Logic Synthesis as an Efficient Means of Minimal Model Discovery from Multivariable Medical Datasets

Niku Gorji
The University of Sydney
niku.gorji@sydney.edu.au

Simon Poon
The University of Sydney
simon.poon@sydney.edu.au

Abstract

In this paper we review the application of logic synthesis methods for uncovering minimal structures in observational/medical datasets. Traditionally used in digital circuit design, logic synthesis has taken major strides in the past few decades and forms the foundation of some of the most powerful concepts in computer science and data mining. Here we provide a review of current state of research in application of logic synthesis methods for data analysis and provide a demonstrative example for systematic application and reasoning based on these methods.

1. Introduction

Constructing data-driven models of complex, real-world problems is the focus of much research across the science and engineering disciplines. In the domain of medicine and health, it is important for any model derived from data to be interpretable and reflective of the known inter-relationships among the variables under investigation.

The problem of model transparency in medical investigations has provided the driving force behind efforts in developing "correct-by-construction" [1] models based on logic synthesis methods.

To facilitate analysis on large, high dimensional datasets, various data mining methods have been proposed to uncover the underlying structure and the links between study variables. These methods include factor analysis, principal component analysis, multidimensional scaling and cluster analysis [2]. The connection between machine learning, knowledge discovery in databases and logic synthesis methods is examined in [3] and [4] demonstrating that the tools and methods employed in these procedures can effectively compete with standard machine learning algorithms.

Originally used by the digital design community to handle the increasing complexity in digital circuits, logical optimization and verification tools under the umbrella of logic synthesis methods have come to a level of maturity thanks to the wealth of research into this area since the 1950s [5].

Given the state of the art and advances of synthesis-based methods for succinct representation and effective handling of large functions, often containing thousands of variables, analyzing the applicability of the latest developments in this area for logical reasoning in medical fields seems promising. However, through our review of recent works we found that research on the parallels between medical reasoning and logic synthesis is limited. In this paper, our objective is to review some of the well-established procedures for construction of interpretable models and generating medical hypothesis and subsequent reasoning.

This paper is organized into seven sections. In Section 2, we briefly provide a background on logic synthesis methods and provide a demonstrative example to elaborate some of the concepts and procedures covered throughout the paper. In Section 3 and 4 we cover respectively the most common data structures and well known algorithms used for these methods. We then review some application in Section 5 and then provide an example of minimal model discovery from a medical dataset in Section 6. Finally, in Section 7, offer some concluding comments with future directions.

2. Overview

Logic synthesis is the process of "automatic production of logic components" [6] based on algebraic notations and Boolean logic minimization and optimization techniques. The foundational core of these methods are built on mathematical notations of Boolean algebra. Application of logic synthesis methods for discovery of patterns or structures in data has been proven useful by various researchers since the 1980s, for example [3], [4], [7], [8], [9], [10]. In order to enhance the scalability and performance of logic synthesis methods, potential

URI: http://hdl.handle.net/10125/50242
ISBN: 978-0-9981331-1-9

enhancements are continuously investigated. Logic synthesis methods can generally be divided into exact and approximate procedures. The exact methods focus on finding a deterministic solution for a given function and therefore face limitations in terms of the size and complexity of the function they can analyze.

The approximate methods apply heuristics to optimize larger functions [11].

In this section we will touch upon some of the well-known methods and for a complete review of some of the recent contributions we refer the reader to [12] and [6].

2.1. Terminology

A Boolean variable is a variable that takes a binary value $B = \{true, false\}$, or $\{1, 0\}$, under a truth assignment. A literal is l a Boolean variable or its complement ($\neg l$). A minterm is a product (AND combination) of terms. An implicants is a minterm that leads to (implies) a specific outcome. On-set specifies the set of minterms that lead to outcome presented by the Boolean variable 1. Off-set minterms are those that lead to outcome 0. 'Don't care' terms are minterms in which their presence or absence in a function is either not specified or is irrelevant.

2.2. Demonstrative example

We use a simple example summarized in Table 1. to provide a demonstrative overview of the methods covered throughout the paper. Consider the study of the expression level of four genes (variables v_1 to v_4) from a sample of 10 hypothetical individuals, 6 of whom are diagnosed with cancer (phenotype) and 4 are controls. The on-set cases represent all cases for which a patient is diagnosed with cancer. The expression levels of genes for which are given below.

Table 1. Example on-set cases

Case	v_1	v_2	v_3	v_4
1	0.17	0.17	0.51	0.51
2	0.17	0.83	0.83	0.51
3	0.83	0.17	0.17	0.17
4	0.83	0.17	0.17	0.51
5	0.83	0.83	0.17	0.17
6	0.51	0.51	0.17	0.83
Mode	0.83	0.17	0.17	0.51

Real life datasets often contain a mixture of data types, i.e., nominal, ordinal, and various numeric types, but since logic synthesis methods are traditionally based on a core of Boolean algebra, the input variables for use with these methods need to be Boolean or dichotomous. This means a pre-processing step for calibration of variables into dichotomies is often required for using these methods with most real life datasets. In practice dichotomizing variables means some information nevertheless will be lost, however, once the underlying minimal structural representations are uncovered, the complete expression of the data can be studied with conventional methods using the minimal structures as a base for reasoning. This lets us appreciate the potential of logic synthesis methods for minimal structural discovery. There are many methods for binary classification of variables that are widely used in machine learning. Classification and regression trees [13] is an example. Using statistical properties such as median and mode and subjective expert evaluation are other methods. For our hypothetical example, we will use a very simple threshold to dichotomize the variables. Any gene with expression level above 0.5 will be represented by 1 or $true$ otherwise will take a 0 or $false$ value. Following this step, the on-set will be represented by the function:

$$f(v_1, v_2, v_3, v_4) = \{0011, 0111, 1000, 1001, 1100, 1101\}$$

We will refer to this function and use it as a demonstrative example in the following sections.

3. Data Structures in Logic Synthesis

There are several data structures and related optimization algorithms for logic synthesis. In this section, we briefly go through some of the most well-known and widely used representations which are mostly based on the introductory work covered in [6] and [14]. In the next section, we will review some of the widely known algorithms that are based on these representations.

3.1. Truth tables

Truth tables are canonical representations of Boolean functions, which means if the truth tables corresponding to two Boolean functions are equivalent, these two functions are equal. In practice, a truth table is only effective in representing functions with a limited number of input variables. Table 2. is an example of a fully specified function, in which all cases have a clear associated outcome. Often times functions representations in logic

synthesis contain don't care terms. Inclusion or exclusion of don't care terms for case-oriented research has been the focus of a substantial amount of investigations, and we refer the reader to obtain a better overview of this matter from [15] and [16].

Table 2. Example Truth Table

v_1	v_2	v_3	v_4	f
0	0	1	1	1
0	1	1	1	1
0	1	0	1	0
1	0	0	0	1
1	0	0	1	1
1	1	1	0	0
1	1	0	0	1
0	1	1	0	0
1	1	0	1	1
0	0	1	0	0

3.2. Sum-of-product (SOP)s

Some of the pioneering methods of logic synthesis are based the representation of Boolean functions in Sum-of-Products (SOP) forms. SOPs are a form of Boolean formulas consisting of disjunction (AND) of conjunction (OR) of literals which are also called product terms or cubes. Literals can also be complemented (i.e., inverted). SOPs are known as disjunctive normal forms (DNF) in computer science. The SOP representation is not canonical, therefore comparison of two function in SOP form requires efficient algorithms for performing Boolean operations. The on-set function of the example given in Section 2.2 using the SOP form can be represented by the following function:

$$f = (v_1 . \neg v_3) + (v_3 . v_4)$$

Determining the minimal form of a function in SOP form is a well-known NP-complete problem [6]

and is known as two-level logic minimization [6].

3.3. Products-of-sums (POS)s

Also known as conjunctive normal forms (CNF) is another representation which unlike the SOP representation is canonical. Canonical formulas are equivalent "if and only if they are syntactically equivalent" [13]. This is a useful property in many application of logic synthesis methods such as functional equivalence checking and verification, but some data structures are not canonical. POSs have found a wide applicability in satisfiability (SAT) checkers. Satisfiability checking refers to determining whether or not the variables of a given Boolean function can be assigned $true$ and $false$ values in such a way that the whole function equates to $true$. Transforming Boolean functions to their CNF representation prior to the application of SAT algorithms creates a more compact representation of the function and can improve satisfiability check runtimes [6]. Therefore, the two-level representation of functions in POS form has become the standard data structure for use in SAT solvers.

Our example on-set function given in Section 2.2 can have the following representation in POS form:

$$f = (v_1 + v_3).(\neg v_3 + v_4)$$

3.4. Binary Decision Diagrams (BDD)s

Another data structure for representing Boolean functions is Binary Decision Diagram [17] and its standardized variant Reduced Ordered Binary Decision Diagram (ROBDD) [18].

Table 3. Data representations for the demonstrative example given in Section 2.2

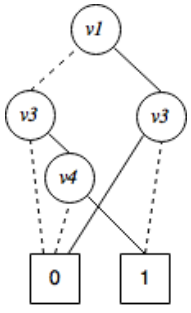

Figure 1. ROBDD

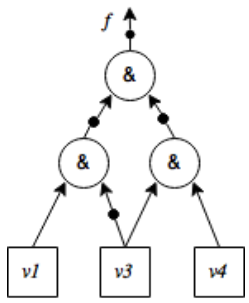

Figure 2. AIG

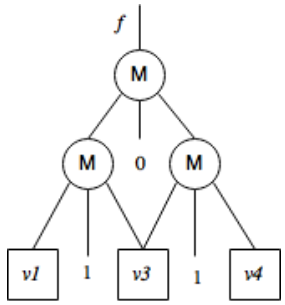

Figure 3. MIG

ROBDDs are canonical representation forms for a provided order of variables and are based on nested if-then-else formulas. BDDs can be recursively decomposed into co-factors until the terminal nodes are reached. This property was first demonstrated by Boole [19] and popularized by Shannon [5] and is called Boole's expansion theorem or Shannon's expansion. Most references to BDDs in the literature refer to their standard reduced and ordered forms.

While the on-set and off-set functions for SOP and POS representations need to be specified separately, a BDD diagram can represent both functions simultaneously. The full representation of our given example function from Section 2.2 (including both off-set and on-set minterms) can be visualized through the reduced binary tree shown in Figure 1. Solid lines represent ifs and dashed line represent the else parts of the formula.

To reduce the depth and complexity of the BDD format, direct acyclic graphs (DAG)s were introduced [21]. DAGs are part of multi-level logic network representations in which AND, OR, complementation and if-then-else operations can be employed as basic functions. These formats can support more scalable optimization and synthesis tools [14].

3.5. AND Inverter Graphs (AIG)s

AIGs are a subset of AND/OR Inverter graphs (AOIG)s [14]. A logic network can be represented through an AIG by decomposing its input functions into two-input AND and Inverters. This form of representation is called a homogenous logic network and refers to DAGs in which all internal nodes represent the same logic function and have the same number of incoming that can appear either in regular or complemented format to preserve universality

AIGs are widely used in logic optimization. AIG optimization algorithms are typically based on structural hashing but they can also incorporate the traditional Boolean techniques. The AIG representation is not canonical. An AIG can have many possible representations. The AIG representation of our example on-set function from Section 2.2 in SOP form:

$$f = (v_1 . \neg v_3) + (v_3 . v_4)$$

can be constructed by converting the OR to AND using the equivalent representation:

$$f = \neg (\neg(v_1 . \neg v_3). \neg (v_3 . v_4))$$

Visualized in Figure 2. The dots on the edges in the AIG diagram represent complements.

3.6. Majority-normal-forms (MNF)s

A fairly recent development in the logic synthesis methods is the introduction of MNFs as an alternative to POS or SOP forms. Majority normal forms are based on majority (a combination of AND and OR operators) and complementation entirely and can be represented with Majority Inverter Graphs(MIG)s [14]. Similar to POS and SOPs, MNFs can represent any Boolean function. MNFs can have compact representations and are suitable for use as an underlying data structure in SAT solvers. The SOP form of our example on-set function given in Section 2.2, in the language of MNF is represented by:

$$M(M(v_1 , \neg v_3, 0), M(v_3, v_{4,}, 0),1)$$

Notice how the AND and OR operators are presented using the majority of two variables and zero or one. Figure 3. demonstrates an example of the Majority Inverter Graphs for our on-set function.

4. Algorithms

4.1. Exact Methods

Logic minimization is used to present a given set of logic function while minimizing a given cost function such as the number of product terms, and is the main part of a logic synthesis process.

Table 4. Time complexity of operations with different structures

	AND	OR	Complement	Satisfiability	Tautology
SOP	quadratic	constant	exponential	constant	coNP-complete
POS	constant	quadratic	exponential	NP-complete	constant
ROBDD	exponential	exponential	constant	constant	constant
AIG	constant	constant	constant	NP-complete	coNP-complete
MNF	constant	constant	constant	NP-complete	coNP-complete

The basic principle in minimization is if two Boolean expressions differ in only one condition yet produce the same outcome, then the condition that distinguishes the two expressions can be considered irrelevant and can be removed.

In two-level logic minimization, the cost function often includes parameters such as the number of product terms or the number of literals.

4.1.1. Karnaugh map. One of the earliest techniques that provides a paper and pen method of minimizing SOP forms is Karnaugh map [22]. In this technique, for an n variables Boolean function, an $n \times n$ matrix is constructed where each element contains the outcome of corresponding minterms. Although simple, this technique is impractical to use with more than 5 variables since the visualization of the dimensions on the map becomes difficult.

4.1.2. Quine-McCluskey. Initially proposed by [23], and then later extended by [24]. The algorithm is used for finding a minimum representation of a Boolean function in SOP form and is based on repeatedly applying a few laws of logic: absorption (e.g. $v_1.\neg v_2 + v_1.v_2 = v_1$), idempotency or redundancy (e.g. $v_1 + v_1 = v_1$), and the law of excluded middle (e.g. $v_1 + \neg v_1 = 1$). The algorithm involves two key steps:

1) Finding all prime implicants (product terms that cannot be minimized any further) from the input Boolean function or the truth table by repeatedly applying the tree laws mentioned above.
2) Using those prime implicants to find the minimum sum or minimum cover of the function. Minimum sum is formed as the sum of the fewest prime implicants which when taken together will equal "one" for all required rows of the table of combinations.

This two–level minimization procedure made Quine-McCluskey the standard algorithm in Boolean minimization for Qualitative Comparative Analysis [25]. However, the second part of the procedure, the set-covering problem, is a well-known NP-Complete problem [6] and is the main bottleneck in many logic synthesis methods that are based on the Quine-McCluskey procedure.

4.2. Approximate Methods

4.2.1. ESPRESSO. Since solving the exact two-level logic minimization problem is a computationally intractable problem, heuristic approaches have been proposed. The family of ESPRESSO logic minimizers [26] including ESPRESSO-II [27] and ESPRESSO-MV [28], was introduced in 1980s, and became the state-of-the-art tool for heuristic two-level logic minimization. ESPRESSO-II combines the two basic steps of Quine-McCluskey algorithm into one, therefore reduces the number of implicants that needs to be processed.

An initial un-optimized cover of the input function is first obtained, and then refinements [27] are iteratively applied to this initial cover to obtain the smallest set of product terms that are still a cover of the input function and cannot be minimized any longer. ESPRESSO-II has three operators in its main algorithm:

1) EXPAND – turns each implicant into a prime implicant by enlarging it
2) IRREDUNDANT – removes redundant implicants to make the cover irredundant
3) REDUCE – modify the cover in a way that is possible to be improved during the next iteration. This is done by giving a weight to implicants and sorting them in descending order, so that the ones that are larger and overlap with many other implicants are processed first.

ESPRESSO-II can deal with large number of variables using a heuristic method to almost always arrive at a near-minimum or minimum solution. However, for functions with more than 100 input variables this algorithm faces quality and runtime difficulty [29]. We used this algorithm for uncovering minimal models from a medical dataset reported in Section 6.

4.2.2. PALMINI. Observing the limitations of two-level logic minimization algorithms, in [30] the concept of minimal implicants was introduced. Instead of solving the covering problem with prime implicants, the authors of PALMINI reduced the minimization problem to that of coloring "the graph of incompatibility of implicants". Coloring a graph is the problem of partitioning a set of nodes according to a given set of rules and is an instance of an NP-complete problem [31]. Incompatible implicants are those that have at least two differing minterms. Compatibility refers to sets of product implicants which are disjoint with the off-set minterms of the function. The underlying theorem behind this method is that the minimal number of compatible sets of product implicants is the minimal cover of the function [30]. For instance, for our example on-set function given in Section 2.2, the matrix of incompatibility and the "Graph of Incompatibility of Implicants" GIM is shown below in Figure 4.

node	1	2	3	4	5	6
1	0	0	1	1	1	1
2		0	1	1	1	1
3			0	0	0	1
4				0	1	0
5					0	0
6						0

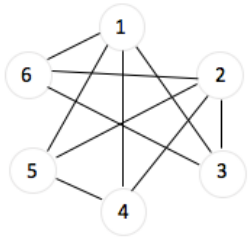

Figure 4. Matrix and Graph of Incompatibility of Implicants for the demonstrative example given in Section 2.2

The resulting graph in Figure 4. is then colored, ensuring that any two incompatible product implicants will be partitioned in two different classes. For the graph above, the minimum number of colors needed is two. We color minterms 1 and 2 with color A, minterms 3 and 4 with color B and finally 5 and 6 with color C. The small dash for the minterm representations below denotes a don't care.

By matching minterms with color A, we get:

$$0011 + 0111 = 0 - 11$$

Matching minterms with color B:

$$1000 + 1001 = 100 -$$

Similarly, by matching minterms with color C:

$$1100 + 1101 = 110 -$$

So $f(v_1, v_2, v_3, v_4) = \{0 - 11, 100 -, 110 -\}$

Which is equal to:

$$(\neg v_1 . v_3 . v_4) + (v_1 . \neg v_2 . \neg v_3) + (v_1 . v_2 . \neg v_3)$$

4.2.3. EXPRESSO-Signature. Inspired by the concept of minimal implicants presented in [30], a new algorithm for exact two-level logic optimization was presented [32] that produces a smaller covering matrix heuristically.

In this method, the unique set of primes forming the covering problem is presented by the largest product of their intersection, called a signature cube. Signature cubes that are not contained in any other cube are called essential, and their set forms the minimum canonical cover that represents the covering problem implicitly.

ESPRESSO-Signature therefore is able to generate a smaller covering matrix without needing to compute all the prime implicants. However, it is still limited by the size of the covering matrix it produces [33].

4.2.4. Scherzo. Since Quine-McCluskey based methods produce the set of prime implicants explicitly, for complex functions in terms of the number of implicants or when the size of the covering matrix is large, these methods become ineffective. Using two different forms of BDDs produces [33] [34] as the underlying data structure, Scherzo represents prime and essential prime implicants implicitly, allowing the minimization of larger, more difficult functions [34]. Scherzo was shown to be as much as a 100 times faster than the previously mentioned methods [34]. However, using BDDs as the underlying data structure means that he complexity of the minimization problem is shifted to generating the BDD representation [6] instead.

4.2.5. BOOM and FC-MIN. Unlike the earlier minimization procedures that produce implicants starting from an initial cover, and reducing to find the minimum cover, BOOM [29] starts from the simplest possible expressions and gradually adds variables until a minimum solution is found. BOOM is positioned for problems with a large number of input variables, but for a large number of outputs the runtime grows rapidly, and the algorithm becomes inefficient.

The second version of BOOM, BOOM-II [35], combines the initial algorithm with another one called FC-Min [36] in order to solve minimization problems with a large number of output variables. It is very fast for this purpose, the runtime grows almost linearly with growing number of both the input and output variables [36].

However, the results this method produces does not reach the quality of other minimizers. Therefore, the developers of this method proposed its application as a pre-processor to other methods such as ESPRESSO to significantly reduce total minimization time, while fully retaining the result quality [35].

4.2.6. SAT-ESPRESSO. A Boolean satisfiability checker accepts a Boolean formula in as input and determines whether or not the formula is satisfiability. Even though determining satisfiability of a Boolean formula in POS form is an NP-complete

problem, SAT checkers perform well with real world formulae containing hundreds of thousands of variables [37]. Replacing the third step of the main algorithm in ESPRESSO-II with a SAT checker, the authors of [37] introduced a new procedure that could perform 5-20 times faster than ESPRESSO-II finding the same cover, and 3-5 times faster compared to BOOM.

5. Applications

Logic Synthesis methods have a wide range of application apart from digital circuit design and verification. For example, in artificial intelligence, reliability analysis and as underlying reasoning engines used in gene-network analysis.

One of the early methods employing the Quine-McCluskey algorithm for analyzing observational data is Qualitative Comparative Analysis, developed by Charles Ragin [15] [16] [38] used for case-oriented research handling a limited number of variables. This method has found a wide range of application among researchers from a variety of disciplines such as social science [39], business and economics [40] [41], management [42], and health policy research [43] among many.

Beside the exact synthesis methods, the heuristic methods capable of handling a larger number of variables and propositional formulas are used for hypothesis generation and model discovery in larger datasets.

The researchers in [7] used the underlying algorithm in an open-source SAT checker package MINISAT [44] to generate minimal models representing the underlying structure in an E.Coli dataset. In their own words, they "translated laws of biochemical reactions into propositional formulas" to compute these minimal models. The results of their analysis is therefore defendable since the generated models are logically sound. They later generalized their method in [8] for prediction of gene knockout effects.

The researchers in [1] used logic synthesis methods based on BDDs and satisfiability checkers for developing a predictive system model and tested it with biological data. In order to achieve high predictability based on observational data, they constructed an algorithm that maximizes the number of don't care (unspecified) terms as long as the model stays satisfiable.

Since logic synthesis methods are based on Boolean logic, the connection between Boolean minimization procedures and multi-valued structure learning is explored in [45] and [9] further demonstrating that these methods can help achieve interpretable models with high accuracy from real world data which is often multi-valued and fuzzy in nature. In a study published in 2016, researchers in [10] presented a methodology using Boolean networks and satisfiability checkers for logic-based synthesis and analysis [46] of gene interaction networks and applied their methodology to derive a highly predictive explanation of known behaviors while using a much smaller set of components and rules compared to the conventional methods.

6. Analyzing Observational Data

We analyzed the dataset of Corticosteroid Randomization after Significant Head Injury (CRASH) trial [47] using an implementation of ESPRESSO-II in R [48]. Study variables include demographics, injury characteristics and computed tomography (CT) findings which are clinically important predictors of TBI outcome [48] [49] [50]. Outcome measure is death or severe disability versus moderate disability or good recovery at 6 months. For a full description of the dataset we refer the readers to [50]. Table 5. lists the 7 variables used in our model. Various statistical techniques have been used in the past to generate explanatory and predictive models for TBI [52] [53]. In contrast to conventional statistical methods, a logical analysis procedure is not concerned with generating a single model from data, rather, focuses on uncovering sets of configurations or models that collectively explain the observed outcomes.

Analysis of 6945 cases with no missing values out of 10008 cases in CRASH trials demonstrated that 9 different configurations of 7 admission variables covered 57% of all cases of favorable outcomes in our sample. Furthermore, 44.5% of the cases of unfavorable outcome could be explained by 20 configurations. These configurations of variables are the prime implicants that collectively cover the whole of our on-set (favorable) or off-set (unfavorable) functions. Variables in the CRASH dataset needed to be dichotomized prior to the application of the ESPRESSO-II algorithm. For this purpose, we used RPART to split ordinal variables in two classes. Binarization of variables let us reach a certain level of generalizability that would be difficult to attain without, as the generated models would become too specific and too complex for interpretation.

To evaluate the usefulness of the logic synthesis method (which we will call Logsyn), we compared its predictive ability with that of a simple Binary Logistic Regression (Logit) model.

Table 5. Model variables

Variable category	Variable (abbreviation)	Category	Total cases with no missing values
Epidemiology	Age (age)	0: Younger than 45	4896
		1: 45 or older	2049
Assessment	Eye opening (eye)	1: No response	2680
		0: Any Response	4265
	Motor response (motor)	1: No response ~ Withdrawal	2456
		0: Localizes or Follows commands	4489
	Verbal response (verbal)	1: No response	3764
		0: Single words or more	3181
	Pupillary response (pupils)	0: Both reactive	5791
		1: No response unilateral or No response	1154
Image findings	Obliterated 3rd ventricle or basal cisterns (oblt)	1: Yes	1663
		0: No	5282
	Midline shift (mdls)	1: Yes	1021
		0: No	5924
Outcome	Outcome at 6 months	0: Death or severe disability	2763
		1: Moderate disability or good recovery	4182

The two methods are based on very different assumptions. The Logit model assumes a linear relationship between independent variables, is additive in nature, and generates one model for the whole dataset. The Logsyn method in contrast, takes configurations of variables in each case into account, outputs multiple models covering parts of the dataset.

The Logit model has an overall predictive accuracy of 0.750 on the whole dataset. On the other hand, Logsyn does not explain the whole dataset in one single model, rather, it produces multiple configurations of variables.

The Logsyn method covers 57% of all cases of favorable outcome with 9 models, and 44.5% of all cases of unfavorable outcome with 20 models, and has an overall prediction accuracy of 0.848 for the fraction of the dataset covered by these models.

For demonstrative purposes, we provide these 9 models covering 57% of favorable outcome cases in the diagram of Figure 5. Dots on edges denote complemented variables. A variable can exist in a model either in its original or complemented form.

215 other models for the cases of favorable outcome and 174 models for cases of unfavorable outcome are needed to explain all of the cases in the dataset according to the Logsyn method. Some of these models cover single occurances of a combination of variables.

These results reflect the findings of our earlier study using the same dataset, but with Quine-McCluskey as the underlying algorithm [54]. For elaboration of the comparison with Logit model, please refer to [54].

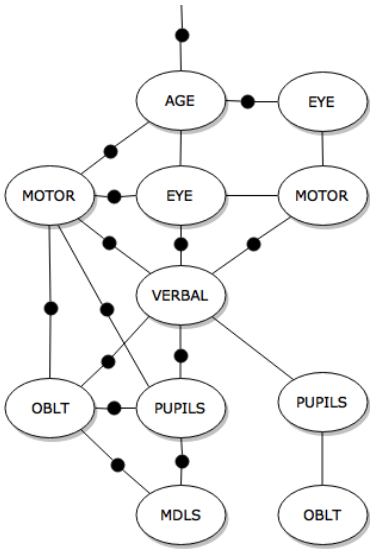

Figure 5. The combined model of configurations for favorable outcome

The results of our analysis are largely in line with the findings of previous TBI studies. Specifying a combination of admission parameters that almost always lead to a specific outcome are appealing from a clinician's perspective and can serve as a set of typical scenarios that are suggestive of favorable versus unfavorable outcome.

7. Concluding Remarks

In this paper we have reviewed some of the important concepts, representations and algorithms for logic synthesis methods and their applications for analyzing observational data for building transparent data-driven models. Logic Synthesis methods can be used to systematically uncover sets of minimal structures that collectively explain a given dataset. Similar to the minimization procedures that is the core of logic synthesis methods for optimizing circuit representations, reducing redundancy in decision systems is handled by minimization of the number of decision rules and unnecessary attributes. Finding the minimum cover of the final Boolean function resembles the discovery of reliable constructs in principal component analysis. Logic synthesis methods have the capability to be applied for dealing with functions with thousands of variables, and therefore are promising. We are currently examining concepts from the field of social network analysis and information theory to evaluate the usefulness of such measures for developing better heuristic algorithms for data analysis. For instance, by modifying the third step of the ESPRESSO-II algorithm to incorporate a measure of Shannon's Entropy [55] while searching for a minimum cover.

8. References

[1] Kugler, Hillel, et al. ""Don't Care" Modeling: A Logical Framework for Developing Predictive System Models," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Heidelberg, 2007.

[2] Dillon WR., Goldstein M., Multivariate analysis: Methods and applications, Wiley, 1984.

[3] Goldman, Jeffrey A., Mark L. Axtell, "On using logic synthesis for supervised classification learning.," in Tools with Artificial Intelligence, 1995.

[4] Files, C M., M A. Perkowski, "Multi-valued functional decomposition as a machine learning method.," in Multiple-Valued Logic, 1998 IEEE

[5] Shannon C. "The synthesis of two-terminal switching circuits." Bell Labs Technical Journal. 1949, 28(1):59-98.

[6] Jiang JH, Devadas S., Logic synthesis in a nutshell. Electronic Deisng Automation: Synthesis, Verification, and Test., K. C. a. Y. C. L.-T. Wang, Ed., Morgan Kaufmann Publishers, 2009.

[7] Soh, T, Inoue K, "Identifying Necessary Reactions in Metabolic Pathways by Minimal Model Generation," ECAI. Vol. 215, 2010.

[8] Sohy, T, et al, "Evaluation of the prediction of gene knockout effects by minimal pathway enumeration.," 2012.

[9] Gobi AF, Pedrycz W. Logic minimization as an efficient means of fuzzy structure discovery. IEEE Transactions on Fuzzy Systems. 2008 Jun;16(3):553-66.

[10] Yordanov, Boyan, et al, "A method to identify and analyze biological programs through automated reasoning.," NPJ systems biology and applications 2, 2016.

[11] R. K. Brayton, F. Somenzi, "An exact minimizer for boolean relations," in Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers, 1989.

[12] S. P. Khatri, K. Gulati, Advanced Techniques in Logic Synthesis, Optimizations and Applications, Springer, 2011.

[13] Breiman L, Friedman J, Stone CJ, Olshen RA., Classification and regression trees, CRC press, 1984.

[14] Amaru, L.G., New Data Structures and Algorithms for Logic Synthesis and Verification., Springer, 2015.

[15] C. C. Ragin, The comparative method: Moving beyond qualitative and quantitative methods, Berkeley: University of California, 1987.

[16] C. C. Ragin, Fuzzy-set social science, University of Chicago Press, 2000.

[17] S. B. Akers, "Binary decision diagrams," IEEE Trans. Computers 27.6, pp. 509-516, 1978.

[18] Bryant, Randal E., "Symbolic Boolean manipulation with ordered binary-decision diagrams.," ACM Computing Surveys (CSUR) 24.3, pp. 293-318, 1992.

[19] G. Boole, An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities, 1854, p. 72.

[21] M. S. K Thulasiraman, Graphs: Theory and Algorithms, Willey, 1992.

[22] M. Karnaugh, "The map method for synthesis of combinational logic circuits.," Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics 72.5, pp. 593-599, 1953.

[23] W. V. Quine, "The problem of simplifying truth functions," The American Mathematical Monthly 59.8, pp. 521-531, 1952.

[24] E. J. McCluskey, "Minimization of boolean functions," Bell system technical Journal 35.6, pp. 1417-1444, 1956.

[25] C. C. Ragin, The comparative method: Moving beyond qualitative and quantitative strategies, Univ of California Press, 2014.

[26] Brayton RK, et.al., “A comparison of logic minimization strategies using ESPRESSO: An APL program package for partitioned logic minimization," in Proceedings of the International Symposium on Circuits and Systems, 1982.

[27] Brayton, RK., G. Hachtel, C. McMullen and A. Sangiovanni, "ESPRESSO-II : A new logic minimizer for programmable logic arrays," in Custom Integrated Circuits Conference, 1984.

[28] R. Rudell A. Sangiovanni-Vincentelli, "Espresso-mv: Algorithms for multiple-valued logic minimization," in Proc. IEEE Custom Integrated Circuits Conf, 1985.

[29] J. Hlavička P. Fišer., "BOOM: a heuristic boolean minimizer," in Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, 2001.

[30] Nguyen, Loc Bao, M. A. Perkowdki, Nahum B. Goldstein, "PALMINI—fast Boolean minimizer for personal computers.," Proceedings of the 24th ACM/IEEE Design Automation Conference., 1987.

[31] Garey, Michael R., David S. Johnson, "The complexity of near-optimal graph coloring," Journal of the ACM (JACM) 23.1, pp. 43-49, 1976.

[32] McGeer, Patrick C., et al., "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions," IEEE Transactions on Very Large Scale Integration (VLSI) Systems 1.4 , pp. 432-440, 1993.

[33] O. Coudert, "Doing Two-Level Logic Minimization 100 Times Faster," SODA, 1995.

[34] O. Coudert, "Two-level logic minimization: an overview." Integration, the VLSI journal 17.2, pp. 97-140, 1994.

[35] P. Fišer, H. Kubátová, "Two-level boolean minimizer boom-ii," in Proc. 6th Int. Workshop on Boolean Problems (IWSBP'04) Freiberg, Germany. Vol. 23. No. 24.9., 2004.

[36] P. Fišer, J. Hlavi H. Kubátová, "FC-Min: A Fast Multi-Output Boolean Minimizer," in Proc. Euromicro Symposium on Digital Systems Design (DSD'03), Antalya (TR), 2003.

[37] Sapra, S, M. Theobald, E. Clarke, "SAT-based algorithms for logic minimization," in 21st International Conference on. IEEE Computer Design, 2003.

[38] C. C. Ragin, Redesigning social inquiry: Fuzzy sets and beyond, University of Chicago Press, 2008.

[39] Cress, D. M., Snow., D. A., "Mobilization at the margins: Resources, benefactors, and the viability of homeless social movement organizations.," American Sociological Review, pp. 1089-1109, 1996.

[40] Evans AJ, Aligica PD., "The spread of the flat tax in Eastern Europe: A comparative study.," Eastern European Economics, pp. 49-67, 2008.

[41] Valliere D, Ni N, Wise S., "Prior relationships and M&A exit valuations: a set-theoretic approach," The Journal of Private Equity, pp. 60-72, 2008.

[42] Greckhamer T, "Cross-cultural differences in compensation level and inequality across occupations: A set-theoretic analysis," Organization Studies, pp. 85-115, 2011.

[43] Thomas J, O’Mara-Eves A, Brunton G., "Using qualitative comparative analysis (QCA) in systematic reviews of complex interventions: a worked example.," Systematic reviews., p. 67, 2014.

[44] Eén, Niklas, Niklas Sörensson, "An extensible SAT-solver.," in International conference on theory and applications of satisfiability testing, 2003.

[45] Gobi, Adam F., W. Pedrycz, "Fuzzy modelling through logic optimization," International Journal of Approximate Reasoning 45.3, pp. 488-510, 2007.

[46] Morris MK, Saez-Rodriguez J, Sorger PK, Lauffenburger DA., "Logic-Based Models for the Analysis of Cell Signaling Networks," Biochemistry, 2010.

[47] Collaborators MC, Perel P, et al. "Predicting outcome after traumatic brain injury: practical prognostic models based on large cohort of international patients," Bmj. Feb 23;336(7641), pp. 425-9, 2008.

[48] W. Stiehl, Truth Table Logic Optimizer, 2016.

[49] Tagliaferri F, et al., "A systematic review of brain injury epidemiology in Europe," Acta neurochirurgica, pp. 255-68, 2006.

[50] Zador Z, Sperrin M, King AT., "Predictors of outcome in traumatic brain injury: new insight using receiver operating curve indices and Bayesian network analysis.," PLoS one, 2016.

[51] Lingsma HF, Roozenbeek B, Steyerberg EW, Murray GD, Maas AI., "Early prognosis in traumatic brain injury: from prophecies to predictions.," The Lancet Neurology, pp. 543-54, 2010.

[52] Hukkelhoven CW, et al. "Predicting outcome after traumatic brain injury: development and validation of a prognostic score based on admission characteristics," Journal of neurotrauma, pp. 1025-39, 2005.

[53] Maas AI, et al., "IMPACT recommendations for improving the design and analysis of clinical trials in moderate to severe traumatic brain injury," Neurotherapeutics, pp. 127-34, 2010.

[54] Gorji, N., Zador, Z., Poon, S.K. (2017), “A Configurational Analysis of Risk Patterns for Predicting the Outcome After Traumatic Brain Injury”, Proceedings of the American Medical Informatics Association (AMIA) 2017

[55] C. Shannon "A Mathematical Theory of Communication". Bell System Technical Journal. 27, 1948