A Continuous Approach to Inductive Inference[*]

Anil P. Kamath
Narendra K. Karmarkar
K.G. Ramakrishnan
Mauricio G.C. Resende

Mathematical Sciences Research Center
AT&T Bell Laboratories, Murray Hill, NJ 07974 USA

March 1991
(Revised Version 1.03 - January 1992)

Abstract

In this paper we describe an interior point mathematical programming approach to inductive inference. We list several versions of this problem and study in detail the formulation based on hidden Boolean logic. We consider the problem of identifying a hidden Boolean function $\mathcal{F} : \{0,1\}^n \to \{0,1\}$ using outputs obtained by applying a limited number of random inputs to the hidden function. Given this input-output sample, we give a method to synthesize a Boolean function that describes the sample. We pose the Boolean Function Synthesis Problem as a particular type of Satisfiability Problem. The Satisfiability Problem is translated into an integer programming feasibility problem, that is solved with an interior point algorithm for integer programming. A similar integer programming implementation has been used in a previous study to solve randomly generated instances of the Satisfiability Problem. In this paper we introduce a new variant of this algorithm, where the Riemannian metric used for defining the search region is dynamically modified. Computational results on 8-, 16- and 32-input, 1-output functions are presented. Our implementation successfully identified the majority of hidden functions in the experiment.

Key words: Inductive Inference, Boolean Function Synthesis, Satisfiability, Artificial Intelligence, Integer Programming, Interior Point Method, Riemannian Geometry.

[*]To appear in *Mathematical Programming*.

1. Introduction

In this paper, we explore the application of a function minimization method, based on *continuous* mathematics to the problem of inductive inference, which in its many forms appears to be an inherently *discrete* problem. Inductive inference is the process of hypothesizing a general rule from examples. This problem is of central importance in artificial intelligence and machine learning. A survey on inductive inference systems is given in [1]. Inductive inference involves the following steps:

- Inferring rules from examples, finding compact abstract models of data or hidden patterns in the data.
- Making predictions based on abstractions.
- By comparing predictions with actual results, one can modify the abstraction. This is the process of learning.
- Designing questions to generate new examples.

In this study, we first focus on an aspect of this problem, concerned with inferring rules from examples or discovering patterns in data. For example, given the sequence $2, 4, 6, 8, \ldots$, we may ask "What comes next?" One could pick any number and justify it by fitting a fourth degree polynomial through the 5 points. However, the answer "10" is considered the most "intelligent". That is so because it is based on the first-order polynomial $2n$, which is linear and hence *simpler* than a fourth degree polynomial. The answer to an inductive inference problem is not unique. In inductive inference, one wants a *simple* explanation that fits a given set of observations. Simpler answers are considered better answers.

A recent body of literature has associated inductive inference with the notion of identification in the limit over the strings of a finite alphabet. In this paper we use the classic notion of inductive inference, as in scientific discovery, where rules are inferred from incomplete observations of the data. We enphasize that the observations are from *incomplete* data sets, differing from the notion of *deductive inference* where a complete set in observed.

One therefore needs a way to measure simplicity. For example, in finite automaton inference, the number of states could be a measure of simplicity. In Boolean circuit inference, the measure could be the number of gates and wires. Inductive inference is in fact an optimization problem, where one wants to maximize simplicity, or find a model no more complex than some specified measure.

There are many ways to formalize this optimization problem. In this paper, we study the most basic Boolean model with single output. In this model there is a black box with n Boolean input variables $x_1, \ldots, x_n$ and a single Boolean output variable y. The black box contains a hidden Boolean function $\mathcal{F} : \{0,1\}^n \rightarrow \{0,1\}$ that maps inputs to outputs. Given a limited number of inputs and corresponding outputs, the question to be answered is: Does there exist a logical circuit with no more than K gates which matches this behavior? If so, what is it?

Because the problem formulation belongs to the domain of discrete mathematics or logic does not necessarily imply that the algorithm for attacking the problem must be restricted to concepts from the same fields. There are many ways of formulating a discrete problem as a continuous minimization problem, but many lead to an excessive, often exponential, number of *spurious* local minima, i.e. those that are not global minima and hence do not correspond to the solution of the original problem. In [10], Karmarkar describes a class of potential functions with good topological properties, such as having connected level sets, which has led to successful interior point methods for several problems, e.g. [12], [9], [13], [18]. Guided by these results, we investigate the application of a similar approach for inductive inference. For related work, see e.g. [2], [5] and [25].

Before we continue, we require some definitions. Consider the Boolean function $\mathcal{F} : \{0,1\}^n \rightarrow \{0,1\}$. An element of the domain of $\mathcal{F}$ is called a *minterm* of $\mathcal{F}$. The set of minterms for which $\mathcal{F}$ evaluates to 1 (0) is called the ON-set (OFF-set). An *incompletely specified* Boolean function is one for which $|\text{ON-set}| + |\text{OFF-set}| < 2^n$. Table 1.1 incompletely specifies a Boolean function with 3 input variables x_1, x_2, x_3 and an output y. In that specification, ON-set $= \{101, 100, 011\}$ and OFF-set $= \{001, 111\}$.

x_1	x_2	x_3	y
1	0	1	1
0	0	1	0
1	0	0	1
1	1	1	0
0	1	1	1

Table 1.1: An example

An *algebraic expression* for an incompletely specified Boolean function $\mathcal{F}$ is a Boolean expression (written with Boolean sums (OR) and products (AND)) that evaluates to 1 (0) for all minterms in the ON-set (OFF-set) and evaluates to either 1 or 0 for all other minterms. An algebraic expression for $\mathcal{F}$ can always be written in sum-of-products (disjunctive normal) form. Each product term in the algebraic sum-of-products expression is called a *disjunct*. Let P_i^0 and P_i^1, respectively, denote the index set of 0 and 1 values of the i-th element of the ON-set of $\mathcal{F}$. The *canonical expansion*,

$$y = \sum_{i \in \text{ON-set}} \left\{ \prod_{j \in P_i^1} x_j \times \prod_{j \in P_i^0} \overline{x}_j \right\}$$

is a sum-of-products algebraic expression for $\mathcal{F}$. The major drawback of the canonical expansion is that it has $|\text{ON-set}|$ disjuncts, each having n variables. The canonical expansion for the function of Table 1.1 is

$$y = x_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_3. \tag{1.1}$$

It requires 3 disjuncts.

Given an ON-set and OFF-set of minterms, the *Boolean Function Synthesis Problem* is to find an algebraic sum-of-products expression for $\mathcal{F}$ having a specified number of disjuncts. The corresponding decision problem is NP-complete [3] [7]. The function of Table 1.1 can be expressed by a Boolean

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	y
1	1	1	1	0	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0	0
1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0
1	1	1	0	1	0	0	1	0	1	0	0	1	1	0	0	0	1	0	0	0
0	1	0	1	0	0	1	1	0	1	0	0	1	1	1	0	0	1	0	1	0
1	1	1	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0
1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	0	1	0	0	1	1	1	1	0	0	1	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	0	0
1	1	1	1	1	0	1	1	1	0	1	0	0	1	0	0	1	0	0	0	0
0	1	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	1	1	1	0	1	0	0	1	1	0	1	1	1	0	0	0	0
1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	1	0	1	1	0	0
1	1	1	0	1	1	0	1	1	1	0	0	1	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	0
0	0	1	1	1	1	0	1	0	0	0	0	0	1	0	0	0	1	1	0	0
0	1	1	1	1	0	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0	1	0	1	1	0	0	0	1	1
1	0	1	0	1	1	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	1	0	1	1
1	1	0	0	1	1	1	1	0	1	1	0	0	1	1	0	0	0	1	1	1
1	1	0	0	0	1	1	0	1	0	0	1	1	0	0	0	1	1	0	1	1
0	1	1	1	0	1	0	1	1	1	0	0	1	1	0	1	0	1	0	0	1
1	1	0	1	0	0	0	0	0	1	0	0	1	0	1	1	0	1	0	1	1
0	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1
0	1	1	1	1	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	1
1	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	0	1	1	1	1
0	1	1	1	0	1	1	0	1	0	1	1	1	0	1	0	0	1	0	0	1
0	1	1	1	1	1	0	0	1	1	0	0	1	1	1	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	1	0	0	0	0	0	1	1	0	1	0	1
1	0	1	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1

Table 1.2: Specification for a 20 variable function with |ON-set| = 14 and |OFF-set| = 16.

expression having only 2 disjuncts, since

$$\begin{aligned} y &= x_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_3 \\ &= x_1\overline{x}_2(x_3+\overline{x}_3) + \overline{x}_1x_2x_3 \\ &= x_1\overline{x}_2 + \overline{x}_1x_2x_3. \end{aligned} \tag{1.2}$$

A closer examination of Table 1.1 shows that the output is independent of input variable x_3 and the function can be further simplified to

$$y = x_1\overline{x}_2 + \overline{x}_1x_2. \tag{1.3}$$

The synthesis of the function in Table 1.1 can be accomplished by inspection. For slightly larger instances, synthesis may not be so trivial. Consider the specification in Table 1.2. An integer programming formulation with 216 variables, 1710 constraints with 4168 nonzero elements in the constraint matrix can be formulated to produce the following 4 term Boolean function for this incomplete specification,

$$y = x_2\overline{x}_{12}x_{19} + \overline{x}_5x_9\overline{x}_{19} + \overline{x}_{14}x_{16}x_{20} + \overline{x}_8x_{15}x_{18}.$$

Note that no more than 3 variables are present in each term, making this circuit easy to implement. Also observe that 9 of the 20 input variables $(x_1, x_3, x_4, x_6, x_7, x_{10}, x_{11}, x_{13}, x_{17})$ are irrelevant. We will describe in this paper such a formulation and show how it can be solved. For this instance, the code used in this study took 22.67 CPU seconds on a VAX 6700.

Our approach in this study is to randomly generate a limited number of inputs and apply them to the hidden logic, thus building an incomplete specification of a Boolean function. For this specification a Boolean algebraic expression is synthesized. Besides the application of this study, the Boolean Function Synthesis Problem has applications in Programmable Logic Arrays (PLA's), an important type of digital integrated circuit design. A Boolean function in sum-of-products form can be implemented on a two level AND/OR circuit. Figures 1.1 and 1.2 show the circuits for the Boolean expressions given in (1.1) and (1.3), respectively. The application of the techniques presented in this paper to circuit design is the subject of future study.

The classical approach to tackle the Boolean Function Minimization Problem (where one wishes to minimize the number of disjuncts in the sum-of-products form) was developed by Quine [19] [20] and McCluskey [14]. To describe the idea behind this approach, a few definitions are first needed. If a product term evaluates to 1 for a given minterm, it is said to *contain* that minterm. An *implicant* of a function $\mathcal{F}$ is a product term that does not contain any minterm of the OFF-set of $\mathcal{F}$. A product term $\mathcal{P}$ is said to *contain* the product term $\mathcal{Q}$ if $\mathcal{P}$ evaluates to 1 for every minterm that $\mathcal{Q}$ evaluates to 1. A *prime implicant* of $\mathcal{F}$ is an implicant that is not contained in any other implicant of $\mathcal{F}$. The Quine–McCluskey method first generates all prime implicants (often an exponential number of them [15]) and then selects a minimum subset of prime implicants that realizes the specification. The generation of the set of prime implicants can be carried out with standard techniques, e.g. [21], [17], [24]. This minimum subset can be found by solving a set covering problem [7]. Exact approaches are limited to instances having few variables. For example, the state-of-the-art code ESPRESSO-EXACT [22] took 18,365.0 VAX 6700 seconds to produce an optimal 3 product-term expression for the 20 variable, 30 minterm problem of Table 1.2. Because exact versions of the Quine-McCLuskey method fail to handle large instances, many heuristic approaches have been developed. These generally generate an initial solution and improve it iteratively. They include MINI [8], PRESTO [4], and ESPRESSO-MV [3]. ESPRESSO-MV is widely used in the circuit design industry.

In this paper we consider an interior point mathematical programming approach to the Boolean Function Synthesis Problem and use this procedure for inductive inference. The synthesis problem is formulated as a type of Satisfiability Problem that can be described as an integer programming problem, using a standard transformation. We apply the interior point algorithm for integer programming described in [10] and [12] to synthesize Boolean functions. In particular, we use an implementation derived from the one described in [9], that is suited for finding a satisfiable truth assignment in instances of the Satisfiability Problem. The interior point algorithm is used to attempt to find a feasible ± 1 integer solution w to the following integer program:

$$\begin{aligned} B^\top w &\leq b \\ w_j &= \pm 1, \quad j = 1, \ldots, n, \end{aligned} \tag{1.4}$$

where $B^\top \in \Re^{m \times n}$, $b \in \Re^m$, and $w \in \Re^n$. In [9] it is shown how the Satisfiability Problem can be formulated as an integer programming problem of this form. Let $A = \left[B \vdots I \vdots -I \right]$, where I is an $n \times n$ identity matrix and let $c^\top = (b^\top, 1, \ldots, 1)$.

Starting with an interior point solution

$$w^0 \in \left\{ w \in \Re^n | A^\top w < c \right\},$$

a trust region method, similar to the one described in Moré and Sorensen [16], is applied to the nonconvex optimization problem

$$\text{minimize } \log(n - w^\top w) - \frac{1}{m}\sum_{k=1}^{m}\log(c_k - a_k^\top w).$$

At each iteration, a quadratic approximation of the potential function

$$\varphi(w) = \log(n - w^\top w) - \frac{1}{m}\sum_{k=1}^{m}\log(c_k - a_k^\top w)$$

is optimized over an ellipsoid inscribed in the polytope defined by

$$\left\{w \in \Re^n | A^\top w \leq c\right\},$$

and centered at the current iterate, to produce a descent direction. The new iterate is determined by moving in that direction by a fixed step length, such that the new point is in the interior of the ellipsoid. A rounding heuristic is applied to the fractional solution and feasibility of the rounded solution is tested. A description of the rounding heuristic and initial interior point solution used in this study can be found in [9].

Traditionally, optimization techniques implicitly use a Euclidean metric defined on the underlying domain. For example, the *steepest descent* direction for a function implicitly assumes a Euclidean metric for measuring the *steepness*. However, there exists a curved Riemannian space that is naturally associated with interior point methods. In [11], a Riemannian metric was introduced for problems formulated with equality and nonnegativity constraints. In this paper, we use a similar metric for problems stated in all-inequality form. This Riemannian metric is used for defining the search region. For a polytope defined by

$$\left\{w \in \Re^n \mid A^\top w \leq c\right\}$$

the associated Riemannian metric is given by $g_{ij}(w)dw_i dw_j$, where

$$g_{ij}(w) = \sum_k \frac{a_{ik}a_{jk}}{d_k^2}$$

and

$$d_k = c_k - a_k^\top w_k$$

is the k-th slack variable. In this metric, an infinitesimal ball centered at a strictly interior point w is given by

$$\sum_{i,j} g_{ij}(w)\ dw_i\ dw_j \leq \epsilon^2.$$

An exact search over a ball in this Riemannian metric would have taken into account the variation of the metric $g_{ij}(w)$ within the ball. For a ball of finite radius r, working in this metric is computationally difficult. Therefore, we approximate $g_{ij}(w)$ by a constant $g_{ij}(w^0)$, where w^0 is the center of the ball, and use it throughout the region. The resulting ball is given by

$$\sum_{i,j} g_{ij}(w^0)(w - w^0)_i(w - w^0)_j \leq r^2, \tag{1.5}$$

which is in fact an ellipsoid. A more accurate approximation is possible using higher order terms. In this paper, we report computational results obtained by using the approximation of the Riemannian ball with the Euclidean ellipsoid.

An outline of the remainder of this paper is given next. In Section 2 we formulate the problem of synthesis of Boolean functions as a Satisfiability Problem. In Section 3 we describe how the interior-point algorithm augments the system of inequalities dynamically. This dynamic change of metric has not been used in previous investigations of our algorithm. In Section 4 we present computational results. Concluding remarks are made in Section 5.

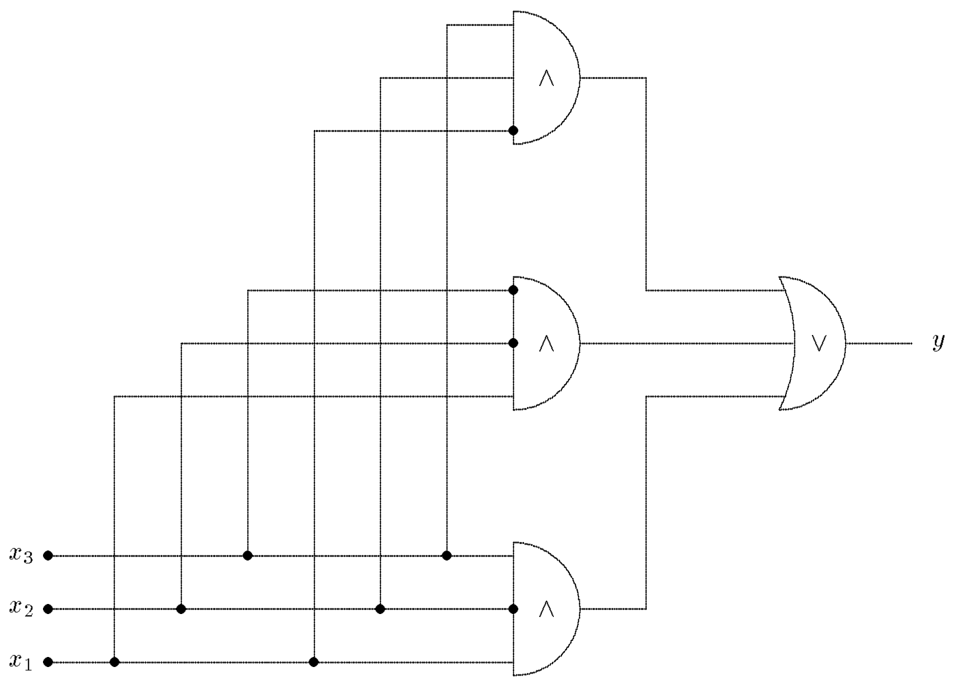

Figure 1.1: Circuit representation of Boolean expression $y = x_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_3$.

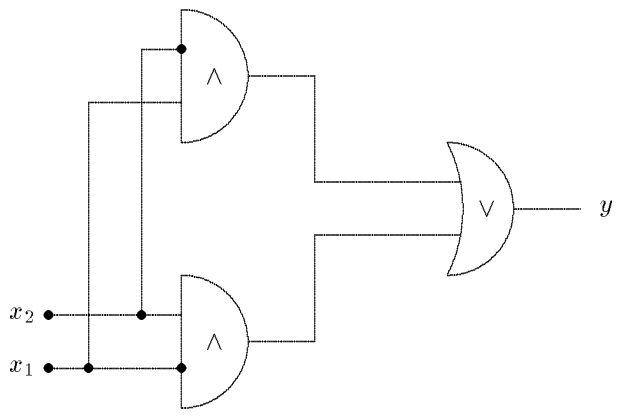

Figure 1.2: Circuit representation of Boolean expression $y = x_1\overline{x}_2 + \overline{x}_1x_2$.

2. Satisfiability Problem formulation

In this Section, we formulate the Boolean Function Synthesis Problem as a Satisfiability Problem. We then use a standard transformation, described in [9], to derive the integer programming formulation for the problem.

Before stating the Satisfiability Problem we need some definitions. Let z be a Boolean variable. Boolean variables can be combined by the logical connectives OR ($\vee$), AND ($\wedge$) and NOT ($\overline{z}$) to form Boolean formulae. A variable or a single negation of the variable is called as *literal*. A Boolean formula consisting of only literals combined by the $\vee$ operator is called a *clause*. The Satisfiability Problem (SAT) can now be stated.

SAT: Given m clauses $\mathcal{C}_1, \ldots, \mathcal{C}_m$ involving n variables $z_1, \ldots, z_n$, does the formula

$$\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \cdots \wedge \mathcal{C}_m \tag{2.1}$$

evaluate to 1 for some input vector? If so, we say that the formula is satisfiable. Otherwise, it is said to be unsatisfiable. To prove satisfiability, it suffices to produce an input vector that evaluates expression (2.1) to 1.

Let us consider the three layer Boolean circuit representation of a Boolean expression and use that representation to aid our modeling. Figure 2.1 illustrates this Boolean circuit. We assume that as input we are given the ON-set and OFF-set of the Boolean function $\mathcal{F}$ that we wish to synthesize. The function has n input variables $x_1, \ldots, x_n$ and a single output variable y and we require a sum-of-products expression with K product terms. Let $R = |\text{OFF-set}|$ be the number of minterms in the OFF-set and $A = |\text{ON-set}|$ be the number of minterms in the ON-set. The three layered circuit has in the output layer an OR gate with a single output y. This OR gate takes its input from K AND gates in the middle layer. Each AND gate in the middle layer takes as its input the output of the n pairs of OR gates from the input layer. Each of these n pairs of gates in the input layer corresponds to an input variable. One gate has input in unnegated form (x_i), while the other in negated ($\overline{x}_i$). Finally, each input layer OR gate has two inputs. The one that corresponds to the unnegated input variable has that variable and a decision variable s as inputs. The other has as input the negated variable and another decision variable s'. We wish to construct a three layer logical circuit with K AND gates that realizes the given specification.

We identify AND gates as $\wedge_1, \ldots, \wedge_K$. The output of all AND gates is input into the output layer OR gate $\vee_0$. The OR gates from the input layer are $\vee_{ji}^k$, for $j = 1, \ldots, K$, $i = 1, \ldots, n$ and $k = 0, 1$. Gate $\vee_{ji}^0$ has as input $\overline{x}_i$ and the decision variable s'_{ji} with its output serving as input to gate $\wedge_j$. Similarly, gate $\vee_{ji}^1$ has as input x_i and the decision variable s_{ji} and its output is input to gate $\wedge_j$.

Decision variables s_{ji} and s'_{ji} determine which literal (x_i or $\overline{x}_i$, if any) is part of the j-th disjunctive term,

$$s_{ji} = \begin{cases} 0 & \text{if } x_i \text{ is in the } j\text{-th disjunctive term} \\ 1 & \text{if } x_i \text{ is not in the } j\text{-th disjunctive term} \end{cases}$$

$$s'_{ji} = \begin{cases} 0 & \text{if } \overline{x}_i \text{ is in the } j\text{-th disjunctive term} \\ 1 & \text{if } \overline{x}_i \text{ is not in the } j\text{-th disjunctive term.} \end{cases}$$

Consequently, s_{ji} and s'_{ji} also determine the number of disjunctive terms. This unnatural choice of values for s and s' allows the use of OR gates in the input layer. Note that if x_i is present in the j-th product term, the output of input gate $\vee^1_{ji}$ is x_i, while if it is not in the term, the output is 1 and x_i has no effect on the output of the middle layer AND gate $\wedge_j$. A similar behavior is observed for s' and $\overline{x}$.

Since x_i and $\overline{x}_i$ cannot simultaneously be part of the j-th disjunctive term, we have that

$$s_{ji} \vee s'_{ji}, \quad i = 1, \ldots, n, \; j = 1, \ldots, K \tag{2.2}$$

must be satisfied. Note that if both x_i and $\overline{x}_i$ are in the j-th disjunctive term, then $s_{ji} = 0$ and $s'_{ji} = 0$ and consequently $s_{ji} \vee s'_{ji} = 0$.

We group the remaining clauses into two classes, corresponding to: (1) OFF-set minterms; (2) ON-set minterms. We first consider the case of OFF-set minterms. For every OFF-set minterm $r = 1, \ldots, R$, let P_r be the set of indices of x for which $x_i = 1$ in the minterm, and $\overline{P}_r$ be the set of indices of x for which $x_i = 0$. Since $y = 0$, then the output of every AND gate must be 0. Consequently, for every AND gate $\wedge_j$, $j = 1, \ldots, K$, at least one input must be 0, i.e.

$$(\bigvee_{i \in P_r} \neg s'_{ji}) \vee (\bigvee_{i \in \overline{P}_r} \neg s_{ji}), \quad j = 1, \ldots, K, \; r = 1, \ldots, R \tag{2.3}$$

must be satisfied.

We next consider the case corresponding to ON-set minterms. In this case, at least one output of every AND gate has value 1. Let the output of AND gate $\wedge_j$ for minterm a be the Boolean variable z^a_j. Consider now the input layer OR gate pair $\vee^1_{ji}$ and $\vee^0_{ji}$ for minterm a. If $x_i = 1$ in minterm a, then the output of $\vee^1_{ji}$ will be 1 and the value of s_{ji} will be irrelevant. The output of $\vee^0_{ji}$ is s'_{ji}. Similarly, if $x_i = 0$ in minterm a, then the output of $\vee^0_{ji}$ will be 1 and the value of s'_{ji} will be irrelevant. The output of $\vee^1_{ji}$ is s_{ji}.

Consequently, since the outputs of the OR pair are combined by the AND operator, the resulting value is s_{ji} if $x_i = 0$ in minterm a or s'_{ji} if $x_i = 1$. To model this, define

$$\sigma^a_{ji} = \begin{cases} s'_{ji} & \text{if } x_i = 1 \text{ in ON-set minterm } a \\ s_{ji} & \text{if } x_i = 0 \text{ in ON-set minterm } a. \end{cases}$$

Variables z^a_j and σ^a_{ji}, $j = 1, \ldots, K$, $a = 1, \ldots, A$, must satisfy

$$z^a_j = \bigwedge_{i=1}^{n} \sigma^a_{ji}. \tag{2.4}$$

To guarantee that $z^a_j = 1$, for at least one gate $\wedge_j$, we require that

$$\bigvee_{j=1}^{K} z^a_j, \quad a = 1, \ldots, A \tag{2.5}$$

be satisfied. Furthermore, to satisfy (2.4) we must disallow $z^a_j = 1$ when, for any $i = 1, \ldots, n$, $\sigma^a_{ji} = 0$. This can be accomplished with the following clauses,

$$\sigma^a_{ji} \vee \neg z^a_j, \quad i = 1, \ldots, n, \; j = 1, \ldots, K, \; a = 1, \ldots, A. \tag{2.6}$$

Summarizing, in the Satisfiability Problem we wish to find Boolean values for s_{ji}, s'_{ji} and z^a_j, such that the following clauses all evaluate to 1.

$$s_{ji} \vee s'_{ji} \quad , \qquad i = 1, \ldots, n,\ j = 1, \ldots, K \tag{2.7}$$

$$(\bigvee_{i \in P_r} \neg s'_{ji}) \vee (\bigvee_{i \in \overline{P}_r} \neg s_{ji}), \qquad j = 1, \ldots, K,\ r = 1, \ldots, R \tag{2.8}$$

$$\bigvee_{j=1}^{K} z^a_j, \qquad a = 1, \ldots, A \tag{2.9}$$

$$\sigma^a_{ji} \vee \neg z^a_j, \qquad i = 1, \ldots, n,\ j = 1, \ldots, K,\ a = 1, \ldots, A. \tag{2.10}$$

This Satisfiability Problem has $(2n + A)K$ variables, $K(n(A + 1) + R) + A$ clauses, and $K(2n(1 + A) + nR + A)$ literals.

Proposition: Satisfiability Problem (2.7-2.10) is satisfiable if $K = A$.

Proof: The canonical expansion has A disjunctive terms and realizes the i/o specification. □

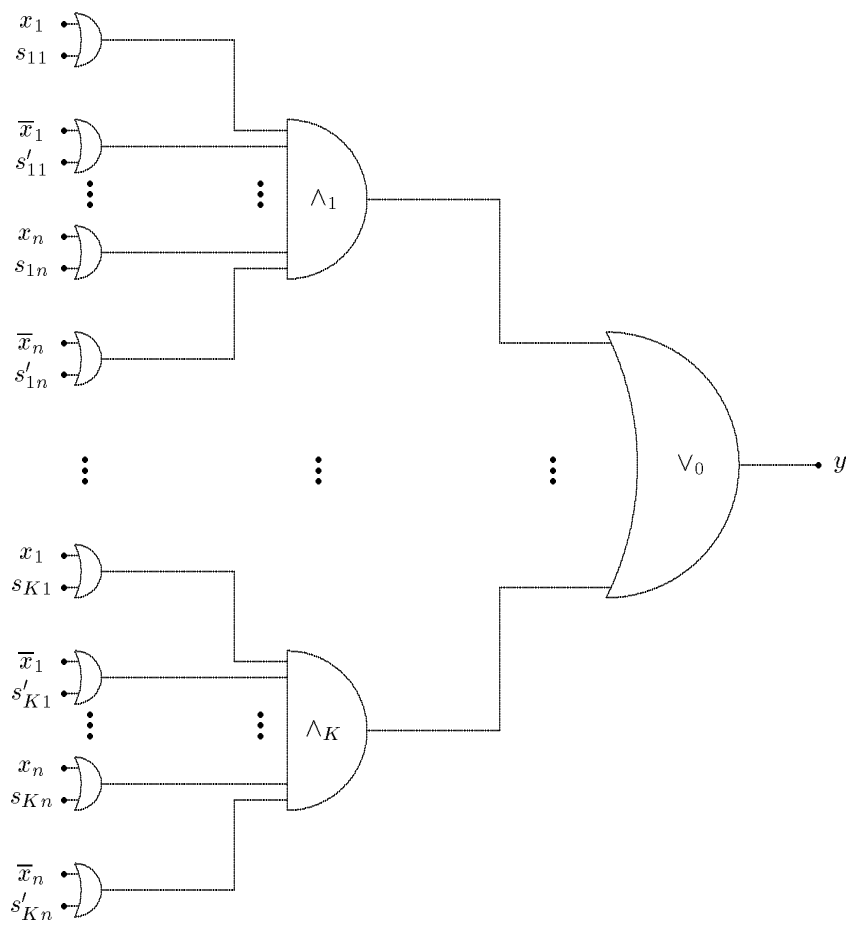

Figure 2.1: 3-layer Boolean circuit

2.1. An example

We now formulate the Satisfiability Problem for the example of Table 2.1. We seek a 2 product-

x_1	x_2	x_3	y
1	0	1	1
0	0	1	0
1	0	0	1
1	1	1	0
0	1	1	1

Table 2.1: An example

term expression to realize this function. In this example, $n = 3$, $K = 2$, $R = 2$, $A = 3$, $P_1 = \{3\}$, $\overline{P}_1 = \{1, 2\}$, $P_2 = \{1, 2, 3\}$, and $\overline{P}_2 = \emptyset$. The ON-set $= \{101, 100, 011\}$ and consequently $\sigma^1_{j1} = s'_{j1}$, $\sigma^1_{j2} = s_{j2}$, $\sigma^1_{j3} = s'_{j3}$, $\sigma^2_{j1} = s'_{j1}$, $\sigma^2_{j2} = s_{j2}$, $\sigma^2_{j3} = s_{j3}$, $\sigma^3_{j1} = s_{j1}$, $\sigma^3_{j2} = s'_{j2}$, $\sigma^3_{j3} = s'_{j3}$.

The Satisfiability Problem has 18 variables, 31 clauses and 66 literals. Clauses of type (2.7) are:

$$s_{11} \vee s'_{11}$$

$$s_{21} \vee s'_{21}$$

$$s_{12} \vee s'_{12}$$

$$s_{22} \vee s'_{22}$$

$$s_{13} \vee s'_{13}$$

$$s_{23} \vee s'_{23}$$

Clauses of type (2.8) are:

$$\neg s'_{13} \vee \neg s_{11} \vee \neg s_{12}$$

$$\neg s'_{23} \vee \neg s_{21} \vee \neg s_{22}$$

$$\neg s'_{11} \vee \neg s'_{12} \vee \neg s'_{13}$$

$$\neg s'_{21} \vee \neg s'_{22} \vee \neg s'_{23}$$

Clauses of type (2.9) are:

$$z^1_1 \vee z^1_2$$

$$z^2_1 \vee z^2_2$$

$$z^3_1 \vee z^3_2$$

Finally, clauses of type (2.10) are:

$$s'_{11} \vee \neg z^1_1$$

$$s'_{11} \vee \neg z^2_1$$

$$s_{11} \vee \neg z_1^3$$
$$s'_{21} \vee \neg z_2^1$$
$$s'_{21} \vee \neg z_2^2$$
$$s_{21} \vee \neg z_2^3$$
$$s_{12} \vee \neg z_1^1$$
$$s_{12} \vee \neg z_1^2$$
$$s'_{12} \vee \neg z_1^3$$
$$s_{22} \vee \neg z_2^1$$
$$s_{22} \vee \neg z_2^2$$
$$s'_{22} \vee \neg z_2^3$$
$$s'_{13} \vee \neg z_1^1$$
$$s_{13} \vee \neg z_1^2$$
$$s'_{13} \vee \neg z_1^3$$
$$s'_{23} \vee \neg z_2^1$$
$$s_{23} \vee \neg z_2^2$$
$$s'_{23} \vee \neg z_2^3$$

The algebraic expression $y = x_1\overline{x}_2 + \overline{x}_1 x_2$ that realizes the function in Table 2.1 corresponds to a solution to the above Satisfiability Problem with $s_{11} = s'_{21} = s'_{12} = s_{22} = 0$, $s'_{11} = s_{21} = s'_{12} = s_{13} = s'_{13} = s_{12} = s'_{22} = s_{23} = s'_{23} = 1$, $z_1^1 = z_1^2 = z_2^3 = 1$, and $z_1^3 = z_2^1 = z_2^2 = 0$.

2.2. Integer programming formulation

The Satisfiability Problem can be formulated as an integer programming feasibility problem. Let $\mathcal{C}_1, \ldots, \mathcal{C}_m$ denote the m clauses of a Satisfiability Problem with n variables $z_1, \ldots, z_n$. Denote

$$I_{\mathcal{C}} = \{i \mid z_i \text{ is in clause } \mathcal{C}\}$$

$$J_{\mathcal{C}} = \{i \mid \overline{z}_i \text{ is in clause } \mathcal{C}\}.$$

Associate with each Boolean literal z_i, the integer variable w_i, such that

$$w_i = \begin{cases} 1 & \text{if } z_i = 1 \\ 0 & \text{if } z_i = 0, \end{cases}$$

and with each Boolean literal $\overline{z}_i$ the integer variable $\overline{w}_i = 1 - w_i$. To satisfy each clause $\mathcal{C}$, it is necessary that at least one literal with index in $I_{\mathcal{C}}$ or in $J_{\mathcal{C}}$ be set to 1, i.e.

$$\sum_{j \in I_C} w_j + \sum_{j \in J_C} \overline{w}_j \geq 1,\ \mathcal{C} = \mathcal{C}_1, \ldots, \mathcal{C}_m.$$

Substituting for $\overline{w}_i$, we get

$$\sum_{j \in I_C} w_j - \sum_{j \in J_C} w_j \geq 1 - |J_{\mathcal{C}}|,\ \mathcal{C} = \mathcal{C}_1, \ldots, \mathcal{C}_m.$$

Applying the above transformation to the Satisfiability Problem formulated in Section 2, we get the following integer programming feasibility problem: Find $(0,1)$ values for s_{ji}, s'_{ji} and z_j^a, for $i = 1, \ldots, n$, $j = 1, \ldots, K$, $a = 1, \ldots, A$, such that:

$$\begin{aligned} s_{ji} + s'_{ji} &\geq 1,\ \ i = 1, \ldots, n,\ j = 1, \ldots, K \\ \sum_{i \in P_r} s'_{ji} + \sum_{i \in \overline{P}_r} s_{ji} &\leq n - 1,\ \ r = 1, \ldots, R,\ j = 1, \ldots, K \\ \sum_{j=1}^{K} z_j^a &\geq 1,\ \ a = 1, \ldots, A \\ z_j^a &\leq \sigma_{ji}^a,\ \ i = 1, \ldots, n,\ j = 1, \ldots, K,\ a = 1, \ldots, A. \end{aligned}$$

3. Dynamic change of metric

In this Section we consider the integer program where one wants to find $w \in \{-1,1\}^n$ such that

$$A^\top w \leq c, \tag{3.1}$$

where $A^\top$ is an $m \times n$ real matrix and c a real m vector. We assume that (3.1) includes $2n$ box constraints of the type

$$-1 \leq w_j \leq 1,\ j = 1,\ldots,n.$$

Let

$$\begin{aligned} \mathcal{I} &= \{w \in \{-1,1\}^n \mid A^\top w \leq c\} \\ \mathcal{L} &= \{w \in \Re^n \mid A^\top w \leq c\} \\ \mathcal{S} &= \{w \in \Re^n \mid A^\top w < c\} \end{aligned}$$

denote the sets of solutions of the integer program, solutions of the linear programming relaxation of the integer program, and interior point solutions of the linear programming relaxation, respectively.

In [12], Karmarkar, Resende and Ramakrishnan introduce an interior point algorithm for integer programming, based on finding a local minimum of the potential function

$$\varphi(w) = \log\left\{\frac{n - w^\top w}{\prod_{k=1}^m (c_k - a_k^\top w)^{1/m}}\right\}$$

by an iterative method. Given an interior solution $w \in \mathcal{S}$, the denominator of the log term of $\varphi(w)$ is the geometric mean of the slacks and is maximized at what is known as the center of the polytope defined by $\mathcal{L}$. Since $-e \leq w \leq e$, where $e = (1,\ldots,1)^\top$, the numerator is minimized at a ± 1 vertex of $\mathcal{L}$. In [12] it is shown that if $\mathcal{I} \neq \emptyset$, w^* is a global minimum of this potential function if and only if $w^* \in \mathcal{I}$.

Let $w^k \in \mathcal{S}$ denote the current iterate. The next iterate w^{k+1} is obtained by moving in a descent direction Δw from w^k, i.e. a direction such that $\varphi(w^{k+1}) = \varphi(w^k + \alpha\Delta w) < \varphi(w^k)$, where α is an appropriate step length. Each iteration of our method is similar to the trust region approach described in Moré and Sorensen [16], except that the construction of the search region is based on making good global approximations to the polytope, instead of being based on local considerations as in a trust region method. A descent direction Δw of the potential function is found by searching over a region that is an approximation, like (1.5), to the Riemannian ball. The Riemannian metric is based on an augmented system of inequalities

$$\mathcal{A}^\top w \leq c,$$

where $\mathcal{A} = \left[A \vdots \tilde{A}\right]$, A is the matrix corresponding to the original system of linear inequalities and $\tilde{A}$ corresponds to the augmented part of $\mathcal{A}$. The procedure for dynamic augmentation will be described later. For a given matrix A, the ellipsoidal approximation (as in (1.5)) to the Riemannian ball is given by

$$(\Delta w)^\top A D_A^{-2} A^\top \Delta w \leq r^2,$$

where $D_A = \operatorname{diag}(c_1 - a_1^\top w^k, \ldots, c_m - a_m^\top w^k)$. Hence, the problem to be solved at each iteration is:

$$\text{minimize} \ \frac{1}{2}(\Delta w)^\top H \Delta w + h^\top \Delta w \tag{3.2}$$

$$\text{subject to:} \ (\Delta w)^\top A D_A^{-2} A^\top \Delta w \leq r^2 < 1. \tag{3.3}$$

where H and $h^\top$ are the Hessian and gradient of $\varphi(w)$, respectively. Ye [26] has shown that (3.2-3.3) can be solved in polynomial time. Our algorithm, which is similar to Ye's, solves (3.2-3.3) by solving a series of systems of linear equations of the form

$$(H + \mu A D_A^{-2} A^\top)\Delta w = -h,$$

where $\mu > 0$ is a real scalar. This system arises from the first-order Karush-Kuhn-Tucker optimality condition for (3.2-3.3).

To find a descent direction Δw for the potential function, the algorithm in this paper solves

$$\text{minimize} \ \frac{1}{2}(\Delta w)^\top H \Delta w + h^\top \Delta w \tag{3.4}$$

$$\text{subject to:} \ (\Delta w)^\top \mathcal{A} D_{\mathcal{A}}^{-2} \mathcal{A}^\top \Delta w \leq r^2 < 1, \tag{3.5}$$

where

$$D_{\mathcal{A}} = \begin{bmatrix} D_A & 0 \\ 0 & D_{\tilde{A}} \end{bmatrix},$$

and $D_{\tilde{A}} = \operatorname{diag}(\tilde{c}_1 - \tilde{a}_1^\top w^k, \ldots, \tilde{c}_{m'} - \tilde{a}_{m'}^\top w^k)$ are slacks with respect to the m' added constraints. The system of equations that is to be solved has the form

$$(H + \mu \mathcal{A} D_{\mathcal{A}}^{-2} \mathcal{A}^\top)\Delta w = -h. \tag{3.6}$$

Let $f_0 = n - {w^k}^\top w^k$. Substituting H and $\mathcal{A} D_{\mathcal{A}}^{-2} \mathcal{A}^\top$ in (3.6) we get

$$\begin{aligned} \Delta w &= -\left[-\frac{2}{f_0} I - \frac{4}{f_0^2} w^k {w^k}^\top + \frac{1}{m} A D_A^{-2} A^\top + \mu A D_A^{-2} A^\top + \mu \tilde{A} D_{\tilde{A}}^{-2} \tilde{A}^\top \right]^{-1} h \\ &= -\left[-\frac{2}{f_0} I - \frac{4}{f_0^2} w^k {w^k}^\top + (\frac{1}{m} + \mu) A D_A^{-2} A^\top + \mu \tilde{A} D_{\tilde{A}}^{-2} \tilde{A}^\top \right]^{-1} h \end{aligned}$$

As in [12], let $\gamma = 1/(\mu + \frac{1}{m}))$, then

$$\begin{aligned} \Delta w &= -\left[-\frac{2}{f_0} I - \frac{4}{f_0^2} w^k {w^k}^\top + \frac{1}{\gamma} A D_A^{-2} A^\top + \frac{1}{\gamma}(1 - \frac{\gamma}{m}) \tilde{A} D_{\tilde{A}}^{-2} \tilde{A}^\top \right]^{-1} h \\ &= -\left[-\frac{2\gamma}{f_0} I - \frac{4\gamma}{f_0^2} w^k {w^k}^\top + A D_A^{-2} A^\top + (1 - \frac{\gamma}{m}) \tilde{A} D_{\tilde{A}}^{-2} \tilde{A}^\top \right]^{-1} \gamma h \\ &= -\left[-\frac{2\gamma}{f_0} I - \frac{4\gamma}{f_0^2} w^k {w^k}^\top + \mathcal{A} {D'}_{\mathcal{A}}^{-2} \mathcal{A}^\top \right]^{-1} \gamma h, \end{aligned}$$

where

$${D'}_{\mathcal{A}}^{-2} = \begin{bmatrix} D_A^{-2} & 0 \\ 0 & (1 - \frac{\gamma}{m}) D_{\tilde{A}}^{-2} \end{bmatrix}.$$

This system of linear equations has the same structure as the linear system solved in [12] and therefore the same computational machinery used there can be used to solve this system. In Section 4, we report computational results of an interior point code that uses a diagonal preconditioned conjugate gradient algorithm used in [9] to solve this system.

We conclude this section by describing the dynamic augmentation of the system of linear inequalities. Our objective is to modify the metric of the descent direction search so as to change the region where the search for the descent direction of the potential function is carried out. The potential function is *not* modified, as is the case in the restart procedure of [12], and hence, H and $h^\top$ are left unchanged. Let $w^k \in \mathcal{S}$ denote the current iterate, an interior non-integral solution. Define, for $j = 1, \ldots, n$,

$$\tilde{a}_j = \begin{cases} +1 & \text{if } w_j^k > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Periodically, we add the constraint

$$\tilde{a}^\top w \leq \tilde{c} = n - 2 \tag{3.7}$$

to the system of linear inequalities used to define the ellipsoid in (3.3). Assuming $\tilde{a} \notin \mathcal{I}$ (otherwise the algorithm terminates with $w = \tilde{a}$ as the solution), it is shown in [12] that constraint (3.7) excludes the infeasible solution $\tilde{a}$, but no other integer feasible solution. In the computational results reported in Section 4 we add a constraint of this type every 10 iterations of the interior point algorithm. Another strategy is to add a constraint every time the interior solution changes orthant, i.e. every time there is a change in the sign of a component of the interior solution.

4. Computational results

In this Section, we report on the computational testing of the integer programming algorithm described in [12] and [9] with the modification of Section 3 on randomly generated instances of a class of inductive inference problems. As in [9], we use rounding scheme $\mathcal{B}$, a greedy scheme based on the ordering of the fractional values of the variables of the current interior solution, and offer no special treatment for local minima. To offer some insight as to how the interior point algorithm compares with a standard Satisfiability algorithm we run an implementation of the Davis-Putnam algorithm [6] on the test problems. We use the C language implementation by Selman, Mitchell and Levesque [23].

The experiment was conducted on a VAX 6700 running 10th Edition UNIX®. The code is written in FORTRAN and C and was compiled on the `f77` and `cc` compilers with the optimization flag `-O` set. Running times are measured with the system function `times()`.

We consider the problem of inferring the logic in an n-input, 1-output "black box", with $n = 8, 16,$ and 32. For each value of n we consider five black boxes. For each black box, we record the output corresponding to randomly generated input samples and build an incomplete truth table, for which we attempt to synthesis a Boolean expression having a given number of product terms. Tables 4.1, 4.4 and 4.7 show problem statistics. For each black box, these tables display the hidden logic, identify the problem instance with a name (8A1, 8A2, ..., 16A1, 16A2, ...,32A1, 32A2, ...), and display the number of random i/o samples generated, the specified number of product terms in the synthesized expression (AND gates), and statistics for the corresponding Satisfiability Problem (number of variables, clauses and literals per clause). Tables 4.2, 4.5 and 4.8 show statistics for the solution obtained with the integer programming algorithm, the inferred logic and the accuracy of the inferred logic. This accuracy (prediction accuracy) is measured by randomly generating 10,000 input samples and comparing outputs for the black box and the inferred logic.

Tables 4.3, 4.6 and 4.9 show statistics for the Davis-Putnam algorithm.

We make the following observations regarding the computational testing.

- Satisfiability problems having from 66 variables and 186 clauses, up to 1,728 variables and 24,792 clauses were solved. Of the 41 problems, 15 were solved in less than one CPU minute. Only 6 of the 41 problems required more than 10 CPU minutes.
- On all instances tested, the integer programming code synthesized algebraic expressions that produced the correct output, given as input the minterms in the incompletely specified truth table.
- Input-output sample sizes varied from 10 to 1000. Increasing sample size increases the Satisfiability Problem size, but generally increases prediction accuracy and frequently reduces CPU time. For example, problem 8A3 took 29.77s and 40 iterations with a sample size of 50, while 8A4 (with same hidden logic as 8A3) required only 9.33s and a single iteration with a sample size of 100.
- For all instance classes but one (32E), the algorithm correctly predicted the hidden logic. For the instance where an exact prediction was not obtained, the algorithm inferred logic with

.98 accuracy. Note that for this instance the algorithm produced an algebraic expression that realizes the function described in the incomplete truth table. The expression produced had as few terms as the hidden function. Increasing the number of input-output samples up to 1000 did not improve the prediction accuracy. With truth tables having more than 1000 input-output examples, the algorithm converged to local minima. However, the number of violated constraints of the integer solution corresponding to the local minima is small (less than 100 constraints out of over 20,000). Consequently, the resulting circuit, while not realizing the given truth table, may still be a good approximation to the hidden logic.

- The number of input-output samples required to achieve a good prediction accuracy was small. All 8-input hidden logic that was correctly predicted required no more than 100 samples, except 8B, that required 200. 90% prediction accuracy was obtained with as few as 50 samples. For the 16-input logic, 16B, 16C and 16E required 400 input samples for an exact prediction, while 16A and 16D required 100 and 200, respectively. For the four 32-input logic instances for which exact predictions were obtained (32A1, 32B4, 32C4, 32D3), the required number of samples was 250, 300, 1000 and 400, respectively. In several instances, as few as 50 and 100 samples sufficed to obtain predictions with over 80% and 90% accuracy, respectively.

- As shown in Section 2, for a fixed number of input-output examples, the number of variables and clauses in the Satisfiability Problem grows linearly with Kn, where K is the number of gates and n is the number of inputs to the black box. For this reason, we limited the number of gates in the 32-input instances to 3 or 4, even though for the 8- and 16-input instances we allowed circuits with as many as 20 gates.

- Even though in many instances the number of product terms allowed was much higher than the number present in the hidden logic, the algorithm produced functions with duplicate and covered terms, resulting in expressions with few terms. For example, for problem 16A1 we used $K = 15$, but the algorithm synthesized an expression with only 4 terms. As mentioned in the previous bullet item, because of memory limitations, we could not search for expressions with a large number of product terms in the 32-input instances.

- Tables 4.3, 4.6 and 4.9 illustrate the difficulty encountered by the Davis-Putnam algorithm to find satisfiable assignments for the Satisfiability problems formulated in the computational experiment. We limited the running time for each instance to 12 CPU hours (43200 s). In that running time the Davis-Putnam code succeeded in finding satisfiable assignments only for the smaller Satisfiability problems. It found assignments for all the 8-input instances, the smallest (in the number of variables) of the 16-input instances and 5 of the 6 smallest 32-input instances. It should be noted that since the 16-input instances had larger values of K (AND gates) than the 32-input instances, the Satisfiability problems corresponding to the 16-input instances are larger than those of the 32-input instances.

- Since the bulk of the computational effort in this code corresponds to solving a system of linear equations with the conjugate gradient method, a parallel implementation of such procedure is expected to speed up the solution process significantly.

hidden logic	instance id	i/o samples	AND gates	Satisfiability Problem vars	clauses	lit/clause
$y = x_4\bar{x}_7 + \bar{x}_3x_4 + x_1x_2\bar{x}_6$	8A1	10	3	66	186	2.4
	8A2	25	6	180	800	2.6
	8A3	50	6	264	1552	2.6
	8A4	100	6	396	2798	2.7
$y = \bar{x}_1\bar{x}_4x_6 + \bar{x}_2x_8 + x_2$	8B1	50	3	168	1054	2.2
	8B2	100	6	576	4088	2.3
	8B3	150	10	1360	10100	2.3
	8B4	200	6	1068	8214	2.2
$y = x_5 + x_6\bar{x}_8 + x_7$	8C1	50	10	510	3065	2.4
	8C2	100	10	950	6689	2.3
$y = \bar{x}_6 + \bar{x}_2 + \bar{x}_3\bar{x}_7$	8D1	50	10	530	3207	2.3
	8D2	100	10	930	6547	2.3
$y = x_8 + x_2x_5 + \bar{x}_3x_5$	8E1	50	10	520	3136	2.4
	8E2	100	10	870	6121	2.4

Table 4.1: Problem statistics: 8-input, 1-output variables

instance id	iters	CPU	inferred logic	prediction accuracy
8A1	1	0.42s	$y = x_4\bar{x}_7 + \bar{x}_3x_4 + \bar{x}_3\bar{x}_6$	.86
8A2	47	21.37s	$y = x_4\bar{x}_7 + \bar{x}_3x_4 + x_1\bar{x}_3$	.87
8A3	40	29.77s	$y = x_4\bar{x}_7 + \bar{x}_3x_4 + x_1x_2$	.92
8A4	1	9.33s	$y = x_4\bar{x}_7 + \bar{x}_3x_4 + x_1x_2\bar{x}_6$	exact
8B1	1	2.05s	$y = x_2 + x_8$	.97
8B2	59	97.07s	$y = x_2 + x_8 + \bar{x}_1x_3\bar{x}_5x_6$	.97
8B3	37	167.07s	$y = x_2 + x_8 + x_3\bar{x}_5x_6\bar{x}_7$	.96
8B4	33	122.62s	$y = x_2 + x_8 + \bar{x}_1\bar{x}_4x_6$	exact
8C1	1	8.02s	$y = x_5 + x_7$	.94
8C2	28	84.80s	$y = x_5 + x_7 + x_6\bar{x}_8$	exact
8D1	86	116.98s	$y = \bar{x}_6 + \bar{x}_2 + \bar{x}_3\bar{x}_7$	exact
8D2	13	45.72s	$y = \bar{x}_6 + \bar{x}_2 + \bar{x}_3\bar{x}_7$	exact
8E1	90	122.82s	$y = x_8 + x_2x_5 + \bar{x}_3x_5$	exact
8E2	1	16.68s	$y = x_8 + x_2x_5 + \bar{x}_3x_5$	exact

Table 4.2: Interior point solution statistics: 8-input, 1-output variables

instance id	CPU time
8A1	0.11s
8A2	25.15s
8A3	77.98s
8A4	43.05s
8B1	0.25s
8B2	2.87s
8B3	6.00s
8B4	9.48s
8C1	2.20s
8C2	5.62s
8D1	9.53s
8D2	11.78s
8E1	4.35s
8E2	8.35s

Table 4.3: Davis-Putnam solution statistics: 8-input, 1-output variables

	instance	i/o	AND	Satisfiability Problem		
hidden logic	id	samples	gates	vars	clauses	lit/clause
$y = x_1\bar{x}_{12} + x_2x_3\bar{x}_5 + x_9 + \bar{x}_7$	16A1	100	15	1650	19368	2.3
	16A2	300	6	1602	23281	2.3
$y = x_3x_{12}x_{15} + \bar{x}_3\bar{x}_{11}+$	16B1	300	8	1728	24792	2.6
$\bar{x}_2\bar{x}_{10}\bar{x}_{16} + x_1x_2$	16B2	400	4	1076	16121	2.6
$y = x_4\bar{x}_7x_{11} + x_4x_{10}x_{14}+$	16C1	100	20	1580	16467	3.0
$\bar{x}_9\bar{x}_{14}x_{15} + \bar{x}_3x_8$	16C2	400	4	924	13803	2.8
$y = \bar{x}_5\bar{x}_8\bar{x}_{10}x_{16} + \bar{x}_2\bar{x}_{12}\bar{x}_{16}+$	16D1	200	10	1230	15901	3.0
$\bar{x}_1\bar{x}_{12} + x_3\bar{x}_5x_6$	16D2	400	4	836	12461	3.0
$y = x_1\bar{x}_2x_3\bar{x}_4 + x_5x_6\bar{x}_7x_8+$	16E1	200	15	1245	14766	4.2
$x_9\bar{x}_{10}\bar{x}_{11}\bar{x}_{12} + \bar{x}_{13}x_{14}\bar{x}_{15}x_{16}$	16E2	400	4	532	7825	4.2

Table 4.4: Problem statistics: 16-input, 1-output variables

instance id	iters	CPU	inferred logic	prediction accuracy
16A1	264	2038.55s	$y = x_1\bar{x}_{12} + x_2x_3\bar{x}_5 + x_9 + \bar{x}_7$	exact
16A2	78	607.80s	$y = x_1\bar{x}_{12} + x_2x_3\bar{x}_5 + x_9 + \bar{x}_7$	exact
16B1	1	78.27s	$y = \bar{x}_3\bar{x}_{11} + x_1x_2 + x_3x_{12}x_{15} + \bar{x}_2\bar{x}_{10}\bar{x}_{16} + \bar{x}_2x_3x_6\bar{x}_{10}x_{11}$	.99
16B2	39	236.07s	$y = x_3x_{12}x_{15} + \bar{x}_3\bar{x}_{11} + \bar{x}_2\bar{x}_{10}\bar{x}_{16} + x_1x_2$	exact
16C1	105	757.55s	$y = \bar{x}_1x_4\bar{x}_9 + x_4\bar{x}_7x_{11} + x_6x_8x_{10}x_{13} + \bar{x}_3x_8 + x_3\bar{x}_6\bar{x}_8x_{12}x_{15}$	.87
16C2	98	520.60s	$y = x_4\bar{x}_7x_{11} + x_4x_{10}x_{14} + \bar{x}_9\bar{x}_{14}x_{15} + \bar{x}_3x_8$	exact
16D1	215	1546.78s	$y = \bar{x}_1\bar{x}_{12} + \bar{x}_2\bar{x}_{12}\bar{x}_{16} + \bar{x}_5\bar{x}_8\bar{x}_{10}x_{16} + x_3\bar{x}_5x_6$	exact
16D2	106	544.25s	$y = \bar{x}_1\bar{x}_{12} + \bar{x}_2\bar{x}_{12}\bar{x}_{16} + \bar{x}_5\bar{x}_8\bar{x}_{10}x_{16} + x_3\bar{x}_5x_6$	exact
16E1	231	2156.42s	$y = x_1\bar{x}_4x_8x_9x_{11}\bar{x}_{13} + x_1\bar{x}_2x_3\bar{x}_4 + \bar{x}_{13}x_{14}\bar{x}_{15}x_{16} + x_9\bar{x}_{10}x_{11}\bar{x}_{12} + x_5x_6\bar{x}_7x_8$	.99
16E2	89	375.83s	$y = x_1\bar{x}_2x_3\bar{x}_4 + x_5x_6\bar{x}_7x_8 + x_9\bar{x}_{10}\bar{x}_{11}\bar{x}_{12} + \bar{x}_{13}x_{14}\bar{x}_{15}x_{16}$	exact

Table 4.5: Interior point solution statistics: 16-input, 1-output variables

instance id	CPU time
16A1	$*$
16A2	$*$
16B1	$*$
16B2	$*$
16C1	$*$
16C2	$*$
16D1	$*$
16D2	$*$
16E1	$*$
16E2	20449.20s

$*$ Did not find satisfiable assignment in 43200s.

Table 4.6: Davis-Putnam solution statistics: 16-input, 1-output variables

	instance	i/o	AND	Satisfiability Problem		
hidden logic	id	samples	gates	vars	clauses	lit/clause
$y = x_1\bar{x}_{12} + x_2\bar{x}_5x_{32} + x_{19}\bar{x}_{23}x_{26}$	32A1	250	3	459	9212	3.6
$y = x_1x_2\bar{x}_9\bar{x}_{12}x_{31} + x_{19}\bar{x}_{23}x_{26}+$	32B1	50	3	228	1374	4.5
$x_2\bar{x}_5\bar{x}_{20}x_{32}$	32B2	100	3	261	2558	4.7
	32B3	250	3	348	5734	5.1
	32B4	300	3	381	6918	5.1
$y = x_2\bar{x}_9\bar{x}_{12}x_{31} + x_2\bar{x}_{20}x_{32}+$	32C1	50	3	225	1280	4.8
$x_1x_2x_{19}\bar{x}_{23}x_{26}$	32C2	100	3	249	2182	5.3
	32C3	150	3	279	3272	5.3
	32C4	1000	3	759	20862	5.5
$y = x_4x_{11}\bar{x}_{22} + x_2x_{12}\bar{x}_{15}\bar{x}_{29}+$	32D1	50	4	332	2703	3.4
$\bar{x}_3x_9x_{20} + \bar{x}_{10}x_{11}\bar{x}_{29}x_{32}$	32D2	100	4	404	5153	3.5
	32D3	400	4	824	19478	3.6
$y = x_9x_{10}x_{23} + x_2x_{29}\bar{x}_{31}+$	32E1	50	3	222	1186	5.0
$x_2\bar{x}_4x_6\bar{x}_7x_{19}\bar{x}_{32}$	32E2	100	3	267	2746	4.5
	32E3	200	3	330	5680	4.8
	32E4	300	3	387	7106	5.0
	32E5	400	3	450	9380	5.0

Table 4.7: Problem statistics: 32-input, 1-output variables

instance id	iters	CPU	inferred logic	prediction accuracy
32A1	44	176.73s	$y = x_1\bar{x}_{12} + x_2\bar{x}_5x_{32} + x_{19}\bar{x}_{23}x_{26}$	exact
32B1	1	5.02s	$y = \bar{x}_{12}\bar{x}_{23}x_{26} + x_{19}\bar{x}_{23}x_{26}+ x_{14}\bar{x}_{17}\bar{x}_{21}x_{31}$	.83
32B2	32	56.65s	$y = x_2\bar{x}_5\bar{x}_{20}x_{32} + x_{19}\bar{x}_{23}x_{26}+ x_2x_3\bar{x}_{14}\bar{x}_{21}x_{22}$	.96
32B3	51	189.80s	$y = x_{19}\bar{x}_{23}x_{26} + x_2\bar{x}_5\bar{x}_{20}x_{32}+ x_2\bar{x}_9\bar{x}_{10}\bar{x}_{12}\bar{x}_{14}\bar{x}_{21}$	.97
32B4	61	259.43s	$y = x_{19}\bar{x}_{23}x_{26} + x_2\bar{x}_5\bar{x}_{20}x_{32}+ x_1x_2\bar{x}_9\bar{x}_{12}x_{31}$	exact
32C1	23	23.85s	$y = x_2x_{19}x_{32} + x_2\bar{x}_7\bar{x}_8x_{13}+ x_2x_7x_{15}\bar{x}_{29}$	.80
32C2	1	9.38s	$y = x_2\bar{x}_{20}x_{32} + x_2x_{18}x_{19}+ x_2x_5x_7x_{14}\bar{x}_{17}\bar{x}_{29}$	.88
32C3	1	14.27s	$y = x_2\bar{x}_{20}x_{32} + x_2\bar{x}_{18}\bar{x}_{21}\bar{x}_{23}x_{32}+ x_2x_{15}x_{18}x_{19}\bar{x}_{24}\bar{x}_{30}x_{31}$	.92
32C4	1	154.62s	$y = x_2\bar{x}_{20}x_{32} + x_2\bar{x}_9\bar{x}_{12}x_{31}+ x_1x_2x_{19}\bar{x}_{23}x_{26}$	exact
32D1	49	65.65s	$y = \bar{x}_{22}x_{28}\bar{x}_{29} + x_{12}\bar{x}_{17}\bar{x}_{25}x_{27}+ \bar{x}_3x_9x_{20} + x_{11}x_{12}\bar{x}_{16}\bar{x}_{32}$	.74
32D2	78	178.10s	$y = x_9x_{11}\bar{x}_{22}\bar{x}_{29} + x_4x_{11}\bar{x}_{22}+ \bar{x}_3x_9x_{20} + x_{12}\bar{x}_{15}\bar{x}_{16}\bar{x}_{29}$	.91
32D3	147	1227.40s	$y = x_4x_{11}\bar{x}_{22} + \bar{x}_{10}x_{11}\bar{x}_{29}x_{32}+ \bar{x}_3x_9x_{20} + x_2x_{12}\bar{x}_{15}\bar{x}_{29}$	exact
32E1	5	8.33s	$y = x_2x_{29}\bar{x}_{31} + \bar{x}_{11}\bar{x}_{26}x_{29}\bar{x}_{31}$	.86
32E2	1	9.67s	$y = x_2x_{29}\bar{x}_{31} + x_2x_5x_{23}\bar{x}_{31}+ x_9x_{10}x_{23}$	.97
32E3	40	132.83s	$y = x_2x_{29}\bar{x}_{31} + x_2\bar{x}_4x_6\bar{x}_7\bar{x}_{30}+ x_9x_{10}x_{23}$	.98
32E4	63	276.93s	$y = x_2x_{29}\bar{x}_{31} + x_9x_{10}x_{23}+ x_2\bar{x}_4x_6\bar{x}_7\bar{x}_{13}\bar{x}_{30}$	.98
32E5	71	390.22s	$y = x_2x_{29}\bar{x}_{31} + x_9x_{10}x_{23}+ x_2\bar{x}_4\bar{x}_7x_{12}\bar{x}_{13}\bar{x}_{30}$	.98

Table 4.8: Interior point solution statistics: 32-input, 1-output variables

instance id	CPU time
32A1	$*$
32B1	159.68s
32B2	$*$
32B3	$*$
32B4	$*$
32C1	18.62s
32C2	176.32s
32C3	$*$
32C4	$*$
32D1	$*$
32D2	$*$
32D3	$*$
32E1	54.57s
32E2	144.40s
32E3	$*$
32E4	$*$
32E5	$*$

$*$ Did not find satisfiable assignment in 43200s.

Table 4.9: Davis-Putnam solution statistics: 32-input, 1-output variables

5. Concluding remarks

Inductive inference is an essential ingredient in systems that exhibit truly intelligent behavior. Since *intelligent* answers to questions are often ranked by their simplicity, optimization plays an important role in effective inductive inference systems. In this paper, we applied a method of continuous function minimization to implement an inductive inference system.

This approach is based on an integer programming formulation for Boolean function synthesis. The introduction of on-line cuts, described in Section 3, enabled our implementation to solve a larger number of problems than our previous implementation, used in [9].

The preliminary computational results presented in this study indicate the feasibility of this approach to inductive inference. In forthcoming papers, we will use similar methods to approach other inference problems. These include:

Multiple output Boolean model. This model is similar to the single output Boolean model, except that the hidden function is of the form $\mathcal{F} : \{0,1\}^n \rightarrow \{0,1\}^m$, i.e. it has m outputs.

Boolean model with correlated variables. This model has no input or output variables. Instead, one is given generic Boolean variables and examples of variable values, with the objective of inferring a relationship between the variables.

Threshold gate model. Given a black box with a fixed network of threshold gates (in place of Boolean gates) and a partial list of input-output examples, the problem is to infer the weights so that the input-output sample is duplicated by the inferred network.

Continuous inputs. All of the above models can be formulated with continuous inputs in place of Boolean inputs.

Finite state machine model. A finite state machine is defined by an input alphabet Σ, a state set S, a start state $s_0 \in S$, a transition function f_t and a set of accept states $A \subseteq S$. Let $L \subseteq \Sigma^*$ be the language accepted by this finite state machine, where Σ^* is the set of all strings formed from the input alphabet Σ. Given two sets of strings U, T, where $U \subseteq L$ and $T \subseteq \Sigma^* - L$, one wishes to pose the question: Does there exist a k-state deterministic finite automaton whose behavior matches the partially specified finite state machine?

Probabilistic model. In this model one is given n events $E_1, E_2, \ldots, E_n$ and a sample of states, each indicating for each event, if that event has occurred or not. One wishes to infer compact relations between events, such as $E_1 \Leftrightarrow (E_2 \vee E_3) \wedge E_4$, that hold with high probability.

Markov chain model. In this model, a hidden Makov chain outputs elements from an output alphabet with each state transition. This output is observed and one wishes to infer the matrix of one-step transition probabilities.

Approximate model. In this model, noise is present in the observation of the input and output of the hidden logic in the black box. One wishes to infer this hidden logic so that with high probability the inferred logic will reproduce the observed input-output patterns.

References

[1] D. Angluin and C.H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15:237–265, 1983.

[2] E. Boros, P.L. Hammer, and J.N. Hooker. Predicting cause-effect relationships from incomplete discrete observations. Technical report, RUTCOR, Rutgers University, Piscataway, NJ, 1991.

[3] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI minimization*. Kluwer Academic, 1985.

[4] D.W. Brown. A state-machine synthesizer-SMS. In *Proceedings of the 18th Design Automation Conference*, pages 301–304, June 1981.

[5] Y. Crama, P.L. Hammer, and T. Ibaraki. Cause-effect relationships and partially defined Boolean functions. *Annals of O.R.*, 16:299–325, 1988.

[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[7] J.F. Gimpel. A method of producing a boolean function having an arbitrarily prescribed prime implicant table. *IEEE Trans. Computers*, 14:485–488, 1965.

[8] S.J. Hong, R.G. Cain, and D.L. Ostapko. MINI: A heuristic approach for logic minimization. *IBM J. Res. Develop.*, pages 443–458, Sept. 1974.

[9] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishan, and M.G.C. Resende. Computational experience with an interior point algorithm on the Satisfiability problem. *Annals of O.R.*, 25:43–58, 1990.

[10] N. Karmarkar. An interior-point approach to NP-complete problems. *Contemporary Mathematics*, 114:297–308, 1990.

[11] N. Karmarkar. Riemannian geometry underlying interior-point methods for linear programming. *Contemporary Mathematics*, 114:51–75, 1990.

[12] N.K. Karmarkar, M.G.C. Resende, and K.G. Ramakrishan. An interior point algorithm to solve computationally difficult set covering problems. *Mathematical Programming*, 52:597–618, 1991.

[13] N.K. Karmarkar, M.G.C. Resende, and K.G. Ramakrishnan. An interior-point approach to the maximum independent set problem in dense random graphs. In *Proceedings of the XV Latin American Conference on Informatics*, pages 241–260, July 1989.

[14] E.J. McCluskey. Minimization of Boolean functions. *Bell Syst. Tech. J.*, 35:1417–1444, 1956.

[15] R.E. Miller. *Switching theory, Vol.1: Combinatorial circuits*. John Wiley and Sons, 1965.

[16] J.J. Moré and D.C. Sorensen. Computing a trust region step. *SIAM J. Sci. Stat. Comput.*, 4:553–572, 1983.

[17] E. Morreale. Recursive operators for prime implicant and irredundant normal form determination. *IEEE Trans. Comput.*, C-19:504, 1970.

[18] R. Pai, N. Karmarkar, and S.S.S.P. Rao. A global router based on Karmarkar's interior point method. Technical report, CSE, Indian Institute of Technology, April 1988.

[19] W.V. Quine. The problem of simplifying truth functions. *Am. Math. Monthly*, 59, 1952.

[20] W.V. Quine. A way to simplify truth functions. *Am. Math. Monthly*, 62, 1955.

[21] J.P. Roth. A calculus and an algorithm for the multiple-output 2-level minimization problem. Technical Report RC 2007, IBM Thomas J. Watson Research Center, 1968.

[22] R. Rudell and A. Sangiovanni-Vincentelli. Exact minimization of multiple-valued functions for PLA optimization. In *Proceedings of the IEEE Int. Conf. Computer-Aided Design*, pages 352–355, Nov 1986.

[23] B. Selman, D. Mitchell, and H.J. Levesque. A new method for solving large Satisfiability problems. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1991.

[24] J.R. Slagle, C.L. Chang, and R.C.T. Lee. A new algorithm for generating prime implicants. *IEEE Trans. Comput.*, C-19:304, 1970.

[25] E. Triantaphyllou, A.L. Soyster, and S.R.T. Kumara. Generating logical expressions from positive and negative examples via a branch-and-bound approach. Technical report, Industrial and Management Systems Engineering, Pennsylvania State University, University Park, PA, 1991.

[26] Y. Ye. On the interior algorithms for nonconvex quadratic programming. Technical report, Integrated Systems Inc., Santa Clara, CA, 1988. To appear in *Mathematical Programming.*