To appear in Combinatorica. A preliminary version of this paper appeared in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1994.

Independence and Port Oracles for Matroids, with an Application to Computational Learning Theory

Collette R. Coullard*
Department of Industrial Engineering and Management Sciences
Northwestern University
Evanston, IL 60208

Lisa Hellerstein†
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

Abstract

Given a matroid M with distinguished element e, a *port oracle* with respect to e reports whether or not a given subset contains a circuit that contains e. The first main result of this paper is an algorithm for computing an e-based ear decomposition (that is, an ear decomposition every circuit of which contains element e) of a matroid using only a polynomial number of elementary operations and port oracle calls. In the case that M is binary, the incidence vectors of the circuits in the ear decomposition form a matrix representation for M. Thus, this algorithm solves a problem in computational learning theory; it learns the class of *binary matroid port* (BMP) functions with membership queries in polynomial time. In this context, the algorithm generalizes results of Angluin, Hellerstein, and Karpinski [1], and Raghavan and Schach [17], who showed that certain subclasses of the BMP functions are learnable in polynomial time using membership queries. The second main result of this paper is an algorithm for testing independence of a given input set of the matroid M. This algorithm, which uses the ear decomposition algorithm as a subroutine, uses only a polynomial number of elementary operations and port oracle calls. The algorithm proves a constructive version of an early theorem of Lehman [13], which states that the port of a connected matroid uniquely determines the matroid.

1 Introduction

Given a matroid M with distinguished element e, the *port* of M with respect to e is the set of circuits of M that contain e. A *port oracle* for M with respect to e reports whether or not a given subset contains a circuit that contains e. In this paper we present port oracle algorithms for solving two problems.

We begin with an algorithm that computes an e-based ear decomposition (that is, an ear decomposition every circuit of which contains element e) of a matroid. It runs in polynomial time (in the

*Research partially funded by NSF PYI Grant No. DDM-91-96083
†Research partially funded by NSF Grant No. CCR-92-10957

number of elements of the matroid), and performs a polynomial number of port oracle calls. The algorithm also has an application in the field of computational learning theory; it learns the class of *binary matroid port* functions with membership queries in polynomial time. (This learning theory application was described in a preliminary version of this paper [8]). The algorithm generalizes results of Angluin, Hellerstein, and Karpinski [1], and Raghavan and Schach [17], who showed that certain subclasses of the BMP functions are learnable in polynomial time using membership queries.

We also present an algorithm that simulates an independence oracle for a connected matroid M using a port oracle for M with respect to a fixed element e. This algorithm, which uses our first algorithm as a subroutine, makes a polynomial number of port oracle calls and runs in polynomial time. Thus port oracles and independence oracles are polynomially equivalent for connected matroids. It is well known that other matroid oracles, such as basis, circuit, and rank oracles are polynomially equivalent to independence oracles. The question of oracle equivalence for general independence systems (not just matroids) has been studied by Hausmann and Korte [7]. Our algorithm proves a constructive version of an early theorem of Lehman [13], which states that the port of a connected matroid uniquely determines the matroid.

Before presenting our algorithms, we begin in Section 2 by discussing relevant previous work in computational learning theory, and describe the relation of our ear decomposition algorithm to that work. In Section 3, we explain the terminology from matroid theory that is used in this paper. Then, in Section 4 we define the computational model that we use to determine the running time of our algorithms. The remainder of the paper is devoted to the algorithms and conclusions.

2 Computational learning theory and binary matroid port functions

In this section we describe the relation of our ear decomposition algorithm to problems in computational learning theory. A brief explanation of the relevant terms from matroid theory can be found in the next section.

A binary matroid port (BMP) function is a boolean function that is computed by the port oracle of a binary matroid. More precisely, it is a boolean function $f : \{0,1\}^V \rightarrow \{0,1\}$ such that $V = \{x_1, \ldots, x_n\}$, and there exists a binary matroid M on the elements $V \bigcup \{e\}$ (where $e \notin V$), such that for all $a \in \{0,1\}^V$, $f(a) = 1$ iff the port oracle of M with respect to e would output "yes" on the set $\{x_i | a(x_i) = 1\} \bigcup \{e\}$.

BMP functions are *monotone* boolean functions. A boolean function $f : \{0,1\}^V \rightarrow \{0,1\}$ is monotone if given any pair $a, a' \in \{0,1\}^V$ such that for all $x_i \in V$, $a(x_i) \leq a'(x_i)$, the following relation holds: $f(a) \leq f(a')$.

Lehman gave a characterization of BMP functions in terms of minterms and maxterms [13]. A subset $S \subseteq V$ is a *minterm* of a monotone boolean function $f : \{0,1\}^V \rightarrow \{0,1\}$ if for all assignments $a \in \{0,1\}^V$ setting the variables in S to 1, $f(a) = 1$, and S is minimal with respect to this property. A *maxterm* of f is defined dually; $T \subseteq V$ is a *maxterm* of f if for all assignments $a \in \{0,1\}^V$ setting the variables in T to 0, $f(a) = 0$, and T is minimal with respect to this property. Lehman proved that a BMP function is a monotone boolean function with the property that for every minterm S of the function, and for every maxterm T of the function, $|S \bigcap T|$ is odd.

Seymour gave a forbidden minor characterization of *binary clutters* [19]. A binary clutter is the set of all minterms of a BMP function.

Our ear decomposition algorithm gives an algorithm for learning BMP functions in the *membership query* model of learning. The membership query model is one of the simplest models in computational learning theory. In this model, the learning algorithm must identify a hidden function f taken from a fixed (and known) class of functions C. The algorithm is given the domain of f and has access to a black box which answers *membership queries* for f. A membership query asks for the value of f on a point in its domain. Using membership queries, the algorithm must determine and output (a representation of) the function f. In the context of algebraic functions, this model is sometimes called *black box interpolation.* An algorithm *learns a class C of functions with membership queries* if, given access to a membership oracle for any function f in C, the algorithm will output (a representation of) the function f. When C is the class of binary matroid port functions, then the membership oracle for a function $f \in C$ is essentially equivalent to the port oracle for the associated binary matroid M with respect to e.

There are few classes of boolean functions known to be learnable in polynomial time with membership queries. Angluin, Hellerstein, and Karpinski gave a polynomial time membership query algorithm for learning the class of read-once formulas over the basis (AND, OR) [1]. A boolean formula over (AND,OR) is a rooted tree whose internal nodes are each labeled AND or OR, and whose leaves are labeled with variables. It computes a boolean function in the natural way, from leaves to root. A formula is *read-once* if no variable appears at more than one leaf. Read-once formulas are an example of a class of formulas learnable in polynomial time with membership queries, but not in Valiant's PAC learning model [1] (unless RP = NP).

Raghavan and Schach subsequently showed that a generalization of read-once (AND,OR) formulas, the class of switch configurations, is also learnable in polynomial time with membership queries [17]. A switch configuration is a monotone boolean function that is defined by a connected graph containing two distinguished vertices s and t. The variables of the function are the edges in the graph, and the output of the function is 1 on an assignment if the variables set to 1 contain an s-t path. Read-once formulas over (AND,OR) compute the same set of functions as switch configurations of series-parallel graphs.

Switch configurations can be shown to compute the class of BMP functions defined on graphic matroids (graphic matroids form a proper subclass of the binary matroids), as follows. Consider the graph of a switch configuration, and consider adding a new edge e between s and t. Let M be the associated graphic matroid. Then the output of the switch configuration function is 1 iff a subset of the edges set to 1 form a circuit with e, or equivalently, if the port oracle of M with respect to e outputs "yes" given e and the edges set to 1.

It can also be easily shown that switch configurations satisfy Lehman's alternate characterization of BMP functions. A minterm of a switch configuration is the set of edges in an s-t path. A maxterm is the set of edges in a minimal s-t cut. Every s-t cut separates the graph into an s-component and a t-component. Since each s-t path begins in the s-component and ends in the t-component, each path and each minimal cut intersect in an odd number of edges. Therefore a switch configuration satisfies the property that for each minterm S and each maxterm T, $|S \bigcap T|$ is odd.

There are classes of BMP functions other than switch configurations. Seymour gave a forbidden minor characterization of switch configurations and presented a few BMP functions that are not switch configurations [20]. For example, let G be a graph. Define a boolean function on the edges of G whose output is 1 on a given assignment iff the edges set to 1 contain an odd cycle of G. This is a BMP function, but is not, in general, a switch configuration.

The problem of learning switch configurations with membership queries is motivated by the following reverse-engineering problem. (See Figure 1.)

Consider a black box containing n electrical switches arranged in some unknown interconnection pattern (graph) with two terminals s and t. Joining the two terminals are a battery and a light bulb, connected in series. Each electrical switch inside the black box is controlled by a toggle on the outside of the box that can open and close the switch. The bulb lights up when an electrical circuit is formed with the bulb and a subset of the closed switches. Thus the function determining whether the light bulb is lit is precisely the switch configuration function of the graph. The goal of the reverse-engineering problem is to determine the configuration of the switches by repeatedly manipulating the toggles and observing the state of the bulb. Such experiments are equivalent to asking membership queries about the switch configuration function. In fact, it is often impossible to determine the hidden configuration exactly because many functionally equivalent switch configurations can exist. Raghavan and Schach's algorithm outputs a graph (i.e. switch configuration) that is functionally equivalent to the hidden one.

In matroid theory terms, Raghavan and Schach's algorithm builds the graph of a connected graphic matroid using a port oracle. In fact, their algorithm builds a graph as it learns and is quite complex for that reason. The output of our algorithm is an ear decomposition of the matroid. An ear decomposition of a connected binary matroid forms a basis for the circuit space of the matroid, and gives a representation of the BMP function. This representation, like the graphic representation, is *polynomially evaluatable*. That is, given an input to the associated BMP function, the output of the function can be computed from the representation in polynomial time. If the BMP function is, in fact, a switch configuration, then existing graph realization techniques [4, 3, 5, 24] can be used to convert the basis representation into a graph representation in polynomial time.

We note that there are classes of boolean functions other than BMP functions that are learnable in polynomial time with membership queries. An example is the class of read-once formulas over the basis of boolean threshold gates [2].

It is possible to show that the consistency problem for BMP functions is NP-complete. Therefore, BMP functions cannot be learned in polynomial time in Valiant's PAC learning model unless RP = NP [16]. Because read-once formulas represent a subclass of the BMP functions, it follows immediately from the results in [11, 12] that, unless certain cryptographic schemes can be broken in polynomial time, BMP functions also cannot be learned in polynomial time in the more general PAC prediction model.

3 Matroid definitions

We give some definitions concerning matroids. For a thorough introduction, we suggest [14] or [25].

A *matroid* M is defined by giving its ground set E and its collection $\mathcal{C}$ of subsets of E called *circuits*. In order for $M = (E, \mathcal{C})$ to be a matroid, the following *circuit axioms* must hold:

C1. $\emptyset \notin \mathcal{C}$,

C2. no member of $\mathcal{C}$ is a proper subset of another, and

C3. if C_1 and C_2 are distinct members of $\mathcal{C}$ with $e \in C_1 \cap C_2$, then there is a $C \in \mathcal{C}$ with $C \subseteq (C_1 \cup C_2) - \{e\}$.

Axiom C3 is called the *circuit elimination* axiom. In fact, the following *strong circuit elimination* property, which we will use throughout the paper, also follows from the above axioms.

Property 1 (Strong Circuit Elimination) *If C_1 and C_2 are members of $\mathcal{C}$ with $e \in C_1 \cap C_2$ and $f \in C_1 - C_2$, then there is a $C \in \mathcal{C}$ with $f \in C$ and $C \subseteq (C_1 \cup C_2) - \{e\}$.*

A subset A of E is *independent* if A contains no circuit of M. An *independence oracle* for matroid M is a function on subsets A of E that has value 0 (or "no") if A contains a circuit and 1 (or "yes") otherwise.

Given matroid $M = (E, \mathcal{C})$ and distinguished element $e \in E$, the *port* M_e of M with respect to e is the collection $\mathcal{C}_e$ of circuits of M that contain e. A *port oracle* for M, with respect to e, is a function on subsets A of E that has value 1 (or "yes") if A contains a circuit of M containing e (that is, A contains a member of $\mathcal{C}_e$), and 0 (or "no") otherwise.

Given a subset X of E, $M \setminus X$ (read "M delete X") is defined as follows. The ground set is $E' = E - X$, and the collection of circuits is $\mathcal{C}' = \{C : C \in \mathcal{C} \text{ and } C \subseteq E - X\}$. The pair $(E', \mathcal{C}')$ is a matroid. Given a subset Y of E, M/Y (read "M contract Y") is defined as follows. The ground set is $E'' = E - Y$, and the collection of circuits is $\mathcal{C}''$, the minimal nonempty members of $\{C - Y : C \in \mathcal{C}\}$. The pair $(E'', \mathcal{C}'')$ is a matroid. Contraction and deletion commute. Given disjoint subsets X and Y of E, the matroid $M \setminus X/Y$ is called a *minor* of M.

A *block* of a matroid $M = (E, \mathcal{C})$ is a maximal nonempty subset $\bar{E}$ of E with the property that for every pair of elements e, f in $\bar{E}$, M has a circuit containing both e and f. It is a well-known property of matroids that if elements e and f are in some circuit of M and elements f and g are in some circuit of M, then e and g are in some circuit of M. (See Proposition 4.1.2 of [14].) Therefore, the blocks $E_1, E_2, \ldots, E_k$ of M form a partition of E. Moreover, each circuit of M is completely contained in one block. Thus, the blocks define a partition $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ of the circuits $\mathcal{C}$. The matroids $M_i = (E_i, \mathcal{C}_i)$ are called the *connected components* (or simply *components*) of M. If M has only one component, then M is said to be *connected.*

A *partial ear decomposition* of matroid M is a nonempty sequence $C_1, C_2, \ldots, C_k$ of circuits of M such that for every $i \in \{2, \ldots, k\}$ the following three properties hold:

E1. $C_i \cap (C_1 \cup \cdots \cup C_{i-1}) \neq \emptyset$;

E2. $C_i - (C_1 \cup \cdots \cup C_{i-1}) \neq \emptyset$; and

E3. no circuit C'_i of M satisfying E1 and E2 also has $C'_i - (C_1 \cup \cdots \cup C_{i-1})$ properly contained in $C_i - (C_1 \cup \cdots \cup C_{i-1})$.

For $i = 2 \ldots k$, the *lobe* of circuit C_i is the set $C_i - (C_1 \cup \cdots \cup C_{i-1})$, which we denote by $\tilde{C}_i$. Property E3 is called the *minimality property* of the lobe $\tilde{C}_i$ of circuit C_i.

An *ear decomposition* of M is a partial ear decomposition in which $C_1 \cup \cdots \cup C_k = E$. A single circuit comprises a partial ear decomposition, and any partial ear decomposition can be extended to a (full) ear decomposition, provided M is connected. Thus, every connected matroid M has an ear decomposition.

Given an element e of M, an ear decomposition (or partial ear decomposition) of M is *e-based* if each of its circuits contains e. Every connected matroid M has an e-based ear decomposition, with respect to any fixed element e. This can be seen as follows. First, since M has a circuit C_1 containing e, this circuit comprises a partial e-based ear decomposition. Now given a partial e-based ear decomposition $C_1, ..., C_{i-1}$, let $C_1, ..., C_{i-1}, C'_i$ be a partial ear decomposition. Suppose C'_i does not contain e. Let C_j be a member of the partial ear decomposition such that $C'_i \cap C_j \neq \emptyset$. Let $f \in C'_i \cap C_j$ and let $g \in \tilde{C}'_i$. Now in the minor $M \backslash (E - (C_j \cup C'_i))$, since e and f are in a circuit and f and g are in a circuit, some circuit C_i contains both e and g. It follows that $C_1, ..., C_{i-1}, C_i$ is a partial e-based ear decomposition of M.

Given that C_1 is the first circuit in some e-based ear decomposition, the *lobe* $\tilde{C}_1$ is defined to be the set $C_1 - \{e\}$.

The first algorithm in this paper constructs an e-based ear decomposition using a port oracle.

The statement "$A \subset B$" means A is a proper subset of B.

A matroid $M = (E, \mathcal{C})$ is *binary* if there exists a vector space $\mathcal{V}$ over GF[2] that satisfies the following properties:

1. The vectors of $\mathcal{V}$ are binary, of length $|E|$, and are indexed by the elements of E.

2. For $\vec{v} \in \mathcal{V}$, let $supp(\vec{v}) = \{e \in E | \vec{v}[e] = 1\}$. Then $\mathcal{C}$ corresponds to the nonzero vectors of minimal support in $\mathcal{V}$. That is, letting $\vec{0}$ denote the zero vector, $\mathcal{C} = \{supp(\vec{v}) |\ \vec{v} \in \mathcal{V} - \{\vec{0}\}$ and $\nexists \vec{v'} \in \mathcal{V} - \{\vec{0}\}$ with $supp(\vec{v'}) \subset supp(\vec{v})\}$.

The space $\mathcal{V}$ is unique and is called the *circuit space* of M. If $\vec{A}$ is a nonzero vector in $\mathcal{V}$ then $supp(\vec{A})$ is the union of disjoint circuits of M.

If M is a connected binary matroid, then the characteristic vectors of an ear decomposition of M form a basis for the circuit space of M. Because this is not a standard fact about binary matroids, we present a proof.

Claim 3.1 *Let $C_1, ..., C_i$ be an ear decomposition of a connected binary matroid M. Let $\vec{C}_1, ..., \vec{C}_i$ denote the characteristic vectors of $C_1, ..., C_i$. Then $\{\vec{C}_1, ..., \vec{C}_i\}$ is a basis for the circuit space of M.*

Proof: We show by induction on j that for each $j = 1 \ldots i$, every circuit $C \subseteq C_1 \cup \ldots \cup C_j$ is generated by the circuits $C_1, \ldots, C_j$. For the base case, the only circuit contained in C_1 is C_1 itself. If $C \subseteq C_1 \cup \ldots \cup C_{j-1}$, then we are done by induction. If not, then $C - (C_1 \cup \ldots \cup C_{j-1}) = \tilde{C}_j$, by the minimality of lobe $\tilde{C}_j$ and the fact that $\tilde{C}_j$ cannot contain C. But then the symmetric difference D of C and C_j is the disjoint union of circuits contained in $C_1 \cup \ldots \cup C_{j-1}$ and thus, by induction, is generated by $C_1, \ldots, C_{j-1}$. Now, C is the symmetric difference of D and C_j. □

Therefore, for connected binary matroids, an ear decomposition uniquely determines the matroid. Moreover, given a basis for the circuit space of a binary matroid, it is possible in polynomial time to simulate a port oracle as follows. Form a binary matrix whose rows span the space orthogonal to the circuit space. Determining whether a set A contains a circuit containing e is equivalent to determining whether some subset of the columns indexed by the elements of A is minimally linearly dependent (over GF[2]), and includes the column indexed by e. This latter problem can be solved by standard linear algebraic techniques. This justifies our earlier claim that, when used to learn BMP function, our ear decomposition algorithm outputs a polynomially evaluatable hypothesis.

An important subclass of the binary matroids are the graphic matroids. A graphic matroid $M = (E, \mathcal{C})$ corresponds to a graph with edge set E. The circuits in $\mathcal{C}$ are the edge sets of the simple cycles in the graph.

4 Computational model and running times

In determining the running time of our algorithms, we use the model of Angluin, Hellerstein, and Karpinski [1]. This model is essentially a unit-cost random access machine modified to handle queries. Queries are given as bit strings placed in consecutive registers of the machine, one bit per register. A query is performed by an instruction that specifies the location of the query and the part of memory where the answer to the query should be placed. Queries are assumed to be answered in constant time. If two queries differ in only a constant number of bits, then the first query can be modified to produce the second in constant time. However, it takes $O(n)$ time to set up an n-bit query from scratch.

The time bound given for the algorithm of Raghavan and Schach for learning switch configurations is $O(|E|^2)$ in the above model. It is difficult to verify the complexity of their algorithm as crucial sections of it are only sketched in their paper. However, they do show that the total number of times a query bit is set is indeed $\mathrm{O}(|E|^2)$. With respect to the reverse engineering problem for switch configurations, this corresponds to the number of times the position of an electrical switch is set. Our ear decomposition algorithm, which learns the broader class of BMP functions, runs in time $O(|E|^3)$. The number of times query bits are set is also $O(|E|^3)$. The algorithm makes $O(|E|^2)$ queries.

5 Constructing an e-Based Ear Decomposition

Let M be a connected matroid on E and let e be a member of E. In this section we give a polynomial algorithm for finding an e-based ear decomposition of M using a port oracle.

The first step in constructing an e-based ear decomposition of M is finding a circuit of M containing element e. That is the purpose of the *greedy reduction procedure* presented next.

Procedure Greedy Reduction

Input:	Finite set E, $e \in E$, connected matroid M on E given via port oracle with respect to e, and $A \subseteq E$.
Output:	Circuit C of M containing e contained in A, if such a circuit exists.

Query the port oracle on input A.
If A does not contain a circuit containing e, then
  return "Fail."
For each element $a \in A - \{e\}$,
  Query the port oracle on input $A - \{a\}$.
  If $A - \{a\}$ contains a circuit containing e, then
    remove a from A.
Return A.

Next we describe how to extend a partial (but not full) e-based ear decomposition by one more circuit. Note that if we run the greedy reduction procedure on an arbitrary input A, it may return a circuit that does not extend the partial e-based ear decomposition, or is perhaps even a circuit that is already in the current decomposition. Our solution is to construct a special "test set" of inputs. The construction guarantees that if we run the greedy reduction procedure on the inputs in the test set (as described below), and if the partial e-based ear decomposition is not full, then we will find a circuit that extends it.

The construction and use of the test set is explained and justified in the next five lemmas. In each of these lemmas, assume the following: $C_1, C_2, \ldots, C_i$ is a partial e-based ear decomposition. For $j = 1, \ldots, i$, let $E_j = C_1 \cup \cdots \cup C_j$. Lobe $\tilde{C}_1 = C_1 - \{e\}$ and for $j = 2, \ldots, i$, lobe $\tilde{C}_j = C_j - E_{j-1}$. Also assume set $G = \{g_1, g_2, \ldots, g_i\}$ is a fixed arbitrary set of members of $\tilde{C}_1, \tilde{C}_2, \ldots \tilde{C}_i$, respectively.

Lemma 5.1 *Let C be a circuit of M contained in E_i and let C_j be the last circuit in the partial ear decomposition whose lobe meets C. Then $\tilde{C}_j \subseteq C$.*

Proof. Circuit C cannot be contained in $\tilde{C}_j$, because no circuit properly contains another. Thus, C meets both E_{j-1} and $\tilde{C}_j$. The lemma now follows from the minimality property of $\tilde{C}_j$. □

Lemma 5.2 *The set $E_i - \{g_1, \ldots, g_i\}$ is independent in M, and for each $j = 1, \ldots, i$ and each $q \in \tilde{C}_j$, the set $(C_j \cup \tilde{C}_{j+1} \cup \cdots \cup \tilde{C}_i) - \{q, g_{j+1}, \ldots, g_i\}$ is independent in M. (Note that the element q may or may not be equal to g_j.)*

Proof. Both conclusions follow directly from Lemma 5.1. □

Lemma 5.3 *For every circuit C that meets both E_i and $E - E_i$, there is a circuit C' that meets both E_i and $E - E_i$, contains no member of $\{g_1, \ldots, g_i\}$, and such that $C' - E_i \subseteq C - E_i$.*

Proof. Let C be a circuit that meets both E_i and $E - E_i$. Let C' be a circuit that meets both E_i and $E - E_i$, has $C' - E_i \subseteq C - E_i$, and such that the last in the sequence $g_1, \ldots, g_i$ belonging to C' is as early in the sequence as possible. Suppose $C' \cap \{g_1, \ldots, g_i\} \neq \emptyset$. Let g_k be the last in the sequence $g_1, \ldots, g_i$ that is in C'. Let f be in $C' - E_i$. By strong circuit elimination, there is a circuit $C'' \subseteq (C_k \cup C') - \{g_k\}$ with $f \in C''$. But this C'' contradicts the choice of C'. □

For a given $G = \{g_1, \ldots, g_i\}$ as defined above, the *test set* of G is the collection of subsets of E defined as follows. $T_G^1 = E - G$ is in the test set, and for each $j = 1, \ldots, i$ and each element $q \in \tilde{C}_j$, the set $T_G(j, q) = (C_j \cup \tilde{C}_{j+1} \cup \cdots \cup \tilde{C}_i \cup (E - E_i)) - \{q, g_{j+1}, \ldots, g_i\}$ is in the test set.

Lemma 5.4 *For every circuit C that meets both E_i and $E - E_i$, there is a member T of the test set and a circuit C'' containing e, such that $C'' \subseteq T$ and $C'' - E_i \subseteq C - E_i$.*

Proof. Let C be a circuit that meets both E_i and $E - E_i$. By Lemma 5.3, there is a circuit C' that meets both E_i and $E - E_i$ and contains no member of G and such that $C' - E_i \subseteq C - E_i$. If C' contains e, then we are finished, because $C' \subseteq T_G^1$. Assume C' does not contain e. Let $\tilde{C}_j$ be the first lobe from the sequence $\tilde{C}_1, \ldots, \tilde{C}_i$ that meets C', and let q be in $\tilde{C}_j \cap C'$. By strong circuit elimination, M has a circuit $C'' \subseteq (C_j \cup C') - \{q\}$ that contains e. Now $C'' - E_i \subseteq C - E_i$ and $C'' \subseteq T_G(j, q)$, as desired. $\Box$

Let $y_1, \ldots, y_k$ be a fixed ordering of the elements in $E - E_i$. For each member T of the test set that contains a circuit containing e, let $C_T \subseteq T$ be the circuit containing e remaining after applying greedy reduction to T beginning with the elements of $E - E_i$ in the fixed order, followed by the elements in E_i in some arbitrary fixed order. Let C_{i+1} be a circuit in $\{C_T \mid T \text{ is a test set}\}$ whose incidence vector on $\{y_1, \ldots, y_k\}$ is lexically smallest. (Binary vector x is *lexically* smaller than binary vector y if the binary number represented by x is less than the binary number represented by y.)

Lemma 5.5 *Circuit C_{i+1} is well defined and meets both E_i and $E - E_i$. Moreover, there is no circuit C that meets both E_i and $E - E_i$ with $C - E_i \subset C_{i+1} - E_i$.*

Proof. Since M is connected, there is some circuit that meets both E_i and $E - E_i$. Thus, by Lemma 5.4, at least one test set contains a circuit containing e, and thus C_{i+1} is well defined. Since C_{i+1} contains e, it certainly meets E_i, and by Lemma 5.2, it must also meet $E - E_i$. Finally, suppose a circuit C exists that meets both E_i and $E - E_i$ and satisfies $C - E_i \subset C_{i+1} - E_i$. Then by Lemma 5.4, there is a test set T and a circuit C'' containing e such that $C'' \subseteq T$ and $C'' - E_i \subseteq C - E_i \subset C_{i+1} - E_i$. First suppose $C'' - E_i = C_T - E_i$. Then, since $C'' - E_i \subset C_{i+1} - E_i$, we have that C_{i+1} is lexically greater than C_T, a contradiction. Therefore, $C'' - E_i \neq C_T - E_i$. Let y_p be the first from $y_1, \ldots, y_k$ contained in only one of C'', C_T. Suppose $y_p \in C'' - C_T$. Then since $C'' - E_i \subset C_{i+1} - E_i$, we have that C_{i+1} is lexically greater than C_T, a contradiction. Therefore, $y_p \in C_T - C''$. But at the time y_p was considered during greedy reduction of test set T, what remained of T still contained C'' after y_p was removed. Thus y_p should have been left out of C_T, contradicting $y_p \in C_T$. $\Box$

By Lemma 5.5, C_{i+1} is an appropriate next circuit in our partial ear decomposition. Thus, Lemmas 5.2 through 5.5 provide the justification for the following, preliminary, ear-decomposition algorithm. We will modify this algorithm below to obtain our final algorithm, which has lower query complexity.

Ear Decomposition Algorithm I

Input: Finite set E, distinguished element $e \in E$, connected matroid M on E given via port oracle with respect to e.
Output: e-based ear decomposition of M.

[*Construct 1st circuit*]
Apply greedy reduction to E to obtain circuit C_1 containing e.
Set $i = 1$.
Until $E - E_i = \emptyset$, do the following:
  [*Construct $(i+1)$st Circuit*]
  Order members of $E - E_i$, obtaining $\{y_1, \ldots, y_k\}$;
    and give the members of E_i a fixed order.
  Select $g_i \in \tilde{C}_i$.
  Construct test set.
  For each member T of test set,
    Apply greedy reduction to T beginning in order with $\{y_1, \ldots, y_k\}$, followed by the members of E_i in the fixed order. If a circuit is obtained, call it C_T.
  Let C_{i+1} be a member of $\{C_T \mid T \text{ is a test set}\}$ whose incidence vector on $\{y_1, \ldots, y_k\}$ is lexically smallest.
  Increment i.

Theorem 5.1 *If $M = (E, \mathcal{C})$ is a connected matroid and e is an element of E, then Ear Decomposition Algorithm I finds an e-based ear decomposition of M. Moreover, this algorithm makes at most $O(|E|^3)$ port-oracle queries, and can be implemented to run in time $O(|E|^3)$.*

Proof. The correctness of the algorithm follows from Lemmas 5.2 through 5.5. To see that the algorithm makes $O(|E|^3)$ queries, first note that $O(|E|)$ circuits are constructed. To construct each circuit, $O(|E|)$ sets comprise the test set, and greedy reduction on each requires $O(|E|)$ port oracle queries and $O(|E|)$ time. Finally, choosing a circuit with lexically smallest incidence vector on $\{y_1, \ldots, y_k\}$ can be done with at most $O(|E|^2)$ bit comparisons. Thus, constructing each circuit requires $O(|E|^2)$ oracle queries, yielding an algorithm that makes $O(|E|^3)$ queries and runs in time $O(|E|^3)$. $\square$

Note that the Ear Decomposition Algorithm can be modified so that when applied to an arbitrary (not necessarily connected) matroid, the result is an ear decomposition of the component containing element e. It suffices to add that the algorithm terminates if either the construction of C_1 fails, or if the set $\{C_T \mid T \text{ is a test set}\}$ is empty.

Next we give a modification of the above algorithm which performs only $O(|E|^2)$ queries and runs in time $O(|E|^3)$. The idea is that we can avoid performing greedy reduction of every test set member.

Ear Decomposition Algorithm II

Input: Finite set E, distinguished element $e \in E$, connected matroid M on E given via port oracle with respect to e.

Output: e-based ear decomposition of M.

[*Construct 1st circuit*]
Apply greedy reduction to E to obtain circuit C_1 containing e.
Set $i = 1$.
Until $E - E_i = \emptyset$, do the following:
    [*Construct $(i+1)$st Circuit*]
    Order members of $E - E_i$, obtaining $\{y_1, \ldots, y_k\}$.
    Select $g_i \in \tilde{C}_i$.
    Construct test set, and order its members.
    For $l = 1, \ldots, k$,
        Consider in order the test set members, stopping if a "yes" answer is obtained:
            choose a test set member T and query the port oracle on input $T - \{y_l\}$.
        If a "yes" answer was obtained, then
            delete from test set all members yielding "no" answer;
            replace each remaining T with $T - \{y_l\}$.
    Choose first remaining member T of test set.
    Apply greedy reduction to T to obtain C_{i+1}.
    Increment i.

Theorem 5.2 *If $M = (E, \mathcal{C})$ is a connected matroid and e is an element of E, then Ear Decomposition Algorithm II finds an e-based ear decomposition of M. Moreover, this Algorithm makes at most $O(|E|^2)$ port oracle queries and can be implemented to run in time $O(|E|^3)$.*

Proof. The correctness of the algorithm follows from Lemmas 5.2 through 5.5, provided we can argue that a circuit C_{i+1} is obtained in each iteration, and that it is a circuit lexically smallest on $\{y_1, \ldots, y_k\}$, just like in Algorithm I. First, note that before the test set is altered in any way, at least one member of the test set contains a circuit containing e.

Algorithm II differs from Algorithm I in that the greedy reduction on the members of the test set is done in parallel; that is, the element y_1 is checked for removal from each, followed by y_2, and so on. In addition, instead of checking each set, we only check until one set is found to still contain a circuit containing e after the current y_j is removed. Then we continue under the assumption that the remaining (unchecked) sets also contain such a circuit, removing y_j from them as well. We discard all checked sets found not to contain such a circuit. After such a pass, the first set remaining on the list certainly contains a circuit containing e. This is clearly still true after a pass where no "yes" answer was obtained, since no sets are changed in such a pass. Thus, after all the members of $\{y_1, \ldots, y_k\}$ have been considered, the first test set member remaining contains a circuit containing e, as required.

The fact that the circuit C found is lexically smallest on $\{y_1, \ldots, y_k\}$ is seen as follows. Let T be the test set member containing C. If some test set member T' contains a lexically smaller such circuit C', then let y_l be the first element in C that is not in C'. Now T' would have passed all checks passed by T up until y_l was considered, and T' would have passed the y_l check, whereas T would have failed it, a contradiction.

The number of queries is calculated as follows. There are at most $|E|$ circuits that are added to the partial ear decomposition throughout the algorithm. The first circuit in the decomposition is generated with $O(|E|)$ queries. Let us count the oracle queries needed to construct each additional circuit C_{i+1}. For each $l = 1, \ldots, k$, for which no "yes" answer was obtained, there is one "no" answer for each of the at most $|E|$ test set members; that is, at most $|E|$ "no" answers. This happens once for each y_l in $\tilde{C}_{i+1}$, for a total of at most $|E|\tilde{C}_{i+1}$ queries. That accounts for some of the "no" answers. Each remaining "no" answer is accompanied by one "yes" answer and results in deletion of a test set member. Thus, there are at most $|E|$ remaining "no" answers. Since there is at most one "yes" answer for each value of l, there are at most $|E|$ "yes" answers. That accounts for all the "no" and "yes" answers per circuit. The total number of queries performed by the algorithm is therefore at most $\sum_{i=1}^{r*}(|E||\tilde{C}_{i+1}| + 2|E|)$, where r^* is the number of circuits in the final ear decomposition. Since the sets $\tilde{C}_1, \ldots, \tilde{C}_{r^*}$ are disjoint, this sum is $\mathrm{O}(|E|^2)$. The time complexity of $\mathrm{O}(|E|^3)$ can be achieved by a straightforward implementation that spends time $\mathrm{O}(|E|)$ per query. □

6 Simulating Independence Oracle with Port Oracle

Let M be a connected matroid on ground set E and let e be an element of E. It is easy to show that a port oracle of M, with respect to e, can be efficiently simulated via an independence oracle of M: To determine whether a given set A contains a circuit containing e, given an independence oracle, build a maximal independent subset B of $A - \{e\}$ one element at a time. Now A contains a circuit containing e if and only if $e \in A$ and $B \cup \{e\}$ is dependent. In this section we show that, conversely, an independence oracle for M can be simulated in polynomial time via a port oracle of M with respect to e. That is, given an arbitrary subset A of E, we can determine whether A is independent in M in polynomial time using only a polynomial number of port oracle queries.

The algorithm for testing independence of A works as follows. First we construct an e-based ear decomposition $C_1, \ldots, C_m$ of M. By Lemma 5.1, if A contains no lobe of this ear decomposition (that is, $\tilde{C}_i - A \neq \emptyset$, for $i = 1, \ldots, m$), then A is independent in M. Otherwise, the problem is reduced by focusing on the last lobe $\tilde{C}_l$ contained in A. The following proposition enumerates the three cases in which A contains a circuit of M. Algorithmically speaking, the first two cases reduce the problem size by at least $|\tilde{C}_l|$. The third is the terminal case.

Proposition 6.1 *Let M be a connected matroid on ground set E, let $e \in E$, and let $C_1, \ldots, C_m$ be an e-based ear decomposition of M. Let A be a subset of E with $e \notin A$. Assume $\tilde{C}_i \subseteq A$, for some index i, and let l be the maximum such index. Let A_e be the elements of A in the same component as e in $M/\tilde{C}_l \setminus (\tilde{C}_{l+1} \cup \cdots \cup \tilde{C}_m)$, and let $\bar{A}_e = A - (A_e \cup \tilde{C}_l \cup \cdots \cup \tilde{C}_m)$.*

Then A contains a circuit of M if and only if one of the following three conditions holds:

1. *A_e contains a circuit of $M/\tilde{C}_l \setminus (\tilde{C}_{l+1} \cup \cdots \cup \tilde{C}_m)$;*
2. *$\bar{A}_e$ contains a circuit of $M \setminus (\tilde{C}_l \cup \cdots \cup \tilde{C}_m)$;*
3. *$\bar{A}_e \cup (C_l - \tilde{C}_l)$ contains a circuit of M containing e.*

Proof. ($\Rightarrow$) Assume A contains a circuit of M. By Lemma 5.1, every circuit in A is contained in $E - (\tilde{C}_{l+1} \cup \ldots \cup \tilde{C}_m)$. First suppose $C \cap \tilde{C}_l = \emptyset$, for all circuits C in A. Then any circuit C in A is also a circuit of $M/\tilde{C}_l \setminus (\tilde{C}_{l+1} \cup \cdots \cup \tilde{C}_m)$, and is, by definition of components, completely contained in either A_e or $\bar{A}_e$. In the former case, (1) holds, and in the latter case, (2) holds. Now assume A contains some circuit C with $C \cap \tilde{C}_l \neq \emptyset$. Then, by Lemma 5.1, $\tilde{C}_l \subseteq C$. Thus, $C' = C - \tilde{C}_l$ is a circuit of $M/\tilde{C}_l \setminus (\tilde{C}_{l+1} \cup \cdots \cup \tilde{C}_m)$, and by definition of component, C' is completely contained in one component of this minor. It follows that $C' \subseteq A_e$ or $C' \subseteq \bar{A}_e$. If $C' \subseteq A_e$, then (1) holds, and we are finished. Assume $C' \subseteq \bar{A}_e$. Then $C \subseteq \bar{A}_e \cup \tilde{C}_l$. Since $e \notin C$, the circuits C and C_l are distinct. Let $f \in \tilde{C}_l \subseteq C \cap C_l$. By strong circuit elimination, there is a circuit C'' of M containing e and contained in $(C \cup C_l) - \{f\}$. Note that $C'' \subseteq C_1 \cup \cdots \cup C_l$. Thus, by minimality of $\tilde{C}_l$, it follows that $C'' \cap \tilde{C}_l = \emptyset$. Therefore $C'' \subseteq (C \cup C_l) - \tilde{C}_l \subseteq \bar{A}_e \cup (C_l - \tilde{C}_l)$. That is, (3) holds.

($\Leftarrow$) That (1) and (2) both imply A contains a circuit is immediate. Consider (3). Let $C \subseteq \bar{A}_e \cup (C_l - \tilde{C}_l)$ be a circuit of M containing e. Let $f \in \tilde{C}_l$; by strong circuit elimination, M has a circuit C' containing f and contained in $(C \cup C_l) - \{e\}$. Since C' cannot be properly contained in C_l, we know that $C' \cap \bar{A}_e \neq \emptyset$. Also, since $C \cup C_l \subseteq C_1 \cup \cdots \cup C_l$, minimality of $\tilde{C}_l$ implies that $\tilde{C}_l \subseteq C'$. It follows that $C' - \tilde{C}_l$ is a circuit of $M/\tilde{C}_l \setminus (\tilde{C}_{l+1} \cup \cdots \cup \tilde{C}_m)$. By definition of component, this circuit must be entirely contained in $\bar{A}_e$. That is, $C' \subseteq \bar{A}_e \cup \tilde{C}_l \subseteq A$, as desired. □

Proposition 6.1 yields the following algorithm for testing independence of a subset A of E. Note that the algorithm includes recursive calls involving minors of the input matroid. Executing these recursive calls requires the ability to simulate a port oracle for a minor of the input matroid, using the port oracle for the input matroid. Given subsets X and Y of $E - \{e\}$, a port oracle for minor $M \setminus X/Y$ can be simulated using a port oracle for M as follows: Given input set A, query the port oracle for M on the set $A \cup Y$.

Independence Algorithm

Input: Finite set E, distinguished element $e \in E$, connected matroid M on E given via port oracle with respect to e, and subset A of E.

Output: Answer to the question: Is A independent or dependent in M?

1. If $e \in A$, then
   Query the port oracle for M on input A.
   If A contains a circuit containing e, then
     return "dependent,"
   else remove e from A.
2. Construct e-based ear decomposition $C_1, \ldots, C_m$ of M.
3. If $\tilde{C}_i - A \neq \emptyset$ for all $i \in \{1, \ldots m\}$, then
   return "independent."
   else, let l be the maximum index with $\tilde{C}_l \subseteq A$.
4. Let $M' := M/\tilde{C}_l \setminus (\tilde{C}_{l+1} \cup \cdots \cup \tilde{C}_m)$.
   Construct e-based ear decomposition $D_1, \ldots, D_p$ of the component of M' containing e.
   Let $A_e := A \cap (D_1 \cup \cdots \cup D_p)$.
   Let $\bar{A}_e := A - (D_1 \cup \cdots \cup D_p \cup \tilde{C}_l \cup \cdots \cup \tilde{C}_m)$.
5. Query the port oracle for M on input $\bar{A}_e \cup (C_l - \tilde{C}_l)$.
   If $\bar{A}_e \cup (C_l - \tilde{C}_l)$ contains a circuit of M containing e, then
   return "dependent."
6. Recursively run the Independence Algorithm to answer the question: Does $\bar{A}_e$ contain a circuit of $M'' := M \setminus (\tilde{C}_l \cup \cdots \cup \tilde{C}_m)$?
   If yes, then
   return "dependent."
7. Recursively run the Independence Algorithm to answer the question: Does A_e contains a circuit of M'?
   If yes, then
   return "dependent."
8. Return "independent."

The above algorithm states explicitly how to compute the sets A_e and $\bar{A}_e$, using the e-based ear decomposition $D_1, \ldots, D_p$ of the component of M' containing element e. Clearly, the members of A contained in the union of the circuits of this ear decomposition are in the same component as e in the minor M', which is how A_e is defined. The set $\bar{A}_e$ is the rest of the members of A in M'.

Now let us analyze the computational complexity of the Independence Algorithm. The algorithm is recursive. Let $c(n, m)$ denote the maximum number of calls to the algorithm (the initial call, plus all recursive calls) when it is initially run on matroid $M = (E, C)$ and set A, where $|E| = n$ and $|A| = m$. When $|A| = 0$, the algorithm makes no recursive calls, and hence $c(n, 0) = 1$. If $|E| = 1$, then $E = \{e\}$ and the algorithm makes no recursive calls. Hence $c(1, m) = 1$. If $n \geq 2$ and $m \geq 1$, then the algorithm makes at most two recursive calls. These calls involve A_e and $\bar{A}_e$, which are disjoint subsets of A. They also involve proper minors of M, which can have at most $|E| - 1$ elements. Thus $c(n, m) \leq 1 + \max_{0 \leq i \leq m}(c(n-1, m-i) + c(n-1, i))$. A simple inductive argument shows that $c(n, m) \leq \max\{1, 2mn\}$, and hence $c(n, m) = O(nm)$.

We now analyze the time spent and number of queries made by each recursive call (independent of any work caused by further recursive calls). In each recursive call, the algorithm constructs at most two ear decompositions, in Steps 2 and 4. Constructing an ear decomposition takes $O(|E|^3)$ time and induces $O(|E|^2)$ queries. Step 1 takes time $O(|E|)$ and induces at most one query. In Step 3, checking for $\tilde{C}_i - A \neq \emptyset$ can easily be done in time $O(|E|^2)$ time per C_i, totaling $O(|E|^3)$ overall. In Step 5, one oracle query is performed. All other computation performed during a recursive call is easily accomplished in time $O(|E|^3)$. Thus the total time spent per recursive call is $O(|E|^3)$, and the total number of queries performed per call is $O(|E|^2)$.

Since there are $O(|E||A|)$ total recursive calls, each taking time $O(|E|^3)$ and making $O(|E|^2)$ queries, and $|A| \leq |E|$, the algorithm runs in time $O(|E|^5)$ and makes $O(|E|^4)$ queries.

7 Conclusions, Extensions and Open Questions

We have given an algorithm to compute an ear decomposition of a general matroid using a port oracle. The algorithm runs in time polynomial in the number of elements of the matroid. We noted in Section 3 that if the matroid is connected and binary, then the ear decomposition uniquely determines the matroid. This is not the case for arbitrary connected matroids, or even for arbitrary connected ternary matroids.

In this paper we have considered the problem of learning the port functions of binary matroids, but not of matroids representable over fields other than GF[2]. However, we have shown that an independence oracle for an arbitrary connected matroid can be simulated in polynomial time by a port oracle. It follows that the port functions of matroids representable over a field $\mathcal{F}$ can be learned in polynomial time with membership queries, provided that there is a polynomial algorithm for the following problem: Given an independence oracle for a matroid known to be representable over $\mathcal{F}$, construct a representation matrix for the matroid. This problem is itself interesting.

In the case of binary matroids, the *fundamental circuit matrix* of the matroid is a representation matrix. The fundamental circuit matrix has the form $[I|D]$, where I is an identity matrix with rows indexed on some base of the matroid, and the columns of D are indexed on the remaining, nonbasic, elements. For each nonbasic element f, the column of D corresponding to f is the incidence vector of $C_f - \{f\}$, where C_f is the fundamental circuit of f with respect to the base. It is easy to show how to construct the fundamental circuit matrix in polynomial time using an independence oracle. (Note that, since the first algorithm in this paper learns binary matroid port functions directly, it is unnecessary to learn them by simulating an independence oracle using the port oracle.)

In the case of ternary matroids, a representation matrix *can* be obtained via a polynomial number of independence oracle queries. One such algorithm is identical to a procedure for constructing a totally unimodular real representation of a given regular matroid (See, for example, [14], Section 6.6, or [23], Corollary 9.2.7.), and proceeds as follows: First construct a fundamental circuit matrix for the matroid. The remaining effort involves "signing" the submatrix D, that is, determining the sign for each of the 1's. This is done via the adjacency graph $G(D)$ (the bipartite graph with node classes indexed on the columns and rows of D, in which two nodes are adjacent if the corresponding entry of D is nonzero). The entries corresponding to a spanning tree of D are signed $+1$. The remaining entries are signed in an order ensuring that as entry ij is being considered, edge ij is contained in some chordless cycle C of G in which all the other edges have already been signed. The sign of ij will be whichever provides the correct determinant for the submatrix obtained by taking the columns of D corresponding to the column nodes in C, together with the columns of I corresponding to the row nodes in C.

That the above algorithm works is really equivalent to the unique representability of connected ternary matroids, which was first proved by Brylawski and Lucas [6]. Kahn [10] proved a unique-representability result for matroids over the field GF[4] and Oxley et al [15] established the extent to which representability over GF[5] is unique. For these fields, the definition of uniqueness is more complicated than for GF[3], involving field automorphisms, and requiring 3-connectivity. The open question here is whether there exists a polynomial algorithm for constructing a representation matrix over GF[4] for a 3-connected quaternary matroid, where the matroid is given via an independence oracle. The analogous question for GF[5] representable matroids is also open.

Turning to general matroids, it is known that there are a doubly exponential number of matroids on a set of n elements [14], and it easily follows that there also are a doubly exponential number of connected matroids on n elements. Therefore, any system for representing arbitrary matroids, or even arbitrary connected matroids, uses a number of bits exponential in n. Moreover, any oracle algorithm that makes a number of queries polynomial in n, and uses an oracle that answers only "yes" or "no", only obtains a polynomial number of bits of information. Thus we cannot hope to find a port oracle algorithm that outputs in time polynomial in n a representation of the arbitrary connected matroid defined by the port oracle.

Because there are no compact explicit representations of arbitrary matroids, algorithms that take matroids as input typically take them in the form of an independence oracle. We have shown that, for connected matroids, any problem that can be solved in polynomial time given an independence oracle (such as testing for graphicness [22]) can be solved in polynomial time given a port oracle.

The work in the first part of this paper represents the first use of matroid theory techniques in solving computational learning theory problems. As shown by this work, matroid techniques can be useful in developing learning algorithms for functions based on graphs.

An open question is whether there exist efficient algorithms for learning generalizations of switch configurations. In the switch configurations discussed here, a switch can only complete a circuit in one of its two positions. In actual electrical wiring, a switch may be able to complete a circuit in either of its two positions, by making two different contacts. Such behavior allows the wiring of a room with a single bulb and two light switches, where changing the position of either light switch changes the state of the bulb. It would be interesting to model and learn generalizations of switch configurations which include this behavior. Some simple generalizations of this type are easily shown not to be learnable in polynomial time with membership queries alone. One generalization includes the class of read-once branching programs, for which a learning algorithm has been developed in the more powerful membership and equivalence query model [18].

8 Acknowledgments

We thank David Hartvigsen for suggesting the use of ear decompositions in this work, and Nader Bshouty and Andras Frank for useful discussions. We also thank Andras Sebö and the referees for their careful reading and very helpful suggestions.

References

[1] D. Angluin, L. Hellerstein, and M. Karpinski, Learning read-once formulas with queries. *J. of the Association for Computing Machinery* **40** (1993) 185-210.

[2] N. Bshouty, T. Hancock, L. Hellerstein, and M. Karpinski, An algorithm to learn read-once threshold formulas, and transformations between learning models. *Computational Complexity* **4** (1994) 37-61.

[3] R. E. Bixby, Matroids and operations research. in: *Advanced Techniques in the Practice of Operations Research,* (H. J. Greenberg, F. H. Murphy, and S. H. Shaw, eds.), North-Holland Publishers (1980) 333-459.

[4] R. E. Bixby and W. H. Cunningham, Converting linear programs to network problems, *Mathematics of Operations Research* **5** (1980) 321-357.

[5] R.E. Bixby and D.K. Wagner, An almost linear time algorithm for graph realization. *Mathematics of Operations Research* **13** (1988) 99-123.

[6] T.H. Brylawski and D. Lucas, Uniquely representable combinatorial geometries. *Teorie Combinatorie* (Proc. 1973 Internat. Colloq.) pp. 83-104, Accademia nazionale dei Lincei, Rome, (1976).

[7] D. Hausmann and B. Korte, The relative strength of oracles for independence systems. in: *Special Topics of Applied Mathematics,* (J. Frehse, D. Pallaschke, and U Trottenberg, eds.), North-Holland Publishers (1980) 195-211.

[8] L. Hellerstein and C. Coullard, Learning binary matroid ports. *Proceedings of the 5th Annual SIAM Symposium on Discrete Algorithms* (1994) 328-335.

[9] P. M. Jensen and B. Korte. Complexity of matroid property algorithms. *SIAM Journal of Computation* **11** (1) (1982) 184-190.

[10] J. Kahn, On the uniqueness of matroid representations over GF(4). *Bull. London Math Soc.* **20** (1988) 5-10.

[11] M. Kearns, M. Li, L. Pitt, and L. Valiant, On the learnability of boolean formulae. *Proc. 19th ACM Symposium on Theory of Computing* (1987) 285-295.

[12] M. Kearns and L. Valiant, Cryptographic limitations on learning boolean formulae and finite automata. *Proc. 21st ACM Symposium on Theory of Computing* (1989) 433-444.

[13] A. Lehman, A solution of the Shannon switching game. *Journal of the Society of Industrial and Applied Mathematics* **12**:4 (1964) 687-725.

[14] J.G. Oxley, *Matroid Theory.* Oxford University Press, New York, (1992).

[15] J.G. Oxley, D. Vertigan and G. Whittle, On inequivalent representations of matroids over finite fields. Technical Report, Department of Mathematics, Louisiana State University, Baton Rouge, LA 70803, (1994).

[16] L. Pitt and L. Valiant, Computational limitations on learning from examples. *J. ACM* **35** (1988) 965-984.

[17] V. Raghavan and S. Schach, Learning switch configurations. *Proceedings of Third Annual Workshop on Computational Learning Theory,* Morgan Kaufmann Publishers (1990) 38-51.

[18] V. Raghavan and D. Wilkins, Learning μ-branching programs with queries. *Proceedings of the Sixth Annual Workshop on Computational Learning Theory,* ACM Press (1993) 27-36.

[19] P. D. Seymour, The forbidden minors of binary clutters. *J. London Math. Soc.* (2) **12** (1975) 356-360.

[20] P. D. Seymour, A note on the production of matroid minors. *J. of Combinatorial Theory (B)* **22** (1977) 289-295.

[21] P. D. Seymour, The matroids with the max-flow min-cut property. *J. of Combinatorial Theory (B)* **23** (1977) 189-222.

[22] P. D. Seymour, Recognizing graphic matroids. *Combinatorica* **1** (1981) 75-78.

[23] K. Truemper, *Matroid Decomposition.* Academic Press, San Diego, (1992).

[24] W. T. Tutte, An algorithm for determining whether a given binary matroid is graphic. *Proc. Amer. Math. Soc.* **11** (1960) 905–917.

[25] D.J.A. Welsh, *Matroid Theory.* Academic Press, London, (1976).

Figure 1: Configuration of electrical switches hidden in black box. Toggles at bottom open and close associated electrical switches.