A Polynomial-time Algorithm for Learning k-variable Pattern Languages from Examples

(extended abstract)

Michael Kearns*
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts

Leonard Pitt†
Department of Computer Science
University of Illinois
Urbana, Illinois

1 Introduction

In this paper we give, for each constant k, a polynomial-time algorithm for learning the class of k-variable *pattern languages* in the learning model introduced by Valiant [**?**]. A *pattern* is a string of constant and variable symbols, for instance the 3-variable pattern $p = 10x_1x_2x_21x_300x_1$. The associated language $L(p)$ is obtained by substituting for each variable in the pattern any constant string. Thus, the string 1011011011110110011 is contained in $L(p)$, since it can be obtained from p by the substitutions $x_1 = 11, x_2 = 011, x_3 = 1011$.

For any constant k, our algorithm learns a k-variable target pattern p by producing a polynomial-sized disjunction of patterns, each of between 0 and k variables. (See below for a discussion of the potential difficulty of learning *general* patterns — where the number of variables is not bounded by a constant.) We assume the algorithm has access to a random source of negative examples, generated according to an arbitrary distribution, and a random source of positive examples of the target pattern p in which the k-tuple of substitution strings (which entirely determines the positive example generated) is drawn not from an *arbitrary* distribution, but from any product distribution $D = D_1 \times \cdots \times D_k$, where each D_i is an arbitrary distribution on substitution strings for variable x_i. This appears to be a natural and very general class of distributions.

Our algorithm runs in time polynomial in parameters $|\Sigma|, n, s, \frac{1}{\epsilon}$, and $\frac{1}{\delta}$ (and in time doubly exponential in the constant k), where Σ is the alphabet of constant symbols, n is the length of the target pattern, s is the maximum length substitution string for any variable, and ϵ and δ are "accuracy" and "confidence" parameters. The algorithm outputs a disjunction of patterns, each over k or fewer variables, that with probability at least $1 - \delta$, has accuracy of classification at least $1 - \epsilon$ on future positive and negative examples generated randomly from the distributions on which the algorithm was run. Thus the algorithm *PAC*-learns (*Probably Approximately Correctly* learns) the class of k-variable patterns *in terms of* the class of disjunctions of k or fewer variable patterns.

*Supported by an A.T. & T. Bell Laboratories Ph.D. Scholarship, ONR grant N00014-85-K-0445, and by a grant from the Siemens Corporation.

†Supported in part by NSF grant IRI-8809570, and by the Department of Computer Science, University of Illinois at Urbana-Champaign.

The algorithm presented here is a positive result in a model that has recently seen a number of strongly negative results. In particular, the work of [**?**, **?**] suggests that for language classes in the Chomsky hierarchy (e.g., regular languages), distribution-free learning is too ambitious a goal to be accomplished in polynomial time. For this reason it is heartening to discover a natural and general class of languages that is learnable in polynomial time under a wide class of distributions. The pattern languages are particularly interesting in light of the results in [**?**, **?**], since they are incomparable to the classes discussed there. For example, it is shown in [**?**] that learning regular languages is hard under certain cryptographic assumptions. However, Angluin [**?**] shows that pattern languages and context-free languages are incomparable. Thus, our results suggest that perhaps there are large classes of languages that are efficiently learnable from examples, but that these classes constitute a "cut" of the complexity hierarchy other than the standard Chomsky classification.

It should be noted that our algorithm works when the empty string λ is allowed as a substitution for a variable; thus each variable may be viewed as a "placeholder" for a substitution that may or may not be present in any given positive example of p. Previous results on pattern languages (discussed below) typically do not allow λ-substitutions.

We can also extend our techniques to learn a wider class of patterns, where restricted homomorphisms of the original variables may also appear in the target pattern. In particular, let $t = t(n, s, |\Sigma|)$ be any polynomial, and $h_1, \ldots, h_t$ a collection of bijective maps $h_i : \Sigma \to \Sigma$ such that for each $1 \leq i \leq t$, $h_i(a) \neq a$. Extend h_i to strings in the usual way: If $a = a_1 \cdots a_l$, then $h_i(a) = h_i(a_1) \cdots h_i(a_l)$. An example of such a homomorphism (over $\Sigma = \{0, 1\}$) is $h(0) = 1$, and $h(1) = 0$. Thus $h(a) = \overline{a}$, the complement of a. A target pattern including such homomorphisms might be $x_1 10 x_2 h_2(x_1) 0 h_1(x_2) 110$, and positive examples are obtained by substituting constant strings for the variables x_1 and x_2, and applying the homomorphisms as indicated.

Given our initial requirement that the positive distribution be a product distribution, the allowance of the maps $h_i(x_j)$ in the target pattern is important in that it reintroduces a natural form of dependence within the example strings. While we still require that the substitution for variable x_1 is drawn independent of the substitution for x_2, we may regard the transformations $h_1(x_1), \ldots, h_t(x_1)$ as a polynomial-sized collection of "variables" whose substitution is strongly dependent on the substitution for x_1. The additional running time is a small polynomial increase depending on t, in particular, there is no increase in the exponent of the running time. Thus in some sense, the algorithm can handle a polynomial number of variables, as long as all but a constant number k are highly dependent, and the remaining k variables are independent. In this extended abstract, we will restrict our attention to basic patterns (without homomorphisms). The modified algorithm and analysis for the homomorphisms described above will be given in the full paper.

A brief comment on the potential difficulty of learning general patterns from examples is now in order. Previous algorithms for learning patterns either limit their attention to a constant number of variables, place strong restrictions on the order of the appearance of the variables, or allow various types of queries. While the problem of learning general patterns from examples by some hypothesis space in polynomial time remains open, there appear to be significant obstacles to overcome in this direction. First, Angluin [**?**] showed that for general patterns, deciding membership in the corresponding language is NP-complete. This suggests difficulties in the design of an algorithm that learns k-variable patterns in time *polynomial* in k. In particular, such an algorithm would not be able to use a general membership test to determine the consistency of a candidate pattern with a set of examples, a tool commonly used in the PAC-learning model. Second, it has recently been

shown by Ko, Marron and Tzeng [**?**] that the problem of finding any pattern consistent with a set of positive and negative examples is NP-hard; the results of [**?**] (see also [**?**]) can be applied to show that the problem of learning general patterns by an algorithm outputting a single pattern in the PAC-learning model is NP-hard.

Our work in this area was motivated by the results of Angluin [**?**], Marron and Ko [**?**], and Marron [**?**]. Angluin's main result is a polynomial-time algorithm for finding a *descriptive pattern* for a positive sample S of an unknown 1-variable target pattern: a pattern p such that $S \subseteq L(p)$, and for any 1-variable pattern p', $L(p') \not\subset L(p)$. Angluin's algorithm has no obvious application to the PAC-learning model, and there appear to be significant obstacles in extending her approach to finding descriptive k-variable patterns for constant $k > 1$ [**?**, **?**]. Shinohara [**?**, **?**] gives a polynomial-time algorithm for finding a descriptive pattern when there is at most one occurrence of each variable, and when λ-substitutions are allowed as well. He also gives an algorithm for patterns where all occurrences of variable x_i appear before any occurrence of x_{i+1}. These results also do not have obvious applications in the PAC-learning model, where negative examples are present as well as positive, and the source of examples is an arbitrary distribution.

Ibarra and Jiang [**?**] give a polynomial-time algorithm for exact identification of general patterns in a model where the algorithm may conjecture an hypothesis pattern, and is provided with a *shortest* counterexample to the equivalence of the conjectured pattern and the target pattern (or is told they are equivalent). Angluin [**?**] shows that if each equivalence query may be answered by an *arbitrary* counterexample, but the algorithm is also provided with membership queries (access to an oracle deciding membership in the target pattern language) and subset queries (access to an oracle deciding if an hypothesis pattern accepts a subset of the target pattern language), exact identification must take exponential time. Again, these query results have no clear application to learning pattern languages in the PAC-learning model.

Marron and Ko [**?**] considered necessary and sufficient conditions on a finite positive initial sample that would allow *exact* identification of a target k-variable pattern from the initial sample and from polynomially many *membership queries*. Subsequently, Marron [**?**] considered the learnability of k-variable patterns in the same model, but where the initial sample consisted of only a single positive example of the target pattern. In both of these papers, a greedy learning algorithm (used as a subroutine here) was proposed. For the case of 1-variable and 2-variable patterns, Marron gave a careful analysis of the structural properties of initial examples that can cause this algorithm to fail. He also showed that only a small fraction of strings possess these properties.

We study the learnability of pattern languages in a model where the examples are generated randomly according to an unknown product distribution. Thus, the fact that the number of strings causing the greedy algorithm to fail is small compared to the total number of strings is no guarantee of success, since the underlying distribution may give large weight to these bad strings. We tackle this problem with a careful probabilistic analysis of the failures of the greedy algorithm together with a new algorithm that avoids these pitfalls with high probability. More precisely, we show that the "bad" portion of the target distribution that causes the greedy algorithm to fail in learning the k-variable target pattern can in fact be covered (within any ϵ) by a polynomial number of $k-1$ variable patterns (the exact number depending on ϵ, and the probability of the bad portion of the distribution). However, the greedy algorithm may of course also fail to learn these covering patterns, and some of them may cover a fraction of the distribution that is too small to notice in polynomial time.

Our algorithm works by repeatedly running the greedy algorithm in an attempt, for each $0 \leq$

$d \leq k$, to learn all patterns of d variables that cover a significant portion of the positive distribution. The main tool introduced in the analysis is a data structure called an *event tree*. The event tree has depth at most k, and its nodes consist of subsets of possible k-tuples of substitution strings (any such subset is called an *event*.) Thus each node represents a set of positive examples of the target pattern. The root of the tree represents the set of all positive examples, and the children of any node represent disjoint subevents of the parent.

The correctness of our algorithm essentially hinges on a proof that the algorithm will learn the leaves of the event tree with high probability. To ensure that our algorithm runs in polynomial time, we define the event tree so as to simultaneously provide an upper bound on the branching factor at any node, and a lower bound on the probability associated with the node. We then show that the leaves of the event tree represent disjoint subevents of the root covering an overwhelming fraction of the positive distribution. Furthermore, for each leaf at depth d, $0 \leq d \leq k$, there is a corresponding pattern over at most d variables that is a subpattern of the target pattern, obtained by replacing $k - d$ of the variables with particular constant substitutions, and whose positive examples include all positive examples of the target pattern that can be obtained from substitution k-tuples corresponding to the leaf.

These leaf patterns are shown to be contained in a larger pool of patterns constructed by repeated application of the greedy algorithm. Our main algorithm then eliminates patterns in this pool whose inclusion would incur too much error on the negative distribution. It then applies an approximation algorithm for the partial set cover problem [**?**] to find a disjunction of the remaining patterns, whose size exceeds the number of leaves in the event tree by at most a logarithmic factor. This disjunction is shown to be small enough to be a $(1 - \epsilon)$-accurate hypothesis by applying a generalization of the technique known as Occam's Razor [**?**, **?**].

2 Definitions and Notation

Let Σ be a finite alphabet. A *k-variable pattern* is an element of $(\Sigma \cup \{x_1, \ldots, x_k\})^*$. For any k-variable pattern p, and any set of k strings $u_1, \ldots, u_k \in \Sigma^*$, let $p[x_1 : u_1, \ldots, x_k : u_k]$ denote the string $w \in \Sigma^*$ obtained by substituting u_i for each occurrence of x_i in the pattern p. The *language generated by p* is defined by $L(p) = \{p[x_1 : u_1, \ldots, x_k : u_k] : u_1 \ldots u_k \in \Sigma^*\}$. Let P_k consist of all k-variable patterns.

For this abstract, we will assume that all substitution strings for variables have maximum length s, where s will typically be polynomially related to the length n of the *target pattern* to be learned. See [**?**] for the justification and motivation of such an assumption. Also, without loss of generality [**?**, **?**] we assume that n and s are given as input to the learning algorithm. Let $\Sigma^{[s]}$ denote strings of any length between 0 and s. Call any such string a *substitution string*.

A *binding* is a list $x_{i_1} : u_1, \ldots, x_{i_d} : u_d$, where $1 \leq d \leq k$, the x_{i_j}'s are distinct variables, and each u_i is a substitution string. The pattern $p[x_{i_1} : u_1, \ldots, x_{i_d} : u_d]$ is the pattern p with strings $u_1, \ldots u_d$ substituted in for variables $x_{i_1}, \ldots x_{i_d}$, respectively. Thus $p[x_{i_1} : u_1, \ldots, x_{i_d} : u_d]$ is a $k - d$ variable pattern over variable set $\{x_1, \ldots, x_k\} - \{x_{i_1}, \ldots x_{i_d}\}$. Such a pattern is called a *subpattern* of p. Note that if $d = k$ then a binding of (all k) variables produces a "subpattern" that is a constant string.

For any string w, let $w(i, j)$ denote the substring of w beginning at position i, and having length j.

For each variable x_i ($1 \leq i \leq k$), we assume an arbitrary probability distribution D_i on substitution strings. Let $\mathbf{Pr} : (\Sigma^{[s]})^k \to [0,1]$ be the product distribution $D_1 \times \cdots \times D_k$, thus $\mathbf{Pr}(\langle u_1, \ldots, u_k \rangle) = \Pi_{i=1}^k D_i(u_i)$. Positive examples of the target pattern are generated according to distribution $\mathbf{Pr}$. That is, the k-tuple $\langle u_1, \ldots, u_k \rangle$ is generated with probability $\mathbf{Pr}(\langle u_1, \ldots, u_k \rangle)$, and for each i, the string u_i is substituted in place of variable x_i in p to obtain an example string $p[x_1 : u_1, \ldots, x_k : u_k]$. Let the distribution on positive examples of p induced by probability measure $\mathbf{Pr}$ be denoted by D_s^+.

We assume an arbitrary probability distribution D_s^- on negative examples of target pattern p of length at most ns. Note that a positive example has length at most ns, since each substitution string has length at most s.

Definition 1 *The class P_k of k-variable patterns is* PAC-learnable *(with respect to a positive product distribution and an arbitrary negative distribution) in terms of hypothesis space H iff there exists an algorithm A such that on input of any parameters n, s, ϵ, and δ, for any pattern p of P_k of length at most n, and for any product distribution D_s^+ on positive examples of p, and any distribution D_s^- on negative examples of p, if A has access to D_s^+ and D_s^-, then A produces an element $h \in H$ such that, with probability at least $1 - \delta$, $\sum_{h(w)=0} D_s^+(w) \leq \epsilon$ and $\sum_{h(w)=1} D_s^-(w) \leq \epsilon$. The run time of A is required to be polynomial in $n, s, \frac{1}{\epsilon}, \frac{1}{\delta}$, and $|\Sigma|$.*

Let $SUB = (\Sigma^{[s]})^k$, the set of all possible k-tuples of substitution strings. An *event* E is a subset of SUB. For any event E, let $E[x_{i_1} : u_1, \ldots, x_{i_d} : u_d]$ be the subset of elements of E where the i_jth component is the string u_j, for $1 \leq j \leq d$.

A variable x_i is *bound* in event E if there exists a substitution string u such that $E = E[x_i : u]$. In this case, we say that x_i is bound to u. (There can be at most one such u.) Define $bound(E) = \{i : 1 \leq i \leq k$ and x_i is bound in $E\}$. Define $free(E) = \{1, 2, \ldots, k\} - bound(E)$. Define $bindings(E)$ to be the list of bindings for each bound variable of E. That is, if $bound(E) = \{i_1, \ldots, i_d\}$, and for each $j, 1 \leq j \leq d$, variable x_{i_j} is bound to substitution string u_j, then $bindings(E) = x_{i_1} : u_1, \ldots, x_{i_d} : u_d$. Given an event E, there is a subpattern of the target pattern p that is induced by the bound variables of E. This induced subpattern is the pattern $p[bindings(E)]$, hereafter written p_E. Notice that if $bound(E) = \{1, 2, \ldots, k\}$ (i.e., all variables are bound), then p_E is a string of Σ^+, which is also interpreted as a 0-variable pattern.

The key property of p_E that we will exploit is that for any event E, and for any $\langle u_1, \ldots, u_k \rangle \in E$, $p[x_1 : u_1, \ldots, x_k : u_k] \in L(p_E)$. In other words, the pattern p_E "covers" the portion of the language $L(p)$ when we consider only substitution k-tuples $\langle u_1, \ldots, u_k \rangle$ drawn from the set E.

3 The Algorithm COVER

We describe an algorithm COVER, that *PAC*-learns (with respect to an arbitrary product distribution on positive examples, and an arbitrary distribution on negative examples) the class of k-variable patterns in terms of the union of at most $poly(n, s, \frac{1}{\epsilon}, \frac{1}{\delta}, |\Sigma|)$ of patterns over k or fewer variables.

Before describing algorithm COVER, we describe a simple variant of a greedy algorithm that was used in [**?**, **?**]. GREEDY (Figure 1 – last page) takes a string w and a list of d variable bindings of ($1 \leq d \leq k$). On input w and list $x_{i_1} : u_1, \ldots, x_{i_d} : u_d$, GREEDY attempts to find, one symbol at a time, a d-variable pattern p_d (over variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_d}\}$) such that $p_d[x_{i_1} : u_1, \ldots, x_{i_d} : u_d] = w$.

Initially, GREEDY looks for the first match in w of any substitution string u_j (breaking ties in favor of shorter strings). If u_j is the first match at some position of w, GREEDY assumes that all prior characters of w are constants of p_d, and that the substring u_j found was generated by x_{i_j}. It outputs x_{i_j}, and continues in this fashion, iteratively finding the next matching substitution that does not overlap with the portion of the input string "parsed" so far.

Algorithm COVER uses GREEDY as a subroutine. COVER, on input parameters n, s, ϵ, δ, and Σ, first computes the values $m_P = m_1 + m_2$ and m_N, described later, each bounded above by a polynomial in the parameters n, s, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and $|\Sigma|$. Then COVER executes the code in Figure 2 (last page). Briefly, COVER takes the first m_1 positive examples, and from each generates many candidate patterns by guessing a binding (of d between 1 and k variables), and calling GREEDY to produce a pattern (of d variables). The candidate set is then pruned so that all remaining candidate patterns are consistent with all m_N negative examples obtained, and finally, a small collection of patterns from the remaining candidate set is produced that accepts most of the remaining m_2 positive examples. This last step is achieved by using a greedy heuristic for the partial set cover problem [**?**]. It is easily argued that COVER runs in polynomial time for constant k.

4 Good Things and Bad Things

The greedy algorithm takes as input a string w and a binding $x_{i_1}:u_1, \ldots, x_{i_d}:u_d$ of variables to substrings of w, and outputs a pattern $GREEDY(w, x_{i_1}:u_1, \ldots, x_{i_d}:u_d)$. For any event E, the event $GOOD(E)$, will consist of k-tuples $\langle u_1, \ldots, u_k\rangle \in E$ such that if the set POS contains $w = p[x_1:u_1, \ldots, x_k:u_k]$, then in step 3 of algorithm COVER, the pattern p_E is produced by GREEDY when supplied with the correct bindings.

Definition 2 *Let E be any event with d free variables. If $d = 0$, then define $GOOD(E) = E$. (Note that there exists at most one element $\langle u_1, \ldots, u_k\rangle \in E$ if $d = 0$.) Otherwise, if $x_{i_1}, \ldots x_{i_d}$ are the d free variables of E, then*

$$GOOD(E) = \{\langle u_1, \ldots, u_k\rangle \in E : \text{GREEDY}(p[x_1:u_1, \ldots, x_k:u_k], x_{i_1}:u_{i_1}, \ldots, x_{i_d}:u_{i_d}) = p_E\}.$$

Let $BAD(E) = E - GOOD(E)$. Suppose for some event E, a GOOD thing does not happen for the k-tuple $\langle u_1, \ldots, u_k\rangle$ of E. That is, the positive example string induced by the k-tuple does not result in the pattern p_E being included in C during the run of COVER. It can be argued that there are at most a polynomial number of reasons that $\langle u_1, \ldots, u_k\rangle \in BAD(E)$. For example, suppose that GREEDY is called with example string $p[x_1:u_1, \ldots, x_k:u_k]$, and binding $x_1:u_1, \ldots, x_d:u_d$ (i.e., the bindings given to GREEDY are correct). If the pattern begins with x_j, but in fact the supplied substitution string u_i is a proper prefix of u_j, then GREEDY will incorrectly output x_i as the first symbol of the pattern. Thus, substitution tuples $\langle u_1, \ldots, u_k\rangle$ such that for some i and j, u_i is a prefix of u_j, may cause GREEDY to fail to output the correct pattern when given the correct bindings. Thus if for all i, j, we define

$$B(E, i, j) = \{\langle u_1, \ldots, u_k\rangle : \langle u_1, \ldots, u_k\rangle \in E \textit{ and } u_i \textit{ is a prefix of } u_j\},$$

we obtain at most k^2 subevents of E that cover part of $BAD(E)$. For our purposes, we can define the events $B(E, i, j)$ to be disjoint.

A careful analysis of all additional ways that the GREEDY algorithm might fail when given a string and the correct bindings that produced that string yields a collection of at most $3k^2 + 3k$ reasons for failure, and corresponding subevents $B_l(E)$ (after appropriate renaming — e.g., each event $B(E, i, j)$ above is renamed as some $B_l(E)$ for a subscript l based on i and j). We can then prove that these subevents satisfy the following lemma, whose statement is implicit in [**?**]:

Lemma 1 *For any event E, there are at most $c = 3k^2 + 3k$ sets $\{B_l(E)\}_{l=1}^{c}$ such that the sets $B_l(E)$ are pairwise disjoint, each is a subset of E, and such that $BAD(E) \subseteq \bigcup_{l=1}^{c} B_l(E)$.*

We next show that for each of these bad subevents $B_l(E)$ of any event E, we can restrict one of the variables of $free(E)$ to polynomially many substitution strings, and still cover most of the bad event $B_l(E)$. The size of the restricted set of substitution strings depends inversely on $\mathbf{Pr}(B_l(E))$; for this reason in the subsequent analysis it will be necessary to avoid events E such that $\mathbf{Pr}(B_l(E))$ is too small. As an example, consider the bad event $B(E, i, j)$ above, consisting of k-tuples of E such that the substitution string u_i is a prefix of u_j. In this case, we have the following lemma:

Lemma 2 *Let $E \subseteq SUB$ be an event, and let $i, j \in free(E)$. Let $q = \mathbf{Pr}(B(E, i, j))$. Then for any $0 < \gamma < 1$ there is a set of strings U_i of size at most $\frac{s}{\gamma q}$ such that*

$$\mathbf{Pr}(\{\langle u_1, \ldots, u_k\rangle : \langle u_1, \ldots, u_k\rangle \in B(E, i, j) \text{ and } u_i \notin U_i\}) \leq \gamma q.$$

Proof: The set U_i will be given as a union of recursively defined subsets U_i^r. Let $U_i^0 = \emptyset$. Then for $t \geq 1$, let

$$B(E, i, j)_t = \{\langle u_1, \ldots, u_k\rangle : \langle u_1, \ldots, u_k\rangle \in B(E, i, j) \text{ and } u_i \notin \bigcup_{r=0}^{t-1} U_i^r\}.$$

Thus $B(E, i, j)_t$ is the set of all k-tuples $\langle u_1, \ldots, u_k\rangle$ in the set $B(E, i, j)$ whose coordinate u_i is not already contained in one of the previous U_i^r. Let

$$q_t = \mathbf{Pr}(B(E, i, j)_t).$$

Note that $B(E, i, j)_1 = B(E, i, j)$ and $q_1 = q$. For any substitution string v, define the set

$$S_t(v) = \{\langle u_1, \ldots, u_k\rangle : u_i \text{ is a prefix of } v \text{ and } u_i \notin \bigcup_{r=0}^{t-1} U_i^r\}.$$

This is the set of all k-tuples $\langle u_1, \ldots, u_k\rangle$ with a coordinate u_i that is a prefix of v and is not contained in one of the previous U_i^r. Now

$$\mathbf{Pr}(\{\langle u_1, \ldots, u_k\rangle : u_i \text{ is a prefix of } u_j \text{ and } u_i \notin \bigcup_{r=0}^{t-1} U_i^r\}) \geq \mathbf{Pr}(B(E, i, j)_t) = q_t.$$

But we also have

$$\mathbf{Pr}(\{\langle u_1, \ldots, u_k\rangle : u_i \text{ is a prefix of } u_j \text{ and } u_i \notin \bigcup_{r=0}^{t-1} U_i^r\})$$

$$= \sum_v \mathbf{Pr}(\{\langle u_1, \ldots, u_k\rangle : u_j = v\})\mathbf{Pr}(S_t(v)).$$

Thus, for some fixed string v_t we must have $\mathbf{Pr}(S_t(v_t)) \geq q_t$. We then let

$$U_i^t = \{u : u \textit{ is a prefix of } v_t\}.$$

We now establish two claims we need to prove the lemma.

Claim 1. For all $t \geq 0$, $q_{t+1} \leq q_t$. This follows from the fact that $B(E,i,j)_t \supseteq B(E,i,j)_{t+1}$.

Claim 2. For all $t \geq 1$, $q_t \leq \frac{1}{t}$. Suppose for contradiction that $q_t > \frac{1}{t}$. Then by Claim 1, $q_1, \ldots, q_{t-1} > \frac{1}{t}$. Thus, $q_1 + \cdots + q_t > 1$. This implies that $\mathbf{Pr}(S_1(v_1)) + \cdots + \mathbf{Pr}(S_t(v_t)) > 1$. This is a contradiction, since the sets $S_r(v_r)$ are disjoint (for $r_2 > r_1$, the definition of $S_{r_2}(v_{r_2})$ explicitly excludes all $\langle u_1, \ldots, u_k\rangle$ whose component u_i is a prefix of v_{r_1}, and $S_{r_1}(v_{r_1})$ contains only such $\langle u_1, \ldots, u_k\rangle$). Thus we must have $q_t \leq \frac{1}{t}$.

We now complete the proof of the lemma. Let

$$U_i = \bigcup_{t=0}^{\frac{1}{\gamma q}} U_i^t.$$

Since $|U_i^r| \leq s$ we have $|U_i| \leq \frac{s}{\gamma q}$. Furthermore,

$$\mathbf{Pr}(\{\langle u_1, \ldots, u_k\rangle : \langle u_1, \ldots, u_k\rangle \in B(E,i,j) \textit{ and } u_i \notin U_i\})$$

$$= \mathbf{Pr}(B(E,i,j)_{\frac{1}{\gamma q+1}}) = q_{\frac{1}{\gamma q+1}} \leq \gamma q$$

by Claim 2. □

Similar lemmas corresponding to each of the sets $B_j(E)$ can be proved. In the above example lemma, we obtained the bound $|U_i| \leq \frac{s}{\gamma q}$. The upper bounds we obtain for the lemmas corresponding to other sets $B_j(E)$ is at worst $|U_i| \leq \frac{2ns^4}{\gamma q}$. Each such lemma implies that if we restrict the pattern p_E to one of a polynomial number of substitution strings for one of the free variables, then the resulting patterns cover all but γ of the examples induced by k-tuples of $B_j(E)$. This follows because in all but γ of $B_j(E)$, some particular (free) variable x_i always receives a substitution u_i from a polynomially sized set U_i of possibilities. In particular, we have:

Lemma 3 *For each $j \leq c$, for any γ, $0 < \gamma < 1$, and for all events E, if $B_j(E)$ is nonempty, then there exists an index i_j between 1 and k such that $i_j \in$ free(E), and a set U_{i_j} of size at most $\frac{2ns^4}{\gamma \mathbf{Pr}(B_j(E))}$ such that the events in the collection $\{B_j(E)[x_{i_j} : u]\}_{u \in U_{i_j}}$ of subevents of E are disjoint, and satisfy*

$$\sum_{u \in U_{i_j}} \mathbf{Pr}(B_j(E)[x_{i_j} : u]) \geq (1-\gamma)\mathbf{Pr}(B_j(E)).$$

Proof: Omitted from this abstract.

5 The Event Tree

We define an *event tree* T with the following properties: (1) The root (depth 0) of T is the set SUB of all possible k-tuples; (2) Each node in the tree will be an event, and the children of an event/node will be disjoint subevents of the node; (3) If E' is a child of event E in the tree, then $|bound(E')| \geq |bound(E)| + 1$. Thus for any event E in the tree, $|bound(E)| \geq depth(E)$, and the maximum depth will be k.

T will have other properties that we discuss later. T is defined inductively. Let the root be SUB, the set of all k-tuples. If E is an event/node at depth $d \leq k$, then, inductively, $|bound(E)| = l \geq d$. Let $bound(E) = \{i_1, \ldots i_l\}$, and let $bindings(E) = x_{i_1} : u_1, \ldots, x_{i_l} : u_l$. Let γ be a number $(0 < \gamma < 1)$ to be specified later. If $\mathbf{Pr}(GOOD(E)) \geq \gamma \mathbf{Pr}(E)$, then E is a leaf of the tree. Intuitively, if $\mathbf{Pr}(GOOD(E))$ is large enough, we'll have a reasonable chance of including p_E in the candidate set C. However, if $\mathbf{Pr}(GOOD(E))$ is small, then it may be ignored (since $BAD(E)$ accounts for most of the event E), and we will cover most of the event $BAD(E)$ with a polynomial number of leaves.

Note that if $d = k$, then since $bound(E)$ increases by at least one as the depth increases by one (this is inductively implicit in our construction), we have $GOOD(E) = E$. Thus if $d = k$, E is a leaf by the criterion just given. Thus all leaves are at depth k or less.

Otherwise, $\mathbf{Pr}(GOOD(E)) < \gamma \mathbf{Pr}(E)$. Now by Lemma 1, the event $BAD(E)$ is covered by events $B_1(E), \ldots B_c(E)$, where $c = 3k^2 + 3k$. The children of event E will be chosen by obtaining, from *some* of the events $B_j(E)$, a polynomial number of subevents of $B_j(E)$.

For each j between 1 and c, if $\mathbf{Pr}(B_j(E)) < \frac{\gamma \mathbf{Pr}(E)}{c}$, then ignore $B_j(E)$ entirely, and obtain no children from it.

For each j between 1 and c such that $\mathbf{Pr}(B_j(E)) \geq \frac{\gamma \mathbf{Pr}(E)}{c}$, we obtain at most $\frac{2ns^4}{\gamma \mathbf{Pr}(B_j(E))} \leq \frac{2ns^4 c}{\gamma^2 \mathbf{Pr}(E)}$ subevents of $B_j(E)$ that are children of the node/event E as follows. Let $i_j \in free(B_j(E)) \subseteq free(E)$ be such that the sets $\{B_j(E)[x_{i_j} : u]\}_{u \in U_{i_j}}$ are disjoint subevents of $B_j(E)$, as guaranteed by Lemma 3, where

$$\sum_{u \in U_{i_j}} \mathbf{Pr}(B_j(E)[x_{i_j} : u]) \geq (1 - \gamma) \mathbf{Pr}(B_j(E)),$$

and such that

$$|U_{i_j}| \leq \frac{2ns^4}{\gamma \mathbf{Pr}(B_j(E))} \leq \frac{2ns^4 c}{\gamma^2 \mathbf{Pr}(E)},$$

the latter inequality following from the lower bound on $\mathbf{Pr}(B_j(E))$.

Further note that each event of the form $B_j(E)[x_{i_j} : u]$ is disjoint from any other event $B_{j'}(E)[x_{i_{j'}} : u']$ for any $j' \neq j$, and for any $x_{i_{j'}}$ and u', since $B_j(E)$ and $B_{j'}(E)$ are disjoint.

Now, for any $u \in U_{i_j}$, let $B_j(E)[x_{i_j} : u]$ be a child of E only if

$$\mathbf{Pr}(B_j(E)[x_{i_j} : u]) \geq \frac{\gamma^3 (\mathbf{Pr}(E))^2}{2ns^4 c^2}.$$

Lemma 4 *If event E in event tree T is not a leaf, then the children $\{E_i\}_{i=1}^r$ of E are disjoint events, $|bound(E_i)| > |bound(E)|$, and $\mathbf{Pr}(\bigcup_{i=1}^r E_i) \geq (1 - 4\gamma) \mathbf{Pr}(E)$.*

Proof: That the children are disjoint, and that each child has at least one more bound variable than its parent, follows immediately from the construction of the event tree.

We can categorize the portion of event E that is not covered by any child of E into the following groups:

1. $GOOD(E)$ is not covered, which has probability at most $\gamma(\mathbf{Pr}(E))$.

2. If event $B_j(E)$ is such that $\mathbf{Pr}(B_j(E)) < \frac{\gamma \mathbf{Pr}(E)}{c}$, then $B_j(E)$ is ignored completely. There are at most c sets $B_j(E)$ (and hence at most as many ignored), thus the total fraction of E that is uncovered due to the ignored events $B_j(E)$ is at most $\gamma \mathbf{Pr}(E)$.

3. For each $B_j(E)$ that was not ignored, the sets $B_j(E)[x_{i_j} : u]$ do not completely cover all of the event $B_j(E)$. But we do have the bound

$$\mathbf{Pr}(\bigcup_{u \in U_{i_j}} B_j(E)[x_{i_j} : u]) \geq (1-\gamma)\mathbf{Pr}(B_j(E)).$$

   Since the sets $B_j(E)$ are mutually disjoint, taking the union of each side over all unignored sets $B_j(E)$, we obtain

$$\mathbf{Pr}\left(\bigcup_{j:B_j(E) \text{ was not ignored}} \bigcup_{u \in U_{i_j}} B_j(E)[x_{i_j} : u]\right) \geq (1-\gamma)\mathbf{Pr}(\bigcup_{j:B_j(E) \text{ was not ignored}} B_j(E)).$$

   Thus the children induced by the sets $B_j(E)$ cover all but at most $\gamma \mathbf{Pr}(\cup B_j(E))$ of the events $\cup B_j(E)$. Thus the amount uncovered is at most $\gamma \mathbf{Pr}(E)$.

4. Finally, for any unignored $B_j(E)$, some of the elements of $\{B_j(E)[x_{i_j} : u]\}_{u \in U_{i_j}}$ are not necessarily included as children of E. For each element not included, a subset of event E of probability at most $\frac{\gamma^3(\mathbf{Pr}(E))^2}{2ns^4c^2}$ is not covered. The number of such sets is at most the product of the number of sets $B_j(E)$ and the number of possible u's in any set U_{i_j}. Thus we fail to cover at most $\frac{2ns^4c^2}{\gamma^2 \mathbf{Pr}(E)}$ sets, each of probability at most $\frac{\gamma^3(\mathbf{Pr}(E))^2}{2ns^4c^2}$, a total loss of at most $\gamma \mathbf{Pr}(E)$.

By the above analysis, the disjoint events consisting of the children of E have total probability at least $(1-4\gamma)\mathbf{Pr}(E)$. □

For any event E in the tree, define $leaves(E)$ to be the set of events that are leaves in the subtree rooted at E. Note that if E is itself a leaf, then $leaves(E) = \{E\}$. Let the *height* of a node in the tree be the length of the longest path from it to one of its leaves, and let the *depth* of a node in the tree be the length of the path from the root to the node.

Lemma 5 *If E is an event in T at height h, then $\sum_{L \in leaves(E)} \mathbf{Pr}(L) \geq (1-4\gamma)^h \mathbf{Pr}(E)$.*

Proof: The proof is by induction on h. If $h = 0$, then E is a leaf, and the lemma follows trivially. Assume inductively that the lemma is true for events at a height $h-1 \leq k-1$, and consider any event E at height $h \leq k$. By the construction of the event tree T, we know that if $E_1, \ldots, E_t$ are the children of event E, then $\sum_{i=1}^{t} \mathbf{Pr}(E_i) \geq (1-4\gamma)\mathbf{Pr}(E)$. The children of E are mutually disjoint events, and each is at height at most $h-1$. Further, each leaf of the tree rooted at E must be a leaf of a tree rooted at one of the children E_i of E. Then, applying the inductive hypothesis and the above observations, we have

$$\sum_{i=1}^{t} \sum_{L \in leaves(E_i)} \mathbf{Pr}(L) \geq \sum_{i=1}^{t} (1-4\gamma)^{h-1} \mathbf{Pr}(E_i) \geq (1-4\gamma)^{h-1} \sum_{i=1}^{t} \mathbf{Pr}(E_i) \geq (1-4\gamma)^h \mathbf{Pr}(E).$$

□

Lemma 6 *If E is an event in the tree at depth $d \geq 0$ then*

$$\mathbf{Pr}(E) \geq \left(\frac{\gamma^3}{2ns^4(3k^2+3k)^2}\right)^{2^d-1}.$$

Proof: We prove the lemma by induction on d. If $d = 0$, then the event E is at the root, and $\mathbf{Pr}(E) = 1$, which satisfies the bound required by the lemma. Assume inductively that the lemma holds for events E at depth $d-1 < k$. Let E be an event at depth d in the tree. Let E' be the parent of E in the tree, where E' has depth $d-1$. Then by construction, E would not have been included as a child of E' unless $\mathbf{Pr}(E) \geq \frac{\gamma^3(\mathbf{Pr}(E'))^2}{2ns^4(3k^2=3k)^2}$. Applying the inductive hypothesis to $\mathbf{Pr}(E')$, we obtain

$$\mathbf{Pr}(E) \geq \frac{\gamma^3}{2ns^4(3k^2+3k)^2}\left(\left(\frac{\gamma^3}{2ns^4(3k^2+3k)^2}\right)^{2^{d-1}-1}\right)^2 \geq \left(\frac{\gamma^3}{2ns^4(3k^2+3k)^2}\right)^{2^d-1}.$$

□

6 Putting it All Together

We have now defined the event tree T rooted at SUB, and obtained lower bounds on the probability of events in T and the portion of the distribution they cover in terms of the unspecified parameter γ. We now argue that for appropriate γ, T has polynomial size, and its leaves cover almost all of the distribution.

Lemma 7 *Let γ be such that $(1-4\gamma)^k \geq 1 - \frac{\epsilon}{4}$. Then $\sum_{E \in leaves(SUB)} \mathbf{Pr}(E) \geq 1 - \frac{\epsilon}{4}$ and*

$$|leaves(SUB)| \leq l = \left(\frac{2ns^4(3k^2+3k)^2}{\gamma^3}\right)^{2^k-1}.$$

Furthermore, for any $E \in leaves(SUB)$, $\mathbf{Pr}(GOOD(E)) \geq \frac{\gamma}{l}$.

Proof: Follows from Lemmas 5, 6 and the disjointness of the leaf events. Note that $\frac{1}{\gamma}$ is polynomial in $\frac{1}{\epsilon}$, and that l (and hence $\frac{\gamma}{l}$) is polynomial in the relevant parameters. □

Thus, we have shown that the event tree T has a polynomial number of leaves, each of significant probability. We now use these facts to argue that the set C of candidate patterns constructed by COVER contains a subset whose disjunction has error at most $\frac{\epsilon}{4}$ on distribution D_s^+, and no error on D_s^-.

Lemma 8 *Let $m_1 \geq \lceil \frac{l}{\gamma} \ln \frac{4l}{\delta} \rceil$ and $C = \{p_1, \ldots, p_r\}$ be the pool of candidate patterns constructed by calls to* GREEDY *in step 3 of* COVER. *Then with probability at least $1 - \frac{\delta}{4}$ there are l patterns $\{p_{i_1}, \ldots, p_{i_l}\} \subseteq C$ satisfying*

$$\mathbf{Pr}(\langle u_1, \ldots, u_k \rangle : p[x_1 : u_1, \ldots, x_k : u_k] \in L(p_{i_1}) \cup \cdots \cup L(p_{i_l})) \geq 1 - \frac{\epsilon}{4} \quad (1)$$

$$L(p_{i_1}) \cup \cdots \cup L(p_{i_l}) \subseteq L(p). \quad (2)$$

Proof: The l patterns are the patterns p_E for each $E \in \mathit{leaves}(SUB)$. If $m_1 \geq \lceil \frac{l}{\gamma} \ln \frac{4l}{\delta} \rceil$, then with probability at least $1 - \frac{\delta}{4}$ a sample of m_1 positive examples contains for each leaf E at least one element of $GOOD(E)$. This follows from the lower bound on $\mathbf{Pr}(GOOD(E))$ given in Lemma 7 and Chernoff bounds. By definition of $GOOD(E)$ we have that with probability at least $1 - \frac{\delta}{4}$, C contains, for each leaf E, the pattern p_E. By Lemma 7 and our choice of γ, the probability that a random $\langle u_1, \ldots, u_k \rangle \in SUB$ is contained in some leaf E is at least $1 - \frac{\epsilon}{4}$. Thus the example $p[x_1 : u_1, \ldots, x_k : u_k]$ is also a positive example of p_E, proving Equation (1). Equation (2) follows immediately from the fact that p_E is a subpattern of p, and thus $L(p_E) \subseteq L(p)$. It can be shown that the size r of the candidate set C satisfies $r \leq m_1(ns^2)^k k^{k+1}$, so r is bounded above by a polynomial in n, s, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$. □

In addition to the l patterns satisfying Equations (1) and (2), the pattern pool C may initially contain some patterns p_i such that $L(p_i) \not\subseteq L(p)$. To eliminate the most offensive of these, in step 4 COVER discards any pattern $p_i \in C$ such that one of the m_N negative examples is contained in $L(p_i)$. There is a constant c_1 such that if m_N is chosen so that $m_N \geq c_1 \frac{r}{\epsilon} \ln \frac{r}{\delta}$, then with probability at least $1 - \frac{\delta}{4}$, any p_i that has probability at least $\frac{\epsilon}{r}$ of accepting a negative example will in fact accept one of the m_N negative examples drawn. Thus *any* disjunction of patterns from C has probability at most $r\frac{\epsilon}{r} = \epsilon$ of accepting a negative example. In other words, with high probability any disjunction of remaining patterns has error at most ϵ with respect to the negative distribution. Without loss of generality, let $C' = \{p_1, \ldots, p_{r'}\}$ be the set of remaining patterns.

The goal of COVER in step 5 is to discover among the remaining patterns a good approximation to the disjunction of l patterns satisfying Equations (1) and (2). Note that Equation (1) is a probabilistic statement; in order to apply the partial cover algorithm of [**?**], we need the following:

Lemma 9 *Let $m_2 \geq c_2 \frac{1}{\epsilon} \ln \frac{1}{\delta}$ for some constant c_2. Let R be a multiset of m_2 random positive examples. Then for $\{p_{i_1}, \ldots, p_{i_l}\}$ satisfying Equation* (1) *above, with probability at least $1 - \frac{\delta}{4}$ we have that*
$|R \cap (L(p_{i_1}) \cup \cdots \cup L(p_{i_l}))| \geq (1 - \frac{\epsilon}{2})|R|$, where the cardinalities are multiset cardinalities.

Proof: Follows from Equation (1) and application of Chernoff bounds. □

For each $p_i \in C'$, let the multiset S_i be defined by $S_i = L(p_i) \cap R$. Then by Lemma 9, with probability at least $1 - \frac{\delta}{4}$, among the multisets $S_1, \ldots, S_{r'}$ there is a collection of l of the S_i that collectively cover a fraction $1 - \frac{\epsilon}{2}$ of the multiset R.

The results of [**?**] give a polynomial-time algorithm that will find at most $c_3 l \ln m_2$ of the S_i that cover $1 - \frac{\epsilon}{2}$ of R for some constant c_3. Without loss of generality, let $C_{cover} = \{p_1, \ldots, p_{c_3 l \ln m_2}\}$ be the corresponding patterns obtained in step 5 of COVER. Then the hypothesis $L(C_{cover}) = L(p_1) \cup \cdots \cup L(p_{c_3 l \ln m_2})$ is consistent with a fraction $1 - \frac{\epsilon}{2}$ of the positive sample R. We now argue that with high probability, the error of this hypothesis is at most ϵ.

Theorem 10 *Let $C_{cover} = \{p_1, \ldots, p_{c_3 l \ln m_2}\}$ be the the set of patterns output by algorithm COVER in step* 5. *Then with probability at least $1 - \delta$,*

$$L(C_{cover}) = L(p_1) \cup \cdots \cup L(p_{c_3 l \ln m_2})$$

has error at most ϵ with respect to both the positive and negative distributions.

Proof: The probability that $L(C_{cover})$ has error greater than ϵ on the negative distribution D_s^- is at most $\frac{\delta}{4}$; this follows immediately from the removal of patterns from C in step 4 discussed above.

By Lemma 8, the probability that the collection C produced in step 3 of COVER fails to contain a subcollection $\{p_{i_1}, \ldots, p_{i_l}\}$ satisfying equation (1) is at most $\frac{\delta}{4}$. By Lemma 9, the probability that the set $\{p_{i_1}, \ldots, p_{i_l}\}$ fails to cover at least $1 - \frac{\epsilon}{2}$ of R is at most $\frac{\delta}{4}$. We invoke a generalization of Occam's Razor from [**?**] to argue that $L(C_{cover})$ in fact has error at most ϵ with respect to the positive distribution. Note that the total number of symbols needed to represent the patterns in C_{cover} is at most $(c_3 l \ln m_2)ns$; thus the *effective hypothesis space* $H = H(n, s, \epsilon, \delta)$ satisfies $|H| \leq (k + |\Sigma|)^{(c_3 l \ln m_2)ns}$. (If A is a *PAC*-learning algorithm for P_k with hypothesis space H, then the *effective hypothesis space* of A on target patterns of length n is denoted by $H(n, s, \epsilon, \delta)$, and consists of that subset of H that A might ever output when run with input parameters n, s, ϵ, δ.) By the results of [**?**] there is a constant c_4 such that if $m_2 \geq c_4(\frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{1}{\epsilon} \ln(k + |\Sigma|)^{(c_3 l \ln m_2)ns})$ then with probability at least $1 - \frac{\delta}{4}$, $L(C_{cover})$ has error at most ϵ with respect to the positive distribution D_s^+.

In summary, the probability that $L(C_{cover})$ has error greater than ϵ on the negative distribution is at most $\frac{\delta}{4}$, and the probability that $L(C_{cover})$ has error greater than ϵ on the positive distribution is at most $\frac{\delta}{4} + \frac{\delta}{4} + \frac{\delta}{4} = \frac{3\delta}{4}$. Thus with probability at least $1 - \delta$, $L(C_{cover})$ has error at most ϵ on both positive and negative distributions. Note that the total number of positive examples $m_P = m_1 + m_2$ and the total number of negative examples m_N are polynomial in $n, s, \frac{1}{\epsilon}, \frac{1}{\delta}$, and $|\Sigma|$. □

7 Conclusions and Future Research

We have given a polynomial-time algorithm for learning the class of k-variable pattern languages from positive and negative examples under a wide and natural class of distributions on the examples. Our algorithm allows empty substitutions and can be extended to handle restricted homomorphisms on the substitution strings, to be discussed in the full paper.

The algorithm and analysis presented here immediately suggest a number of interesting areas for further research. Perhaps the most obvious of these is that of finding either positive or negative results for learning k-variable patterns in the *PAC*-learning model under *arbitrary* distributions on the positive examples (here we considered only product distributions).

It would also be nice to have results for learning general patterns in polynomial time. While the results mentioned in the Introduction seem to suggest that this is computationally difficult, the problem of learning general patterns by an hypothesis space other than single patterns in the *PAC*-learning model in polynomial time remains open.

Another interesting question concerns the performance of the algorithm presented here. For instance, it is possible that a tighter analysis of the event tree might yield considerably improved sample and time complexity bounds for our algorithm. The algorithm may also demonstrate performance better than the worst-case analysis on more restrictive classes of distributions. Finally, extensions to our algorithm might be found by investigating larger classes of homomorphisms and other operations on the substitution strings.

References

GREEDY$(w, x_{i_1} : u_1, \ldots, x_{i_d} : u_d)$:

1. $a \leftarrow 1$. (Initialize pointer into string w.)
2. If $a \geq |w| + 1$ then HALT.
3. Let MATCH $= \{j : u_j = w(a, |u_j|)\}$. (MATCH contains indices of supplied substitution strings that appear as substrings of w starting at the current position a of w.)
4. If MATCH $= \emptyset$ then (the next symbol of the pattern is a constant)
   (a) Output $w(a, 1)$. (Output the constant.)
   (b) $a \leftarrow a + 1$. (Move the pointer.)
   (c) Return to step 2.
5. (Else MATCH $\neq \emptyset$.) Choose an arbitrary element j of the set $\{j : j \in$ MATCH and for all $l \in$ MATCH, $|u_j| \leq |u_l|\}$. (Pick a shortest matching substring.)
6. Output x_{i_j}. (Hopefully, x_{i_j} is what produced u_j.)
7. $a \leftarrow a + |u_j|$. (Move the pointer to the next unparsed character of the input.)
8. Return to step 2.

Figure 1: Algorithm GREEDY

COVER$(n, s, \epsilon, \delta, \Sigma)$:

1. From distributions D_s^+ and D_s^-, obtain sets POS of $m_P = m_1 + m_2$ positive examples, and NEG of m_N negative examples.
2. $C \leftarrow \emptyset$. (C will be a set of candidate patterns).
3. For each of the first m_1 elements w of POS do:
   (a) For each d between 1 and k, and for each list $x_{i_1} : u_1, \ldots, x_{i_d} : u_d$ of bindings of some subset of d variables of $\{x_1, \ldots, x_k\}$ to d substrings $u_1, \ldots, u_d$ of w, each of length at most s, add the pattern GREEDY$(w, x_{i_1} : u_1, \ldots, x_{i_d} : u_d)$ to C
   (b) Add the (0-variable) pattern w to C
4. Eliminate from C any pattern q such that $L(q) \cap$ NEG $\neq \emptyset$.
5. Run the greedy partial set cover algorithm of [**?**] to obtain, with high probability, a subcollection C_{cover} of elements of C that are consistent with $1 - \frac{\epsilon}{4}$ of the remaining m_2 positive examples.

Figure 2: Algorithm COVER