Sampling Minimal Frequent Boolean (DNF) Patterns

Geng Li and Mohammed J. Zaki
Rensselaer Polytechnic Institute, Troy, NY
{lig2, zaki}@cs.rpi.edu

ABSTRACT

We tackle the challenging problem of mining the simplest Boolean patterns from categorical datasets. Instead of complete enumeration, which is typically infeasible for this class of patterns, we develop effective sampling methods to extract a representative subset of the minimal Boolean patterns (in disjunctive normal form – DNF). We make both theoretical and practical contributions, which allow us to prune the search space based on provable properties. Our approach can provide a near-uniform sample of the minimal DNF patterns. We also show that the mined minimal DNF patterns are very effective when used as features for classification.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications - Data Mining

Keywords: Minimal Generator, Boolean Expression Patterns, Sampling, Pattern-based Classification

1. INTRODUCTION

Frequent pattern mining, long a mainstay of data mining, is moving away from complete enumeration methods to approaches that can effectively sample the pattern space for the most interesting patterns. This shifting trend is in keeping with the growing complexity of the data as well as the types of patterns sought. Whereas much research in the past has focused on itemset mining, i.e., conjunctive patterns, our focus is on the entire class of Boolean patterns in the disjunctive normal form (DNF), i.e., disjunctions over conjunctive patterns. Such Boolean patterns can help discover interesting relationships among attributes (e.g., gene expression mining [22]).

Complete enumeration of all frequent Boolean patterns is prohibitive in most real-world datasets, and thus the main issue is how to effectively sample a representative subset, which can in turn be used as features to build classification models. Furthermore, we focus on the problem of mining only the most simple Boolean patterns that completely characterize a subset of the data, i.e., the minimal DNF expressions. Our work makes number of novel contributions:

i) We propose the first approach to generate a near-uniform sample of the minimal Boolean expressions. Our method, based on Markov Chain Monte Carlo (MCMC) sampling, yields a succinct subset of the simplest frequent Boolean patterns.

ii) We propose a novel theoretical characterization of the minimal DNF expressions, which allows us to prune the pattern search space effectively. When combined with other optimization techniques, our approach is also practically effective. For instance, we are able to sample interesting "support-less" patterns, i.e., where the minimum frequency threshold is set to one. The pruning techniques can be applied by any method (even a complete one) for mining Boolean expressions.

iii) We perform an extensive set of experiments to demonstrate the effectiveness of our method. In particular, we classify a variety of datasets from the UCI Machine Learning Repository [13], and show that minimal DNF patterns make very effective features for classification; more so than purely conjunctive features. We also study the sample quality of our approach, as well as its scalability.

1.1 Preliminaries

Dataset: Let $\mathcal{Z} = \{z_1, z_2, \ldots, z_m\}$ be a set of binary-valued attributes or *items*, and let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a set of transactions identifiers or *tids*. A dataset $\mathcal{D}$ is a binary relation $\mathcal{D} \subseteq \mathcal{Z} \times \mathcal{T}$. $\mathcal{D}$ can also be considered as a set of tuples of the form $(t, t.X)$ where $t \in \mathcal{T}$ and $t.X \subseteq \mathcal{Z}$. Note that any categorical dataset can easily be converted into this format by assigning an item for each attribute-value pair.

Given dataset $\mathcal{D}$, we call $\mathcal{D}^T$ the *vertical* or transposed dataset comprising tuples of the form $(z, z.Y)$ where $z \in \mathcal{Z}$ and $z.Y \subseteq \mathcal{T}$. Table 1 shows an example dataset $\mathcal{D}$ and its transpose $\mathcal{D}^T$. The dataset has six items $\mathcal{Z} = \{A, B, C, D, E, F\}$ and five transactions $\mathcal{T} = \{1, 2, 3, 4, 5\}$. For example, the tuple $(2, ACDF) \in \mathcal{D}$ denotes the fact that tid 2 has four items A, C, D, F, whereas the tuple $(E, 134) \in \mathcal{D}^T$ denotes the fact that item E is contained in transactions $1, 3, 4$. For convenience, we write subsets without commas. Thus $\{A, C, D, F\}$ is written as ACDF, and so on.

tid	set of items
1	ABE
2	ACDF
3	BEF
4	ADE
5	BF

(a)

item	tidset
A	124
B	135
C	2
D	24
E	134
F	235

(b)

Table 1: Dataset $\mathcal{D}$ (a) and its transpose $\mathcal{D}^T$ (b)

Boolean Expressions: Let AND, OR, and NOT denote the logical operators. We denote a negated item (NOT z)

as $\bar{z}$. We also call $\bar{z}$ the complement of z. We use the symbols $\wedge$ and $\vee$ to denote AND and OR, respectively. For example, $A \vee B$ and $A \wedge B$ denote logical expressions A OR B, A AND B, respectively. For conciseness we also use $|$ in place of $\vee$ and we usually omit the $\wedge$ operator. For example, $A|BC|D$ denotes the Boolean expression A OR (B AND C) OR D.

A *literal* is either an item z or its complement $\bar{z}$. A clause is either the logical AND or the logical OR of a set of literals. An AND-clause contains only the AND operator over all its literals, e.g., BCD. Likewise, an OR-clause contains only the OR operator over all its literals, e.g., $C|E|F$. We assume that a clause does not contain both a literal and its complement – e.g., $A \wedge \bar{A}$ leads to contradiction, and $\bar{A} \vee A$, to a tautology.

We adopt the disjunctive normal form (DNF) to represent Boolean expressions. A Boolean expression Z is said to be in DNF if it consists of OR of AND-clauses, with the NOT operator (if any) directly preceding only literals, written as:

$$Z = \bigvee_{i=1}^{k} Z_i = \bigvee_{i=1}^{k} (z_{i1} \wedge z_{i2} \wedge \cdots \wedge z_{im_i})$$

Here each z_{ik} is a literal and each $Z_i = (z_{i1} \wedge \ldots \wedge z_{im_i})$ is an AND-clause. The size or length of a Boolean expression Z is the number of literals in Z, denoted $|Z| = \sum_{i=1}^{k} m_i$.

Tidset and Support: Given a tuple $(t, t.X) \in \mathcal{D}$, and a literal l, the *truth value* of l in t is 1 if $l \in t.X$, and 0 otherwise. Likewise, the truth value of $\bar{l}$ is 1 if $l \notin t.X$, and 0 otherwise. We say t *satisfies* a Boolean expression Z, if after replacing every literal in the Boolean expression with its truth value, the Boolean expression evaluates to true. The set of all satisfying transactions is called the *tidset* of Z, and is denoted as

$$T(Z) = \{t \in \mathcal{T} | t \text{ satisfies } Z\}$$

The number of satisfying transactions is called the *support* of Z in $\mathcal{D}$, denoted $sup(Z) = |T(Z)|$.

Minimal Boolean Expressions: Given DNF expressions $X = \bigvee_{i=1}^{m} X_i$ and $Y = \bigvee_{j=1}^{n} Y_j$, where X_i and Y_j are AND-clauses, we say that X is a *subset* of Y, denoted $X \subseteq Y$, iff there exists an injective (or into) mapping $\phi : X \rightarrow Y$, that maps each clause X_i to $\phi(X_i) = Y_{j_i}$, such that $X_i \subseteq Y_{j_i}$. If $X \subset Y$ and $|X| = |Y| - 1$, we say that X is a parent of Y, and Y is a child of X.

Definition 1. *A DNF expression Z is said to be* minimal *or* minDNF *(with respect to support) if there does not exist any expression $Y \subset Z$, such that $T(Y) = T(Z)$. A minimal AND-clause is called minAND for short. A minimal Boolean expression is also called a* minimal generator.

A minDNF expression Z is thus the simplest DNF expression with tidset $T(Z)$. Given a user-specified minimum support threshold $\sigma_{\min}$, we say that a DNF expression Z is frequent if $sup(Z) \geq \sigma_{\min}$. However, note that the support of a minDNF expression is not monotonic, since the addition of an item to a clause causes the support to drop, whereas, the addition of an item as a new clause causes the support to increase. For example, $sup(A) = 3$, since $T(A) = 124$, but $sup(AB) = 1$ (since $T(AB) = 1$) and $sup(A|E) = 4$ (since $T(A|E) = 1234$). Thus, the support of Z's children can be higher or lower. Further, any infrequent clause can be made frequent by adding additional clauses. For example, if $\sigma_{\min} = 2$, then C is infrequent, but $C|F$ is frequent. To prevent such "trivial" clauses, we also impose a minimum support threshold $\sigma^c_{\min}$ on the clauses. For any DNF expression $Z = \bigvee_{i=1}^{m} Z_i$, we say that Z is *frequent* if $sup(Z) \geq \sigma_{\min}$, and $\sup(Z_i) \geq \sigma^c_{\min}$ for all $i = 1, \ldots, m$. Since a clauses' support cannot exceed the support of the whole DNF expression, we have the condition that $\sigma^c_{\min} \leq \sigma_{\min}$ (usually, we just set them equal).

Given $\sigma_{\min}$ and $\sigma^c_{\min}$, the *complete minDNF mining task* is to enumerate all frequent minDNF expressions. However, given the huge search space, it is typically not feasible to mine the complete set of minDNF expressions. Instead, we will focus on sampling a representative subset.

Markov Chain Monte Carlo (MCMC) Methods: A *Markov chain* is a mathematical model for stochastic systems with discrete or continuous states controlled by transition probabilities. A Markov chain satisfies the property that the current state depends only on the previous state, i.e., $P(X_{t+1} = s_{t+1} | X_t = s_t, X_{t-1} = s_{t-1}, \ldots, X_0 = s_0) = P(X_{t+1} = s_{t+1} | X_t = s_t)$ for all $t \in \mathcal{N}$ and $s_t \in \mathcal{S}$, where $\mathcal{N}$ and $\mathcal{S}$ denote the time space and state space, respectively. Let $p^t(i,j)$ denote the t-step transition probability, i.e., $p^t(i,j) = P(X_{n+t} = s_{n+t} | X_n = s_n)$. If for all states $s_i, s_j \in \mathcal{S}, i \neq j$, there $\exists\ t' \in \mathcal{N}$, s.t. $P^t(i,j) > 0$ is satisfied for all $t > t'$, we call the Markov chain *aperiodic*. Given two states s_i and s_j, we say s_j is *reachable* from s_i, if $\exists\ t, s.t.\ P^t(i,j) > 0$, and we denote it as $s_i \rightarrow s_j$. If all state pairs are mutually reachable, we call the Markov chain *irreducible*. A Markov chain that is aperiodic and irreducible is called *ergodic*. An ergodic Markov chain has a stationary distribution $\pi = (\pi_i | s_i \in \mathcal{S})$, that satisfies three properties: (1) $\pi_i > 0$. (2) $\sum_{s_i \in \mathcal{S}} \pi_i = 1$. (3) $\pi P = \pi, i.e., \sum_{s_i \in \mathcal{S}} \pi_i P(i,j) = \pi_j$. The stationary distribution for an ergodic Markov chain is unique. A Markov chain is *time reversible* iff it has a stationary distribution π that satisfies the balance condition $\forall s_i, s_j \in \mathcal{S}$, $\pi_i P(i,j) = \pi_j P(j,i)$.

In the context of minDNF mining, each Markov state can be taken to be a Boolean expression, with transitions allowed, for example, only between parent and child expressions. Starting from the empty expression, we can then use Monte Carlo methods to perform random walks in the expression space to sample the minimal expressions. The main challenges include efficiency, and guaranteeing sampling quality. We address these questions below.

1.2 Related Work

Mining frequent AND-clauses has been extensively studied within the context of itemset mining [14]. The notion of minimal AND-clauses (called minimal itemset generators) was proposed in [3]. Methods that can mine minAND expressions include [11] (that focuses on finding the succinct minimal generators), CHARM-L [20] and Blosom [22]. The task of mining minimal monotone DNF expressions was proposed in [19], whereas Blosom [22] proposed a complete framework to extract the minimal DNF and pure AND-clauses, as well as the minimal CNF (conjunctive normal form – AND of OR-clauses) and pure OR-clauses. Blosom uses a two-step process to mine the minDNFs. It first mines all minimal AND-clauses, treats them as new items, as then extracts minimal OR-clauses over these composite AND-items. Disjunctive association rules have also been considered in [18]; they first mine all frequent AND-clauses, and then greedily select good OR combinations. The notion of disjunctive emerging patterns (EPs) for classification was proposed in [17]. Disjunctive EPs are Boolean expressions in CNF form, such that their support is high for the positive class and low for the negative class. However, they consider restricted CNF expressions that must contain a clause for each attribute. We mine general DNF expressions, without any constraints.

As we shall see, complete mining is infeasible for all but very high support values. Thus, the focus in recent work has shifted to sampling based approaches. One of the earliest use of sampling was for mining maximal itemsets via randomization [15]. In the context of frequent graph mining, [8] proposed a randomized sampling method to generate a small representative set of frequent maximal graph patterns; the method did not provide any sampling guarantee. The first method to sample maximal graph patterns with uniform sampling via MCMC was presented in [16]. In [2], the authors introduced a generic sampling framework to sample the output space of frequent subgraphs, which is based on MCMC algorithm as well. In the context of itemset mining, [5] proposed a randomized approximation method for counting the number of frequent itemsets. In [4] a Metropolis-Hastings algorithm for sampling closed itemsets is given. More recently, [6] presented a direct sampling approach for mining AND-clauses. Unfortunately, direct sampling cannot be used for sampling minDNFs since the pattern space of minDNFs is not connected, as it is for closed AND-clauses. We thus focus on MCMC sampling, designing an appropriate transition probability matrix that ensures near-uniform sampling of the set of minDNF patterns.

2. MINIMAL BOOLEAN EXPRESSIONS

In this section we prove some properties of minDNF expressions, which will allows us to design effective pruning strategies while sampling.

Lemma 1. *Any subset of a minimal AND-clause must also be a minimal AND-clause.*

Proof. Let X be a minAND expression, and let $Y \subset X$. Assume that Y is not minimal. Then there exists a minAND expression $Z \subset Y$, such that $t(Z) = t(Y)$. However, in this case, $t((X \setminus Y) \cup Z) = t(X)$, which contradicts the fact that X is minimal. Thus, Y must be a minAND expression. □

Theorem 1. *A DNF expression $Z = \bigvee_{i=1}^{n} Z_i$ is minDNF iff it satisfies the following two properties:*

a) For any Z_i, $(i = 1, \ldots, n)$, we have $T(Z_i) \nsubseteq \bigcup_{j \neq i} T(Z_j)$. In other words, for any tidset of a clause, it is cannot be a subset of the unions of tidsets over the other clauses.

b) If we delete any item z_{ja} from a clause Z_j to yield a new clause $Z'_j = Z_j \setminus z_{ja}$, then for the resulting DNF expression $Z' = (\bigvee_{i \neq j} Z_i) \vee Z'_j$, we have $T(Z') \neq T(Z)$.

Proof. If (a) is violated, we can simply delete Z_i without changing support, which would contradict the fact that Z is minimal. Likewise, if (b) is violated, it would contradict Z's minimality. For the reverse direction, suppose a DNF expression $Z = \bigvee_{i=1}^{n} Z_i$ satisfies properties (a) and (b). We have to show that Z is a minDNF. Assume that Z is not minimal. Then there exists a minDNF $Y = \bigvee_{j=1}^{m} Y_j$, such that $Y \subset Z$ and $T(Y) = T(Z)$, which implies that there exists an injective mapping ϕ that maps each $Y_j \in Y$ to $\phi(Y_j) = Z_i \in Z$, such that $Y_j \subseteq Z_i$ and $T(Y_j) \supseteq T(Z_i)$. There are two cases to consider: (1) If ϕ is a bijection, then $m = n$, and there exist a clause $Y_j \in Y$, such that $Y_j \subseteq \phi(Y_j) = Z_i \in Z$. However, in this case property (b) of Z is violated, since we can delete some item from $(Z_i \setminus Y_j)$, and the resulting expression will still have the same support at Z. (2) If ϕ is not a bijection, then $m < n$, and there exists a clause $Z_k \in Z$, such that $\phi^{-1}(Z_k) \notin Y$. However, in this case property (a) of Z violated, since $T(Y) = T(Z)$ implies that $T(Z_k) \subseteq \bigcup_{i \neq k} T(Z_i)$. Therefore, Z must be a minimal DNF expression. □

Lemma 2. *A minDNF consists of OR of minAND expressions, i.e., if $Z = \bigvee_{i=1}^{n} Z_i$ is minDNF, then each Z_i must be a minimal AND-clause.*

Proof. Assume some Z_i is not a minimal AND-clause. Then there exists a literal $l \in Z_i$, such that $T(Z_i \setminus l) = T(Z)$. In this case we can delete l from Z_i without affecting $T(Z)$, which violates property (b) in Theorem 1. □

Lemma 3. *If $Z = \bigvee_{i=1}^{n} Z_i$ is minDNF, for any $Z_i, Z_j \in Z$, we have $Z_i \nsubseteq Z_j$. In other words, no clause is a subset of another clause.*

Proof. Suppose $Z_i \subseteq Z_j$. Thus $T(Z_i) \supseteq T(Z_j)$ and property (a) in Theorem 1 is violated. □

Please note that Theorem 1 is a sufficient condition for Lemmas 2 and 3 but not a necessary condition. As such a DNF expression that satisfies Lemmas 2 and 3, need not be a minimal generator. We use this observation to reduce the random walk state space.

Corollary 1. *Any clause-wise subset (obtained by deleting an entire clause) of a minDNF expression Z is also minDNF.*

Proof. The proof is similar to Lemma 1. Suppose a clause-wise subset $Z_s \subset Z$ is not minDNF. Then we can replace Z_s with its equivalent minDNF, say Z'_s, in Z, without affecting the tidset of Z. This would contradict the minimality of Z. □

For example, for the example in Table 1, $B|DF|E$ is a minimal DNF generator, with tidset $T(B|DF|E) = 12345$. Thus all clause-wise subsets, namely B, DF, E, $B|DF$, $B|E$, $DF|E$ are minDNF expressions.

Corollary 2. *Disallowing the empty pattern $\emptyset$ as a valid minDNF, then a single item is always a minimal AND-clause and thus a minDNF as well.*

In our running example in Table 1, items A, B, C, D, E and F are all minDNFs.

3. MINDNF SAMPLING ALGORITHM

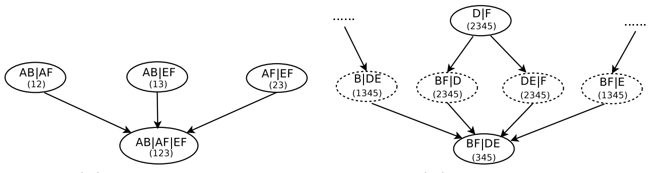
The state space for the Markov chain for the minDNF mining consists of DNF expressions linked by immediate subset-superset or parent-child relationships. The DNF partial order is generated via the following four operations: a) Add_As_Clause (AAC): add a new clause comprising a single item into the DNF. b) Add_To_Clause (ATC): add an item to an existing clause. c) Delete_The_Clause (DTC): Delete a single item clause from the DNF. d) Delete_From_Clause (DFC): Delete an item from a clause (with at least two items) in the DNF. The added or deleted item can be either an item or its complement.

Lemma 4. *The partial order graph of minimal DNF generators is disconnected.*

Consider the example in Table 1. We ignore negated items for now. $AB|AF|EF$ is a minDNF with $T(AB|AF|EF) = 123$. However, all its parents are not minimal DNFs. Take its parent $B|AF|EF$ as an example, $T(B) = 135$, $T(AF) = 2$, $T(EF) = 3$, thus property (a) in Theorem 1 is violated. Similarly, all its children are not minimal DNF generators. For instance, $AB|ACF|EF$ is not a minimal DNF generator since property (b) in Theorem 1 is violated. So, if we only keep minDNF expressions in the partial order graph, $AB|AF|EF$ would become an isolated point, and would never be reached! We provide two sampling solutions below.

3.1 Sampling in Clause-wise State Space

The first, rather naive, solution is to sample in the clause-wise state space. In particular, rather than adding or deleting an item by AAC, ATC, DTC or DFC each time, we add or delete a clause instead. From Lemma 2, every minDNF consists of only minimal AND-clauses. Also by Corollary 1, any clause-wise subset is also a a minDNF. Thus, the partial order graph is connected and all parents of the nodes in the graph will be minDNFs as shown in Figure 1(a). The solid ovals represent nodes which are minDNFs. However, this naive idea requires that we first mine the complete set of minimal AND-clauses from the dataset. As we shall see in the experiments, for reasonable support values mining all possible minimal AND-clauses is very expensive or intractable.



(a) Clause-wise (b) Item-wise

Figure 1: Clause- and Item-wise State Space

3.2 Sampling in Item-wise State Space

Given that the item-wise state space is disconnected, in order to guarantee all minDNFs are reachable, we also need to keep non-minimal DNFs in the graph. However, the goal is to reduce the number of non-minimal DNF in the graph to as few as possible while retaining all possible minDNFs. The following two lemmas help in this direction.

LEMMA 5. *Let Z be a DNF that violates Lemma 2. No extension of Z (by AAC or ATC operations) can result in a minDNF.*

PROOF. At least one of the clauses in Z is not a minimal AND-clause. Any future extension by adding a literal to Z cannot be a minDNF, since all its minimal AND-clause subsets must also be minimal by Lemma 1. ☐

This lemma states that any DNF that violates Lemma 2 should be removed from the graph, which can greatly reduce the size of the search space. Furthermore, we also remove any DNF node that violates Lemma 3. Note that this pruning still keeps the graph connected since it is always possible to find other paths. As an example, for the data in Table 1, $DE|E$ should be removed since one of its parents $DE|EF$, which is a minimal generator, can be reached by another path, namely from $DE|F$. As mentioned earlier, Definition 1 is a sufficient condition for Lemma 2 and Lemma 3, but not a necessary condition. There still exist DNFs that satisfy Lemma 2 and Lemma 3 but are not minimal generators.

Transition Probability Matrix: It is well known that a regular random walk on the partial order graph favors the nodes with higher degree [16]. In fact, if the edges are weighted, and the graph is undirected (i.e., symmetric weights: $w(u, v) = w(v, u)$ for all connected node pairs), then nodes are sampled proportional to the total weight of a node $s(u) = \sum_v w(u, v)$. We state without proof from [16]: In an ergodic random walk with an associated weighted connected (undirected) graph, the stationary distribution of a vertex is directly proportional to the sum of the edge weight incident to that vertex.

Consider a random walk on the partial order graph where there are both minimal and non-minimal DNF generators, with the transition probability matrix P given as:

$$P(u, v) = \begin{cases} \frac{w(u,v)}{\sum_{x \in N_u} w(u,x)} & \text{if } v \in N_u \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Here N_u denotes the *neighbors* of u, i.e., the set of nodes adjacent to u. Further, let $N_u^m = \{v \in N_u | v \text{ is a minDNF}\}$ and $N_u^n = \{v \in N_u | v \text{ is not a minDNF}\}$ denote set of u's neighbors that are minDNF and not minDNF, respectively. Also, let $d_u^m = |N_u^m|$ and $d_u^n = |N_u^n|$ be the minDNF degree and non-minDNF degree of expression u. Clearly the degree of u is given as $d_u = |N_u| = d_u^m + d_u^n$. We define the weight $w(u, v)$ on each edge (u, v) as follows:

$$w(u, v) = \begin{cases} \frac{(1-\alpha)c}{\max\{d_u^m, d_v^m\}} & \text{if } u \text{ and } v \text{ are minDNFs} \\ \frac{\alpha c}{d_v^n} & \text{if } v \text{ is minDNF but } u \text{ is not} \\ \frac{\alpha c}{d_u^n} & \text{if } u \text{ is minDNF but } v \text{ is not} \\ 1 & \text{if } u \text{ and } v \text{ are not minDNFs} \end{cases} \tag{2}$$

Here $0 < \alpha < 1$ is a weighting term, and $c > 0$ is a scaling constant. We shall see that α controls the degree of non-uniformity in the sampling, and along with c also affects the convergence rate of the sampling method (it has no impact on the correctness). From the definition, one can verify that the edge weights in the graph are symmetric and the transition probability matrix is stochastic. Moreover, the weights favor transitions to minDNF nodes. We prove that the defined random walk converges to a stationary distribution.

THEOREM 2. *An ergodic random walk on the weighted graph where the transition matrix P is defined via Eq.* (1), *using the weight function in Eq.* (2) *is reversible. Further, the random walk converges to a stationary distribution.*

PROOF. Essentially, we can show that P satisfies the detailed balance condition: $\forall s_i, s_j \in \mathcal{S}$, $\pi_i P(i, j) = \pi_j P(j, i)$. Furthermore, the walk is finite, irreducible, and can be made aperiodic with minor tweaking [16]. Details omitted due to lack of space. ☐

DEFINITION 2. *Let π denote the stationary distribution for a Markov chain, where $\pi(u)$ denotes the probability of visiting node u. The* non-uniformity *of a random walk is defined as the ratio of the maximum to the minimum probability of visiting a minDNF node.*

THEOREM 3. *The non-uniformity of minDNF sampling defined by Eq.* (2) *is bounded by the ratio $1/\alpha$.*

PROOF. The stationary distribution for any node v is given as $\pi(v) = \frac{s(v)}{W}$, where $s(v) = \sum_{y \in N_v} w(v, y)$ is the total weight for node v, and $W = \sum_v s(v)$ is the sum of the weights over all nodes in the Markov chain [16]. W is a constant for a given graph, thus $\pi(u) \propto s(u)$. Let u be a minDNF expression, with minDNF degree d_u^m and non-minDNF degree d_u^n. The total weight for u is given as

$$s(u) = \sum_{v \in N_u} w(u, v) = d_u^n \cdot \frac{\alpha c}{d_u^n} + (1-\alpha)c \sum_{y \in N_u^m} \frac{1}{\max(d_u^m, d_y^m)}.$$

Note that $\sum_{y \in N_u^m} \frac{1}{\max(d_u^m, d_y^m)} \leq d_u^m \cdot \frac{1}{d_u^m} = 1$. Equality is achieved only if $d_u^m \geq d_y^m$ for all $y \in N_u^m$, in which case $s(u) = \alpha c + (1-\alpha)c = c$. On the other hand, in the worst case we may assume that $d_y^m \gg d_u^m$ so that the second term vanishes in the limit, in which case we have $s(u) = \alpha c$. Thus the worst-case non-uniformity in sampling is $\frac{c}{\alpha c} = \frac{1}{\alpha}$. ☐

The pseudo-code for the minDNF sampling algorithm is outlined in Figure 2. The method always starts by picking a

```
minDNF Sampling Algorithm
Input: $\mathcal{D}, \sigma_{\min}, \sigma^c_{\min}, k$
Output: $k$ minimal DNF generators
1. $B$= select a frequent item randomly
2. IF is_minimal($B$)
3.     Output $B$, insert $B$ in $\mathcal{B}$
4. IF $|\mathcal{B}| == k$ THEN return //k minDNFs sampled
5. $\mathcal{F}$ = Compute-Local-Neighborhood($B$)
6. $\mathbf{P}$ = Local-Transition-Matrix($B, \mathcal{F}$) //Eq. (1), (2)
7. Select a DNF $B_{next}$ from $\mathcal{F}$ proportional to $\mathbf{P}$
8. Set $B = B_{next}$, and go to Line 2

Compute-Local-Neighborhood($B$)
9. For each Boolean expression $f$ in neighborhood of $B$
10.   IF $sup(f)$ satisfies $\sigma^c_{\min}$ and $\sigma_{\min}$
11.     If Lemma 2 and Lemma 3 are satisfied
12.       Insert $f$ in $\mathcal{F}$
13.       If conditions (a), (b) in Theorem 1 satisfied
14.         Set $f$.min = True;
```

Figure 2: minDNF Sampling Algorithm

random frequent item (or its negation; we omit that here). Given the current node B, we first check if it minimal, and if so add it to the sampled set of patterns $\mathcal{B}$. If k steps have been performed, we stop (line 4). Otherwise, in line 5, determine all the immediate parents and children of the current node B that satisfy support constraints, as given in the function **Compute-Local-Neighborhood** in lines 9-14. To get all possible parents and children of the current node B, the four operations AAC, ATC, DTC, DFC are used in line 9. In Line 10 we test the support and prune out those patterns that do not satisfied the conditions. In line 11, we use Lemmas 2 and Lemma 3 to determine whether f is qualified to be remain in the partial order graph. We further test property (a) and (b) in Theorem 1 for f to determine its minimality. Returning back to line 6, we compute the transition probability $\mathbf{P}$ according to Equation 1, 2. Then we select a DNF expression B_{next} proportional to P to continue the walk in lines 7-8. Note that ideally, we should have a burn-in period for the random walk before patterns are output. We can start counting patterns after a sufficient number of steps (for line 4 check). We omit these details here, for clarity.

3.3 Optimizations

Transition Probability Matrix: Whereas Eq. (2) gives a good guarantee on the sampling quality, it can be expensive to compute, since we have to determine the minDNF and non-minDNF degree for a node, as well as for all of its neighbors N_u. Instead, we propose another weighting scheme that leads to much faster sampling, without sacrificing the sampling quality too much:

$$w(u,v) = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are minDNFs} \\ \frac{c}{d_v} & \text{if } v \text{ is minDNF but } u \text{ is not} \\ \frac{c}{d_u} & \text{if } u \text{ is minDNF but } v \text{ is not} \\ 0.5 & \text{if } u \text{ and } v \text{ are not minDNFs} \end{cases} \tag{3}$$

We note that if the nodes u and v are both either minDNFs or non-minDNFs then we do not need to compute their degrees. Only if one of the nodes is a minDNF, we have to compute its degree (d_u or d_v), but we do not have to determine its minDNF or non-minDNF degrees. The theorem below, shows that sampling quality is still good:

THEOREM 4. *The sampling non-uniformity of weighting scheme in Eq.* (3) *is bounded by* $1 + d^m/c$, *where* d^m *is maximum minDNF degree of a node.*

PROOF. Let u be a minDNF node, with minDNF degree d^m_u and non-minDNF degree d^n_u. The total weight for u is given as $s(u) = \sum_{v \in N_u} w(u,v) = d^n_u \cdot \frac{c}{d_u} + d^m_u \leq c + d^m_u$. The last step holds since $d^n_u \leq d_u$. The least weight at any node occurs when $d^m_u = 0$, and the most weight is when $d^m_u = \max_u \{d^m_u\} = d^m$. Thus, the non-uniformity is given as $\frac{c+d^m}{c} = 1 + d^m/c$. □

In practice, this weighting scheme works well. One reason for this is that the expected minDNF degree of a node is much better than the worst-case d^m given above.

Random Walks with Jumps and Restarts: Even after pruning, the partial order graph of sampling minimal DNF generators is large. The walker may be thus get trapped in local regions of the graph which consists of non-minimal DNFs. If this happens, our algorithm will not output minimal DNFs even after a long run, although the samples are guaranteed to be uniform. To avoid avoid getting stuck in local parts, we use the following two strategies: i) Random Walks with Random Jumps (RWRJ): In case the algorithm outputs no minimal DNF generators even after r consecutive steps, we abort the current path, and randomly jump to any earlier minimal DNF generator in the history as its new start. Any such node is then deleted, so that it will not again be chosen as a jump point. ii) Random Walks with Restart (RWR): At each step in the random walk, we enable a certain probability r that the walker jumps back to the root node (empty itemset). We confirm empirically that RWRJ is the better strategy.

AND- and OR-Clause Cache: To further improve execution time, we pre-compute the frequent AND-clauses and OR-Clauses of length 2, and store them in a hash table, so that they can be used to quickly test for the valid ATC and AAC operations. For example, when we apply ATC operations on a clause Z_i in DNF Z, we first get all frequent candidate items I_{ij} to be added for each literal $z_{ij} \in Z_i$. Then the candidate items to be added by ATC for the clause Z_i are $\{ \bigcap_j I_{ij} | z_{ij} \in Z_i \}$. We can do so quickly by looking up the candidate items in the hash table. This step avoids searching the whole $\mathcal{Z}$ space when we apply ATC operations and thus improves the efficiency.

Fast Minimality Determination: Since we perform random walks on a partial order graph that consists of both minimal and non-minimal DNFs, Lemma 6 and Lemma 7 mentioned below can help to quickly determine the minimality of a DNF when performing random walks without explicitly testing properties (a), (b) in Theorem 1, which can save a lot of computational overhead.

LEMMA 6. *Let* $Z = \bigvee_{i=1}^m Z_i$, *with* $m \geq 2$, *be a general DNF expression that violates property* (a) *in Theorem 1. Let* $T(Z_k) \subseteq \bigcup_{j \neq k} T(Z_j)$. *By adding an item to clause* Z_k *in* Z, *the resulting DNF cannot be minDNF.*

PROOF. The tidset of a clause is anti-monotonic. By adding an item x to clause Z_k, resulting in clause Z'_k, i.e., $Z'_k = Z_k \wedge x$, we still have $T(Z'_k) \subseteq \bigcup_{j \neq k} T(Z_j)$. Hence property (a) is violated. □

However, note that adding an item to other clauses rather than Z_1 in Z can result in a minDNF node.

LEMMA 7. *Let* $Z = \bigvee_{i=1}^m Z_i$ *be a general DNF expression, with* $m \geq 2$, *and let clause* $Z_k \in Z$ *violate property* (b) *in Theorem 1. Adding an item to clause* Z_k *cannot result in a minimal DNF expression.*

PROOF. Since Z_k violates property (b) in Theorem 1, there exists item $x \in Z_k$, such that $T(Z_k) \cup (\bigcup_{j \neq k} T(Z_j)) = T(Z_k^{'}) \cup (\bigcup_{j \neq k} T(Z_j))$, where $Z_k = Z_k^{\prime} \wedge x$. Let $Z_k^{''} = Z_k \wedge y = Z_k^{\prime} \wedge x \wedge y$ and consider the DNF expression $Z^{\prime\prime} = Z_k^{''} \vee (\bigvee_{j \neq k} Z_j)$. Assume that $Z^{\prime\prime}$ is minDNF, which implies that $(Z_k^{'} \wedge x \wedge y)$ is minAND by Lemma 1. This in turn implies that the difference set $D_y = T(Z_k^{'} \wedge y) - T(Z_k^{'} \wedge x \wedge y)$ is non-empty. We consider three cases: (1) If $D_y \subseteq \bigcup_{j \neq k} T(Z_j)$, then $T(Z_k^{'} \wedge x \wedge y) \cup (\bigcup_{j \neq k} T(Z_j)) = T(Z_k^{'} \wedge y) \cup (\bigcup_{j \neq k} T(Z_j))$, which contradicts the assumption that $Z^{\prime\prime}$ is a minDNF. (2) If $D_y \cap \bigcup_{j \neq k} T(Z_j) = \emptyset$, then we have $D_y \subseteq T(Z_k^{'}) \subseteq T(Z_k^{'}) \cup (\bigcup_{j \neq k} T(Z_j)) = T(Z_k) \cup (\bigcup_{j \neq k} T(Z_j))$, which implies that $D_y \subseteq T(Z_k) = T(Z_k^{'} \wedge x)$. However, by definition, $D_y \subseteq T(Z_k^{'} \wedge y)$. Hence $D_y \subseteq T(Z_k^{'} \wedge x) \cap T(Z_k^{'} \wedge y) = T(Z_k^{'} \wedge x \wedge y)$. But thus implies that $D_y = \emptyset$, which contradicts the assumption that $Z_k^{'} \wedge x \wedge y$ is minAND. (3) If $D_y \nsubseteq \bigcup_{j \neq k} T(Z_j)$ and $D \cap \bigcup_{j \neq k} T(Z_j) \neq \emptyset$ then we can divide D_y into two parts $D_y = D_y^{\prime} \cup D_y^{\prime\prime}$, such that $D_y^{\prime} \subseteq \bigcup_{j \neq k} T(Z_j)$, $D_y^{\prime\prime} \cap \bigcup_{j \neq k} T(Z_j) = \emptyset$, and $D_y^{\prime} \neq \emptyset, D_y^{\prime\prime} \neq \emptyset$. Similar to step (2), $D_y^{\prime\prime} \subseteq T(Z_k^{'} \wedge x \wedge y)$, which implies that $D_y = D_y^{\prime} \cup D_y^{\prime\prime} = T(Z_k^{\prime} \wedge y) - T(Z_k^{\prime} \wedge x \wedge y) = D_y^{\prime}$. However, this implies that $D_y^{\prime\prime} = \emptyset$, which is a contradiction. From the three cases above, we conclude that $Z^{\prime\prime}$ is not minDNF. $\square$

Once again, note that adding an item to $Z_j, j \neq k$, may possibly generate a minimal DNF. Consider an example from Table 1 again, with DNF $D|BF$. It violates property (b) in Theorem 1 since $T(D|BF) = T(D|F)$ and $D|F$ is a minimal DNF generator. However, one extension of this DNF is $BF|DE$, which is a minimal DNF generator. Figure 1(b) shows this example. The solid ovals in the figure are minimal DNF generators, and dashed ovals represent non-minimal DNFs.

3.4 Sampling Minimal AND-clauses

LEMMA 8. *The partial order graph on minimal AND-clause is connected.*

PROOF. According to Lemma 1, any subset of a minimal AND-clause generator is also the minimal AND-clause. Thus, in the partial order graph a node is connected to to all its immediate subsets/parents. $\square$

For example, if ABC is a minAND then ABC has three parents, AB, AC and BC, which are all minANDs. Given this connected state-space, a simple symmetric transition probability matrix suffices to sample minAND expressions:

$$P(u,v) = \begin{cases} \frac{1}{max(d_u, d_v)} & \text{if } v \in N_u \\ 1 - \sum_{x \in N_u} P(u,x) & \text{if } u = v \end{cases} \qquad (4)$$

We can use this matrix in the algorithm in Figure 2 to mine all minAND clauses.

4. EXPERIMENTS

We evaluate the benefits of mining minDNF expressions by using them as features for categorical data classification task. We also evaluate the sampling quality. We compare our results with Blosom [22], which is the only current algorithm that can mine minDNF patterns (although it is a complete method). For pure-AND clauses, we also compare with CHARM-L [21], a state-of-the-art method for mining minimal AND-clauses (i.e., minimal generators.) All experiments are performed on a quad-core Intel i7 3.5GHz CPU, with 16GB memory, and 2TB disk, running Linux (Ubuntu 11.10). The minDNF sampling code was implemented in C++. All datasets and source code are available at: `http://www.cs.rpi.edu/~zaki/www-new/pmwiki.php/Software`.

4.1 Classification Performance

We experimented with a wide range of datasets from the UCI repository [13] as shown in Table 2. First, each of the 34 datasets was converted into a categorical one using entropy-based discretization [12], as implemented in the Orange data mining suite [10]. The total number of transactions and items in each dataset, and the number of classes (ranging from 2 to 24) are shown in the table. Next we ran a linear SVM, on the original (non-discretized) dataset, as well as on the discretized dataset, using 5-fold cross-validation. For a given run of minDNF sampling, we converted each of the mined minDNF patterns into a binary attribute, that takes on the value 1 if a transaction satisfies the minDNF formula, and 0 otherwise. This binary-valued dataset is then classified using linear SVM with 5-fold cross-validation, again using the Orange library (which in turn uses LIBSVM [7]). Since the sampling is randomized, we repeat the sampling 10 times, and report the averages.

For minDNF and minAND sampling the default parameters are as follows: We use the random walks with random jump approach, with $j = 3$. We use $\alpha = 0.9$ and set c to the average transaction length. The minimum support for the DNF and clause were both set to 1, i.e., $\sigma_{\min} = \sigma_{\min}^c = 1$. Finally, we sampled $k = 100$ minDNF patterns. Thus, for minDNF and minAND the number of "items" or features is at most k for all datasets (it can be less after removing duplicates). Note also that we do not perform any feature selection (it may be possible to further improve the performance if this is done).

Table 2 shows the 5-fold cross-validation classification accuracy and standard-error for each algorithm, over each of the datasets, averaged over 10 runs. The best results are shown in bold. In the table, results for minDNF sampling using the weight matrix in Eq. (2) are denoted as minDNF, whereas results using the faster weight computation in Eq. (3) are denoted as minDNF*. SVM-orig and SVMd denote the performance of SVM on the original and discretized dataset, respectively. Finally, non-default parameters are indicated in the table (third row), when we compare minAND with $k = 500$, and minDNF/minAND with $\sigma_{\min} = 5\%$.

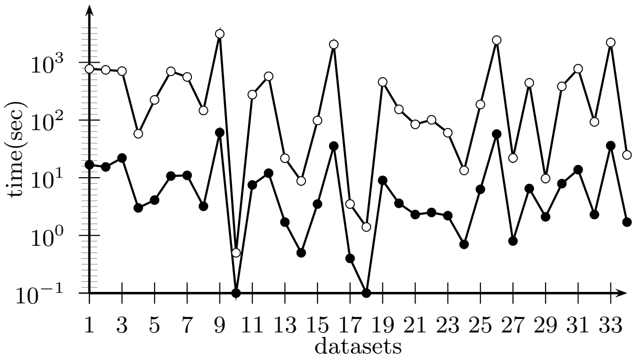


Figure 3: Time (in sec) for minDNF (white dots) and minDNF* (black dots)

minDNF Time: The total time for minDNF and minDNF* is shown in Figure 3. The datasets are numbered in the order they appear in Table 2. The running time is affected by the number of items, since the more the items, the more

	description			SVM		minDNF Sampling			minAND Sampling		
dataset	cls	trans	items	SVM-orig	SVMd	minDNF	minDNF $\sigma_{\min} = 5\%$	minDNF*	minAND	minAND $k = 500$	minAND $\sigma_{\min} = 5\%$
adultsample	3	977	113	59.5±11.7	36.6±0.7	79.8±0.9	**81.8±1.3**	57.1±1.6	48.6±1.4	45.1±0.5	75.2±0.9
anneal	5	898	73	33.6±5.0	43.7±3.5	**97.6±0.4**	97.2±0.4	92.0±0.6	80.0±0.6	91.7±0.7	82.5±0.6
audiology	24	226	154	33.2±2.8	33.2±2.8	**79.7±3.0**	75.1±2.3	48.3±3.3	43.8±2.0	33.4±1.4	37.5±1.5
balancescale	3	625	20	**87.8±1.0**	**87.8±1.0**	71.7±1.4	74.1±1.3	72.0±1.4	46.8±3.9	74.0±1.2	15.7±7.7
breastwisc1	2	683	30	86.7±0.8	60.8±1.6	95.4±0.5	**95.5±0.6**	95.0±0.6	75.6±1.1	93.2±0.8	77.2±0.5
breastwisc2	2	699	90	88.0±1.1	88.0±1.1	**94.4±0.6**	91.9±0.9	89.4±1.0	61.5±1.0	72.2±1.3	86.6±0.9
breastwisc3	2	683	89	84.2±1.5	84.0±1.5	**93.8±0.6**	**93.8±0.9**	87.1±1.1	45.8±1.0	93.6±0.8	52.1±0.3
breastcancer	2	286	41	71.3±1.3	**72.0±1.0**	67.8±3.0	68.5±2.3	64.7±2.1	54.8±2.3	60.3±2.1	68.5±2.4
brownselected	3	186	182	89.8±1.5	39.2±2.1	**99.5±0.4**	99.0±0.6	88.3±2.1	50.4±1.8	64.4±2.5	67.9±2.9
bupa	2	345	7	50.1±3.5	54.8±3.2	**63.2±2.7**	**63.2±2.7**	**63.2±2.7**	54.8±3.2	54.8±3.2	54.8±3.2
car	4	1728	21	64.0±0.8	64.0±0.8	**81.0±0.4**	77.2±0.6	76.7±0.8	71.2±0.4	79.1±0.8	87.2±0.7
crx	2	690	53	74.8±2.5	**85.4±1.1**	83.6±1.5	83.7±1.5	74.1±1.5	62.1±1.5	76.4±1.3	71.9±1.2
glass	6	214	22	48.1±4.7	23.3±3.9	75.7±1.0	**77.2±1.1**	75.2±1.4	62.5±1.6	50.5±0.7	70.0±1.6
hayesroth	3	132	15	46.1±4.6	46.1±4.6	73.5±3.5	**76.8±2.8**	72.3±4.0	68.4±3.2	78.7±3.3	58.9±4.0
heartdisease	2	303	29	70.3±5.5	65.7±2.8	78.5±2.2	**78.9±2.2**	76.0±2.7	67.2±3.5	62.0±1.7	71.6±3.1
ionosphere	2	351	142	83.8±0.7	84.6±1.6	**89.5±1.5**	88.8±1.3	86.3±1.6	49.3±0.8	80.8±1.2	46.5±0.9
iris	3	150	12	93.3±2.4	68.7±2.0	94.9±1.3	**95.5±1.4**	94.9±1.8	95.2±1.4	95.4±1.3	94.7±1.8
lenses	3	24	9	83.0±7.7	83.0±7.7	80.3±6.8	72.5±7.9	61.6±11.1	78.2±6.2	**87.0±8.3**	86.2±8.2
lungcancer	3	32	157	**52.9±10.3**	**52.9±10.3**	52.1±7.5	44.6±7.5	39.3±8.2	33.2±6.1	31.7±4.3	34.4±6.8
lymphography	4	148	50	**82.4±4.1**	81.1±3.0	80.0±3.6	79.1±2.4	70.9±3.3	63.6±3.1	75.9±2.9	68.0±2.6
monks1	2	556	17	67.6±2.6	67.6±2.6	84.3±1.4	83.7±1.4	77.5±1.6	65.4±1.8	**92.7±0.9**	66.6±1.6
monks2	2	601	17	64.2±1.1	64.2±1.1	66.3±1.1	71.9±1.7	62.6±1.4	53.8±1.6	**74.7±1.6**	68.2±1.7
monks3	2	554	17	74.4±0.6	74.4±0.6	93.4±1.1	**96.4±0.9**	89.2±1.2	69.7±1.4	93.6±1.1	84.5±1.0
postoperative	3	90	20	66.7±1.8	**67.8±2.1**	57.4±4.3	55.1±3.8	56.3±3.1	58.4±4.7	58.2±4.2	61.6±3.4
primarytumor	21	339	37	28.6±0.6	28.6±0.6	**40.2±2.0**	38.6±2.1	32.6±2.1	30.0±1.7	36.2±2.5	33.9±2.3
promoters	2	106	228	72.5±6.5	72.5±6.5	**74.2±3.0**	65.2±3.8	62.5±3.9	57.6±3.2	65.5±4.3	66.1±4.6
shuttle	2	253	16	96.5±0.7	96.8±1.0	**97.1±0.9**	94.4±1.2	90.5±1.8	81.2±2.3	96.4±1.1	95.0±1.5
tictactoe	2	958	27	67.8±1.0	67.7±0.9	**78.7±1.2**	76.9±1.3	70.2±1.3	47.4±1.0	68.8±1.6	76.1±1.4
titanic	2	2201	8	76.8±1.1	76.8±1.1	79.0±0.9	77.9±1.0	79.0±0.9	78.9±1.0	**79.1±0.9**	79.0±0.9
voting	2	435	32	91.0±0.7	91.3±0.8	**94.6±1.1**	93.9±0.8	91.8±1.0	71.9±1.1	83.1±1.4	78.3±1.1
wdbc	2	569	64	88.9±4.3	75.9±3.8	95.3±0.8	**95.5±0.8**	92.9±1.0	69.6±1.0	84.8±0.9	85.2±0.8
wine	3	178	37	85.9±4.8	96.7±2.0	**97.9±0.5**	97.6±1.1	94.4±1.3	76.3±2.6	95.9±1.3	74.7±1.9
yeastRPR	3	186	182	89.8±1.5	39.2±2.1	99.3±0.7	**99.7±0.2**	87.6±2.0	59.7±2.2	73.7±1.8	43.6±2.0
zoo	7	101	36	95.1±1.5	95.1±1.5	**96.3±1.6**	96.2±1.5	89.5±2.1	82.5±3.5	95.5±1.9	83.2±3.7

Table 2: Classification Performance: Accuracy ± Standard Error: cls denotes #classes

the number of neighbors, which affects the weight/transition probability computation time. However, note that these results are with support one, and substantial speedup is possible for higher support values. We can observe that minDNF* is typically over an order of magnitude faster than minDNF, though this comes at some penalty in classification performance, as we describe next.

minDNF versus SVMd: Our minDNF sampling algorithm yields a near-uniform sample and we can see from the classification accuracies in columns 7 & 8 that the sampled minDNF patterns make excellent features. In 24 out of the 34 datasets, they yield the best accuracy among all methods, including SVM-orig (on the original) and SVMd (on the discretized datasets). On three datasets, the differences between minDNF and best method is not significant. On two datasets SVMs substantially outperform minDNF (namely, balancescale and postoperative, where the difference is more than 10%).

minDNF versus minAND: Comparing the Boolean expression features comprising minDNF patterns versus minAND patterns (both with $k = 100$), we find that minDNF substantially outperforms minAND (see columns 7 and 10). Although initially unexpected, it is perhaps not that surprising, given the fact that minDNFs can be considered as disjunctive rules, and are much more informative than simple conjunctive rules. Over all the 34 datasets, minDNF sampling yields on average 2.69 clauses per DNF expression, with a standard deviation of 1. For fairness, we also compared minDNF (with $k = 100$) to minAND with $k = 500$ features (see column 11). The larger number of features improves minAND in most cases. However, minDNF (with $k = 100$) is still substantially better than minAND ($k = 500$); only on three datasets (lenses, monks1, and monks2) does minAND outperform minDNF. These results indicate that overall minDNF patterns are more effective than minAND patterns.

minDNF* Sampling: Since minDNF sampling using Eq. (2) does take more time, we also compared with minDNF* that uses the faster weight computation in Eq. (3). We can see that minDNF* sampling suffers in performance compared to minDNF. However, minDNF* still outperforms SVMd on 22, SVM-orig on 21, minAND patterns (with $k = 100$) on 31, and minAND (with $k = 500$) on 18 out of the 34 datasets.

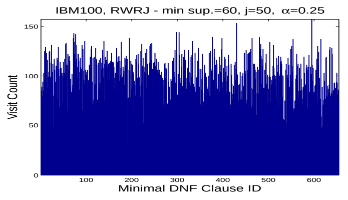
Effect of Support: To see the effect of minimum support on classification accuracy, we ran minDNF and minAND sampling with the minimum support set to 5% of the number of transactions (see columns 8 & 12). Overall, we find that adding the frequency constraint is not that beneficial to for minDNF sampling, since mining frequent minDNFs improves the accuracy only slightly in 13 datasets, when compared to the minDNFs with support one. On the other hand, frequent minANDs are better than support-one minANDs on 26 datasets. Mining frequent expressions obviously lowers running times.

Negated Items: We also experimented with minDNF expressions containing negated items. However, in this case the accuracy was slightly better for the negated items in only 5 out of the 34 datasets, which unfortunately came at the expense of a significant increase in the runtime (sometimes by orders of magnitude). We conclude that negated items do not confer significant advantages in terms of classification (at least for the UCI datasets tested).

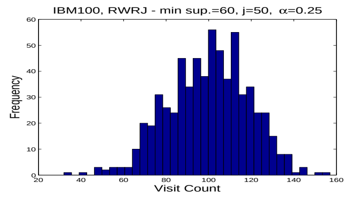
Our results above clearly demonstrate the value of mining/sampling minDNF patterns, especially in support-less mode (i.e., $\sigma_{\min} = 1$).

4.2 Sampling Evaluation

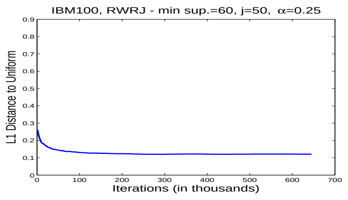
Having shown the effectiveness of minDNF sampling for classification, we now study the sampling quality provided by minDNF and the sensitivity to various parameters. We use both small (first three) and large (last three) datasets shown in Table 3 for these experiments. The last four are taken from the FIMI repository [14]. The Gene dataset is from [22], and IBM100 is a synthetic dataset generated using the IBM itemset generator [1].



(a) Visit Counts

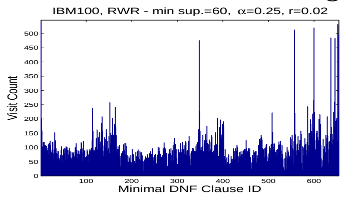


(b) Count Histogram

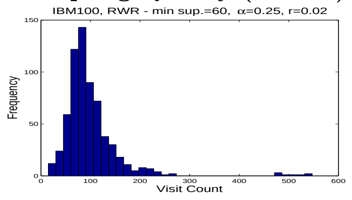


(c) Variation Distance

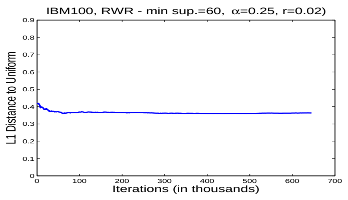
Figure 4: Sampling Quality (RWRJ): IBM100



(a) Visit Counts



(b) Count Histogram



(c) Variation Distance

Figure 5: Sampling Quality (RWR): IBM100

Dataset	trans	items	avg. trans len
IBM100	100	20	9.3
Gene	74	824	86.1
Chess	3196	75	37
Connect	67557	129	43
Retail	88162	16470	10.3
Kosarak	990002	41270	8.1

Table 3: Datasets

Random Walk Type: We first show the effect of the type of random walk, i.e., random walk with random jumps (RWRJ, $j = 50$) versus random walks with restart (RWR, $r = 0.02$), as shown in Figures 4 and 5. With $\sigma_{\min} = 60$, on the IBM100 dataset, there are 654 distinct minDNF patterns (found using Blosom). We ran minDNF sampling for $k = 654 \times 100$ iterations. We use $\alpha = 0.25$ and c is set to the avg. transaction length.

If the dataset has f minDNF patterns and we perform the uniform sampling for $k = f \times t$ steps, the number of times m a specific pattern will be sampled is described by the binomial distribution, $\mathcal{B}(m|k,p)$, where $p = \frac{1}{f}$. The expected number of times a minDNF pattern is visited is given as $kp = f \cdot t \cdot \frac{1}{f} = t$, and the standard deviation is $\sqrt{kp(1-p)} = \sqrt{\frac{t(f-1)}{f}}$. For the IBM100 dataset, we have $f = 654$ and $t = 100$, and thus in the ideal case we expect to see each pattern 100 times, with standard deviation of $\sqrt{100 \cdot 653/654} = 9.99$. Figures 4 and 5 plot the number of times each pattern is visited (a), and the count histogram (b). The sampling statistics, namely the maximum, minimum, median, and standard deviation of visits counts for RWRJ and RWR are shown in Table 4. It is very clear that RWRJ is much superior to the RWR strategy; its median is closer to the ideal case, and the standard deviation is smaller. Whereas the RWRJ strategy jumps to a node in its history, RWR always restarts from the empty pattern. As such RWR is biased towards sampling patterns close to the origin, and this is reflected in the sampling quality.

	Maximum	Minimum	Median	Std
RWRJ	157	32	101	19.0
RWR	547	15	89	60.5

Table 4: Sampling Statistics: IBM100

Convergence Rate: One important issue in using MCMC sampling is to determine when the initial distribution converges to the stationary distribution and how fast the convergence rate is. It is well known that the mixing time is closely related to the spectral gap, $\gamma = |\lambda_1 - \lambda_2| = |1 - \lambda_2|$, which is defined as the absolute difference between the largest $\lambda_1 = 1$ and the second largest eigenvalue λ_2 of the transition matrix P [9]. The larger the spectral gap, the faster the walk converges. Unfortunately, we cannot compute the entire transition matrix P; the whole point of sampling is to avoid enumerating all the minDNF patterns. An alternative strategy to measure the convergence rate is to compute the total variation distance, defined as $vd(P^t(s,.),\pi) = 0.5\sum_{q\in\mathcal{S}}|P^t(s,q) - \pi(q)|$, where s is the initial state, P^t is the transition matrix at time t, and π is the desired stationary distribution.

Figures 4(c) and 5(c) plot the variation distance for the RWRJ and RWR sampling methods. Here we compute the variation distance empirically. To be more specific, we estimate $P^t(s,q)$ at time t via the count histogram, converted into a probability distribution of visitations, to each pattern q from the initial empty pattern $s = \emptyset$. We compute the variation distance after every 1000 steps. We can see that the distance converges to slightly around 0.12 for RWRJ and to 0.36 for RWR, indicating that RWRJ is the better strategy. We also ran experiments on the Gene and Chess datasets, and obtained similar results (figures omitted due to space constraints).

α, c, j	npats	mean	std	max	min	med	time
$j = 3$	636.2	15.7	10.0	60.2	1	14	98.8s
$j = 5$	635.2	15.7	8.9	48.0	1	15.8	100.2s
$j = 10$	636	15.7	8.1	45.2	1	16.4	100.2s
$j = 50$	629	15.9	7.6	38.2	1	16	100.2s
$j = 100$	637.4	15.7	7.6	39.6	1	16.2	101.6s
$\alpha = 0.1$	654	15.3	5.4	34	1.6	15	74.8s
$\alpha = 0.25$	654	15.3	5.3	34.2	2.6	15	80s
$\alpha = 0.5$	654	15.3	5.6	35.4	2.2	15	89.4s
$\alpha = 0.75$	651.8	15.3	7.0	39.4	1	16	96.8s
$\alpha = 0.9$	634.8	15.8	7.6	37.8	1	16.4	101.7s
$c = 5$	633.2	15.8	7.3	35.8	1	16.4	125.6s
$c = avg(9.3)$	632.8	15.8	7.5	45.4	1	16	100.0s
$c = max(17)$	639	15.7	7.9	47.2	1	16.2	88.8s
$c = 50$	633.2	15.8	8.8	50.6	1	15.8	79.1s

Table 5: Effect of Parameters: IBM100

Effect of α, c, j: Table 5 shows the effect of these three parameters on the sampling quality on IBM100 with $\sigma_{\min} = 60$. We set $k = 10000$ iterations. We run each experiment 5 times, and report the average number of distinct minDNFs sampled (npats), mean, standard deviation (std), maximum, minimum, and median (med) of the visit counts, and the average total time. Ideal sampling should yield a mean visit count of $k/f = 15.3$, and a standard deviation of $\sqrt{k/f(1-1/f)} = 3.9$, since IBM100 has $f = 654$ minDNF patterns for $\sigma_{\min} = 60$. First, we look at the effect of j, fixing $c = avg$ and $\alpha = 0.9$. Larger j results in a smaller standard deviation, and ideally j should not be constrained. However, for many of the classification datasets the random walk could get trapped in a local region, and therefore, we set $j = 3$ in our earlier experiments. Next, we look at the effect of α, setting $j = 50$ and $c = avg$. We find that larger α takes more time, with a slight increase in std, most likely due to the constraint on j. Lastly, we fix $j = 50$, $\alpha = 0.9$ and vary c. Larger c values take lesser time, but also result in higher deviation. The average c value (9.3 for IBM100) offers an acceptable choice.

Dataset	1%	5%	10%	20%
IBM100	1m50s	45.5s	40.2s	20.9s
*	17.5s	1.6s	14.5s	9.9s
Gene	3h45m12s	46m13s	31m1s	3m45s
*	19m11s	6m52s	5m23s	3m36s
Chess	11h26m11s	6h41m34s	5h23m29s	4h28m33s
*	1h6m8s	59m22s	42m15s	28m33s
Connect	15h52m47s	8h23m18s	7h59m22s	6h31m39s
*	4h44m2s	2h19m22s	1h58m53s	1h34m48s
Retail	50m4s	1m6s	35.0s	3.1s
*	59m25s	21.5s	12.8s	4.7s
Kosarak	27h58m43s	2h24m39s	9m55s	3m41s
*	8h31m4s	20m3s	2m14s	1m40s

Table 6: Running Time: minDNF and minDNF*

Scalability: Table 6 shows the time to sample the small (with $k = 1000$) and large datasets (with $k = 100$) for various support thresholds using minDNF and minDNF*. Blosom was unfortunately not able to mine the complete set of patterns for any of these datasets for the support levels shown even after 24hours for the smaller datasets, and 48hrs for the large ones. We note that whereas minDNF provides better theoretical guarantee, minDNF* is significantly faster (by as much as an order of magnitude). We also compared minAND sampling time with Blosom-MA and CHARM-L, both of which can mine minimal AND-clauses. For example, for the Gene dataset with 10% support, minAND took 0.7s to sample 1000 patterns, whereas CHARM-L took 40m54s and Blosom-MA took 2h58m56s. For lower support values, neither of these methods could finish within 24hours, whereas for 1% support minAND finished in 1.3s. These results confirm that complete mining is practically infeasible, whereas sampling provides a viable alternative.

5. CONCLUSIONS

In this paper we presented the first approach to mine the simplest Boolean patterns, namely the minimal DNF expressions. We propose a novel weighting scheme to compute the transition probability matrix for the Markov chain Monte Carlo sampling algorithm, which bounds the amount of non-uniformity in the sampling. Since the method can be slow in practice, we also suggest a faster alternative, that yields effective sampling quality as well. We perform an extensive set of experiments to test various design parameters, and justify our choices. Finally, somewhat surprisingly, we found that the minimal DNF patterns make very effective features for classification. Via an extensive set of experiments on UCI datasets, we show that our method outperforms simple AND-clause based features, as well as the SVM method, typically by a wide margin, though it does suffer in the runtime comparison. However, the faster weight computation approach yields significantly faster running times, with some loss in the classification accuracy. The minDNF features still remain the effective across the different classifiers. Perhaps the most interesting aspect of the classification study is that we use support-less patterns (with minimum support one), and do not perform any feature selection. Our future work will target more effective feature selection by considering other interestingness criteria for the patterns while sampling, such as their discrimination power. Efficiency still remains an issue, which may be tackled by implementing the approach on multi-core processors, as well and utilizing graphics computing units (GPUs), since the MCMC methods are inherently parallel.

Acknowledgments

This work was supported in part by NSF grant EMT-0829835.

6. REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. Verkamo, et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12:307–328, 1996.

[2] M. Al Hasan and M. Zaki. Output space sampling for graph patterns. *Proceedings of the VLDB Endowment*, 2(1):730–741, 2009.

[3] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2), Dec. 2000.

[4] M. Boley, T. Gärtner, H. Grosskreutz, and I. Fraunhofer. Formal concept sampling for counting and threshold-free local pattern mining. In *SIAM Data Mining Conf.*, 2010.

[5] M. Boley and H. Grosskreutz. Approximating the number of frequent sets in dense data. *Knowledge and Information Systems*, 21(1):65–89, 2009.

[6] M. Boley, C. Lucchese, D. Paurat, and T. Gärtner. Direct local pattern sampling by efficient two-step random procedures. In *ACM SIGKDD Conference*, 2011.

[7] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[8] V. Chaoji, M. A. Hasan, S. Salem, J. Besson, and M. J. Zaki. ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns. *Statistical Analysis and Data Mining*, 1(2):67–84, June 2008.

[9] M. Cowles and B. Carlin. Markov chain monte carlo convergence diagnostics: a comparative review. *J. American Statistical Association*, 91(434):883–904, 1996.

[10] T. Curk, J. Demsar, Q. Xu, G. Leban, U. Petrovic, I. Bratko, G. Shaulsky, and B. Zupan. Microarray data mining with visual programming. *Bioinformatics*, 21:396–398, Feb. 2005.

[11] G. Dong, C. Jiang, J. Pei, J. Li, and L. Wong. Mining succinct systems of minimal generators of formal concepts. In *Int'l Conf. Database Systems for Advanced Applications*, 2005.

[12] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Int'l Joint Conf. on AI*, 1993.

[13] A. Frank and A. Asuncion. UCI machine learning repository (http://archive.ics.uci.edu/ml), 2010.

[14] B. Goethals and M. J. Zaki. Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explorations*, 6(1):109–117, June 2004.

[15] D. Gunopulos, H. Mannila, and S. Saluja. Discovering all Most Specific Sentences by Randomized Algorithm. In *6th Int'l Conf. on Database Theory*, 1997.

[16] M. A. Hasan and M. J. Zaki. Musk: Uniform sampling of k maximal patterns. In *9th SIAM Data Mining Conf.*, Apr. 2009.

[17] E. Loekito and J. Bailey. Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. In *ACM SIGKDD Conf.* ACM, 2006.

[18] A. A. Nanavati, K. P. Chitrapura, S. Joshi, and R. Krishnapuram. Mining generalised disjunctive association rules. In *ACM CIKM Conference*, 2001.

[19] Y. Shima, S. Mitsuishi, K. Hirata, and M. Harao. Extracting minimal and closed monotone dnf formulas. In *Int'l Conf. on Discovery Science*, 2004.

[20] M. Zaki and N. Ramakrishnan. Reasoning about sets using redescription mining. In *ACM SIGKDD Conference*, 2005.

[21] M. J. Zaki and C.-J. Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):462–478, Apr. 2005.

[22] L. Zhao, M. J. Zaki, and N. Ramakrishnan. Blosom: A framework for mining arbitrary boolean expressions. In *12th ACM SIGKDD Conference*, Aug. 2006.