Efficiently Depth-First Minimal Pattern Mining

Arnaud Soulet[1] and François Rioult[2]

[1] Université François Rabelais Tours, LI
3 Place Jean Jaurès,
F-41029 Blois, France
arnaud.soulet@univ-tours.fr
[2] Université de Caen, GREYC
Campus Côte de Nacre,
F-14032 Caen Cédex, France
francois.rioult@unicaen.fr

Abstract. Condensed representations have been studied extensively for 15 years. In particular, the maximal patterns of the equivalence classes have received much attention with very general proposals. In contrast, the minimal patterns remained in the shadows in particular because of their difficult extraction. In this paper, we present a generic framework for minimal patterns mining by introducing the concept of minimizable set system. This framework addresses various languages such as itemsets or strings, and at the same time, different metrics such as frequency. For instance, the free and the essential patterns are naturally handled by our approach, just as the minimal strings. Then, for any minimizable set system, we introduce a fast minimality check that is easy to incorporate in a depth-first search algorithm for mining the minimal patterns. We demonstrate that it is polynomial-delay and polynomial-space. Experiments on traditional benchmarks complete our study.

Keywords: Pattern mining, condensed representation, minimal pattern.

1 Introduction

Minimality is an essential concept of pattern mining. Given a function f and a language $\mathcal{L}$, a minimal pattern X is one of the smallest pattern with respect to the set inclusion in $\mathcal{L}$ satisfying the property $f(X)$. Interestingly, the whole set of minimal patterns forms a condensed representation of $\mathcal{L}$ adequate to f: it is possible to retrieve $f(Y)$ for any pattern of Y in $\mathcal{L}$. Typically, the set of free itemsets [1] (also called generators or key itemsets [2]) is a condensed representation of all itemsets (here, f and $\mathcal{L}$ are respectively the frequency and the itemset language). Of course, it is often more efficient to extract minimal patterns rather than all patterns because they are less numerous. In addition, minimal patterns have a lot of useful applications including higher KDD tasks: producing the most relevant association rules [3], building classifiers [4] or generating minimal traversals [5]. Minimality has been studied in the case of different functions (like frequency [6] and condensable functions [7]) and different languages (e.g., itemsets [1] and sequences [8]). Although the minimality has obvious advantages [9],

V.S. Tseng et al. (Eds.): PAKDD 2014, Part I, LNAI 8443, pp. 28–39, 2014.

very few studies are related to the minimality while maximality (i.e., closed patterns) has been widely studied. In particular, to the best of our knowledge, there is no framework as general as those proposed for maximality [10].

We think that a current major drawback of minimal patterns lies in their inefficient extraction. This low efficiency comes mainly from the fact that most existing algorithms use a levelwise approach [1,7,11] (i.e., breadth-first search/generate and test method). As they store all candidates in memory during the generation phase, the extraction may fail due to memory lack. To tackle this memory pitfall, it seems preferable to adopt a depth-first traversal which often consumes less memory and is still very fast. However, check whether the minimality is satisfied or not is very difficult in a depth-first traversal. In the case of frequency with itemsets, the best way for evaluating the minimality for a pattern (saying abc) is to compare its frequency with that of all its direct subsets (here, ab, ac and bc). But, when the pattern abc is achieved by a depth-first traversal, only frequencies of a and ab have previously been calculated. As the frequency of ac and bc are unknown, it is impossible to check whether the frequency of abc is strictly less than that of ac and bc. To cope with this problem, [11,12] have adopted a different traversal with reordered items. For instance, when the itemset abc is reached by this new traversal, c, b, bc, a, ac and bc were previously scanned and their frequency are known for checking whether abc is minimal. Unfortunately, such a method requires to store all the patterns in memory (here, c, b, bc and so on) using a trie [11] or an hash table [12]. For this reason, existing DFS proposals [11,12] do not solve the low memory consumption issue as expected.

Contributions. The main goal of this paper is to present a generic and efficient framework for minimal pattern mining by providing a depth-first search algorithm. We introduce the notion of *minimizable set system* which is at the core of the definition of this framework. This latter covers a broad spectrum of minimal patterns including all the languages and measures investigated in [7,10]. Fast minimality checking in a depth-first traversal is achieved thanks to the notion of *critical objects* which depends on the minimizable set system. Based on this new technique, we propose the DefMe algorithm. It mines the minimal patterns for any minimizable set system using a depth-first search algorithm. To the best of our knowledge, this is the first algorithm that enumerates minimal patterns in polynomial delay and in linear space with respect to the dataset.

The outline of this paper is as follows. In Section 2, we propose our generic framework for minimal pattern mining based on set systems. We introduce our fast minimality checking method in Section 3 and we indicate how to use it by sketching the DefMe algorithm. Section 4 provides experimental results. In Section 5, we discuss some related work in light of our framework.

2 Minimizable Set System Framework

2.1 Basic Definitions

A *set system* $(\mathcal{F}, E)$ is a collection $\mathcal{F}$ of subsets of a *ground set* E (i.e. $\mathcal{F}$ is a subset of the power set of E). A member of $\mathcal{F}$ is called a *feasible set*. A *strongly*

accessible set system $(\mathcal{F}, E)$ is a set system where for every feasible sets X, Y satisfying $X \subset Y$, there is an element $e \in Y \setminus X$ such that $Xe \in \mathcal{F}$[1]. Obviously, itemsets fits this framework with the set system $(2^{\mathcal{I}}, \mathcal{I})$ where $\mathcal{I}$ is the set of items. $(2^{\mathcal{I}}, \mathcal{I})$ is even strongly accessible. But the notion of set system allows considering more sophisticated languages. For instance, it is easy to build a family set $\mathcal{F}_S$ denoting the collection of substrings of $S = abracadabra$ by encoding each substring $s_{k+1}s_{k+2}\dots s_{k+n}$ by a set $\{(s_{k+1}, 1), (s_{k+2}, 2), \dots, (s_{k+n}, n)\}$. The set sytem $(\mathcal{F}_S, E_S = \bigcup \mathcal{F}_S)$ is also strongly accessible. The set system formalism has already been used to describe pattern mining problems (see for instance [10]).

Intuitively, a pattern always describes a set of objects. This set of objects is obtained from the pattern by means of a *cover operator* formalized as follows:

Definition 1 (Cover operator). *Given a set of objects $\mathcal{O}$, a cover operator $cov : 2^E \to 2^{\mathcal{O}}$ is a function satisfying $cov(X \cup Y) = cov(X) \cap cov(Y)$ for every $X \in 2^E$ and $Y \in 2^E$.*

This definition indicates that the coverage of the union of two patterns is exactly the intersection of their two covers. For itemsets, a natural cover operator is the extensive function of an itemset X that returns the set of tuple identifiers supported by X: $cov_{\mathcal{I}}(X) = \{o \in \mathcal{O} \mid X \subseteq o\}$. But, in general, the cover is not the final aim: the cardinality of $cov_{\mathcal{I}}(X)$ corresponds to the frequency of X. In the context of strings, the index list of a string X also define a cover operator: $cov_S(X) = \{i \mid \forall (s_j, j) \in X,\ (s_j, j+i) \in S\}$. Continuing our example with the string $S = abracadabra$, it is not difficult to compute the index lists $cov_S(\{(a,1)\}) = \{0,3,5,7,10\}$ and $cov_S(\{(b,2)\}) = \{0,7\}$ and then, to verify $cov_S(\{(a,1),(b,2)\}) = cov_S(\{(a,1)\}) \cap cov_S(\{(b,2)\}) = \{0,7\}$.

For some languages, the same pattern is described by several distinct sets and then it is necessary to have a canonical form. For example, consider the set $\{(a,1),(b,2),(r,3)\}$ corresponding to the string abr. Its suffix $\{(b,2),(r,3)\}$ encodes the same string br as $\{(b,1),(r,2)\}$. The latter is the canonical form of the string br. To retrieve the canonical form of a pattern, we introduce the notion of canonical operator:

Definition 2 (Canonical operator). *Given two set systems $(\mathcal{F}, E)$ and $(\mathcal{G}, E)$, a canonical operator $\phi : \mathcal{F} \cup \mathcal{G} \to \mathcal{F}$ is a function satisfying (i) $X \subset Y \Rightarrow \phi(X) \subset \phi(Y)$ and (ii) $X \in \mathcal{F} \Rightarrow \phi(X) = X$ for all sets $X, Y \in \mathcal{G}$.*

In this definition, the property (i) ensures us that the canonical forms of two comparable sets with respect to the inclusion remain comparable. The property (ii) means that the set system $(\mathcal{F}, E)$ includes all canonical forms. Continuing our example about strings, it is not difficult to see that $\phi_S : \{(s_k, k), (s_{k+1}, k+1), \dots, (s_{k+n}, n)\} \mapsto \{(s_k, 1), (s_{k+1}, 2), \dots, (s_{k+n}, n-k+1)\}$ satisfies the two desired properties (i) and (ii). For instance, $\phi_S(\{(b,2),(r,3)\})$ returns the canonical form of the string $\{(b,2),(r,3)\}$ which is $\{(b,1),(r,2)\}$.

[1] We use the notation Xe instead of $X \cup \{e\}$.

2.2 Minimizable Set System

Rather than considering an entire set system, it is wise to select a smaller part that provides the same information (w.r.t. a cover operator). For this, it is necessary that this set system plus the cover operator form a *minimizable* set system:

Definition 3 (Minimizable set system). *A minimizable set system is a tuple* $\langle(\mathcal{F}, E), \mathcal{G}, cov, \phi\rangle$ *where:*

- $(\mathcal{F}, E)$ *is a finite, strongly accessible set system. A feasible set in* $\mathcal{F}$ *is called a pattern.*
- $(\mathcal{G}, E)$ *is a finite, strongly accessible set system satisfying for every feasible set* $X, Y \in \mathcal{F}$ *such that* $X \subseteq Y$ *and element* $e \in E$, $X \backslash \{e\} \in \mathcal{G} \Rightarrow Y \backslash \{e\} \in \mathcal{G}$. *A feasible set in* $\mathcal{G}$ *is called a generalization.*
- $cov : 2^E \rightarrow 2^{\mathcal{O}}$ *is a cover operator.*
- $\phi : \mathcal{F} \cup \mathcal{G} \rightarrow \mathcal{F}$ *is a canonical operator such that for every feasible set* $X \in \mathcal{G}$, *it implies* $cov(\phi(X)) = cov(X)$.

Let us now illustrate the role of $\mathcal{G}$ compared to $\mathcal{F}$ in the case of strings. In fact, $\mathcal{G}_S$ gathers all the suffixes of any pattern of $\mathcal{F}_S$. Typically, $\{(b,2),(r,3)\} \in \mathcal{G}_S$ is a generalization of $\{(a,1),(b,2),(r,3)\} \in \mathcal{F}_S$. As said above, $\{(b,2),(r,3)\}$ has an equivalent form in $\mathcal{F}_S$: $\phi_S(\{(b,2),(r,3)\}) = \{(b,1),(r,2)\}$. By convention, we extend the definition of cov_S to $\mathcal{G}_S$ by considering that $cov_S(\phi_S(X)) = cov_S(X)$. In addition, it is not difficult to see that $\mathcal{G}_S$ satisfies the desired property with respect to $\mathcal{F}_S$: for every feasible set $X, Y \in \mathcal{F}_S$ such that $X \subseteq Y$ and element $e \in E_S$, $X \setminus \{e\} \in \mathcal{G}_S \Rightarrow Y \setminus \{e\} \in \mathcal{G}_S$. Indeed, if $X \setminus \{e\}$ is a suffix of X, it means that e is the first letter. If we consider a specialization of X and we again remove the first letter, we also obtain a suffix belonging to $\mathcal{G}_S$. Therefore, $\langle(\mathcal{F}_S, E_S), \mathcal{G}_S, cov_S, \phi_S\rangle$ is a minimizable set system.

Obviously, a minimizable set system can be reduced to a system of smaller cardinality of which the patterns are called the *minimal patterns*:

Definition 4 (Minimal pattern). *A pattern* X *is minimal for* $\langle(\mathcal{F}, E), \mathcal{G}, cov, \phi\rangle$ *iff* $X \in \mathcal{F}$ *and for every generalization* $Y \in \mathcal{G}$ *such that* $Y \subset X$, $cov(Y) \neq cov(X)$. $\mathcal{M}(\mathcal{S})$ *denotes the set of all minimal patterns.*

Definition 4 means that a pattern is minimal whenever its cover differs from that of any generalization. For example, for the cover operator cov_S, the minimal patterns have a *strictly* smaller cover than their generalizations. The string ab is not minimal due to its suffix b because $cov_S(\{(b,2)\}) = cov_S(\{(a,1),(b,2)\}) = \{0,7\}$. For our running example, the whole collection of minimal strings is $\mathcal{M}(\mathcal{S}_S) = \{a, b, r, c, d, ca, ra, da\}$.

Given a minimizable set system $\mathcal{S} = \langle(\mathcal{F}, E), \mathcal{G}, cov, \phi\rangle$, the minimal pattern mining problem consists in enumerating all the minimal patterns for $\mathcal{S}$.

3 Enumerating the Minimal Patterns

This section aims at effectively mining all the minimal patterns in a depth-first search manner (Section 3.3). To do this, we rely on two key ideas: the pruning of the search space (Section 3.1) and the fast minimality checking (Section 3.2).

Before, it is important to recall that the minimal patterns are sufficient to induce the cover of any pattern. From now, we consider a minimizable set system $\mathcal{S} = \langle (\mathcal{F}, E), \mathcal{G}, cov, \phi \rangle$. The minimal patterns $\mathcal{M}(\mathcal{S})$ is a lossless representation of all patterns of $\mathcal{F}$ in the sense we can find the cover of any pattern.

Theorem 1 (Condensed representation). *The set of minimal patterns is a concise representation of $\mathcal{F}$ adequate to cov: for any pattern $X \in \mathcal{F}$, there exists $Y \subseteq X$ such that $\phi(Y) \in \mathcal{M}(\mathcal{S})$ and $cov(\phi(Y)) = cov(X)$.*

Theorem 1 means that $\mathcal{M}(\mathcal{S})$ is really a condensed representation of $\mathcal{S}$ because the minimal pattern mining enables us to infer the cover of any pattern in $\mathcal{S}$. For instance, the cover of the non-minimal pattern $\{(a,1),(b,2)\}$ equals to that of the minimal pattern $\phi(\{(b,2)\}) = \{(b,1)\}$: $cov_S(\{(a,1),(b,2)\}) = cov_S(\{(b,1)\}) = \{0,7\}$. It is preferable to extract $\mathcal{M}(\mathcal{S})$ instead of $\mathcal{S}$ because its size is lower (and, in general, much lower) than the total number of patterns.

3.1 Search Space Pruning

The first problem we face is fairly classical. Given a minimizable set system $\mathcal{S} = \langle (\mathcal{F}, E), \mathcal{G}, cov, \phi \rangle$, the number of patterns $|\mathcal{F}|$ is huge in general (in the worst case, it reaches $2^{|E|}$ patterns). So, it is absolutely necessary not to completely scan the search space for focusing on the minimal patterns. Effective techniques can be used to prune the search space due to the downward closure of $\mathcal{M}(\mathcal{S})$:

Theorem 2 (Independence system). *If a pattern X is minimal for $\mathcal{S}$, then any pattern $Y \in \mathcal{F}$ satisfying $Y \subseteq X$ is also minimal for $\mathcal{S}$.*

The proof of this theorem strongly relies on a key lemma saying that a non-minimal pattern has a direct generalization having the same cover (proofs are omitted due to lack of space):

Lemma 1. *If X is not mininal, there exists $e \in X$ such that $X \setminus \{e\} \in \mathcal{G}$ and $cov(X) = cov(X \setminus \{e\})$.*

For instance, as the string da is minimal, the substrings d and a are also minimal. More interestingly, as ab is not minimal, the string abr is not minimal. It means that the string ab is a cut-off point in the search space. In practice, anti-monotone pruning is recognized as a very powerful tool whatever the traversal of the search space (level by level or in depth).

3.2 Fast Minimality Checking

The main difficulty in extracting the minimal patterns is to test whether a pattern is minimal or not. As we mentioned earlier, this is particularly difficult in a depth-first traversal because all subsets have not yet been enumerated. Indeed, depth-first approaches only have access to the first parent branch contrary to levelwise approaches. To overcome this difficulty, we introduce the concept of *critical objects* inspired from critical edges in case of minimal traversals [13]. Intuitively, the critical objects of an element e for a pattern X are objects that are not covered by X due to the element e. We now give a formal definition of the critical objects derived from any cover operator:

Definition 5 (Critical objects). *For a pattern X, the critical objects of an element $e \in X$, denoted by $\widehat{cov}(X, e)$ is the set of objects that belongs to the cover of X without e and not to the cover of e: $\widehat{cov}(X, e) = cov(X \setminus e) \setminus cov(e)$.*

Let us illustrate the critical objects with our running example. For $\{(a, 1), (b, 2)\}$, the critical objects $\widehat{cov}(ab, a)$ of the element $(a, 1)$ correspond to $\emptyset$ $(= \{0, 7\} \setminus \{0, 3, 5, 7, 10\})$. It means that the addition of a to b has no impact on the cover of ab. At the opposite, for the same pattern, the critical objects of $(b, 2)$ are $\{3, 5, 10\}$ $(= \{0, 3, 5, 7, 10\} \setminus \{0, 7\})$. It is due to the element b that ab does not cover the objects $\{3, 5, 10\}$.

The critical objects are central in our proposition for the following reasons: 1) the critical objects easily characterize the minimal patterns; and 2) the critical objects can efficiently be computed in a depth-first search algorithm.

The converse of Lemma 1 says that a pattern is minimal if its cover differs from that of its generalization. We can reformulate this definition thanks to the notion of critical objects as follows:

Property 1 (Minimality). $X \in \mathcal{F}$ is minimal if $\forall e \in X$ such that $X \setminus e \in \mathcal{G}$, $\widehat{cov}(X, e) \neq \emptyset$.

Typically, as b is a generalization of the string ab and at the same time, $\widehat{cov}(ab, a)$ is empty, ab is not minimal. Property 1 means that checking whether a candidate X is minimal only requires to know the critical objects of all the elements in X. Unlike the usual definition, no information is required on the subsets. Therefore, the critical objects allow us to design a depth-first algorithm if (and only if) computing the critical objects does not also require information on the subsets.

In a depth-first traversal, we want to update the critical objects of an element e for the pattern X when a new element e' is added to X. In such case, we now show that the critical objects can efficiently be computed by intersecting the old set of critical objects $\widehat{cov}(X, e)$ with the cover of the new element e':

Property 2. The following equality holds for any pattern $X \in \mathcal{F}$ and any two elements $e, e' \in E$: $\widehat{cov}(Xe', e) = \widehat{cov}(X, e) \cap cov(e')$.

For instance, Definition 5 gives $\widehat{cov}_S(a, a) = \{1, 2, 4, 6, 8, 9\}$. As $cov_S(b) = \{0, 7\}$, we obtain that $\widehat{cov}_S(ab, a) = \widehat{cov}_S(a, a) \cap cov_S(b) = \{1, 2, 4, 6, 8, 9\} \cap \{0, 7\} = \emptyset$. Interestingly, Property 2 allows us to compute the critical objects

of any element included in a pattern X having information on a single branch. This is an ideal situation for a depth-first search algorithm.

3.3 Algorithm DeFMe

The algorithm DeFMe takes as inputs the current pattern and the current tail (the list of the remaining items to be checked) and it returns all the minimal patterns containing X (based on *tail*). More precisely, Line 1 checks whether X is minimal or not. If X is minimal, it is output (Line 2). Lines 3-14 explores the subtree containing X based on the tail. For each element e where Xe is a pattern of $\mathcal{F}$ (Line 4) (Property 1), the branch is built with all the necessary information. Line 7 updates the cover and Lines 8-11 updates the critical objects using Property 2. Finally, the function DeFMe is recursively called at Line 12 with the updated tail (Line 5).

Algorithm 1. DeFMe$(X, tail)$

Input: X is a pattern, *tail* is the set of the remaining items to be used in order to generate the candidates. Initial values: $X = \emptyset, tail = E$.
Output: polynomially incrementally outputs the minimal patterns.

```
 1: if ∀e ∈ X, ĉov(X, e) ≠ ∅ then
 2:   print X
 3:   for all e ∈ tail do
 4:     if Xe ∈ F then
 5:       tail := tail \ {e}
 6:       Y := Xe
 7:       cov(Y) := cov(X) ∩ cov(e)
 8:       ĉov(Y, e) := cov(X) \ cov(e)
 9:       for all e' ∈ X do
10:         ĉov(Y, e') := ĉov(X, e') ∩ cov(e)
11:       end for
12:       DeFMe(Y, tail)
13:     end if
14:   end for
15: end if
```

Theorems 3 and 4 demonstrate that the algorithm DeFMe has an efficient behavior both in space and time. This efficiency mainly stems from the inexpensive handling of covers/critical objects as explained by the following property:

Property 3. The following inequality holds for any pattern $X \in \mathcal{F}$:

$$|cov(X)| + \sum_{e \in X} |\widehat{cov}(X, e)| \leq |cov(\emptyset)|$$

Property 3 means that for a pattern, the storage of its cover plus that of all the critical objects is upper bounded by the number of objects (i.e., $|cov(\emptyset)|$). Thus, it is straightforward to deduce the memory space required by the algorithm:

Theorem 3 (Polynomial-space complexity). *$\mathcal{M}(\mathcal{S})$ is enumerable in $O(|cov(\emptyset)| \times m)$ space where m is the maximal size of a feasible set in $\mathcal{F}$.*

In practice, the used memory space is very limited because m is small. In addition, the amount of time between each output pattern is polynomial:

Theorem 4 (Polynomial-delay complexity). *$\mathcal{M}(\mathcal{S})$ is enumerable in $O(|E|^2 \times |cov(\emptyset)|)$ time per minimal pattern.*

It is not difficult to see that between two output patterns, DeFMe requires a polynomial number of operations assuming that the membership oracle is computable in polytime (Line 4). Indeed, the computation of the cover and that of the critical objects (Lines 7-11) is linear with the number of objects due to Property 3; the loop in Line 3 does not exceed $|E|$ iterations and finally, the number of consecutive backtracks is at most $|E|$.

4 Experimental Study

The aim of our experiments is to quantify the benefit brought by DeFMe both on effectiveness and conciseness. We show its effectiveness with the problem of free itemset mining for which several prototypes already exist in the literature. Then we instantiate DeFMe to extract the collection of minimal strings and compare its size with that of closed strings. All tests were performed on a 2.2 GHz Opteron processor with Linux operating system and 200 GB of RAM memory.

4.1 Free Itemset Mining

We designed a prototype of DeFMe for itemset mining as a proof of concept and we compared it with two other prototypes: ACminer based on a levelwise algorithm [1] and NDI[2] based on a depth-first traversal with reordered items [11]. For this purpose, we conducted experiments on benchmarks coming from the FIMI repository and the 2004 PKDD Discovery Challenge[3]. The first three columns of Table 1 give the characteristics of these datasets. The fourth column gives the used minimal support threshold. The next three columns report the running times and finally, the last three columns indicate the memory consumption.

The best performances are highlighted in bold in Table 1 for both time and space. ACminer is by far the slowest prototype. Its levelwise approach is particularly penalized by the large amount of used memory. Except on the genomic datasets 74x822 and 90x27679, the running times of NDI clearly outperform those of DeFMe. As a piece of information, Figure 1 details, for various minsup thresholds, the speed of DeFMe. It plots the number of minimal patterns it extracted for each second of computing time.

Concerning memory consumption, DeFMe is (as expected) the most efficient algorithm. In certain cases, the increase of the storage memory would not be sufficient to treat the most difficult datasets. Here, ACminer and NDI are

[2] As this prototype mines non-derivable itemsets, it enable us to compute free patterns when the depth parameter is set to 1.

[3] `fimi.ua.ac.be/data/` and `lisp.vse.cz/challenge/ecmlpkdd2004/`

Table 1. Characteristics of benchmarks and results about free itemset mining

				time (s)			memory (kB)		
dataset	objects	items	minsup	ACMINER	NDI	DEFME	ACMINER	NDI	DEFME
74x822	74	822	88%	fail	fail	**45**	fail	fail	**3,328**
90x27679	90	27,679	91%	fail	fail	**79**	fail	fail	**13,352**
chess	3,196	75	22%	6,623	**187**	192	3,914,588	1,684,540	**8,744**
connect	67,557	129	7%	34,943	**115**	4,873	2,087,216	1,181,296	**174,680**
pumsb	49,046	2,113	51%	70,014	**212**	548	7,236,812	1,818,500	**118,240**
pumsb*	49,046	2,088	5%	21,267	**202**	4,600	5,175,752	2,523,384	**170,632**

not suitable to process genomic datasets even with 200GB of RAM memory and relatively high thresholds. More precisely, Figure 1 plots the ratio between NDI's and DEFME's memory use for various minsup thresholds. It is easy to notice that this ratio quickly leads NDI to go out of memory. DEFME works with bounded memory and then is not minsup limited.

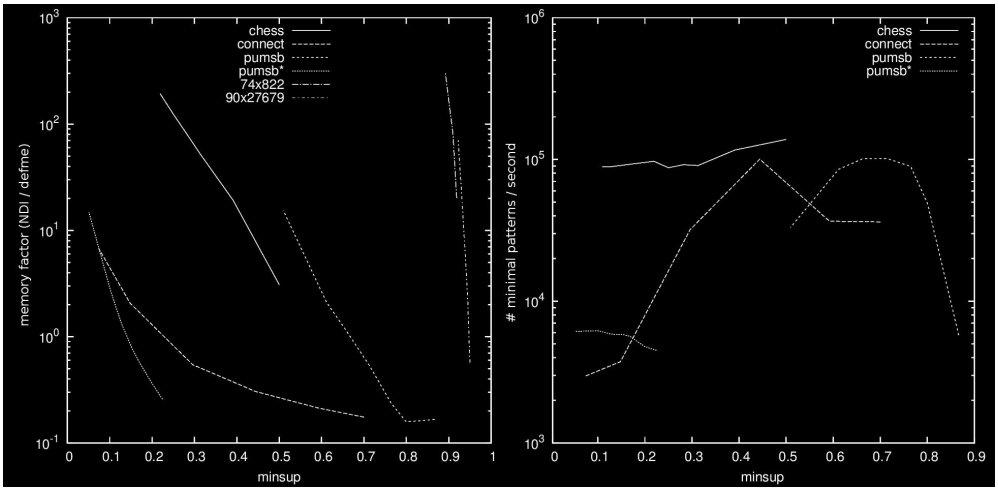

Fig. 1. Ratio of mining speed (left) and memory use (right) of NDI by DEFME

4.2 Minimal String Mining

In this section, we adopt the formalism of strings stemming from our running example. We compared our algorithm for minimal string mining with the MAXMOTIF prototype provided by Takeaki Uno that mines closed strings [10]. Our goal is to compare the size of condensed representations based on minimal strings with those based on all strings and all closed strings. We do not report the execution times because MAXMOTIF developed in Java is much slower than DEFME (developed in C++). Experiments are conducted on two datasets: chromosom[4] and msnbc coming from the UCI ML repository (`www.ics.uci.edu/~mlearn`).

Figure 2 and 3 report the number of strings/minimal strings/closed strings mined in chromosom and msnbc. Of course, whatever the collection of patterns,

[4] This dataset is provided with MAXMOTIF: `research.nii.ac.jp/~uno/codes.htm`

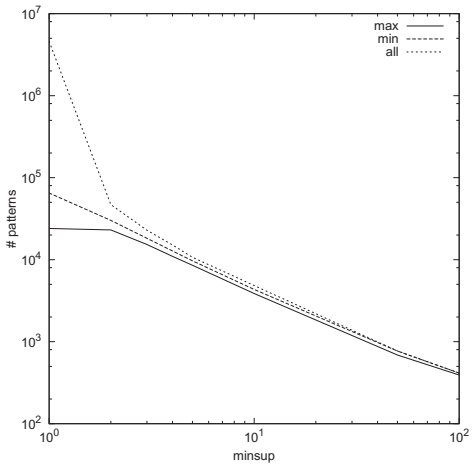

Fig. 2. Number of patterns in chromosom

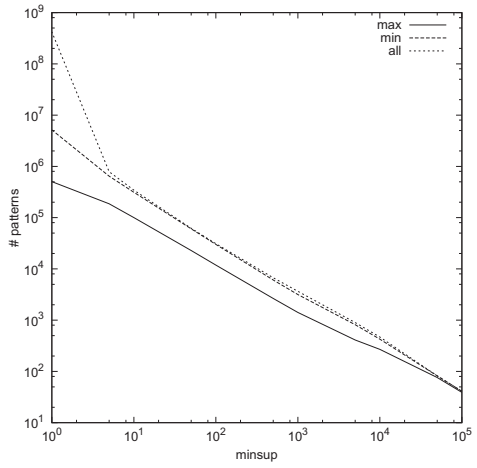

Fig. 3. Number of patterns in msnbc

the number of patterns increases with the decrease of the minimal frequency threshold. Interestingly, the two condensed representations become particularly useful when the frequency threshold is very small. Clearly the number of minimal strings is greater than the number of closed strings, but the gap is not as important as it is the case with free and closed itemsets.

5 Related Work

The collection of minimal patterns is a kind of condensed representations. Let us recall that a condensed representation of the frequent patterns is a set of patterns that can regenerate all the patterns that are frequent with their frequency. The success of the condensed representations stems from their undeniable benefit to reduce the number of mined patterns by eliminating redundancies. A large number of condensed representations have been proposed in literature [6,14]: closed itemsets [2], free itemsets [1], essential itemsets [15], Non-Derivable Itemsets [11], itemsets with negation [16] and so on. Two ideas are at the core of the condensed representations: the closure operator [14] that builds equivalence classes and the principle of inclusion-exclusion. As the inclusion-exclusion principle only works for the frequency, this paper exclusively focuses on minimal patterns considering equivalence classes. In particular, as indicated above the system $\mathcal{S}_{\mathcal{I}} = \langle (2^{\mathcal{I}}, \mathcal{I}), 2^{\mathcal{I}}, cov_{\mathcal{I}}, Id \rangle$ is minimizable and $\mathcal{M}(\mathcal{S}_{\mathcal{I}})$ corresponds exactly to the free itemsets (or generators). The frequency of each itemset is computed using the cardinality of the cover. Replace the cover operator $cov_{\mathcal{I}}$ by $\overline{cov_{\mathcal{I}}} : X \mapsto \{o \in \mathcal{O} \mid X \cap o = \emptyset\}$ leads to a new minimizable set system $\langle (2^{\mathcal{I}}, \mathcal{I}), 2^{\mathcal{I}}, \overline{cov_{\mathcal{I}}}, Id \rangle$ of which minimal patterns are essential itemsets [15]. The disjunctive frequency of an itemset X is $|\mathcal{O}| - |\overline{cov_{\mathcal{I}}}(X)|$.

Minimal pattern mining has a lot of applications and their use is not limited to obtain frequent patterns more efficiently. Their properties are useful for higher KDD tasks. For instance, minimal patterns are used in conjunction of closed patterns to produce non-redundant [3] or informative rules [2]. The sequential rules also benefit from minimality [17]. It is also possible to exploit the minimal patterns for mining the classification rules that are the key elements of associative

classifiers [4]. Our framework is well-adapted for mining all such minimal classification rules that satisfy interestingness criteria involving frequencies.Assuming that the set of objects $\mathcal{O}$ is divided into two disjoint classes $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$, the confidence of the classification rule $X \rightarrow class_1$ is $|\mathcal{O}_1 \cap cov_{\mathcal{I}}(X)|/|cov_{\mathcal{I}}(X)|$. More generally, it is easy to show that any frequency-based measure (e.g., lift, bond) can be derived from the positive and negative covers. In addition, the essential patterns are useful for deriving minimal traversals that exactly corresponds to the maximal patterns of $\mathcal{M}(\langle(2^{\mathcal{I}}, \mathcal{I}), 2^{\mathcal{I}}, \overline{cov_{\mathcal{I}}}, Id\rangle)$. Let us recall that the minimal transversal generation is a very important problem which has many applications in Logic (e.g., satisfiability checking), Artificial Intelligence (e.g., model-based diagnosis) and Machine Learning (e.g., exact learning) [5,13].

The condensed representations of minimal patterns are not limited to frequency-based measures or itemsets. Indeed, it is also possible to mine the minimal patterns adequate to classical aggregate functions like min, max or sum [7]. Minizable set systems are also well-adapted for such measures. For instance, let us consider the function $cov_{min}(X) = \{val(i) | \exists i \in \mathcal{I}, val(i) \leq min(X.val)\}$ that returns all the possible values of val less than $min(X.val)$. This function is a cover operator and $\langle(2^{\mathcal{I}}, \mathcal{I}), 2^{\mathcal{I}}, cov_{\min}, Id\rangle$ is even a minimizable set system. The minimal patterns adequate to min correspond to the minimal patterns of the previous set system. Furthermore, the value $min(X.val)$ could be obtained as follows $\max(cov_{min}(X))$. A similar approach enables us to deal with max and sum. In parallel, several studies have extended the notion of generators to address other languages such as sequences [8,18], negative itemsets [19], graphs [20]. Unfortunately no work proposes a generic framework to extend the condensed representations based on minimality to a broad spectrum of languages as it was done with closed patterns [10]. For instance, [1,2,11,12] only address itemsets or [8,18] focus exclusively on sequences. In this paper, we have made the connection between the set systems and only two languages: itemsets and strings due to space limitation. Numerous other languages can be represented using this set system framework. In particular, all the languages depicted by [10] are suitable.

6 Conclusion

By proposing the new notion of *minimizable set system*, this paper extended the paradigm of minimal patterns to a broad spectrum of functions and languages. This framework encompasses the current methods since the existing condensed representations (e.g., free or essential itemsets) fit to specific cases of our framework. Besides, DefMe efficiently mines such minimal patterns even in difficult datasets, which are intractable by state-of-the-art algorithms. Experiments also showed on strings that the sizes of the minimal patterns are smaller than the total number of patterns.

Of course, we think that there is still room to improve our implementation even if it is difficult to find a compromise between generic method and speed. We especially want to test the ability of the minimal patterns for generating minimal classification rules with new types of data, such as strings. Similarly, it would be interesting to build associative classifiers from minimal patterns.

Acknowledgments. This article has been partially funded by the Hybride project (ANR-11-BS02-0002).

References

1. Boulicaut, J.-F., Bykowski, A., Rigotti, C.: Approximation of frequency queries by means of free-sets. In: Zighed, D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 75–85. Springer, Heidelberg (2000)
2. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient mining of association rules using closed itemset lattices. Inf. Syst. 24(1), 25–46 (1999)
3. Zaki, M.J.: Generating non-redundant association rules. In: KDD, pp. 34–43 (2000)
4. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: KDD, pp. 80–86 (1998)
5. Eiter, T., Gottlob, G.: Hypergraph transversal computation and related problems in logic and AI. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 549–564. Springer, Heidelberg (2002)
6. Calders, T., Rigotti, C., Boulicaut, J.-F.: A survey on condensed representations for frequent sets. In: Boulicaut, J.-F., De Raedt, L., Mannila, H. (eds.) Constraint-Based Mining. LNCS (LNAI), vol. 3848, pp. 64–80. Springer, Heidelberg (2006)
7. Soulet, A., Crémilleux, B.: Adequate condensed representations of patterns. Data Min. Knowl. Discov. 17(1), 94–110 (2008)
8. Lo, D., Khoo, S.C., Li, J.: Mining and ranking generators of sequential patterns. In: SDM, pp. 553–564. SIAM (2008)
9. Li, J., Li, H., Wong, L., Pei, J., Dong, G.: Minimum description length principle: Generators are preferable to closed patterns. In: AAAI, pp. 409–414 (2006)
10. Arimura, H., Uno, T.: Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In: SDM, pp. 1087–1098. SIAM (2009)
11. Calders, T., Goethals, B.: Depth-first non-derivable itemset mining. In: SDM, pp. 250–261 (2005)
12. Liu, G., Li, J., Wong, L.: A new concise representation of frequent itemsets using generators and a positive border. Knowl. Inf. Syst. 17(1), 35–56 (2008)
13. Murakami, K., Uno, T.: Efficient algorithms for dualizing large-scale hypergraphs. In: ALENEX, pp. 1–13 (2013)
14. Hamrouni, T.: Key roles of closed sets and minimal generators in concise representations of frequent patterns. Intell. Data Anal. 16(4), 581–631 (2012)
15. Casali, A., Cicchetti, R., Lakhal, L.: Essential patterns: A perfect cover of frequent patterns. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2005. LNCS, vol. 3589, pp. 428–437. Springer, Heidelberg (2005)
16. Kryszkiewicz, M.: Generalized disjunction-free representation of frequent patterns with negation. J. Exp. Theor. Artif. Intell. 17(1-2), 63–82 (2005)
17. Lo, D., Khoo, S.C., Wong, L.: Non-redundant sequential rules - theory and algorithm. Inf. Syst. 34(4-5), 438–453 (2009)
18. Gao, C., Wang, J., He, Y., Zhou, L.: Efficient mining of frequent sequence generators. In: WWW, pp. 1051–1052. ACM (2008)
19. Gasmi, G., Yahia, S.B., Nguifo, E.M., Bouker, S.: Extraction of association rules based on literalsets. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2007. LNCS, vol. 4654, pp. 293–302. Springer, Heidelberg (2007)
20. Zeng, Z., Wang, J., Zhang, J., Zhou, L.: FOGGER: an algorithm for graph generator discovery. In: EDBT, pp. 517–528 (2009)