Efficient Mining of Niches and Set Routines

Guozhu Dong and Kaustubh Deshpande
Dept. of CSE, Wright State University, Dayton OH 45435, USA.

Abstract. It is widely recognized that successful businesses usually fall into set routines and become limited by their past. To remain successful, they need to discover new opportunities and niches. Niches are *surprising rules that contradict the set routines*; they capture significant, representative client sectors that deserve new, more profitable treatments; they are not merely strong-rule and exception pairs. In this paper we study the efficient mining of set routines and niches. We also introduce a semantic approach to select a set of representative patterns, and present an efficient incremental algorithm to implement the approach.

Key words: Data mining, niches, set routines, exceptions, interestingness, semantic-based selection

1 Introduction

In order to succeed, a starting business is always looking for opportunities. An established business, however, usually falls into set routines and becomes limited by its past. To remain successful, it needs to discover new opportunities and niches. Niches are *surprising rules that contradict the set routines*; they capture significant, representative client sectors that deserve new, targeted, more profitable treatments; they are not merely strong-rule and exception pairs. Niches are also useful for many other applications such as medicine, scientific discovery, and customized treatment of clients.

We illustrate the importance of niches with an example reported in KD-Nuggets [5]. *"Farmers Insurance found a previously unnoticed niche of sports car enthusiasts: married boomers with a couple of kids and a second family car, maybe a minivan, parked in the driveway. Farmers relaxed its underwriting rules and cut rates on certain sports cars for people who fit the profile (and presumably gained market share in this niche)."* An insurance company divides clients into different risk classes, and charges rates according to risk levels. Rules for deciding risk levels are formulated through experience, statistical analysis, or data mining; these rules may remain fixed for a long time and become set routines. A niche here can be a special segment of customers who are less risky than the company currently believes.

To mine niches, we need to capture the set routines first. For a business decision, the set routines (SRS) should correspond to a set of business rules or operational policies; in general, the SRS should correspond to a set of dominant trends (DTs) which are important to the task (decision) at hand. A DT can be captured by an emerging pattern (EP), namely a pattern which occurs more frequently in the undesirable instances than in the desirable instances, or vice versa. The EP should occur at a relatively high support. This corresponds to the fact that the SRS is usually formed by past experience, observations, and even data analysis, because human observations and past data mining algorithms mainly discover high support patterns.

The SRS should contain a relatively small number of DTs, such as 100 or less. More importantly, the DTs should represent different segments of the clients (instances); each DT should capture a unique segment of the instances. We will first mine a set of EPs, using ConsEPMiner [16]. Then we use a novel semantic-based approach, which minimizes overlap and maximized disjointness between DTs, to select an SRS from this set. We associate each pattern with the set of data instances containing the pattern, and we consider two patterns semantically similar if their associated data sets overlap sufficiently. Semantic-based selection ensures that different DTs capture almost disjoint segments of data, and the DTs in the SRS collectively cover as much data as possible; it also helps enhance understandability of the niches and avoid repeated computation.

The exception EPs to the SRS are mined, again using ConsEPMiner, and the semantic-based approach is used to select good representatives as niches. Experiments show that our algorithm is efficient, can find meaningful niches from real data, and can succeed in niche mining at low support levels.

To illustrate how niche mining can be done, consider an auto insurance company which has been in operation for a while. If an SRS cannot be extracted from the business manual, we simply use the company's database, using a threshold on rates to divide the clients into $Risky^{view}$ and $NonRisky^{view}$ (according to the view of the company). We extract an SRS from $Risky^{view}$ and $NonRisky^{view}$. Then we divide the clients into $Risky^{actual}$ and $NonRisky^{actual}$, using a threshold on the total number (or amount) of claims. We can mine the exceptions to the SRS using these two new data sets.

Niche mining can also be useful to a new insurance company, to help it understand how its competitors work. If the new company has access to all the data discussed above, then it can just use the procedure discussed above. Otherwise, it can still gain insights on how its competitors work, by using $Risky^{actual}$ and $NonRisky^{actual}$ for both SRS mining and niche mining.

Most past work on data mining concentrated on discovering high-support patterns or high-support relationships. In contrast, our work is concerned with finding low support exceptions to high support strong rules. Moreover, our niches are exceptions to the set routines of an organization and thus are exceptions in a global sense, and can thus provide the organization with "complete" representatives of niche opportunities. In contrast, past research on exception mining [14, 10] considered strong-rule and exception pairs in a local manner. Producing all strong-rule and exception pairs will make the results hard to understand; moreover, the mining process also becomes unnecessarily expensive, because repeated similar computation is performed for similar strong rules. Other researchers have considered mining of exceptions of other types, e.g. interesting holes in data [8]. Our work is related to interestingness of patterns [11, 12, 2, 9].

Our semantic-based selection of representative patterns is related to [15]. That paper used the semantics-based idea of "cover" to select patterns, but it did not consider minimizing overlap and maximizing disjointness.

Our method can also be used to find semantic representatives of strong-rule and exception pairs. Structures can be added to the DTs in an SRS, e.g. by imposing an ordering; mining of niches for such extensions will be a future topic.

2 Set Routines and Niches

The way an organization operates can be understood from its past operations. We will consider one decision (e.g. is a client risky or not risky) for the organization. For each instance, the decision will be either Yes or No. We will call each of the two decisions a class, and will denote these two classes as P (positive) and N (negative). For C in $\{P, N\}$, we will use NC to mean the opposite class of C. For convenience, we will identify a class with its associated set of instances.

We assume the readers are familiar with transactions, relations, itemsets, and supports of itemsets. We will use the term instance to refer to case, transaction, vector, or tuple. We assume that numerical attributes have been discretized using some binning method such as equal-density or equal-length, and numerical values have been mapped to their containing intervals. For uniformity, we will also refer to these bins as values. Relations can now be viewed as transactions, where an item is an attribute-value pair.

For any organization, the policies regarding the decision are usually formulated under the influence of dominant trends (DTs). There are several requirements on a DT: (a) It should be a pattern (condition). (b) It should capture a significant segment of the instances. (c) It should occur much more frequently in one class than in another class.

(a) ensures that the DT can be tested on instances. (b) and (c) ensure that the DT has influenced the formulation of policies of the organization, because it differentiates between the two classes over a significant segment of instances.

For example, for insurance, statistics shows that sports car owners are usually riskier than other owners. The condition involved in this pattern is "the owner ownes sports cars". This condition captures a significant segment of clients.

An emerging pattern (EP) is a pattern meeting these requirements of a DT. EPs were introduced [3] to capture sharp differences between data classes or emerging trends in time. Suppose we are given two classes, say classes C_1 and C_2, associated with respectively data sets D_1 and D_2. Let $supp_i$(X) denote $supp_{D_i}$(X). The growth rate of an itemset X from D_1 to D_2 is defined as the ratio $\frac{supp_2(X)}{supp_1(X)}$ (letting $\frac{0}{0} = 0$ and $\frac{\neq 0}{0} = \infty$). Given a growth rate threshold $\rho > 1$, if the growth rate of X from D_1 to D_2 is $\geq \rho$, then X is called an emerging pattern from D_1 to D_2 (or from C_1 to C_2, or simply an EP of class C_2); C_1 is called the background class and C_2 the target class; we will write $X : C_2$ to signify the fact that X is an EP of class C_2. We can use EPs to make decisions. If $X : C_2$ is an EP with a growth rate of 40, $|D_1| \approx |D_2|$, and t is an instance containing X, then the probability that t belongs to class C_2 is $97\% = \frac{40}{40+1}$.

The set routines (SRS) of an organization should consist of a set of DTs. They should satisfy these constraints[1] in order to provide an accurate model: (a) The DTs in the SRS should collectively cover as many instances as possible. (b) Different DTs in the SRS should cover disjoint subsets of instances. (c) The SRS should be relatively small, containing perhaps ≤ 100 DTs.

For example, the SRS for a car insurance company should correspond to the way the rates are determined. One DT may be {sports-car-owner : *yes*} : *Risky*. Another DT may be {points-on-license > 3} : *Risky*. A third DT may be $\{3.0 < GPA \leq 4.0\}$: *nonRisky*. There can be more DTs in the SRS.

We now formalize the meaning of cover, disjointness and overlap. For each pattern X, let $Sat(X)$ denote the set of instances (of either class P or N) satisfying (containing or covered by) X. (A more refined approach is possible, by dividing $Sat(X)$ into two classes and adjusting semantic-based selection accordingly. To focus on the spirit of the semantic-based approach, we omit the details of that refinement here.) For a set S of patterns, let $Sat(S) = \cup_{X \in S} Sat(X)$. Difference and overlap between two patterns X and Y will be measured by $|(Sat(X) - Sat(Y)) \cup (Sat(Y) - Sat(X))|$ and $|Sat(X) \cap Sat(Y)|$, resp.

Example 1. Suppose our two classes together contain the instances of Table 1 and we are given the four EPs in Table 2. Then $\textsc{Sat}(\{1,2\}) = \{t_1,t_2,t_3,t_6\}$, $\textsc{Sat}(\{2,3\}) = \{t_1,t_5\}$, and $\textsc{Sat}(\{\{1,2\},\{2,3\}\}) = \{t_1,t_2,t_3,t_5,t_6\}$. The overlap between $\{1,2\}$ and $\{2,3\}$ is $\{t_1\}$ and the difference between them is $\{t_2,t_3,t_5,t_6\}$.

Transaction id	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Items	{1,2,3}	{1,2,4}	{1,2}	{2,4,5}	{2,3,4}	{1,2}	{4,5}	{1,4,5}

Table 1: Transactions

EP
{1,2}
{2,4}
{2,3}
{4,5}

Table 2: EPs

We now consider how to capture niches. Similar to dominant trends, niches should naturally be closely related to the decision under consideration; so we will also capture a niche by an EP. A niche should satisfy the following requirements: (a) A niche should be an exception to some DT in SRS. It should capture a subset of the instances captured by the DT and lead to a decision reversing that of the DT. (b) A niche should not be implied by other DTs of the SRS. We say that $XY : C$ is implied by an EP $Z : C$ if Z is a subset of XY.

For the auto insurance example, the EP {sports-car-owner:yes, age:[40..60], married:yes, #kids $\geq$ 2, second-family-car:yes}:NonRisky is a niche. It is an exception to the {sports-car-owner:yes}:Risky DT and it is not implied by any other DT in the SRS. However, the EP {age:[18..25], GPA:[3.0..4.0]}:NonRisky is not a niche: Although an exception to the DT {age:[18..25]}:Risky, it is implied by the EP {GPA:[3.0..4.0]}:NonRisky, which is in SRS.

[1] Syntactical difference is not good for capturing SRS, because syntactically disjoint patterns may be semantically similar: {small-car-owner:yes} and {age:[18..25]} are syntactically different but may cover nearly the same segment of drivers.

It is sometimes possible to extract the SRS for an organization from its operational manual. If that is not possible, an SRS can be mined from operational data of the organization; we will address this problem in Section 4.

3 The Niche Mining Problem and Our Algorithm

The **niche-mining problem** is the following: Given two classes P and N, mine an SRS (if not given) and niches satisfying a minsupp threshold on DTs.

```
Niche Miner (SRS S, classes P, N, minDTsupp)
(1) if S is empty then
(2)     mine sets E1 of EPs from P to N and E2 from N to P;
(3)     select an SRS S from E1 and E2;
(4) mine exceptions to DTs in S;
(5) remove implied exceptions;
(6) select representative exceptions as niches
```

Fig. 1. Niche Miner Pseudo Code

The psuedo code of our algorithm is given in Figure 1. We first mine EPs with growth rate greater than a threshold value (e.g. 5) and remove EPs whose growth rates are relatively low (e.g. not among the top 40%). We then select DTs that are semantically distinct in terms of their SATs. Finally we mine EPs contradicting the DTs and semantically select the niches.

By selecting an SRS as a semantic representation of all possible dominant trends, we avoid the excessive computation needed over a huge number of possible DTs. This allows us to mine at lower support thresholds for the niches. Also, the resulting niches will be more understandable.

Mining EPs: We will use ConsEPMiner (Constraint Based EP Miner) [16] to mine EPs. The algorithm uses constraints, either explicitly given or inherently implied by the data/pattern type, to efficiently mine EPs, from large high dimensional data sets. It uses an improvement constraint to ensure that a representative grid of EPs (in the complete set-theoretic lattice of all EPs) is returned; the set of returned EPs is much smaller than the set of all possible EPs; more specifically, given that one EP X is chosen to be returned, then another EP Y will not be returned if Y is a superset of X but the growth rate of Y is not significantly larger (specified by the improvement constraint) than that of X, then Y will not be returned. It also uses upper estimates of supports and growth rates, obtained from the counts of candidates (regular or lookahead) already considered, to prune candidates. Pruning and dynamic reordering of items are performed at three different stages: before counting, after the background data set is counted, after both data sets are counted. Using these ideas, ConsEPMiner overcomes the problem of combinatorial explosion of candidate itemsets.

Semantic-based Pattern Selection: In the next section we will propose efficient, semantic-based methods to select an SRS and select the representatives of exceptions as niches.

Finding niches efficiently: Given an SRS, we need to mine exception EPs that contradict the SRS, and select representative exceptions as niches. To select representative exceptions as niches, we also use the SAT-EP-Select algorithm discussed in the next section. The exception EPs are required to satisfy some appropriate growth rate threshold. Observe that the DTs usually have very high growth rates, and large reverse growth rates indicate that the exception EPs are significant reverse "trends."

Our approach to mine exception EPs is: For each DT X we reduce the data sets to contain only the transactions containing X, and then we call ConsEP-Miner on the reduced data sets to find EPs that contradict X. This method is efficient since the reduced data sets are much smaller than the whole data sets. The above process of reducing data sets is called relativization; a similar technique was used in [7] for instance based classification.

A naive method to mine EPs is to call ConsEPMiner once for each class, and select niches from these common sets of EPs. This method suffers from several problems: Many of the mined EPs will be useless, since they do not contradict the EPs in SRS; moreover, to mine exceptions, we need to mine at very small threshold levels of support (such as 0.01%). At such low levels ConsEPMiner generates a huge number of EPs; it is very expensive (with respect to time) to find these EPs and to select from the set of candidate niches. One can improve this approach to make it competitive, by properly seeding ConsEPMiner with the DTs in the SRS. While this improved approach is not yet implemented, we believe that its performance might be comparable with our relativization approach. (The relativization approach may be better if the volume of data is large, as it becomes in memory for each DT after one pass for relativization.)

From the support and growth-rate of a DT X, one can determine the maximum support and growth rate for exception EPs of X. This information can help avoid useless computation: If there are no exception EPs meeting support and growth rate thresholds, then there is no need to call ConsEPMiner for X.

4 Semantic-Based Pattern Selection

When we mine a data set, we get many, perhaps millions of, EPs. The set of EPs can still number in tens of thousands, even after removing those with relatively low supports and growth rates[2]. The objective of semantic-based selection is to select a representative subset of EPs satisfying: two different EPs capture disjoint sets of instances whereas the selected EPs collectively cover as many instances as possible.

The exhaustive approach to selection is clearly infeasible, since the numbers of EPs and of transactions are both very large. In this paper we consider the greedy method sketched in Figure 2; we will give efficient algorithms for the key steps in Section 5. For steps 2 and 5, we use growth-rate to break ties. In step 5, maximizing $|\textsc{Sat}(X) - \textsc{Sat}(S)|$ to $|\textsc{Sat}(X) \cap \textsc{Sat}(S)|$ allows us to maximize disjointness and minimize overlap.

Example 2. We illustrate this algorithm using the transactions and EPs of Example 1. Let $K = 3$. Initially, $\mathcal{P} = \{\{1,2\},\{2,4\},\{2,3\},\{4,5\}\}$, and the Sats are:

[2] While we present the semantics-based selection algorithm for EPs, it can be easily modified to work for other types of patterns.

```
SAT-EP-Select(data set D, EP set 𝒫, K)
;; K is the maximum number of selected EPs
;; returns a set S of selected EPs
1)   compute SAT(X) for all X in 𝒫;
2)   select an EP X from 𝒫 with highest support;
3)   let S = {X} and 𝒫 = 𝒫 − {X};
4)   while (SAT(S) can be expanded) or (|S| < K)
5)      select an EP X s.t. |Sat(X)−Sat(S)| / |Sat(X)∩Sat(S)| is maximal for all EPs in 𝒫;
        ;; if ∃X ∈ 𝒫 s.t. Sat(X) ∩ Sat(S) = ∅, choose X
6)      let S = S ∪ {X} and 𝒫 = 𝒫 − {X};
7)   return S;
```

Fig. 2. Sketch of SAT-EP-Select

$Sat(\{1,2\}) = \{t_1, t_2, t_3, t_6\}$, $Sat(\{2,4\}) = \{t_2, t_4, t_5\}$, $Sat(\{2,3\}) = \{t_1, t_5\}$, and $Sat(\{4,5\}) = \{t_4, t_7, t_8\}$. We choose {1,2} as our first DT, since it has highest support; now $S = \{\{1,2\}\}$ and $\mathcal{P} = \{\{2,4\},\{2,3\},\{4,5\}\}$.

For each iteration, we need to compute, for each EP X in $\mathcal{P}$, $Sat(X) - Sat(S)$ and $Sat(X) \cap Sat(S)$ in order to find their cardinalities. For iteration 1, we get

$|Sat(\{2,4\}) - Sat(S)| = 2$, $|Sat(\{2,4\}) \cap Sat(S)| = 1$;
$|Sat(\{2,3\}) - Sat(S)| = 1$, $|Sat(\{2,3\}) \cap Sat(S)| = 1$;
$|Sat(\{4,5\}) - Sat(S)| = 3$, $|Sat(\{4,5\}) \cap Sat(S)| = 0$.

We choose $X = \{4,5\}$, since it is the only EP whose SAT is disjoint from SAT(S). Now, $S = \{\{1,2\}\} \cup \{\{4,5\}\} = \{\{1,2\},\{4,5\}\}$, and $\mathcal{P} = \{\{2,4\},\{2,3\}\}$.

For Iteration 2: We get $|Sat(\{2,4\}) - Sat(S)| = 1$, $|Sat(\{2,4\}) \cap Sat(S)| = 2$, $|Sat(\{2,3\}) - Sat(S)| = 1$, and $|Sat(\{2,3\}) \cap Sat(S)| = 1$. We choose $X = \{2,3\}$. Since $|S| = K$ (and $Sat(S) = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$ happens to be equal to the entire transaction set), we stop. So the selected EP set is $\{\{1,2\},\{2,3\},\{4,5\}\}$.

The SAT-EP-Select algorithm as given above may select many more EPs for one class than the other. One can avoid this as follows: We choose EPs by switching between the two classes; at any time, if one class C is over represented, then we select the next EP from the class NC (unless NC is exhausted already).

There are two expensive steps in this algorithm. Notice that $\mathcal{P}$ is usually a very large set, containing tens (or even hundreds) of thousands of EPs, the data set is also very large, containing tens of thousands of transactions, and the data may have very high dimension. Step 1, which computes the initial Sats of all EPs in $\mathcal{P}$, can be expensive, since it must check which transactions contain which EPs. We implemented this search using a hash tree, obtaining a speed up of about 50%; the details are omitted. Step 5, which calculates $Sat(X) - Sat(S)$ and $Sat(X) \cap Sat(S)$ for each X in $\mathcal{P}$ at each iteration, can be very expensive, since SAT(X) can be as large as the number of transactions. We will discuss an incremental approach to reduce the cost next.

5 Incremental Computation of Overlap and Differences

The main idea of our technique is to avoid the repeated computations, between consecutive iterations, in computing $|Sat(X) - Sat(S)|$ and $|Sat(X) \cap Sat(S)|$

for each X in $\mathcal{P}$. To this end we store the set $Sat(X) - Sat(S)$ in a variable called $\textsc{SatDif}(X)$. $\textsc{SatDif}(X)$ is initialized to $Sat(X)$ (for $S = \{\}$).

Suppose the EP chosen for a particular iteration is Y, and the set S before Y is added in it is S_0. Let $S_Y = S_0 \cup \{Y\}$. Let $\textsc{SatDif}(X, S_0)$ denote the value of $\textsc{SatDif}(X)$ before Y is added to S, and let $\textsc{SatDif}(X, S_Y)$ denote the value of $\textsc{SatDif}(X)$ after Y is added to S.

We observe this: A transaction t in $\textsc{SatDif}(X, S_0)$ will need to be removed to get $\textsc{SatDif}(X, S_Y)$ iff t is in $\textsc{SatDif}(Y, S_0)$. This observation can be used as follows: In the incremental computation we can use the transactions in $\textsc{SatDif}(Y, S_0)$ to drive the computation of $\textsc{SatDif}(X, S_Y)$. This will improve efficiency because $\textsc{SatDif}(Y, S_0)$ is normally small, especially after a number of iterations have been executed. This idea is formalized in the algorithm below:

```
For each t in SATDIF(Y)
    For each X in P such that t is in SATDIF(X, S_0)
        remove t from SATDIF(X);
```

We now illustrate how the incremental approach is more efficient using the data and EPs of Example 1. The computation with the incremental approach differs from the original naive approach, in that we replace SAT of individual EPs by their SatDif, and that we replace Sat(S) by SatDif(Y) for deriving SatDif(X). More specifically, the computation in iteration 1 is identical to that for the non-incremental approach, as SatDif(X) = Sat(X) for all X at this time. In iteration 2, this is no longer the case; for example, Sat({4,5}) is replaced by SatDif({4,5}). The computation of Sat({2,4}) - Sat(S) is replaced by SatDif({2,4}) - SatDif(Y) (for Y = {4,5}); such replacement is the main reason that the incremental approach is efficient, because SatDif(Y) $=\{t_4,t_7,t_8\}$ is much smaller than Sat(S), and because SatDif(Y) is used to drive the modification of SatDif of all EPs. For example, SatDif({2,4}) is computed through $\{t_4,t_5\} - \{t_4,t_7,t_8\}$ instead of $\{t_2,t_4,t_5\} - \{ t_1,t_2,t_3,t_4,t_6, t_7,t_8\}$.

We will store the set $\textsc{SatDif}(X)$ as a bit vector, and the SatDifs of all EPs in $\mathcal{P}$ as a bit matrix, with EPs as columns and transactions as rows. Initially, position (t, e) of the matrix is 1 iff transaction t contains EP e. Table 3 below shows the initial contents of this matrix for Example 1.

To compute the cardinalities of the differences and the overlaps, we will keep two additional arrays of integers: $OldCounts$ and $CurCounts$. $OldCounts(X)$ stores $|Sat(X)|$ whereas $CurCounts(X)$ stores $|Sat(X) - Sat(S)| = |\textsc{SatDif}(X)|$. We can derive $|Sat(X) \cap Sat(S)|$ from $OldCounts(X) - CurCounts(X)$. The $OldCounts$ array is initialized but never changed.

$CurCounts(X)$ is adjusted only when we modify $\textsc{SatDif}(X)$. Suppose Y has just been selected. For each t in $\textsc{SatDif}(Y)$, if t is in $\textsc{SatDif}(X)$, then we remove t from $\textsc{SatDif}(X)$ and decrement $CurCounts(X)$ by 1.

The algorithm, presented in terms of the SatDif matrix, is given in Figure 3.

We illustrate the algorithm for $Y = \{1, 2\}$. Suppose the matrix before Y is added is given in Table 3. Because t_1 is a member of SatDif(Y), we check if there is any other 1 in the same row; we find that there is a 1 in the column for

;; Suppose Y is chosen as a new DT.
For each transaction t such that the (t,Y) position of the matrix = 1
;; transaction t is contained in SATDIF(Y)
  For each EP $X \neq Y$
    If the (t,X) position of the matrix = 1 then
      change the value to 0 and decrement $CurCounts$ for X;
      ;; remove t from SATDIF(X) and adjust $CurCounts(X)$

Fig. 3. Sketch: Bit-Driven SAT-EP-Select

{2,3}; we change it to 0 and decrement $CurCounts(\{2,3\})$ by 1. Similar actions are taken for t_2, t_3, and t_6. After adding Y to S, the matrix and the $CurCounts$ array become Table 4.

Trans/EPs	{1,2}	{2,4}	{2,3}	{4,5}
t_1	1	0	1	0
t_2	1	1	0	0
t_3	1	0	0	0
t_4	0	1	0	1
t_5	0	1	1	0
t_6	1	0	0	0
t_7	0	0	0	1
t_8	0	0	0	1
CurCounts	4	3	2	3
OldCounts	4	3	2	3

Table 3: SATDIFMatrix for Example 1.

Trans/EPs	{1,2}	{2,4}	{2,3}	{4,5}
t_1	1	0	0	0
t_2	1	0	0	0
t_3	1	0	0	0
t_4	0	1	0	1
t_5	0	1	1	0
t_6	1	0	0	0
t_7	0	0	0	1
t_8	0	0	0	1
CurCounts	4	2	1	3
Oldcounts	4	3	2	3

Table 4: SATDIFMatrix for second iteration

Figure 4 compares the bit-driven incremental approach against the original naive approach, for waveform data with 10000 transactions, 21 attributes, and 14623 EPs. For $K = |SRS| = 100$, it took 2433 seconds for the naive approach but only 86 seconds for the incremental approach. Both approaches use hash for initial SAT computation.

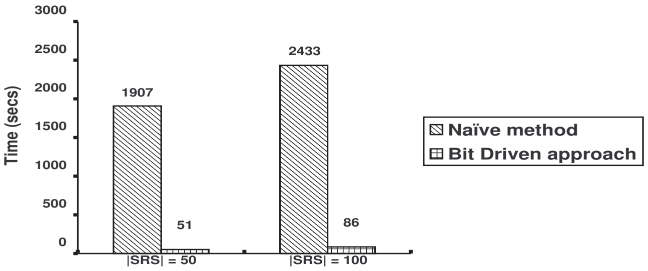

Fig. 4. Cost of SRS Selection

6 Experiments

We report results of two types of experiments: (i) results of mining SRS and niches from real data sets (mostly from the UCI Repository), (ii) results on efficiency and scalability with respect to both volume and dimensions. All experiments, including those reported in previous sections, are carried out on a single node of a multiprocessor system with 195 MHz ip27 processors and a shared main memory of 3698 MB.

SRS and Niches Our results indicate that our algorithm can find meaningful niches from real data, at low support thresholds.

We report results on the Adult data set[3], which has 14 attributes (6 continuous and 8 nominal) and $32561 = 24720 + 7841$ instances. There are two classes defined: 1) People earning $\leq$50K (24720 instances) and 2) people earning >50K (7841 instances). The instances contain personal information (US Census), including these attributes: Age, Workclass, fnlwgt (presumably final wage tax), Education, Education-num, Marital-status, Occupation, Relationship, Race, Sex, Capital-gain, Capital-loss, Hours-per-week & Native-country; the data set was originally used to predict yearly salaries.

We found the SRS given in Table[4] 5. We first used the minsupp = 0.02 threshold, growth rate threshold of 5, and a growth rate improvement threshold of 0.05. We found around 500 EPs for each class. We then selected top 300 EPs for each class. (We found that the Adult data set has very few EPs, unlike the other data sets mentioned above.) Then we selected an SRS from these EPs.

DTs	Supp	GR	Class
age: [-..31.6), Never-married, cg20	0.31	18.23	$<= 50K$
fwt30, edu-num:[13..+), relnship: Husband, White, Male	0.25	25.83	$> 50K$
edu-num:[10..13), Married-civ-s, occup:Exec-mngerial, White, HPW:[40.2..59.8)	0.04	17.39	$> 50K$
age:[31.6..46.2), edu: Prof-school	0.03	17.70	$> 50K$
edu-num:[7..10), Divorced, cg20, HPW:[20.6..40.2), nativ-cntry: USA	0.14	17.48	$<= 50K$
age:[31.6..46.2), fwt30, edu: Masters, occup: Exec-managerial, Male	0.02	18.45	$> 50K$
wrkclss: Private, Separated, cg20, cap-loss: [-..871.2), HPW:[20.6..40.2)	0.04	22.92	$<= 50K$

Table 5: SRS for Adult

cg20 denotes cap-gain: [-..19999.8), fwt30 denotes fnlwgt: [-..306769)

We mined exception EPs and selected niches from them. For the first DT given above, we found 18 niches, whose supports range from 0.76% down to 0.03%. We list three of these below. (We omit the conditions in the DT.)

niches	Supp	GR	Class
edu: Bachelors, relnship: Not-in-family	0.0076	674.00	$> 50K$
edu-num:[13..+), White	0.0018	153.00	$> 50K$
relnship: Not-in-family, Amer-Indian-Eskimo, HPW:[20.6..40.2)	0.0003	GR:∞	$> 50K$

Table 6: Niches for First DT of Table 5

We note that, for real life data sets, niches of support at 0.01% can still be very useful – they may capture a segment of thousands of customers.

Scalability with respect to volume To study how our algorithm performs when size of data set increases, we choose the Adult data set since it has large number of instances, which consists of 32561 instances (14 attributes). We randomly selected sub-data sets of 15K, 20K, 25K, and 30K instances. We set the

[3] Experiments on other data sets, including Musk (40 selected dimensions, $6598 = 5581 + 1017$ instances), Waveform (21 dimensions, $5000 = 1657 + 3343$ instances), and Arabidopsis-DNA (35 dimensions, $44484 = 2305 + 42179$ instances), are not included due to space restrictions.

[4] In the tables we will omit the attributes if such omissions will not lead to confusions.

maximum number of DTs to 50. The graph in Figure 5.a shows that our algorithm is scalable w.r.t. volume, and can find niches from very large data sets.

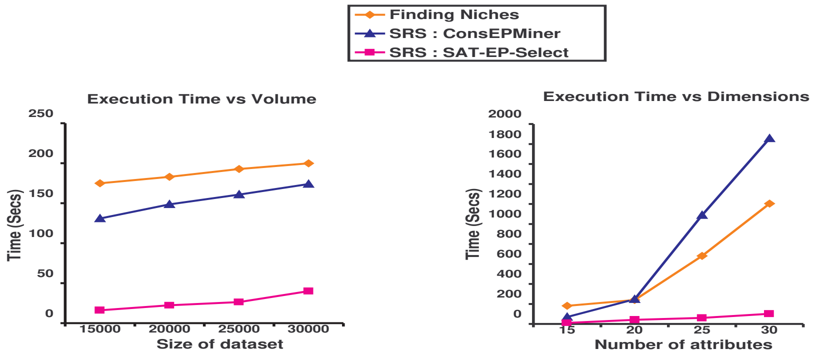

Fig. 5. Execution Time v.s. Volume and Dimensionality

The three curves show the time needed by different components of the algorithm. "SRS:ConsEPMiner" corresponds to the mining of the initial EPs for the selection of the SRS, "SRS:SAT-EP-Select" corresponds to the selection of the DTs from the initial set of EPs, and "Finding Niches" corresponds to finding the niches (including the repeated calls to the relativization, ConsEPMiner, and SAT-EP-Select procedures).

Scalability with respect to Dimensionality To study how our algorithm performs with respect to dimensionality, we generated appropriate data sets by mutating the Musk data set as follows. The original Musk data has 166 attributes and about $6598 = 5581 + 1017$ instances. The dimensionality is high. For any given number N, we randomly choose N attributes of Musk.

However, the number of instances in Musk is small. To increase the number of instances, we add instances. We do not want to add exact copies of existing instances and we do not want to add instances which are totally different from the existing instances. Our solution is to add "mutated copies" of existing instances: We randomly choose an existing instance, and then randomly choose some attributes, and then change these attributes randomly within a range of $\pm 20\%$ of its original value. For each instance, a maximum of 25% of total number of attributes are modified. These ensure that the mutated instances are similar to existing ones but not identical.

The graph in Figure 5.b shows performance of our algorithm w.r.t. the number of dimensions: it is fast and can efficiently deal with high dimensions.

7 Concluding Remarks

In this paper we proposed a way to capture set routines (SRS) and niches, and introduced algorithms to efficiently mine SRS and niches. SRS allows one to understand in a global sense how an organization operates with respect to a decision. By mining niches together with an SRS, we ensure that the niches can indeed provide appropriate representatives of all possible new opportunities, and that they are informative and more understandable.

The semantic-based selection algorithm introduced here is useful for selecting a good set of representatives of patterns. The approach ensures that different selected patterns capture different aspects of the application (equivalently, different segments of data), and collectively they capture as many aspects as possible.

Algorithmically, our niche and SRS mining algorithm is efficient. An important reason is the selection of an SRS before mining exceptions, because this helps avoid the repeated computation for similar dominant patterns. The bit-driven SAT-EP-Select algorithm is also efficient. We also used a hash technique and a relativization technique to improve the efficiency of niche mining.

For future research, the following problems can be considered: How to push the semantic-based selection into a tree-based pattern mining algorithm? How to use niches to improve prediction accuracy in the classification process? It is also interesting to generalize our SAT-EP-Select algorithm, by considering it as a clustering problem for extremely high dimensions.

References

1. R. J. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of ACM-SIGMOD Int'l Conference on Management of Data*, pages 85–93, 1998.
2. G. Dong and J. Li. Interestingness of discovered association rules in terms of neighborhood-based unexpectedness. In PAKDD, 1998.
3. G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. of ACM SIGKDD* 1999.
4. G. Dong, X. Zhang, L. Wong, and J. Li. CAEP: Classification by aggregating emerging patterns. In *Proc. Int'l Conference on Discovery Science*, 1999.
5. KDNuggets. Data mining digs in, July 1999, American demographics. Also KD-Nuggets 99:17, http://www.kdnuggets.com/news/99/n17/i7.html, edited by Gregory Piatetsky-Shapiro.
6. J. Li, G. Dong, and K. Ramamohanarao. Instance-based classification by emerging patterns. In *PKDD 2000.*
7. J. Li, G. Dong, and K. Ramamohanarao. Making use of the most expressive jumping emerging patterns for classification. In *PAKDD 2000.*
8. B. Liu, L.-P. Ku, & W. Hsu. Discovering interesting holes in data. In *IJCAI 1997.*
9. Bing Liu, Wynne Hsu, Lai-Fun Mun, and Hing-Yan Lee. Finding interesting patterns using user expectations. *TKDE*, 11(6):817–832, 1999.
10. H. Liu, H. Lu, L. Feng, and F. Hussain. Efficient search of reliable exceptions. In *PAKDD*, 1999.
11. G. Piatesky-Shapiro and C. J. Matheus. The interestingness of deviations. In *Proc. AAAI KDD Workshop*, 1994.
12. S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. In *VLDB*, 1998.
13. Ron Rymon. Search through systematic set enumeration. In *Proc. the Third Int'l Conference on Principles of Knowledge Representation and Reasoning*, 1992.
14. E. Suzuki and Y. Kodratoff. Discovery of surprising exception rules based on intensity of implication. In *PKDD*, pages 10–18, 1998.
15. H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hätönen, and H. Mannila. Pruning and grouping discovered association rules. In *MLnet Workshop on Statistics, Machine Learning, and Discovery in Databases*, 1995.
16. X. Zhang, G. Dong, and K. Ramamohanarao. Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In *KDD*, 2000.