Dense Itemsets

Jouni K. Seppänen
Jouni.Seppanen@hut.fi

Heikki Mannila
Heikki.Mannila@hut.fi

Laboratory of Computer and Information Science and HIIT Basic Research Unit
P.O.Box 5400, FI-02015 Helsinki University of Technology, Finland

ABSTRACT

Frequent itemset mining has been the subject of a lot of work in data mining research ever since association rules were introduced. In this paper we address a problem with frequent itemsets: that they only count rows where all their attributes are present, and do not allow for any noise. We show that generalizing the concept of frequency while preserving the performance of mining algorithms is nontrivial, and introduce a generalization of frequent itemsets, dense itemsets. Dense itemsets do not require all attributes to be present at the same time; instead, the itemset needs to define a sufficiently large submatrix that exceeds a given density threshold of attributes present.

We consider the problem of computing all dense itemsets in a database. We give a levelwise algorithm for this problem, and also study the top-k variations, i.e., finding the k densest sets with a given support, or the k best-supported sets with a given density. These algorithms select the other parameter automatically, which simplifies mining dense itemsets in an explorative way. We show that the concept captures natural facets of data sets, and give extensive empirical results on the performance of the algorithms. Combining the concept of dense itemsets with set cover ideas, we also show that dense itemsets can be used to obtain succinct descriptions of large datasets. We also discuss some variations of dense itemsets.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database applications—*data mining*

General Terms: Algorithms, Experimentation

Keywords: Frequent itemsets, error tolerance

1. INTRODUCTION

The concept of frequent patterns has been much explored in data mining research for at least ten years, since the inception of association rule mining in [1, 2]. The basic premise has remained the same: to seek *association rules* like

$$\{\,\text{onions}, \text{carrots}\,\} \implies \text{green peas} \quad (\sigma = .03, c = .75)$$

meaning that 75% of customers who buy the two vegetables on the left-hand side will also buy the one on the right-hand side, and that 3% of the database rows support this rule.

The usual way to find these rules is to seek first *frequent itemsets*, i.e., sets of items that often occur together. The concept of frequent itemsets is a useful one, but for understanding the structure of data it is not perfect. For an itemset to be found frequent, *all* of its items must co-occur sufficiently often—in other words, the frequency of an itemset is the result of a full conjunctive database query. Such full conjunctions are rare in real-world data, and connections between items may exist that are manifested by co-occurrence of not the full set of items but of varying subsets.

Generalizing the concept of frequent itemsets turns out to be far from trivial. The most obvious generalization would be to replace the requirement of perfect co-occurrence by the less stringent one of partial co-occurrence: require that an itemset have at least a proportion $1-\varepsilon$ of items present in at least a proportion f of database rows, and say that itemsets meeting this more tolerant requirement have *ε-approximate frequency f*. This definition leads to two problems: first, any frequent itemset will generate many approximately frequent itemsets that do not convey any meaningful information; second, the usual kind of itemset mining algorithms like Apriori are not easily generalized to the new task.

The first problem is illustrated in Table 1a. There, the itemset $ABCDE$ is obviously a frequent one[1], and the data has no further structure. However, a multitude of approximately frequent sets exist with e.g. $\varepsilon = 0.5$: $ABCFGH$, $ABCDFGH$, $ABCDEFGH$ etc., and beyond the fact that $ABCDE$ is frequent, these sets give us no new information.

The second problem can be seen in Table 1b. Now the itemset $ABCD$ has 0.5-approximate frequency 100%, but the approximate frequencies of its subsets are lower: 50% for A, 83% for AB, and 67% for ABC. Thus a set can be approximately frequent even though none of its nontrivial subsets are. This precludes us from pruning the candidate itemsets in the way that Apriori and other algorithms do. A part of this problem is caused by the discrete nature of item presence: the 0.5-approximate frequency of ABC counts only those database rows where two of the three attributes are present, since $\lceil 0.5 \cdot 3 \rceil = 2$.

Both problems are avoided by our definition of dense itemsets. An itemset X is *(σ, δ)-dense*, given two parameters σ and δ, if for any subset $Y \subseteq X$ there is a set r_Y of σ database rows such that in the subdatabase defined by Y and r_Y at least a fraction δ of items are present.

[1]We use $ABCDE$ as shorthand for $\{A, B, C, D, E\}$.

A	B	C	D	E	F	G	H
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0

(a)

A	B	C	D
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1

(b)

Table 1: Two example databases

With this definition, the first problem cannot occur, because every subset of X must have sufficient co-occurrence. The second problem is also solved, since by the definition, the property of being dense is required to hold also for the subsets. For example, in Table 1b all itemsets are $(6, 0.5)$-dense, so a levelwise algorithm would find $ABCD$.

In this paper we formalize the notion of dense itemsets and give some results on their properties. We then give a levelwise algorithm for finding all dense itemsets, and another algorithm for finding the top-k dense itemsets for a given threshold σ, i.e., the sets which are (σ, δ)-dense and have the highest values of δ. We demonstrate the usefulness of the concept and algorithm on a variety of data sets. We also show that the approach can be used to produce intuitive reorderings of the data matrix, and discuss generalizations and variations of the concept of dense itemsets.

We next discuss related work. As far as we are aware, there have been two papers in this direction of research. The first is Pei et al.'s paper on fault-tolerant itemsets. [17] They allow a *fixed* number of missing items on each row in the support of an itemset. This definition leads to an antimonotone concept, but it suffers from a problem like that in Table 1a. To avoid this, Pei et al. require each item in a fault-tolerant set to be itself sufficiently supported. This still causes problems: for example, add to Table 1a sufficiently many rows with only item F present, and $ABCDEF$ will be a fault-tolerant itemset, even though F has no connection to the other items.

The other existing publication is Yang et al.'s paper [20]. They define *strong error-tolerant itemsets* (ETIs), which is the same concept that we have here called ε-approximately frequent sets. They also use the concept of *weak* ETI*s*, which correspond to our dense itemsets without the recursive criterion. That is, a set is dense if and only if all its subsets are weak ETIs. Yang et al. search for weak ETIs, which have a partial monotonicity property, and then select strong ETIs from among the weak ones. They acknowledge the problem illustrated by Table 1a, but the way they deal with it is by searching only a subcollection of all weak ETIs using a heuristic algorithm; it is not very clear how exactly to characterize the resulting itemset family. Since their objective is to obtain a starting point for mixture modeling, this is perhaps not a serious drawback: it is enough to find some assignment of rows into clusters, and the EM algorithm will refine the solution. In contrast, our goal is to find insights about the data directly from dense itemsets, so it is important to understand exactly what kind of sets are found.

The rest of this paper is structured as follows. In Section 2 we formally define dense itemsets and describe some of their properties. Then we discuss algorithmic questions related to mining dense sets in Section 3, giving a levelwise algorithm and a top-k algorithm that helps with parameter selection. In Section 4 we show experimental results on dense itemsets and describe a set cover approach for selecting the most interesting dense sets. The same approach can also be used to reorder the rows and columns of the data matrix. Section 5 is a brief conclusion.

2. DENSE ITEMSETS

In this section we first formalize the idea of dense itemsets, then describe some of their properties. We start by defining binary databases and frequent itemsets [2].

Definition 1. A binary *database* $\mathsf{DB} = \langle R, r \rangle$ consists of a finite set R of *attributes*, also known as *items*, and a finite multiset $r = \{t_1, t_2, \ldots, t_n\}$ of *transactions*, which are subsets of R.

Binary databases are easily visualized as matrices, as we have already done in Table 1, and so one can naturally refer to transactions as rows, and to items as columns. *Itemsets* are, of course, subsets of R. The traditional concept of *frequency* (sometimes called *support*) is the result of a fully conjunctive database query: only those rows count that include every item in the set.

Definition 2. The (absolute) *frequency* of an itemset $X \subseteq R$ in a database $\mathsf{DB} = \langle R, r \rangle$ is the number of transactions that include all the attributes of X:

$$\operatorname{freq}(X) = \bigl|\{\, t \in r \mid t \supseteq X \,\}\bigr|.$$

An itemset is *frequent* if its frequency is greater than or equal to a predefined frequency threshold σ.

As discussed in the introduction, this definition is too strict. It thus needs to be changed to allow some missing attributes: we want to count all transactions, weighting them by the fraction of X's attributes they have.

Definition 3. The *weak density* of an itemset $X \subseteq R$ is

$$\operatorname{wdens}(X, r) = \frac{\sum_{t \in r} |X \cap t|}{|X| \cdot |r|}.$$

The sum in the numerator is taken over all transactions t in r, and the summand is the size of the intersection of X and t; what weak density means is thus the average fraction of items present.

Example 1. In the database $\langle ABCDEFGH, r_a \rangle$ shown in Table 1a, $\operatorname{wdens}(ABCDE, r_a) = 1$, $\operatorname{wdens}(DEF, r_a) = 2/3$, and $\operatorname{wdens}(FGH, r_a) = 0$. In the database $\langle ABCD, r_b \rangle$ of Table 1b, $\operatorname{wdens}(X, r_b) = 1/2$ for every (nonempty) itemset X.

We will generally be interested in the weak density of itemsets over some subdatabases, specifically those subdatabases where the weak density is maximized.

Definition 4. Given a number σ between zero and the size of the relation, the *weak density at support* σ of X is

$$\operatorname{wdens}(\sigma, X, r) = \max_{r'} \operatorname{wdens}(X, r'),$$

where the maximum is taken over all σ-element submultisets r' of r. An itemset is *weakly* (σ, δ)*-dense* if its weak density at support σ exceeds δ, where σ and δ are predefined parameters.

Henceforth, we will omit the multiset r from the notation $\text{wdens}(\sigma, X, r)$. For frequent sets, the definition of frequency has the property of antimonotonicity: if $X \subseteq Y$, then $\text{freq}(X) \geq \text{freq}(Y)$. This property is useful because it enables the Apriori algorithm to prune non-frequent sets. Note that the weakly dense sets do not have this property.

A more fundamental problem with the definition of weakly dense itemsets is the one illustrated in Table 1a: an itemset whose weak density is significantly greater than δ will attract other items that have lower weak density—free-riders. To ensure a more uniform density, we will use minimization over subsets in our stronger definition.

Definition 5. The *density* $\text{dens}(\sigma, X)$ of an itemset X at support level σ is the minimum of the weak densities of all non-empty subsets of X, i.e.,

$$\text{dens}(\sigma, X) = \min_{\emptyset \neq Y \subset X} \text{wdens}(\sigma, Y)$$

An itemset X is *(strongly)* (σ, δ)*-dense*, if $\text{dens}(\sigma, X) \geq \delta$.

Antimonotonicity follows as an immediate corollary from this definition, so dense itemsets can be found by a levelwise approach.

Two questions arise: Is this definition too restrictive? Or is it still too loose? We first consider whether the definition of dense sets is too restrictive. It can be shown that every set X that is weakly but not strongly dense includes a strongly dense subset whose weak density is greater than that of X. In other words, every case where a set is weakly dense but not dense is an instance of the free-rider phenomenon in Table 1a. The role of the weak density threshold δ then is to define the extent to which free-riders are tolerated.

Next we examine a case where the definition might seem to be too loose—see the left-hand side of Table 2. Since the subsets of a dense set are only required to fulfill the weak density criterion *somewhere* in the data, not necessarily all on the same rows, we find that $X = ABCDEF$ is $(4, 0.5)$-dense. Here the connection between the attributes B, C, D, E, and F is the strongest pattern, and A has a weaker connection that actually only occurs outside the main block where the rest of the attributes are found. This apparent discrepancy between dense sets and intuition is caused by the fact that the support level of 4 rows is too low: perhaps the first half of the database contains only noise, but at this resolution the noise is picked up. The right-hand side of the table shows the supports σ at which the listed sets have weak density 0.5: the set $ABCDE$ (like several other sets not shown) has support 4, so if we raise the support threshold above 4, the set $ABCDE$ is no longer dense. The intuitively strong pattern $BCDEF$ is found as the maximal-size (σ, δ)-dense set with e.g. $(\sigma, \delta) = (5, 0.6)$, or $(6, 0.5)$. Thus, the role of the support threshold σ is to define the size of the patterns that we seek—the algorithm's resolution.

A	B	C	D	E	F
1	1	0	0	0	0
1	0	1	0	0	0
1	0	0	1	0	0
1	0	0	0	1	0
1	0	0	0	0	1
0	1	1	1	0	0
0	0	1	1	1	0
0	0	0	1	1	1
0	1	0	0	1	1
0	1	1	0	0	1

set	support
A	10
B	8
AB	9
BC	8
ABC	8
BCD	8
$ABCD$	7
$ABCDE$	4
$BCDEF$	6
$ABCDEF$	5

Table 2: An example database and the supports σ at which the listed sets are weakly $(\sigma, 0.5)$-dense

Next we show an inequality for weak density. Yang et al. showed that all weakly dense itemsets are *accessible* from the empty set by adding single items that keep the intermediate sets weakly dense. [20] Accessibility can be seen as a kind of monotonicity, and it allows weakly dense sets to be found in principle by a levelwise algorithm, whose practical usability is limited by the large number of candidates it needs to consider. The candidates can be pruned to some degree using the following result; however, this does not suffice to make the algorithm usable.

PROPOSITION 1. *If X and Y are itemsets with $X \cap Y = \emptyset$,*

$$\text{wdens}(\sigma, X \cup Y) \leq \frac{|X| \,\text{wdens}(\sigma, X) + |Y| \,\text{wdens}(\sigma, Y)}{|X| + |Y|}.$$

Another application for the inequality is filtering the collection of dense itemsets: if the gap between the upper bound and the actual density of $X \cup Y$ is large, $\{X, Y\}$ may be a better collection of itemsets than $\{X \cup Y\}$. Space considerations preclude further discussion.

3. ALGORITHMS

In this section we give simple algorithms for finding all dense itemsets from large collections of binary data. The algorithms are based on the familiar Apriori idea: for each $h \geq 1$, given dense sets of size h, form candidate sets of size $h+1$, and then do a database pass to verify which candidates indeed satisfy the density condition. We also discuss finding the top-k dense itemsets with respect to density, given the support threshold σ.

The levelwise algorithm

Algorithm 1, DENSE-SETS, performs a levelwise search to find all dense itemsets. The GENERATE-CANDIDATES subroutine is exactly the same as in Apriori [2]. As it is written, Algorithm 1 outputs the weak densities of itemsets, but the strong densities can be obtained by simple post-processing.

Algorithm 1: Find strongly dense itemsets.

Input: A binary database $\mathsf{DB} = \langle R, r \rangle$, a density threshold δ, a support threshold σ
Output: All (σ, δ)-dense itemsets in DB, and their weak densities

```
DENSE-SETS($\mathsf{DB}, \delta, \sigma$)
(1)     $C \leftarrow \{ \{A\} \mid A \in R \}$
(2)     while $C$ is nonempty
(3)         $D \leftarrow$ WEAK-DENSITIES($\mathsf{DB}, C, \sigma$)
(4)         $P \leftarrow \{ X \in C \mid D(X) \geq \delta \}$
(5)         foreach $X \in P$
(6)             print $X$, $D(X)$
(7)         $C \leftarrow$ GENERATE-CANDIDATES($P$)
```

Algorithm 2: Top-k dense itemsets given σ.

Input: A binary database $\mathsf{DB} = \langle R, r \rangle$, a support threshold σ, and a number k.
Output: Some k itemsets in DB such that they are the (σ, δ)-dense itemsets for the given σ and some δ

```
TOP-K-GIVEN-SUPPORT($\mathsf{DB}, \sigma, k$)
(1)       $H \leftarrow$ empty heap
(2)       $F \leftarrow$ empty set-family
(3)       $\delta = \infty$
(4)       $C \leftarrow \{\, \{A\} \mid A \in R \,\}$
(5)       while $k > 0$
(6)           $D \leftarrow$ DENSITIES($\mathsf{DB}, C, \sigma$)
(7)           foreach $X \in C$
(8)               HEAP-PUSH($H, D(X), X$)
(9)           $(d, X) \leftarrow$ HEAP-POP($H$)
(10)          print $X, d$
(11)          $\delta \leftarrow \min(\delta, d)$
(12)          $C \leftarrow$ CANDIDATES($F, X$)
(13)          $F \leftarrow F \cup \{X\}$
(14)          $k \leftarrow k - 1$
(15)      if consistent answer of $\geq k$ sets is wanted
(16)          (run similar loop with fixed $\delta$)
```

The statement $D \leftarrow$ WEAK-DENSITIES(DB, C, σ) assigns to $D(X)$, for all $X \in C$, the weak density of X. We postpone the description of WEAK-DENSITIES for a while.

Finding top k dense sets

Algorithm 1 takes two parameters, a density threshold δ and a support threshold σ, whose roles were discussed at the end of Section 2. Selecting the parameters is not always easy; when mining frequent itemsets, one can first try with a high threshold and gradually lower it until a satisfactory number of sets is found. With two parameters, such continual adjustment becomes more difficult: with too high values, few sets are found, but with too low values, there are so many dense sets that the algorithm takes a prohibitively long time.

However, along the lines of, e.g., [11, 19] there is a way automatically to select one of the two parameters: given the support threshold σ and the number k of itemsets required, we can find the k itemsets that *would have been found* by running Algorithm 1 with the given σ and some δ. This is done by Algorithm 2; an analogous algorithm can be found for the case where δ is given and σ unknown.

This algorithm uses a heap H to store pairs (d, X), where d is the density of X at support σ. The subroutine HEAP-PUSH adds such a pair to the heap, and HEAP-POP removes and returns a pair (d, X) for which d is maximal. The variable F is a family of itemsets that have already been found; in practice, we implement it as a tree structure where the nodes are items, and sets are found by traversing the tree in numerical order of items. The subroutine CANDIDATES(F, X) (omitted) simply enumerates all sets $X \cup \{A\}$ with $A \in R \backslash X$ and returns the collection of those sets whose all subsets are in F.

A detail about Algorithm 2 is that there may not be a family of exactly k sets that are the dense sets given σ and some δ. If exactly k sets are wanted, the algorithm can be stopped at line 15; if a consistent answer is wanted, even though it may include many more than k sets, the algorithm can be continued—the last part is essentially similar to the first part, now with a fixed value of δ.

Database pass

For the database pass we need a way to find the weak density of an itemset at a given support. It turns out that one scan through the database is sufficient, using $O(|X|)$ space for each candidate itemset X.

We next introduce a concept that is used here for efficient computation of the weak density of an itemset.

Definition 6. Given an itemset $X \subseteq R$ with $|X| = \ell$, we define the *intersection profile* H_X as an $(\ell + 1)$-sized vector, indexed by numbers from 0 to ℓ, where $H_X(j)$ is the number of transactions that have *exactly* j attributes in common with X:

$$H_X(j) = \bigl|\bigl\{\, t \in r \bigm| |X \cap t| = j \,\bigr\}\bigr|.$$

To find wdens(σ, X) for an itemset X, it suffices to find the intersection profile H_X and then do some counting. How do we compute the weak density of a set X at support σ? We need to take σ transactions whose intersections with X are maximal. To form the multiset r' of these σ transactions $t \in R$, we simply start from the top: first the $H_X(\ell)$ transactions whose intersection has size $\ell = |X|$, then the next $H_X(\ell - 1)$, etc., until finally adding some $H_X(j)$ tuples would make the subrelation grow to a size $> \sigma$. In this process, the size of the subrelation eventually becomes $\sum_{i=j+1}^{\ell} H_X(i)$, where j is the intersection size at which the subrelation would have grown above σ. Finally, we need to add the remaining transactions, each one having size j. This yields the WEAK-DENSITIES subroutine.

4. EXPERIMENTAL RESULTS

We implemented the above algorithms in Python 2.3 using the kjBuckets library[2] for representing sets, and tested on an Athlon XP 1600+ computer running Linux. The reader should keep in mind that Python is an interpreted scripting language, and an optimized implementation of the algorithm would likely be much faster.

We experimented on the Retail data set [5] obtained from the FIMI web site [10]. The data contains 16470 attributes, out of which we selected only the 1998 most frequent. The data contains 88162 rows (transactions), and we used support thresholds σ ranging from 441 (0.5%) to 8816 (10%). Table 3 shows the number of dense itemsets in the preprocessed data; missing values indicate that the run was stopped because it was taking too long. The results show that there are parameter values for which there is a moderate number of dense itemsets. Table 4 summarizes the running times of our implementation: as noted above, the implementation is not optimized. About 8 minutes are needed just to read and preprocess the data.

Next we consider finding top k dense itemsets using Algorithm 2. The results are shown in Table 5, for $k = 1000$. When searching for the top 1000 sets, one usually obtains some more sets, since no collection of exactly 1000 sets might be a coherent result for a single value of δ: for example, with $\sigma = 8816$, the closest match is a 1007-set collection with $\delta = 0.324$. "Border" here means the number of extra sets considered by the algorithm—sets that have weak density $< \delta$ but whose all immediate subsets are dense. The effect of the border sets is considerable for lower values of σ,

[2]`http://starship.python.net/crew/aaron_watters/kjbuckets/`

σ	0.4	0.5	0.6	0.7	0.8	0.9	1.0
8816	103	55	31	19	15	13	9
4408	3227	547	75	45	31	17	16
2204		8555	391	119	83	53	38
882			2859	965	429	240	159
441					1458	844	580

Table 3: Number of dense itemsets in Retail data; columns=values of δ; rows=values of σ

σ	0.4	0.5	0.6	0.7	0.8	0.9	1.0
8816	10.2	10.0	9.9	9.8	9.4	9.9	9.9
4408	29.9	13.0	10.2	10.2	10.0	10.2	9.9
2204		143.3	12.7	10.9	10.7	10.3	10.2
882			80.7	47.9	33.5	25.7	22.3
441					214.9	164.0	111.5

Table 4: Run-times for Retail data in cpu minutes

σ	sets+border	δ	time/cpumin
8816	1007+ 237	0.324	16.8
4408	1011+ 310	0.445	17.1
2204	1011+ 185	0.549	17.1
882	1004+ 1327	0.697	41.9
441	1001+13041	0.866	157

Table 5: Summary of Top-K-Given-Support in Retail data with $k = 1000$

as can be seen on the last two rows of the table. Of course, the value of k here is too large for most practical uses.

We now move to another data set. The Course data set contains information about course enrollment at the University of Helsinki Computer Science Department. This particular data lists only Masters-level courses, and contains information for 1739 students and 102 courses. Mining dense itemsets with $\sigma = 87$ (i.e., 5%), $\delta = 0.5$ we found 4657 sets.

We used these sets to obtain a novel representation of the data set. Given a dense itemset, we say that its coverage is the number of 1s present in the 87 rows from the data with most attributes within the set. We used a greedy set-cover algorithm: always select the itemset among those found that, when selecting the 87 rows yields the maximal number of 1s not yet covered. The first 8 sets output by this algorithm are shown in Table 6; note that the attributes are listed as rows and the dense itemsets as columns. The densities of these sets vary from 0.52 to 0.58.

The first sets have clear meanings: the first one represents programming (both as a process and as low-level systems knowledge), the second information systems, the third algorithms, etc. An interesting phenomenon is seen in set 8: it is the same as set 1, but lacking the theory course. In the set cover algorithm, set 1 should already have covered those 5% of students who have taken many programming courses, but now such students are found again. The finding could have an interesting interpretation: many students who are studying Computer Science in order to become professional programmers have not yet passed the theory course. Indeed, many students consider the course a difficult one.

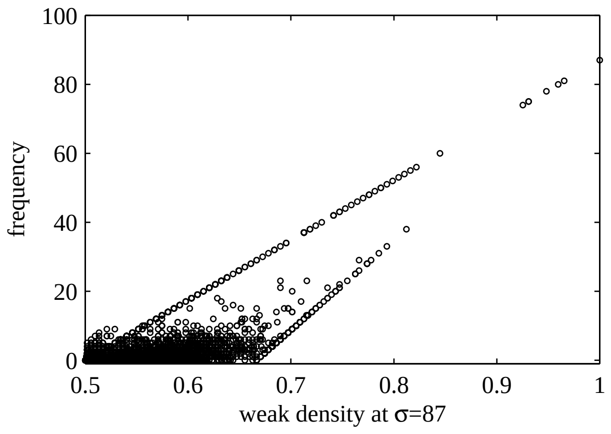

Figure 1: Course data: weak density vs. frequency for $(87, 0.5)$-dense sets X with $|X| \geq 2$

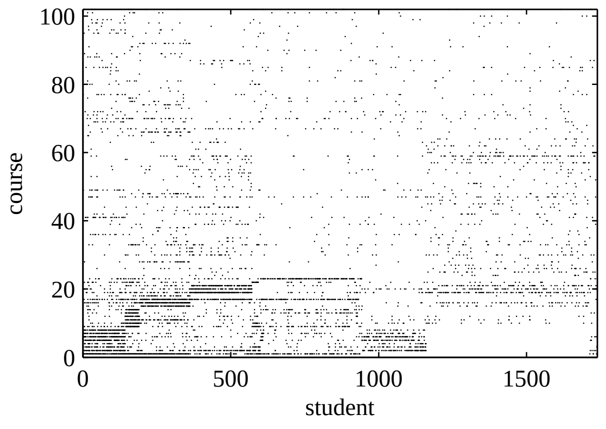

Figure 2: Course data reordered using the 8 itemsets in Table 6.

One of the motivations for the concept of dense itemsets is that the interesting sets will have very low frequency in the classical sense, and would thus not be found by frequent itemset mining. This proved to be the case in the course data: the 8 most useful dense itemsets had frequency 0. As a further illustration, Figure 1 shows the relationship of weak density and frequency for sets with at least two elements. Another motivation is that the family of weakly dense sets is infeasibly large, and this was also confirmed. For example, out of the 4.2 million possible 4-element itemsets in this data, more than 575 thousand had weak density at least 0.5 with $\sigma = 87$. Clearly, finding all weakly dense itemsets even up to size 8 would yield a much larger output than mining the 4657 dense itemsets.

As an illustration of the use of dense itemsets, we also used the greedy approach to reorder the data matrix. We first took the "best" dense itemset (in the coverage sense defined above) and listed its attributes. Then we selected the rows which have at least a frequency of δ for this itemset. The remaining rows and columns are then iteratively ordered using the same method. Figure 2 shows the resulting reordered data matrix.

Dense set	1	2	3	4	5	6	7	8
Theory of Computation	*	*	*	*	*	*	*	
Software Architectures	*			*	*			*
Software Processes	*				*			*
Software Testing	*							*
Data Communication 2	*					*		*
Computer Architecture	*					*		*
Operating Systems 2	*					*		*
Distributed Systems	*					*		*
Structured Documents		*			*		*	
DB Algorithms		*			*			
Data Warehouses		*						
Document Collections		*					*	
Information Extraction		*					*	
Data Mining		*	*					
Algorithm Analysis			*					
Artificial Intelligence			*					
Computer Graphics			*	*		*	*	
String Algorithms			*					
User Interfaces				*				
Distributed OS's				*				
Data Management 2				*				
Database Modeling					*			
Computer-aided Learning						*	*	

Table 6: Set-cover with 8 dense Course itemsets.

5. CONCLUSION

In this paper we have introduced the concept of dense itemsets, and argued that the definition is robust by giving some results on the combinatorial behavior of the density functions. We gave algorithms for finding all or the top-k dense itemsets. The experimental results show that the methods work well in practice. We also demonstrated that one can further select the most interesting dense itemsets using a greedy approach: the application of the method to real datasets shows the usefulness of this technique.

Several open problems remain. Most importantly, we believe that filtering and reordering techniques need to be developed for handling large collections of patterns. The reordering technique of Section 4 is one possibility; another is applying Proposition 1. Related methods include other greedy covering algorithms [12, 15]; matrix ordering and factorization techniques such as NMF [14] and PCA [8]; generalizations of the various subclasses of frequent itemsets [4, 7, 16]; and selection using ROC curves [18]. We are also investigating a generalization of Definition 6 to "external intersection counts", measuring not only how many items of a given set but also how many items of its complement appear in a transaction. These counts lead to filtering techniques reminiscent of segmentation problems [13] and tilings [9]. Also of interest are applications to topic models [3] and query approximation [6].

6. ACKNOWLEDGMENTS

We thank Mohammed Zaki, Aristides Gionis, and Jaakko Hollmén for enlightening discussions, Taneli Mielikäinen for bringing the Pei et al. paper [17] to our attention, and Tom Brijs for donating the Retail data set [5].

7. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93*, pages 207–216, 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI Press, 1996.

[3] E. Bingham, H. Mannila, and J. K. Seppänen. Topics in 0-1 data. In D. Hand, D. Keim, and R. Ng, editors, *Proc. KDD–2002*, pages 450–455, 2002.

[4] J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Approximation of frequency queries by means of free-sets. In *PKDD–2000*, volume 1910 of *LNCS*, pages 75–85. Springer, 2000.

[5] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Knowledge Discovery and Data Mining*, pages 254–260, 1999.

[6] A. Bykowski, J. K. Seppänen, and J. Hollmén. Model-independent bounding of Boolean formulae in binary data. In M. Klemettinen, R. Meo, F. Giannotti, and L. De Raedt, editors, *Knowledge Discovery in Inductive Databases (KDID–02), First International Workshop*, pages 20–31, Helsinki, Finland, 2002. University of Helsinki Department of Computer Science Report B–2002–7.

[7] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Proc. PKDD–2002*, volume 2431 of *LNCS*, pages 74–85. Springer-Verlag, 2002.

[8] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[9] F. Geerts, B. Goethals, and T. Mielikäinen. Mining tiles and tilings. Manuscript in preparation.

[10] B. Goethals and M. J. Zaki, editors. *Proc. Workshop on Frequent Itemset Mining Implementations (FIMI–03)*, volume 90 of *CEUR-WS*, Melbourne, Florida, 2003. `http://CEUR-WS.org/Vol-90/`.

[11] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *ICDM 2002*, pages 211–218, 2002.

[12] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999. doi:10.1016/S0020-0190(99)00031-9.

[13] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Segmentation problems. *J. ACM*, 51(2):263–280, 2004.

[14] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, 2000.

[15] T. Mielikäinen and H. Mannila. The pattern ordering problem. In N. Lavrač, D. Gamberger, L. Todorovski, and H. Blockeel, editors, *Proc. PKDD–2003*, volume 2383 of *LNAI*, pages 327–338. Springer, 2003.

[16] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *LNCS*, 1540:398–416, 1999.

[17] J. Pei, A. K. Tung, and J. Han. Fault-tolerant frequent pattern mining: Problems and challenges. In *Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[18] J. A. Swets. Measuring the accuracy of diagnostic systems. *Science*, 240(4857):1285–93, June 1988.

[19] P. Tzvetkov, X. Yan, and J. Han. TSP: Mining top-k closed sequential patterns. In *ICDM 2003*, pages 347–354, 2003.

[20] C. Yang, U. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. KDD–2001*, pages 194–203. ACM Press, 2001.