Item Set Mining Based on Cover Similarity

Marc Segond and Christian Borgelt

European Centre for Soft Computing
Calle Gonzalo Gutiérrez Quirós s/n, E-33600 Mieres (Asturias), Spain
{marc.segond,christian.borgelt}@softcomputing.es

Abstract. While in standard frequent item set mining one tries to find item sets the support of which exceeds a user-specified threshold (minimum support) in a database of transactions, we strive to find item sets for which the similarity of their covers (that is, the sets of transactions containing them) exceeds a user-specified threshold. Starting from the generalized Jaccard index we extend our approach to a total of twelve specific similarity measures and a generalized form. We present an efficient mining algorithm that is inspired by the well-known Eclat algorithm and its improvements. By reporting experiments on several benchmark data sets we demonstrate that the runtime penalty incurred by the more complex (but also more informative) item set assessment is bearable and that the approach yields high quality and more useful item sets.

1 Introduction

Frequent item set mining and association rule induction are among the most intensely studied topics in data mining and knowledge discovery in databases. The enormous research efforts devoted to these tasks have led to a variety of sophisticated and efficient algorithms, among the best-known of which are Apriori [1], Eclat [27, 28] and FP-growth [13]. However, these approaches, which find item sets whose support exceeds a user-specified minimum in a given transaction database, have the disadvantage that the support does not say much about the actual strength of association of the items in the set: a set of items may be frequent simply because its elements are frequent and thus their frequent co-occurrence can even be expected by chance. As a consequence, the (usually few) interesting item sets drown in a sea of irrelevant ones.

In order to improve this situation, we propose in this paper to change the selection criterion, so that fewer irrelevant items sets are produced. For this we draw on the insight that for associated items their covers—that is, the set of transactions containing them—are more similar than for independent items. Starting from the Jaccard index to illustrate this idea, we explore a total of twelve specific similarity measures that can be generalized from pairs of sets (or, equivalently, from pairs of binary vectors) as well as a generalized form. By applying an Eclat-based mining algorithm to standard benchmark data sets and to the 2008/2009 Wikipedia Selection for schools, we demonstrate that the search times are bearable and that high quality item sets are produced.

2 Frequent Item Set Mining

Frequent item set mining was originally developed for market basket analysis, aiming at finding regularities in the shopping behavior of the customers of supermarkets, mail-order companies and online shops. Formally, we are given a set B of *items*, called the *item base*, and a database T of *transactions*. Each item represents a product, and the item base represents the set of all products on offer. The term *item set* refers to any subset of the item base B. Each transaction is an item set and represents a set of products that has been bought by an actual customer. Note that two or even more customers may have bought the exact same set of products. Note also that the item base B is usually not given explicitly, but only implicitly as the union of all transactions.

We write $T = (t_1, \ldots, t_n)$ for a database with n transactions, thus distinguishing equal transactions by their position in the vector. In order to refer to the index set, we introduce the abbreviation $\mathbb{N}_n := \{k \in \mathbb{N} \mid k \leq n\} = \{1, \ldots, n\}$. Given an item set $I \subseteq B$ and a transaction database T, the *cover* $K_T(I)$ of I w.r.t. T is defined as $K_T(I) = \{k \in \mathbb{N}_n \mid I \subseteq t_k\}$, that is, as the set of indices of transactions that contain I. The *support* $s_T(I)$ of an item set $I \subseteq B$ is the number of transactions in the database T it is contained in, that is, $s_T(I) = |K_T(I)|$. Given a user-specified *minimum support* $s_{\min} \in \mathbb{N}$, an item set I is called *frequent* in T iff $s_T(I) \geq s_{\min}$. The goal of frequent item set mining is to identify all item sets $I \subseteq B$ that are frequent in a given transaction database T.

A standard approach to find all frequent item sets w.r.t. a given database T and a support threshold $s_{\min}$, which is adopted by basically all frequent item set mining algorithms (except those of the Apriori family), is a *depth-first search* in the subset lattice of the item base B. Viewed properly, this approach can be interpreted as a simple *divide-and-conquer* scheme. All subproblems that occur in this scheme can be defined by a *conditional transaction database* and a *prefix*. The prefix is a set of items that has to be added to all frequent item sets that are discovered in the conditional database, from which all items in the prefix have been removed. Formally, all subproblems are tuples $S = (T_C, P)$, where T_C is a conditional transaction database and $P \subseteq B$ is a prefix. The initial problem, with which the recursion is started, is $S = (T, \emptyset)$, where T is the given transaction database to mine and the prefix is empty. A subproblem $S_0 = (T_0, P_0)$ is processed as follows: Choose an item $i \in B_0$, where B_0 is the set of items occurring in T_0. This choice is arbitrary, but usually follows some predefined order of the items. If $s_{T_0}(i) \geq s_{\min}$, then report the item set $P_0 \cup \{i\}$ as frequent with the support $s_{T_0}(i)$, and form the subproblem $S_1 = (T_1, P_1)$ with $P_1 = P_0 \cup \{i\}$. The conditional transaction database T_1 comprises all transactions in T_0 that contain the item i, but with the item i removed. This also implies that transactions that contain no other item than i are entirely removed: no empty transactions are ever kept. If T_1 is not empty, process S_1 recursively. In any case (that is, regardless of whether $s_{T_0}(i) \geq s_{\min}$ or not), form the subproblem $S_2 = (T_2, P_2)$, where $P_2 = P_0$ and the conditional transaction database T_2 comprises all transactions in T_0 (including those that do not contain the item i), but again with the item i removed. If T_2 is not empty, process S_2 recursively.

3 Jaccard Item Sets

We base our item set mining approach on the similarity of item covers rather than on item set support. In order to measure the similarity of a set of item covers, we start with the Jaccard index [16], which is a well-known statistic for comparing sets. For two arbitrary sets A and B it is defined as $J(A,B) = |A \cap B|/|A \cup B|$. Obviously, $J(A,B)$ is 1 if the sets coincide (i.e. $A = B$) and 0 if they are disjoint (i.e. $A \cap B = \emptyset$). For overlapping sets its value lies between 0 and 1. The core idea of using the Jaccard index for item set mining lies in the insight that the covers of (positively) associated items are likely to have a high Jaccard index, while a low Jaccard index indicates independent or even negatively associated items. However, since we consider also item sets with more than two items, we need a generalization to more than two sets (here: item covers). In order to achieve this, we define the *carrier* $L_T(I)$ of an item set I w.r.t. a transaction database T as

$$L_T(I) = \{k \in \mathbb{N}_n \mid I \cap t_k \neq \emptyset\} = \{k \in \mathbb{N}_n \mid \exists i \in I : i \in t_k\} = \textstyle\bigcup_{i \in I} K_T(\{i\}).$$

The *extent* $r_T(I)$ of an item set I w.r.t. a transaction database T is the size of its carrier, that is, $r_T(I) = |L_T(I)|$. Together with the notions of *cover* and *support* (see above), we can define the generalized Jaccard index of an item set I w.r.t. a transaction database T as its support divided by its extent, that is, as

$$J_T(I) \;=\; \frac{s_T(I)}{r_T(I)} \;=\; \frac{|K_T(I)|}{|L_T(I)|} \;=\; \frac{|\bigcap_{i \in I} K_T(\{i\})|}{|\bigcup_{i \in I} K_T(\{i\})|}.$$

Clearly, this is a very natural and straightforward generalization of the Jaccard index. Since for an arbitrary item $a \in B$ it is obviously $K_T(I \cup \{a\}) \subseteq K_T(I)$ and equally obviously $L_T(I \cup \{a\}) \supseteq L_T(I)$, we have $s_T(I \cup \{a\}) \leq s_T(I)$ and $r_T(I \cup \{a\}) \geq r_T(I)$. From these two relations it follows $J_T(I \cup \{a\}) \leq J_T(I)$ and thus that the generalized Jaccard index w.r.t. a transaction database T over an item base B is an anti-monotone function on the partially ordered set $(2^B, \subseteq)$.

Given a user-specified minimum Jaccard value $J_{\min}$, an item set I is called *Jaccard-frequent* if $J_T(I) \geq J_{\min}$. The goal of Jaccard item set mining is to identify all item sets that are Jaccard-frequent in a given transaction database T. Since the generalized Jaccard index is anti-monotone, this task can be addressed with the same basic scheme as the task of frequent item set mining. The only problem to be solved is to find an efficient scheme for computing the extent $r_T(I)$.

4 The Eclat Algorithm

Since we will draw on the basic scheme of the well-known Eclat algorithm for mining Jaccard item sets, we briefly review some of its core ideas. Eclat [27] uses a purely vertical representation of conditional transaction databases, that is, it uses lists of transaction indices, which represent the cover of an item or an item set. It then exploits the obvious relation $K_T(I \cup \{a,b\}) = K_T(I \cup \{a\}) \cap K_T(I \cup \{b\})$, which allows to extend an item set by an item. This is used in the recursive

divide-and-conquer scheme described above by intersecting the list of transaction indices associated with the split item with the lists of transaction indices of all items that have not yet been considered in the recursion.

An alternative to the intersection approach, which is particularly useful for mining dense transaction databases, relies on so-called *difference sets* (or *diffsets* for short) [28]. The diffset $D_T(a \mid I)$ of an item a w.r.t. an item set I and a transaction database T is defined as $D_T(a \mid I) = K_T(I) - K_T(I \cup \{a\})$. That is, a diffset $D_T(a \mid I)$ lists the indices of all transactions that contain I, but not a. Since $s_T(I \cup \{a\}) = s_T(I) - |D_T(a \mid I)|$, diffsets are equally effective for finding frequent item sets, provided one can derive a formula that allows to compute diffsets with a larger conditional item set I without going through covers (using the above definition of a diffset). However, this is easily achieved, because $D_T(b \mid I \cup \{a\}) = D_T(b \mid I) - D_T(a \mid I)$ [28]. This formula allows to formulate the search entirely with the help of diffsets.

5 The JIM Algorithm (Jaccard Item Set Mining)

The diffset approach as it was reviewed in the previous section can easily be transferred in order to find an efficient scheme for computing the *carrier* and thus the *extent* of item sets. To this end we define the *extra set* $E_T(a \mid I)$ as

$$E_T(a \mid I) = K_T(\{a\}) - \textstyle\bigcup_{i \in I} K_T(\{i\}) = \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k\}.$$

That is, $E_T(a \mid I)$ is the set of indices of all transactions that contain a, but no item in I, and thus identifies the extra transaction indices that have to be added to the carrier if item a is added to the item set I. For extra sets we have $E_T(a \mid I \cup \{b\}) = E_T(a \mid I) - E_T(b \mid I)$, which corresponds to the analogous formula for diffsets reviewed above. This relation is easily verified as follows:

$$\begin{aligned}
&E_T(a \mid I) - E_T(b \mid I) \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k\} - \{k \in \mathbb{N}_n \mid b \in t_k \wedge \forall i \in I : i \notin t_k\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k \wedge \neg(b \in t_k \wedge \forall i \in I : i \notin t_k)\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k \wedge (b \notin t_k \vee \exists i \in I : i \in t_k)\} \\
&= \{k \in \mathbb{N}_n \mid (a \in t_k \wedge \forall i \in I : i \notin t_k \wedge b \notin t_k) \\
&\qquad\qquad \vee \underbrace{(a \in t_k \wedge \forall i \in I : i \notin t_k \wedge \exists i \in I : i \in t_k)}_{=\text{false}}\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k \wedge b \notin t_k\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I \cup \{b\} : i \notin t_k\} \\
&= E_T(a \mid I \cup \{b\})
\end{aligned}$$

In order to see how extra sets can be used to compute the extent of item sets, let $I = \{i_1, \ldots, i_m\}$, with some arbitrary, but fixed order of the items that is indicated by the index. This will be the order in which the items are used as

quantity	behavior
n_T	constant
$s_T(I) = \lvert K_T(I)\rvert = \left\lvert \bigcap_{i \in I} K_T(\{i\})\right\rvert$	anti-monotone
$r_T(I) = \lvert L_T(I)\rvert = \left\lvert \bigcup_{i \in I} K_T(\{i\})\right\rvert$	monotone
$q_T(I) = r_T(I) - s_T(I)$	monotone
$z_T(I) = n_T \quad - r_T(I)$	anti-monotone

Table 1. Quantities in terms of which the considered similarity measures are specified, together with their behavior as functions on the partially ordered set $(2^B, \subseteq)$.

split items in the recursive divide-and-conquer scheme. It is

$$\begin{aligned} L_T(I) &= \textstyle\bigcup_{k=1}^{m} K_T(\{i_k\}) \;=\; \bigcup_{k=1}^{m} \left(K_T(\{i_k\}) - \bigcup_{l=1}^{k-1} K_T(\{i_l\})\right) \\ &= \textstyle\bigcup_{k=1}^{m} E(i_k \mid \{i_1, \ldots, i_{k-1}\}), \end{aligned}$$

and since the terms of the last union are clearly all disjoint, we have immediately

$$r_T(I) = \sum_{k=1}^{m} |E(i_k \mid \{i_1, \ldots, i_{k-1}\})| = r_T(I - \{i_m\}) + |E(i_m \mid I - \{i_m\})|.$$

Thus we have a simple recursive scheme to compute the extent of an item set from its parent in the search tree (as defined by the divide-and-conquer scheme).

The mining algorithm can now easily be implemented as follows: initially we create a vertical representation of the given transaction database. The only difference to the Eclat algorithm is that we have two transaction lists per item i: one represents $K_T(\{i\})$ and the other $E_T(i \mid \emptyset)$, which happens to be equal to $K_T(\{i\})$. (That is, for the initial transaction database the two lists are identical, which, however, will obviously not be maintained in the recursive processing.) In the recursion the first list for the split item is intersected with the first list of all other items to form the list representing the cover of the corresponding pair. The second list of the split item is subtracted from the second lists of all other items, thus yielding the extra sets of transactions for these items given the split item. From the sizes of the resulting lists the support and the extent of the enlarged item sets and thus their generalized Jaccard index can be computed.

6 Other Similarity Measures

Up to now we focused on the (generalized) Jaccard index to measure the similarity of sets (covers). However, there is a large number of alternatives. Recent extensive overviews for the pairwise case include [5] and [6].

The JIM algorithm (as presented above) allows us to easily compute the quantities listed in Table 1. With these quantities a wide range of similarity measures for sets or binary vectors can be generalized. Exceptions are those measures that refer explicitly to the number of cases in which a vector $\boldsymbol{x}$ is 1 while the other vector $\boldsymbol{y}$ is 0, and distinguish this number from the number of cases in which $\boldsymbol{y}$ is 1 and $\boldsymbol{x}$ is 0. This distinction is difficult to generalize

Measures derived from inner product:

Russel & Rao [21]	$S_{\mathrm{R}} = \frac{s}{n} = \frac{s}{r+z}$
Kulczynski [19]	$S_{\mathrm{K}} = \frac{s}{q} = \frac{s}{r-s}$
Jaccard [16] Tanimoto [26]	$S_{\mathrm{J}} = \frac{s}{s+q} = \frac{s}{r}$
Dice [8] Sørensen [25] Czekanowski [7]	$S_{\mathrm{D}} = \frac{2s}{2s+q} = \frac{2s}{r+s}$
Sokal & Sneath 1 [24, 22]	$S_{\mathrm{S}} = \frac{s}{s+2q} = \frac{s}{r+q}$

Measures derived from Hamming distance:

Sokal & Michener Hamming [23, 15]	$S_{\mathrm{M}} = \frac{s+z}{n} = \frac{n-q}{n}$
Faith [10]	$S_{\mathrm{F}} = \frac{2s+z}{2n} = \frac{s+\frac{1}{2}z}{n}$
AZZOO [5] $\sigma \in [0,1]$	$S_{\mathrm{Z}} = \frac{s+\sigma z}{n}$
Rogers & Tanimoto [20]	$S_{\mathrm{T}} = \frac{s+z}{n+q} = \frac{n-q}{n+q}$
Sokal & Sneath 2 [24, 22]	$S_{\mathrm{N}} = \frac{2(s+z)}{n+s+z} = \frac{n-q}{n-\frac{1}{2}q}$
Sokal & Sneath 3 [24, 22]	$S_{\mathrm{O}} = \frac{s+z}{q} = \frac{n-q}{q}$
Baroni-Urbani & Buser [3]	$S_{\mathrm{B}} = \frac{\sqrt{sz}+s}{\sqrt{sz}+r}$

Table 2. Considered similarity measures for sets/binary vectors.

beyond the pairwise case, because the number of possible assignments of zeros and ones to the different vectors, each of which one would have to consider for a generalization, grows exponentially with the number of these vectors (here: covers, and thus: items) and therefore becomes quickly infeasible.

By collecting from [6] similarity measures that are specified in terms of the quantities listed in Table 1, we compiled Table 2. Note that the index T and the argument I are omitted to make the formulas more easily readable. Note also that the Hamann measure $S_{\mathrm{H}} = \frac{x+z-s}{n} = \frac{n-2s}{n}$ [14] listed in [6] is equivalent to the Sokal & Michener measure S_{M}, because $S_{\mathrm{H}} + 1 = 2S_{\mathrm{M}}$, and hence omitted. Likewise, the second Baroni-Urbani & Buser measure $S_{\mathrm{U}} = \frac{\sqrt{xz}+x-q}{\sqrt{xz}+o}$ listed in [6] is equivalent to the one given in Table 2, because $S_{\mathrm{U}} + 1 = 2S_{\mathrm{B}}$. Finally, note that all of the measures listed in Table 2 have range $[0, 1]$ except S_{K} (Kulczynski) and S_{O} (Sokal & Sneath 3), which have range $[0, \infty)$.

Table 2 is split into two parts depending on whether the numerator of a measure refers only to the support s or to both the support s and the number z of transactions that do not contain any of the items in the considered set. The former are often referred to as based on the inner product, because in the pairwise case s is the value of the inner (or scalar) product of the binary vectors that are compared. The latter measures (that is, those referring to both s and z) are referred to as based on the Hamming distance, because in the pairwise case q is the Hamming distance of the two vectors and $n - q = s + z$ their Hamming similarity. The decision whether for a given application the term z should be considered in the numerator of a similarity measure or not is difficult. Discussions of this issue for the pairwise case can be found in [22] and [9].

Note that the Russel & Rao measure is simply normalized support, demonstrating that our framework comprises standard frequent item set mining as a

special case. The Sokal&Michener measure is simply the normalized Hamming similarity. The Dice/Sørensen/Czekanowski measure may be defined without the factor 2 in the numerator, changing the range to $[0, 0.5]$. The Faith measure is equivalent to the AZZOO measure (alter zero zero one one) for $\sigma = 0.5$ and the Sokal&Michener measure results for $\sigma = 1$. AZZOO is meant to introduce flexibility in how much weight should be placed on z, the number of transactions which lack all items in I (zero zero) relative to s (one one).

All measures listed in Table 2 are anti-monotone on the partially ordered set $(2^B, \subseteq)$, where B is the underlying item base. This is obvious if in at least one of the formulas given for a measure the numerator is (a multiple of) a constant or anti-monotone quantity or a (weighted) sum of such quantities, and the numerator is (a multiple of) a constant or monotone quantity or a (weighted) sum of such quantities (see Table 1). This is the case for all but S_{D}, S_{N} and S_{B}.

That S_{D} is anti-monotone can be seen by considering its reciprocal value $S_{\mathrm{D}}^{-1} = \frac{2s+q}{2s} = 1 + \frac{q}{2s}$. Since q is monotone and s is anti-monotone, S_{D}^{-1} is clearly monotone and thus S_{D} is anti-monotone. Applying the same approach to S_{B}, we arrive at $S_{\mathrm{B}}^{-1} = \frac{\sqrt{sz}+r}{\sqrt{sz}+s} = \frac{\sqrt{sz}+s+q}{\sqrt{sz}+s} = 1 + \frac{q}{\sqrt{sz}+s}$. Since q is monotone and both s and $\sqrt{sz}$ are anti-monotone, S_{B}^{-1} is clearly monotone and thus S_{B} is anti-monotone. Finally, S_{N} can be written as $S_{\mathrm{N}} = \frac{2n-2q}{2n-q} = 1 - \frac{q}{2n-q} = 1 - \frac{q}{n+s+z}$. Since q is monotone, the numerator is monotone, and since n is constant and s and z are anti-monotone, the denominator is anti-monotone. Hence the fraction is monotone and since it is subtracted from 1, S_{N} is anti-monotone.

Note that all measures in Table 2 can be expressed as

$$S = \frac{c_0 s + c_1 z + c_2 n + c_3 \sqrt{sz}}{c_4 s + c_5 z + c_6 n + c_7 \sqrt{sz}} \tag{1}$$

by specifying appropriate coefficients $c_0, \ldots, c_7$. For example, we obtain S_{J} for $c_0 = c_6 = 1$, $c_5 = -1$ and $c_1 = c_2 = c_3 = c_4 = c_7 = 0$, since $S_{\mathrm{J}} = \frac{s}{r} = \frac{s}{n-z}$. Similarly, we obtain S_{O} for $c_0 = c_1 = c_6 = 1$, $c_4 = c_5 = -1$ and $c_2 = c_3 = c_7 = 0$, since $S_{\mathrm{O}} = \frac{s+z}{q} = \frac{s+z}{n-s-z}$. This general form allows for a flexible specification of various similarity measures. Note, however, that not all selections of coefficients lead to an anti-monotone measure and hence one has to carefully check this property before using a measure that differs from the pre-specified ones.

7 Experiments

We implemented the described item set mining approach as a C program that was derived from an Eclat implementation by adding the second transaction identifier list for computing the extent of item sets. All similarity measures listed in Table 2 are included as well as the general form (1). This implementation has been made publicly available under the GNU Lesser (Library) Public License.[1]

In a first set of experiments we applied the program to five standard benchmark data sets, which exhibit different characteristics, and compared it to a

[1] See `http://www.borgelt.net/jim.html`

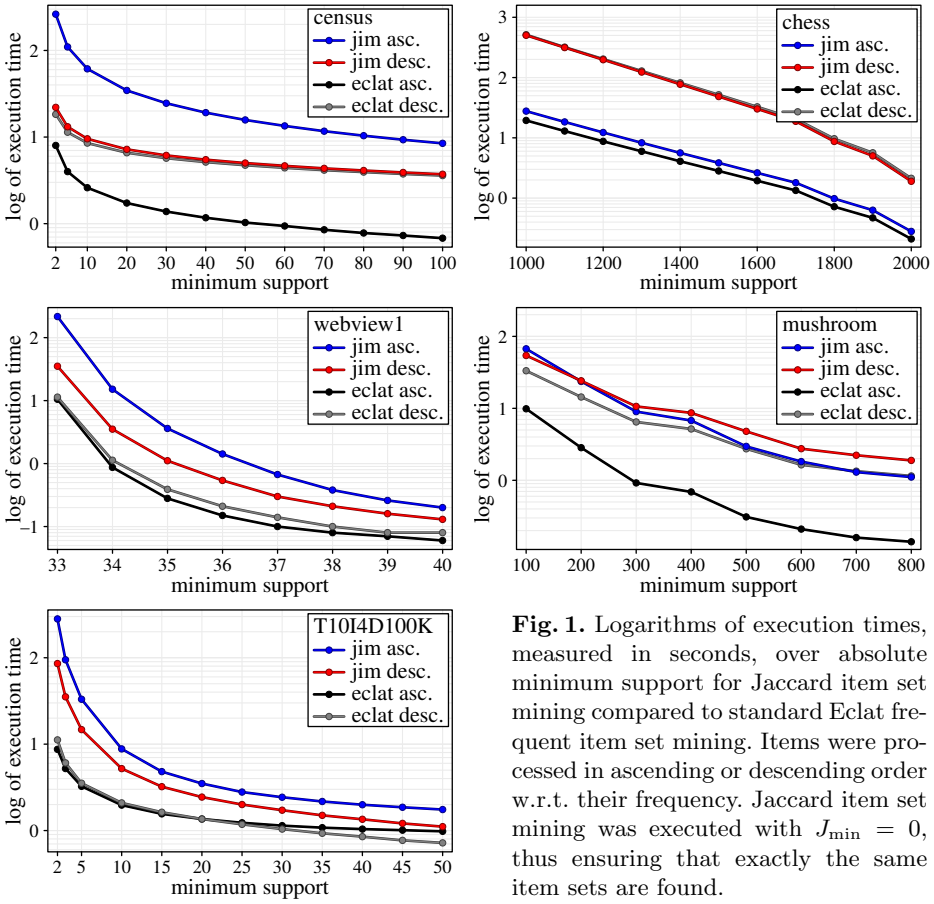

Fig. 1. Logarithms of execution times, measured in seconds, over absolute minimum support for Jaccard item set mining compared to standard Eclat frequent item set mining. Items were processed in ascending or descending order w.r.t. their frequency. Jaccard item set mining was executed with $J_{\min} = 0$, thus ensuring that exactly the same item sets are found.

standard Eclat search. We used BMS-Webview-1 (a web click stream from a leg-care company that no longer exists, which has been used in the KDD cup 2000 [17]), T10I4D100K (an artificial data set generated with IBM's data generator [29]), census (a data set derived from an extract of the US census bureau data of 1994, which was preprocessed by discretizing numeric attributes), chess (a data set listing chess end game positions for king vs. king and rook), and mushroom (a data set describing poisonous and edible mushrooms by different attributes). The first two data sets are available in the FIMI repository [11], the last three in the UCI machine learning repository [2]. The discretization of the numeric attributes in the census data set was done with a shell/gawk script that can be found on the web page given in footnote 1 (previous page). For the experiments we used an Intel Core 2 Quad Q9650 (3GHz) machine with 8 GB main memory running Ubuntu Linux 10.4 (64 bit) and gcc version 4.4.3.

The goal of these experiments was to determine how much the computation of the carrier/extent of an item set affected the execution time. Therefore we ran the JIM algorithm with $J_{\min} = 0$, using only a minimum support threshold.

As a consequence, JIM and Eclat always found exactly the same set of frequent item sets and any difference in execution time comes from the additional costs of the carrier/extent computation. In addition, we checked which item order (ascending or descending w.r.t. their frequency) yields the shortest search times.

The results are depicted in the diagrams in Figure 1. We observe that processing the items in increasing order of frequency always works better for Eclat (black and grey curves)—as expected. For JIM, however, the best order depends on the data set: on census, BMS-Webview-1 and T10I4D100K descending order is better (red curve is lower than blue), on chess ascending order is better (blue curve is lower than red), while on mushroom it depends on the minimum support which order yields the shorter time (red curve intersects blue).

We interpret these findings as follows: for the support computation (which is all Eclat does) it is better to process the items in ascending order, because this reduces the average length of the transaction id lists. By intersecting with short lists early, the lists processed in the recursion tend to be shorter and thus are processed faster. However, for the extent computation the opposite order is preferable. Since it works on extra sets, it is advantageous to add frequent items as early as possible to the carrier, because this increases the size of the already covered carrier and thus reduces the average length of the extra lists. Therefore, since there are different preferences, it depends on the data set which operation governs the complexity and thus which item order is better.

From Figure 1 we conjecture that dense data sets (high fraction of ones in a bit matrix representation), like chess and mushroom, favor ascending order, while sparse data sets, like census, BMS-Webview-1 and T10I4D100K, favor descending order. This is plausible, because in dense data sets intersection lists tend to be long, so it is important to reduce them. In sparse data sets, however, extra lists tend to be long, so here it is more important to focus on them.

Naturally, the execution times of JIM are always greater than those of the corresponding Eclat runs (with the same order of the items), but the execution times are still bearable. This shows that even if one does *not* use a similarity measure to *prune* the search, this additional information can be computed fairly efficiently. However, it should be kept in mind that the idea of the approach is to set a threshold for the similarity measure, which can effectively prune the search, so that the actual execution times found in applications are much lower. In our own practice we basically always achieved execution times that were lower than for the Eclat algorithm (but, of course, with a different output).

In another experiment we used an extract from the 2008/2009 Wikipedia Selection for schools[2], which consisted of 4861 web pages. Each of these web pages was taken as a transaction and processed with standard text processing methods (name detection, stemming, stop word removal etc.) to extract a total of 59330 terms/keywords. The terms occurring on a web page are the items occurring in the corresponding transaction. The resulting data file was then mined for Jaccard item sets with a threshold of $J_{\min} = 0.1$. Some examples of found term associations are listed in Table 3.

[2] See `http://schools-wikipedia.org/`

Table 3. Jaccard item sets found in the 2008/2009 Wikipedia Selection for schools.

item set	s_T	J_T
Reptiles, Insects	12	1.0000
phylum, chordata, animalia	34	0.7391
planta, magnoliopsida, magnoliophyta	14	0.6667
wind, damag, storm, hurrican, landfal	23	0.1608
tournament, doubl, tenni, slam, Grand Slam	10	0.1370
dinosaur, cretac, superord, sauropsida, dinosauria	10	0.1149
decai, alpha, fusion, target, excit, dubna	12	0.1121
conserv, binomi, phylum, concern, animalia, chordata	14	0.1053

Clearly, there are several term sets with surprisingly high Jaccard indices and thus strongly associated terms. For example, "Reptiles" and "Insects" always appear together (on a total of 12 web pages) and never alone. A closer inspection revealed, however, that this is an artifact of the name detection, which extracts these terms from the Wikipedia category title "Insects, Reptiles and Fish" (but somehow treats "Fish" not as a name, but as a normal word). All other item sets contain normal terms, though (only "Grand Slam" is another name), and are no artifacts of the text processing step. The second item set captures several biology pages, which describe different vertebrates, all of which belong to the phylum "chordata" and the kingdom "animalia". The third set indicates that this selection contains a surprisingly high number of pages referring to magnolias. The remaining item sets show that term sets with five or even six terms can exhibit a quite high Jaccard index, even though they have a fairly low support.

An impression of the filtering power can be obtained by comparing the size of the output to standard frequent item set mining: for $s_{\min} = 10$ there are 83130 frequent item sets and 19394 closed item sets with at least two items. A threshold of $J_{\min} = 0.1$ for the (generalized) Jaccard index reduces the output to 5116 (frequent) item sets. From manual inspection, we gathered the impression that the Jaccard item sets contained more meaningful sets and that the Jaccard index was a valuable additional piece of information. It has to be conceded, though, that whether item sets are more "meaningful" or "interesting" is difficult to assess, because this requires an objective measure, which is not available.

However, the usefulness of our method is indirectly supported by a successful application of the Jaccard item set mining approach for concept detection, for which standard frequent item set mining did not yield sufficiently good results. This was carried out in the EU FP7 project BISON[3] and is reported in [18].

8 Conclusions

We introduced the notion of a Jaccard item set as an item set for which the (generalized) Jaccard index of its item covers exceeds a user-specified threshold.

[3] See `http://www.bisonet.eu/`

In addition, we extended this basic idea to a total of twelve similarity measures for sets or binary vectors, all of which can be generalized in the same way and can be shown to be anti-monotone. By exploiting an idea that is similar to the difference set approach for the well-known Eclat algorithm, we derived an efficient search scheme that is based on forming intersections and differences of sets of transaction indices in order to compute the quantities that are needed to compute the similarity measures. Since it contains standard frequent item set mining as a special case, mining item sets based on cover similarity yields a flexible and versatile framework. Furthermore, the similarity measures provide highly useful additional assessments of found item sets and thus help us to select the interesting ones. By running experiments on standard benchmark data sets we showed that mining item sets based on cover similarity can be done fairly efficiently, and by evaluating the results obtained with a threshold for the cover similarity measure we demonstrated that the output is considerably reduced, while expressive and meaningful item sets are preserved.

Acknowledgements

This work was supported by the European Commission under the 7th Framework Program FP7-ICT-2007-C FET-Open, contract no. BISON-211898.

References

1. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. *Proc. 20th Int. Conf. on Very Large Databases (VLDB 1994, Santiago de Chile)*, 487–499. Morgan Kaufmann, San Mateo, CA, USA 1994
2. A. Asuncion and D.J. Newman. *UCI Machine Learning Repository*. School of Information and Computer Science, University of California at Irvine, CA, USA 2007. `http://www.ics.uci.edu/~mlearn/MLRepository.html`
3. C. Baroni-Urbani and M.W. Buser. Similarity of Binary Data. *Systematic Zoology* 25(3):251–259. Oxford University Press, Oxford, United Kingdom 1976
4. R. Bayardo, B. Goethals, and M.J. Zaki, eds. *Proc. Workshop Frequent Item Set Mining Implementations (FIMI 2004, Brighton, UK)*. CEUR Workshop Proceedings 126, Aachen, Germany 2004 `http://www.ceur-ws.org/Vol-126/`
5. S.-H. Cha, C.C. Tappert, and S. Yoon. Enhancing Binary Feature Vector Similarity Measures. *J. Pattern Recognition Research* 1:63–77. San Diego, CA, USA 2006
6. S.-S. Choi, S.-H. Cha, and C.C. Tappert. A Survey of Binary Similarity and Distance Measures. *Journal of Systemics, Cybernetics and Informatics* 8(1):43–48. Int. Inst. of Informatics and Systemics, Caracas, Venezuela 2010
7. J. Czekanowski. Zarys metod statystycznych w zastosowaniu do antropologii [An Outline of Statistical Methods Applied in Anthropology]. Towarzystwo Naukowe Warszawskie, Warsaw, Poland 1913
8. L.R. Dice. Measures of the Amount of Ecologic Association between Species. *Ecology* 26:297–302. Ecological Society of America, Ithaca, NY, USA 1945
9. G. Dunn and B.S. Everitt. *An Introduction to Mathematical Taxonomy*. Cambridge University Press, Cambirdge, United Kingdom 1982
10. D.P. Faith. Asymmetric Binary Similarity Measures. *Oecologia* 57(3):287–290. Springer, Berlin, Germany 1983

11. B. Goethals, ed. *Frequent Item Set Mining Dataset Repository.* University of Helsinki, Finland 2004 `http://fimi.cs.helsinki.fi/data/`
12. B. Goethals and M.J. Zaki, eds. *Proc. Workshop Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL, USA).* CEUR Workshop Proceedings 90, Aachen, Germany 2003 `http://www.ceur-ws.org/Vol-90/`
13. J. Han, H. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX)*, 1–12. ACM Press, New York, NY, USA 2000
14. V. Hamann. Merkmalbestand und Verwandtschaftsbeziehungen der Farinosae. Ein Beitrag zum System der Monokotyledonen. *Willdenowia* 2:639–768. Botanic Garden and Botanical Museum, Free University of Berlin, Germany 1961
15. R.V. Hamming. Error Detecting and Error Correcting Codes. *Bell Systems Tech. Journal* 29:147–160. Bell Laboratories, Murray Hill, NJ, USA 1950
16. P. Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles* 37, 547-579. France 1901
17. R. Kohavi, C.E. Bradley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 Organizers' Report: Peeling the Onion. *SIGKDD Exploration* 2(2):86–93. ACM Press, New York, NY, USA 2000
18. T. Kötter and M.R. Berthold. Concept Detection. *Proc. 8th Conf. on Computing and Philosophy (ECAP 2010).* University of Munich, Germany 2010
19. S. Kulczynski. Classe des Sciences Mathématiques et Naturelles. *Bulletin Int. de l'Acadamie Polonaise des Sciences et des Lettres Série B (Sciences Naturelles) (Supplement II)* 57–203. Warsaw, Poland 1927
20. D.J. Rogers and T.T. Tanimoto. A Computer Program for Classifying Plants. *Science* 132:1115–1118. American Association for the Advancement of Science, Washington, DC, USA 1960
21. P.F. Russel and T.R. Rao. On Habitat and Association of Species of Anopheline Larvae in South-eastern Madras. *J. Malaria Institute* 3:153–178. India 1940
22. P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy.* Freeman Books, San Francisco, CA, USA 1973
23. R.R. Sokal and C.D. Michener. A Statistical Method for Evaluating Systematic Relationships. *University of Kansas Scientific Bulletin* 38:1409–1438. University of Kansas, Lawrence, KS, USA 1958
24. R.R. Sokal and P.H.A. Sneath. *Principles of Numerical Taxonomy.* Freeman Books, San Francisco, CA, USA 1963
25. T. Sørensen. A Method of Establishing Groups of Equal Amplitude in Plant Sociology based on Similarity of Species and its Application to Analyses of the Vegetation on Danish Commons. *Biologiske Skrifter / Kongelige Danske Videnskabernes Selskab* 5(4):1-34. Denmark 1948
26. T.T. Tanimoto. IBM Internal Report, November 17, 1957
27. M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97, Newport Beach, CA)*, 283–296. AAAI Press, Menlo Park, CA, USA 1997
28. M.J. Zaki and K. Gouda. Fast Vertical Mining Using Diffsets. *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2003, Washington, DC)*, 326–335. ACM Press, New York, NY, USA 2003
29. Synthetic Data Generation Code for Associations and Sequential Patterns. Intelligent Information Systems, IBM Almaden Research Center `http://www.almaden.ibm.com/software/quest/Resources/index.shtml`