



arules – A Computational Environment for Mining Association Rules and Frequent Item Sets

Michael Hahsler

Wirtschaftsuniversität Wien

Bettina Grün

Technische Universität Wien

Kurt Hornik

Wirtschaftsuniversität Wien

Abstract

Mining frequent itemsets and association rules is a popular and well researched approach for discovering interesting relationships between variables in large databases. The R package **arules** presented in this paper provides a basic infrastructure for creating and manipulating input data sets and for analyzing the resulting itemsets and rules. The package also includes interfaces to two fast mining algorithms, the popular C implementations of Apriori and Eclat by Christian Borgelt. These algorithms can be used to mine frequent itemsets, maximal frequent itemsets, closed frequent itemsets and association rules.

Keywords: data mining, association rules, frequent itemsets, R.

1. Introduction

Mining frequent itemsets and association rules is a popular and well researched method for discovering interesting relations between variables in large databases. Piatetsky-Shapiro (1991) describes analyzing and presenting strong rules discovered in databases using different measures of interest. Based on the concept of strong rules, Agrawal, Imielinski, and Swami (1993) introduced the problem of mining association rules from transaction data as follows:

Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of n binary attributes called *items*. Let $\mathcal{D} = \{t_1, t_2, \dots, t_m\}$ be a set of transactions called the *database*. Each transaction in \mathcal{D} has an unique transaction ID and contains a subset of the items in I . A *rule* is defined as an implication of the form $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The sets of items (for short *itemsets*) X and Y are called *antecedent* (left-hand-side or LHS) and *consequent* (right-hand-side or RHS) of the rule.

transaction ID	items
1	milk, bread
2	bread, butter
3	beer
4	milk, bread, butter
5	bread, butter

Table 1: An example supermarket database with five transactions.

To illustrate the concepts, we use a small example from the supermarket domain. The set of items is $I = \{\text{milk, bread, butter, beer}\}$ and a small database containing the items is shown in Table 1. An example rule for the supermarket could be $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ meaning that if milk and bread is bought, customers also buy butter.

To select interesting rules from the set of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence. The *support* $\text{supp}(X)$ of an itemset X is defined as the proportion of transactions in the data set which contain the itemset. In the example database in Table 1, the itemset $\{\text{milk, bread}\}$ has a support of $2/5 = 0.4$ since it occurs in 40% of all transactions (2 out of 5 transactions).

Finding frequent itemsets can be seen as a simplification of the unsupervised learning problem called “mode finding” or “bump hunting” (Hastie, Tibshirani, and Friedman 2001). For these problems each item is seen as a variable. The goal is to find prototype values so that the probability density evaluated at these values is sufficiently large. However, for practical applications with a large number of variables, probability estimation will be unreliable and computationally too expensive. This is why in practice frequent itemsets are used instead of probability estimation.

The *confidence* of a rule is defined $\text{conf}(X \Rightarrow Y) = \text{supp}(X \cup Y) / \text{supp}(X)$. For example, the rule $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ has a confidence of $0.2/0.4 = 0.5$ in the database in Table 1, which means that for 50% of the transactions containing milk and bread the rule is correct. Confidence can be interpreted as an estimate of the probability $P(Y|X)$, the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS (see e.g., Hipp, Güntzer, and Nakhaeizadeh 2000).

Association rules are required to satisfy both a minimum support and a minimum confidence constraint at the same time. At medium to low support values, often a great number of frequent itemsets are found in a database. However, since the definition of support enforces that all subsets of a frequent itemset have to be also frequent, it is sufficient to only mine all *maximal frequent itemsets*, defined as frequent itemsets which are not proper subsets of any other frequent itemset (Zaki, Parthasarathy, Ogihara, and Li 1997). Another approach to reduce the number of mined itemsets is to only mine *frequent closed itemsets*. An itemset is closed if no proper superset of the itemset is contained in each transaction in which the itemset is contained (Pasquier, Bastide, Taouil, and Lakhal 1999; Zaki 2004). Frequent closed itemsets are a superset of the maximal frequent itemsets. Their advantage over maximal frequent itemsets is that in addition to yielding all frequent itemsets, they also preserve the support information for all frequent itemsets which can be important for computing additional interest measures after the mining process is finished (e.g., confidence for rules generated from the found itemsets, or *all-confidence* (Omiecinski 2003)).

A practical solution to the problem of finding too many association rules satisfying the support and confidence constraints is to further filter or rank found rules using additional interest measures. A popular measure for this purpose is *lift* (Brin, Motwani, Ullman, and Tsur 1997). The lift of a rule is defined as $\text{lift}(X \Rightarrow Y) = \text{supp}(X \cup Y) / (\text{supp}(X)\text{supp}(Y))$, and can be interpreted as the deviation of the support of the whole rule from the support expected under independence given the supports of the LHS and the RHS. Greater lift values indicate stronger associations.

In the last decade, research on algorithms to solve the frequent itemset problem has been abundant. Goethals and Zaki (2004) compare the currently fastest algorithms. Among these algorithms are the implementations of the Apriori and Eclat algorithms by Borgelt (2003) interfaced in the **arules** environment. The two algorithms use very different mining strategies. Apriori, developed by Agrawal and Srikant (1994), is a level-wise, breadth-first algorithm which counts transactions. In contrast, Eclat (Zaki *et al.* 1997) employs equivalence classes, depth-first search and set intersection instead of counting. The algorithms can be used to mine frequent itemsets, maximal frequent itemsets and closed frequent itemsets. The implementation of Apriori can additionally be used to generate association rules.

This paper presents **arules**, an extension package for R (R Development Core Team 2005) which provides the infrastructure needed to create and manipulate input data sets for the mining algorithms and for analyzing the resulting itemsets and rules. Since it is common to work with large sets of rules and itemsets, the package uses sparse matrix representations to minimize memory usage. The infrastructure provided by the package was also created to explicitly facilitate extensibility, both for interfacing new algorithms and for adding new types of interest measures and associations.

The rest of the paper is organized as follows: In the next section, we give an overview of the data structures implemented in the package **arules**. In Section 2 we introduce the functionality of the classes to handle transaction data and associations. In Section 3 we describe the way mining algorithms are interfaced in **arules** using the available interfaces to Apriori and Eclat as examples. We provide several examples in Section 4. The first two examples show typical R sessions for preparing, analyzing and manipulating a transaction data set, and for mining association rules. The third example demonstrates how **arules** can be extended to integrate a new interest measure. We conclude with a summary of the features and strengths of the package **arules** as a computational environment for mining association rules and frequent itemsets.

2. Data structure overview

To enable the user to represent and work with input and output data of association rule mining algorithms in R, a well-designed structure is necessary which can deal in an efficient way with large amounts of sparse binary data. The S4 class structure implemented in the package **arules** is presented in Figure 1.

For input data the classes **transactions** and **tidLists** (transaction ID lists, an alternative way to represent transaction data) are provided. The output of the mining algorithms comprises the classes **itemsets** and **rules** representing sets of itemsets or rules, respectively. Both classes directly extend a common virtual class called **associations** which provides a common interface. In this structure it is easy to add a new type of associations by adding a new class that

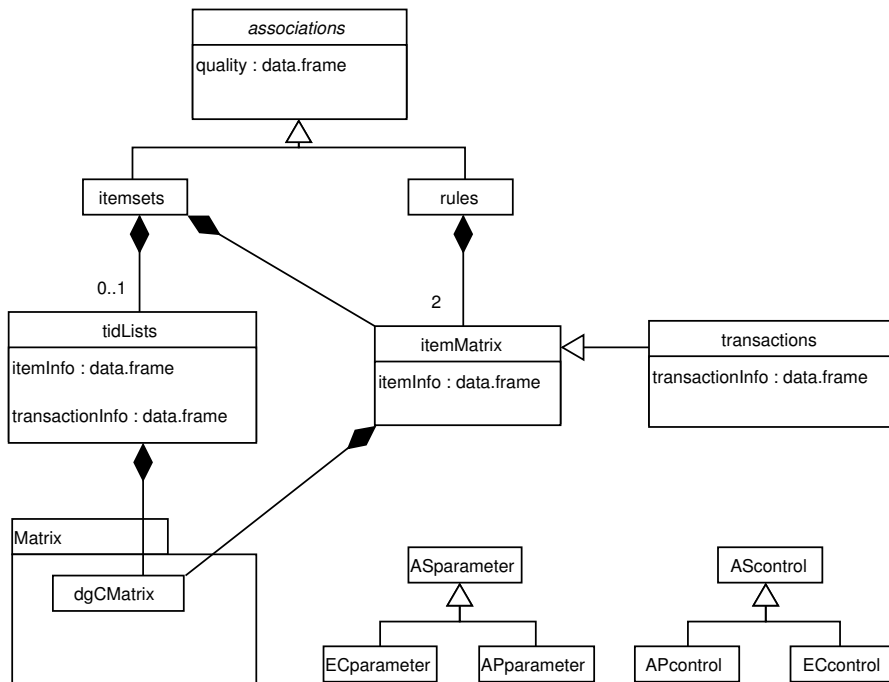


Figure 1: UML class diagram (see [Fowler 2004](#)) of the **arules** package.

extends `associations`.

Items in `associations` and `transactions` are implemented by the `itemMatrix` class which provides a facade for the sparse matrix implementation `dgCMatix` from the R package **Matrix** ([Bates and Maechler 2005](#)).

To control the behavior of the mining algorithms, the two classes `ASparameter` and `AScontrol` are used. Since each algorithm can use additional algorithm-specific parameters, we implemented for each interfaced algorithm its own set of control classes. We used the prefix ‘AP’ for Apriori and ‘EC’ for Eclat. In this way, it is easy to extend the control classes when interfacing a new algorithm.

2.1. Representing collections of itemsets

From the definition of the association rule mining problem we see that transaction databases and sets of associations have in common that they contain sets of items (itemsets) together with additional information. For example, a transaction in the database contains a transaction ID and an itemset. A rule in a set of mined association rules contains two itemsets, one for the LHS and one for the RHS, and additional quality information, e.g., values for various interest measures.

Collections of itemsets used for transaction databases and sets of associations can be represented as binary incidence matrices with columns corresponding to the items and rows corresponding to the itemsets. The matrix entries represent the presence (1) or absence (0) of an item in a particular itemset. An example of a binary incidence matrix containing itemsets for the example database in [Table 1](#) on [Page 2](#) is shown in [Table 2](#). Note that we need to store collections of itemsets with possibly duplicated elements (identical rows), i.e, itemsets

		items			
		i_1	i_2	i_3	i_4
		milk	bread	butter	beer
itemsets	X_1	1	1	0	0
	X_2	0	1	0	1
	X_3	1	1	1	0
	X_4	0	0	1	0

Table 2: Example of a collection of itemsets represented as a binary incidence matrix.

containing exactly the same items. This is necessary, since a transaction database can contain different transactions with the same items. Such a database is still a set of transactions since each transaction also contains a unique transaction ID.

Since a typical frequent itemset or a typical transaction (e.g., a supermarket transaction) only contains a small number of items compared to the total number of available items, the binary incidence matrix will in general be very sparse with many items and a very large number of rows. A natural representation for such data is a sparse matrix format. For our implementation we chose the `dgCMatrix` class defined in package `Matrix`. The `dgCMatrix` is a compressed, sparse, column-oriented matrix which contains the indices of the non-zero rows, the pointers to the initial indices of elements in each column and the non-zero elements of the matrix. Since package `Matrix` does not provide efficient subset selection functionality which directly works on the `dgCMatrix` data structure, we implemented a suitable function in C and interfaced it as the subset selection method (`[]`). Despite the column orientation of the `dgCMatrix`, it is more convenient to work with incidence matrices which are row-oriented. This makes the most important manipulation, selecting a subset of transactions from a data set for mining, more comfortable and efficient. Therefore, we implemented the class `itemMatrix` providing a row-oriented facade to the `dgCMatrix` which stores a transposed incidence matrix¹. At this level also the constraint that the incidence matrix is binary (and not real valued as the `dgCMatrix`) is enforced. In sparse representation the following information needs to be stored for the collection of itemsets in Table 2: A vector of indices of the non-zero elements (row-wise starting with the first row) 1, 2, 2, 4, 1, 2, 3, 3 and the pointers 1, 3, 5, 8 where each row starts in the index vector. The first two pointers indicate that the first row starts with element one in the index vector and ends with element 2 (since with element 3 already the next row starts). The first two elements of the index vector represent the items in the first row which are i_1 and i_2 or milk and bread, respectively. The two vectors are stored in the `dgCMatrix`. Note that indices for the `dgCMatrix` start with zero rather than with one and thus actually the vectors 0, 1, 1, 3, 0, 3, 3 and 0, 3, 4, 7 are stored. However, the data structure of the `dgCMatrix` class is not intended to be directly accessed by the end user of `arules`. The interfaces of `itemMatrix` can be used without knowledge of how the internal representation of the data works. However, if necessary, the `dgCMatrix` can be directly accessed by developers to add functionality to `arules` (e.g., to develop new types of associations or interest measures or to efficiently compute a distance matrix between itemsets for clustering). In this case, the

¹Note that the developers of package `Matrix` recently started adding support for a sparse row-oriented format and for sparse logical matrices. The classes are defined, but the available functionality currently is rather minimal. Once all required functionality is implemented, `arules` will switch the internal representation to sparse row-oriented logical matrices.

`dgCMatrix` should be accessed using the coercion mechanism from `itemMatrix` to `dgCMatrix` via `as()`.

In addition to the sparse matrix, `itemMatrix` stores item labels (e.g., names of the items) and handles the necessary mapping between the item label and the corresponding column number in the incidence matrix. Optionally, `itemMatrix` can also store additional information on items. For example, the category hierarchy in a supermarket setting can be stored which enables the analyst to select only transactions (or as we later see also rules and itemsets) which contain items from a certain category (e.g., all dairy products).

For `itemMatrix`, basic matrix operations including `dim()` and subset selection (`[]`) are available. The first element of `dim()` and `[]` corresponds to itemsets or transactions (rows), the second element to items (columns). For example, on a transaction data set in variable `x` the subset selection `'x[1:10, 16:20]'` selects a matrix containing the first 10 transactions and items 16 to 20.

Since `itemMatrix` is used to represent sets or collections of itemsets additional functionality is provided. `length()` can be used to get the number of itemsets in an `itemMatrix`. Technically, `length()` returns the number of rows in the matrix which is equal to the first element returned by `dim()`. Identical itemsets can be found with `duplicated()`, and duplications can be removed with `unique()`. `match()` can be used to find matching elements in two collections of itemsets.

With `c()`, several `itemMatrix` objects can be combined by successively appending the rows of the objects, i.e., creating a collection of itemsets which contains the itemsets from all `itemMatrix` objects. This operation is only possible if the `itemMatrix` objects employed are “compatible,” i.e., if the matrices have the same number of columns and the items are in the same order. If two objects contain the same items (item labels), but the order in the matrix is different or one object is missing some items, `recode()` can be used to make them compatible by reordering and inserting columns.

To get the actual number of items in the itemsets stored in the `itemMatrix`, `size()` is used. It returns a vector with the number of items (ones) for each element in the set (row sum in the matrix). Obtaining the sizes from the sparse representations is a very efficient operation, since it can be calculated directly from the vector of column pointers in the `dgCMatrix`. For a purchase incidence matrix, `size()` will produce a vector as long as the number of transactions in the matrix with each element of the vector containing the number of items in the corresponding transaction. This information can be used, e.g., to select or filter unusually long or short transactions.

`itemFrequency()` calculates the frequency for each item in an `itemMatrix`. Conceptually, the item frequencies are the column sums of the binary matrix. Technically, column sums can be implemented for sparse representation efficiently by just tabulating the vector of row numbers of the non-zero elements in the `dgCMatrix`. Item frequencies can be used for many purposes. For example, they are needed to compute interest measures. `itemFrequency()` is also used by `itemFrequencyPlot()` to produce a bar plot of item count frequencies or support. Such a plot gives a quick overview of a set of itemsets and shows which are the most important items in terms of occurrence frequency.

Coercion from and to `matrix` and `list` primitives is provided where names and `dimnames` are used as item labels. For the coercion from `itemMatrix` to `list` there are two possibilities. The usual coercion via `as()` results in a list of vectors of character strings, each containing the

		items						transaction ID lists	
		milk	bread	butter	beer			items	
transactions	1	1	1	0	0	items	milk	1, 4	
	2	0	1	1	0		bread	1, 2, 4, 5	
	3	0	0	0	1		butter	2, 4	
	4	1	1	1	0		beer	3	
	5	0	1	1	0				

(a)
(b)

Table 3: Example of a set of transactions represented in (a) horizontal layout and in (b) vertical layout.

item labels of the items in the corresponding row of the `itemMatrix`. The actual conversion is done by `LIST()` with its default behavior (argument `decode` set to `TRUE`). If in turn `LIST()` is called with the argument `decode` set to `FALSE`, the result is a list of integer vectors with column numbers for items instead of the item labels. For many computations it is useful to work with such a list and later use the item column numbers to go back to the original `itemMatrix` for, e.g., subsetting columns. For subsequently decoding column numbers to item labels, `decode()` is also available.

Finally, `image()` can be used to produce a level plot of an `itemMatrix` which is useful for quick visual inspection. For transaction data sets (e.g., point-of-sale data) such a plot can be very helpful for checking whether the data set contains structural changes (e.g., items were not offered or out-of-stock during part of the observation period) or to find abnormal transactions (e.g., transactions which contain almost all items may point to recording problems). Spotting such problems in the data can be very helpful for data preparation.

2.2. Transaction data

The main application of association rules is for market basket analysis where large transaction data sets are mined. In this setting each transaction contains the items which were purchased at one visit to a retail store (see e.g., [Berry and Linoff 1997](#)). Transaction data are normally recorded by point-of-sale scanners and often consists of tuples of the form:

$$\langle \text{transaction ID}, \text{item ID}, \dots \rangle$$

All tuples with the same transaction ID form a single transaction which contains all the items given by the item IDs in the tuples. Additional information denoted by the ellipsis dots might be available. For example, a customer ID might be provided via a loyalty program in a supermarket. Further information on transactions (e.g., time, location), on the items (e.g., category, price), or on the customers (socio-demographic variables such as age, gender, etc.) might also be available.

For mining, the transaction data is first transformed into a binary purchase incidence matrix with columns corresponding to the different items and rows corresponding to transactions.

The matrix entries represent the presence (1) or absence (0) of an item in a particular transaction. This format is often called the *horizontal* database layout (Zaki 2000). Alternatively, transaction data can be represented in a *vertical* database layout in the form of *transaction ID lists* (Zaki 2000). In this format for each item a list of IDs of the transactions the item is contained in is stored. In Table 3 the example database in Table 1 on Page 2 is depicted in horizontal and vertical layouts. Depending on the algorithm, one of the layouts is used for mining. In **arules** both layouts are implemented as the classes `transactions` and `tidLists`. Similar to `transactions`, class `tidLists` also uses a sparse representation to store its lists efficiently. Objects of classes `transactions` and `tidLists` can be directly converted into each other by coercion.

The class `transactions` directly extends `itemMatrix` and inherits its basic functionality (e.g., subset selection, getting itemset sizes, plotting item frequencies). In addition, `transactions` has a slot to store further information for each transaction in form of a `data.frame`. The slot can hold arbitrary named vectors with length equal to the number of stored transactions. In **arules** the slot is currently used to store transaction IDs, however, it can also be used to store user IDs, revenue or profit, or other information on each transaction. With this information subsets of transactions (e.g., only transactions of a certain user or exceeding a specified profit level) can be selected.

Objects of class `transactions` can be easily created by coercion from `matrix` or `list`. If names or `dimnames` are available in these data structures, they are used as item labels or transaction IDs, accordingly. To import data from a file, the `read.transactions()` function is provided. This function reads files structured as shown in Table 3 and also the very common format with one line per transaction and the items separated by a predefined character. Finally, `inspect()` can be used to inspect transactions (e.g., “interesting” transactions obtained with subset selection).

Another important application of mining association rules has been proposed by Piatetsky-Shapiro (1991) and Srikant and Agrawal (1996) for discovering interesting relationships between the values of categorical and quantitative (metric) attributes. For mining associations rules, non-binary attributes have to be mapped to binary attributes. The straightforward mapping method is to transform the metric attributes into k ordinal attributes by building categories (e.g., an attribute income might be transformed into a ordinal attribute with the three categories: “low”, “medium” and “high”). Then, in a second step, each categorical attribute with k categories is represented by k binary dummy attributes which correspond to the items used for mining. An example application using questionnaire data can be found in Hastie *et al.* (2001) in the chapter about association rule mining.

The typical representation for data with categorical and quantitative attributes in R is a `data.frame`. First, a domain expert has to create useful categories for all metric attributes. This task is supported in R by functions such as `cut()`. The second step, the generation of binary dummy items, is automated in package **arules** by coercing from `data.frame` to `transactions`. In this process, the original attribute names and categories are preserved as additional item information and can be used to select itemsets or rules which contain items referring to a certain original attributes. By default it is assumed that missing values do not carry information and thus all of the corresponding dummy items are set to zero. If the fact that the value of a specific attribute is missing provides information (e.g., a respondent in an interview refuses to answer a specific question), the domain expert can create for the attribute a category for missing values which then will be included in the transactions as its own dummy

item.

The resulting `transactions` object can be mined and analyzed the same way as market basket data, see the example in Section 4.1.

2.3. Associations: Itemsets and sets of rules

The result of mining transaction data in **arules** are **associations**. Conceptually, associations are sets of objects describing the relationship between some items (e.g., as an itemset or a rule) which have assigned values for different measures of quality. Such measures can be measures of significance (e.g., support), or measures of interest (e.g., confidence, lift), or other measures (e.g., revenue covered by the association).

All types of association have a common functionality in **arules** comprising the following methods:

- `summary()` to give a short overview of the set and `inspect()` to display individual associations,
- `length()` for getting the number of elements in the set,
- `items()` for getting for each association a set of items involved in the association (e.g., the union of the items in the LHS and the RHS for each rule),
- sorting the set using the values of different quality measures (`SORT()`),
- subset extraction (`[]` and `subset()`),
- set operations (`union()`, `intersect()` and `setequal()`), and
- matching elements from two sets (`match()`).

The associations currently implemented in package **arules** are sets of itemsets (e.g., used for frequent itemsets of their closed or maximal subset) and sets of rules (e.g., association rules). Both classes, `itemsets` and `rules`, directly extend the virtual class `associations` and provide the functionality described above.

Class `itemsets` contains one `itemMatrix` object to store the items as a binary matrix where each row in the matrix represents an itemset. In addition, it may contain transaction ID lists as an object of class `tidLists`. Note that when representing transactions, `tidLists` store for each item a transaction list, but here store for each itemset a list of transaction IDs in which the itemset appears. Such lists are currently only returned by `eclat()`.

Class `rules` consists of two `itemMatrix` objects representing the left-hand-side (LHS) and the right-hand-side (RHS) of the rules, respectively.

The items in the associations and the quality measures can be accessed and manipulated in a safe way using accessor and replace methods for `items`, `lhs`, `rhs`, and `quality`. In addition the association classes have built-in validity checking which ensures that all elements have compatible dimensions.

It is simple to add new quality measures to existing associations. Since the `quality` slot holds a `data.frame`, additional columns with new quality measures can be added. These new measures can then be used to sort or select associations using `SORT()` or `subset()`. Adding

a new type of associations to **arules** is straightforward as well. To do so, a developer has to create a new class extending the virtual `associations` class and implement the common functionality described above.

3. Mining algorithm interfaces

In package **arules** we interface free reference implementations of Apriori and Eclat by Christian Borgelt ([Borgelt and Kruse 2002](#); [Borgelt 2003](#)). The code is called directly from R by the functions `apriori()` and `eclat()` and the data objects are directly passed from R to the C code and back without writing to external files. The implementations can mine frequent itemsets, and closed and maximal frequent itemsets. In addition, `apriori()` can also mine association rules.

The data given to the `apriori()` and `eclat()` functions have to be `transactions` or something which can be coerced to `transactions` (e.g., `matrix` or `list`). The algorithm parameters are divided into two groups represented by the arguments `parameter` and `control`. The mining parameters (`parameter`) change the characteristics of the mined itemsets or rules (e.g., the minimum support) and the control parameters (`control`) influence the performance of the algorithm (e.g., enable or disable initial sorting of the items with respect to their frequency). These arguments have to be instances of the classes `APparameter` and `APcontrol` for the function `apriori()` or `ECparameter` and `ECcontrol` for the function `eclat()`, respectively. Alternatively, data which can be coerced to these classes (e.g., `NULL` which will give the default values or a named list with names equal to slot names to change the default values) can be passed. In these classes, each slot specifies a different parameter and the values. The default values are equal to the defaults of the stand-alone C programs ([Borgelt 2004](#)) except that the standard definition of the support of a rule ([Agrawal et al. 1993](#)) is employed for the specified minimum support required (Borgelt defines the support of a rule as the support of its antecedent).

For `apriori()` the appearance feature implemented by Christian Borgelt can also be used. With argument `appearance` of function `apriori()` one can specify which items have to or must not appear in itemsets or rules. For more information on this feature we refer to the Apriori manual ([Borgelt 2004](#)).

The output of the functions `apriori()` and `eclat()` is an object of a class extending `associations` which contains the sets of mined associations and can be further analyzed using the functionality provided for these classes.

There exist many different algorithms which use an incidence matrix or transaction ID list representation as input and solve the frequent and closed frequent itemset problems. Each algorithm has specific strengths which can be important for very large databases. Such algorithms, e.g. `kDCI`, `LCM`, `FP-Growth` or `Patricia`, are discussed in [Goethals and Zaki \(2003\)](#). The source code of most algorithms is available on the internet and, if a special algorithm is needed, interfacing the algorithms for **arules** is straightforward. The necessary steps are:

1. Adding interface code to the algorithm, preferably by directly calling into the native implementation language (rather than using files for communication), and an R function calling this interface.

2. Implementing extensions for ASparameter and AScontrol.

4. Examples

4.1. Analyzing and preparing a transaction data set

In this example, we show how a data set can be analyzed and manipulated before associations are mined. This is important for finding problems in the data set which could make the mined associations useless or at least inferior to associations mined on a properly prepared data set. For the example, we look at the **Epub** transaction data contained in package **arules**. This data set contains downloads of documents from the Electronic Publication platform of the Vienna University of Economics and Business Administration available via <http://epub.wu-wien.ac.at/> from January 2003 to August 2005.

First, we load **arules** and the data set.

```
> library("arules")
```

```
Loading required package: stats4
```

```
Loading required package: Matrix
```

```
> data("Epub")
```

```
> Epub
```

```
transactions in sparse format with
 3307 transactions (rows) and
 436 items (columns)
```

We see that the data set consists of 3307 transactions and is represented as a sparse matrix with 3307 rows and 436 columns which represent the items. Next, we use the `summary()` to get more information about the data set.

```
> summary(Epub)
```

```
transactions as itemMatrix in sparse format with
 3307 rows (elements/itemsets/transactions) and
 436 columns (items)
```

```
most frequent items:
```

```
doc_11d doc_4c6 doc_2cd doc_71 doc_24e (Other)
   194    115    106    102    93    5323
```

```
element (itemset/transaction) length distribution:
```

```
  1  2  3  4  5  6  7  8  9 10 11 12 13 14
2359 491 201 95 46 24 17 12 11 7 9 4 4 3
```

15	16	17	18	19	20	22	24	25	28	34	38	74	79
1	3	2	3	5	2	1	1	1	1	1	1	1	1

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	1.000	1.000	1.794	2.000	79.000

includes extended transaction information - examples:

```

transactionID      TimeStamp
1 session_4795 2003-01-01 19:59:00
2 session_4797 2003-01-02 06:46:01
3 session_479a 2003-01-02 09:50:38

```

`summary()` displays the most frequent items in the data set, information about the transaction length distribution and that the data set contains some extended transaction information. We see that the data set contains transaction IDs and in addition time stamps (using class `POSIXct`) for the transactions. This additional information can be used for analyzing the data set.

```

> year <- strftime(as.POSIXlt(transactionInfo(Epub)[["TimeStamp"]]),
+                 "%Y")
> table(year)

```

```

year
2003 2004 2005
 988 1375  944

```

For 2003, the first year in the data set, we have 988 transactions. We can select the corresponding transactions and inspect the structure using a level-plot (see Figure 2).

```

> Epub2003 <- Epub[year == "2003"]
> length(Epub2003)

```

```
[1] 988
```

```
> image(Epub2003)
```

The plot is a direct visualization of the binary incidence matrix where the dark dots represent the ones in the matrix. From the plot we see that the items in the data set are not evenly distributed. In fact, the white area to the top right side suggests, that in the beginning of 2003 only very few items were available (less than 50) and then during the year more items were added until it reached a number of around 300 items. Also, we can see that there are some transactions in the data set which contain a very high number of items (denser horizontal lines). These transactions need further investigation since they could originate from data collection problems (e.g., a web robot downloading many documents from the publication site). To find the very long transactions we can use the `size()` and select very long transactions (containing more than 20 items).

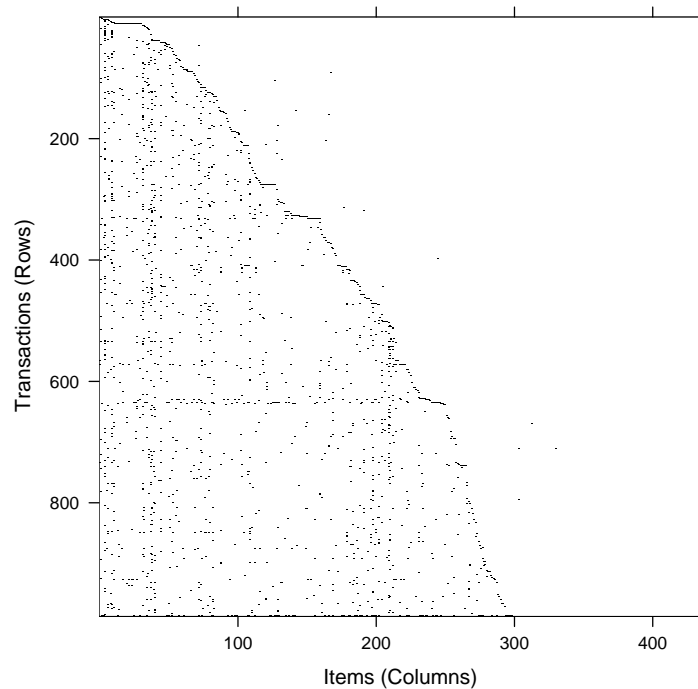


Figure 2: The Epub data set (year 2003).

```
> transactionInfo(Epub2003[size(Epub2003) > 20])
```

	transactionID	TimeStamp
301	session_56e2	2003-04-29 12:30:38
580	session_6308	2003-08-17 17:16:12
896	session_72dc	2003-12-29 19:35:35

We found three long transactions and printed the corresponding transaction information. Of course, `size` can be used in a similar fashion to remove long or short transactions.

Transactions can be inspected using `inspect()`. Since the long transactions identified above would result in a very long printout, we will inspect the first 5 transactions in the subset for 2003.

```
> inspect(Epub2003[1:5])
```

	items	transactionID	TimeStamp
1	{doc_154}	session_4795	2003-01-01 19:59:00
2	{doc_3d6}	session_4797	2003-01-02 06:46:01
3	{doc_16f}	session_479a	2003-01-02 09:50:38
4	{doc_f4, doc_11d, doc_1a7}	session_47b7	2003-01-02 17:55:50
5	{doc_83}	session_47bb	2003-01-02 20:27:44

Most transactions contain one item. Only transaction 4 contains three items. For further inspection transactions can be converted into a list with:

```
> as(Epub2003[1:5], "list")

$session_4795
[1] "doc_154"

$session_4797
[1] "doc_3d6"

$session_479a
[1] "doc_16f"

$session_47b7
[1] "doc_f4" "doc_11d" "doc_1a7"

$session_47bb
[1] "doc_83"
```

Finally, transaction data in horizontal layout can be converted to transaction ID lists in vertical layout using coercion.

```
> EpubTidLists <- as(Epub, "tidLists")
> EpubTidLists
```

```
tidLists in sparse format for
 436 items/itemsets (rows) and
 3307 transactions (columns)
```

For performance reasons the transaction ID list is also stored in a sparse matrix. To get a list, coercion to list can be used.

```
> as(EpubTidLists[1:3], "list")

$doc_154
[1] "session_4795" "session_6082" "session_60dd" "session_67db"
[5] "session_769c" "session_7ee3" "session_bd9d" "session_c591"
[9] "session_ce9f"

$doc_3d6
 [1] "session_4797" "session_4893" "session_48f4"
 [4] "session_4ca3" "session_wu4450a" "session_52c6"
 [7] "session_5712" "session_58e3" "session_5984"
[10] "session_5b20" "session_5c20" "session_5dc0"
[13] "session_5eac" "session_wu4a129" "session_6599"
```

```
[16] "session_673d"      "session_683e"      "session_wu4d25a"
[19] "session_6f2f"      "session_708a"      "session_7a0c"
[22] "session_7de5"      "session_89db"      "session_9227"
[25] "session_9941"      "session_a4d7"      "session_a8c0"
[28] "session_c3c4"      "session_c546"      "session_ca44"
```

```
$doc_16f
```

```
[1] "session_479a" "session_56e2" "session_630c" "session_72dc"
[5] "session_8b3e" "session_91ab" "session_a202" "session_a7b9"
```

In this representation each item has an entry which is a vector of all transactions it occurs in. `tidLists` can be directly used as input for mining algorithms which use such a vertical database layout to mine associations.

In the next example, we will see how a data set is created and rules are mined.

4.2. Preparing and mining a questionnaire data set

As a second example, we prepare and mine questionnaire data. We use the Adult data set from the UCI machine learning repository (Blake and Merz 1998) provided by package **arules**. This data set is similar to the marketing data set used by Hastie *et al.* (2001) in their chapter about association rule mining. The data originates from the U.S. census bureau database and contains 48842 instances with 14 attributes like age, work class, education, etc. In the original applications of the data, the attributes were used to predict the income level of individuals. We added the attribute `income` with levels `small` and `large`, representing an income of \leq USD 50,000 and $>$ USD 50,000, respectively. This data is included in **arules** as the data set `AdultUCI`.

```
> data("AdultUCI")
> dim(AdultUCI)

[1] 48842    15

> AdultUCI[1:2, ]

  age      workclass  fnlwgt  education  education-num
1  39      State-gov  77516  Bachelors          13
2  50  Self-emp-not-inc  83311  Bachelors          13
  marital-status      occupation  relationship  race  sex
1  Never-married      Adm-clerical  Not-in-family  White  Male
2  Married-civ-spouse  Exec-managerial      Husband  White  Male
  capital-gain  capital-loss  hours-per-week  native-country  income
1         2174           0           40  United-States  small
2           0           0           13  United-States  small
```

`AdultUCI` contains a mixture of categorical and metric attributes and needs some preparations before it can be transformed into transaction data suitable for association mining. First, we

remove the two attributes `fnlwgt` and `education-num`. The first attribute is a weight calculated by the creators of the data set from control data provided by the Population Division of the U.S. census bureau. The second removed attribute is just a numeric representation of the attribute `education` which is also part of the data set.

```
> AdultUCI[["fnlwgt"]] <- NULL
> AdultUCI[["education-num"]] <- NULL
```

Next, we need to map the four remaining metric attributes (`age`, `hours-per-week`, `capital-gain` and `capital-loss`) to ordinal attributes by building suitable categories. We divide the attributes `age` and `hours-per-week` into suitable categories using knowledge about typical age groups and working hours. For the two capital related attributes, we create a category called `None` for cases which have no gains/losses. Then we further divide the group with gains/losses at their median into the two categories `Low` and `High`.

```
> AdultUCI[["age"]] <- ordered(cut(AdultUCI[["age"]], c(15,
+   25, 45, 65, 100)), labels = c("Young", "Middle-aged",
+   "Senior", "Old"))
> AdultUCI[["hours-per-week"]] <- ordered(cut(AdultUCI[["hours-per-week"]],
+   c(0, 25, 40, 60, 168)), labels = c("Part-time", "Full-time",
+   "Over-time", "Workaholic"))
> AdultUCI[["capital-gain"]] <- ordered(cut(AdultUCI[["capital-gain"]],
+   c(-Inf, 0, median(AdultUCI[["capital-gain"]][AdultUCI[["capital-gain"]] >
+   0]), Inf)), labels = c("None", "Low", "High"))
> AdultUCI[["capital-loss"]] <- ordered(cut(AdultUCI[["capital-loss"]],
+   c(-Inf, 0, median(AdultUCI[["capital-loss"]][AdultUCI[["capital-loss"]] >
+   0]), Inf)), labels = c("none", "low", "high"))
```

Now, the data can be automatically recoded as a binary incidence matrix by coercing the data set to transactions.

```
> Adult <- as(AdultUCI, "transactions")
> Adult
```

```
transactions in sparse format with
48842 transactions (rows) and
115 items (columns)
```

The remaining 115 categorical attributes were automatically recoded into 115 binary items. During encoding the item labels were generated in the form of `<variable name>=<category label>`. Note that for cases with missing values all items corresponding to the attributes with the missing values were set to zero.

```
> summary(Adult)
```


transactions as `itemMatrix` in sparse format with
48842 rows (elements/itemsets/transactions) and
115 columns (items)

most frequent items:

capital-loss=none	46560	capital-gain=None	44807
native-country=United-States	43832	race=White	41762
workclass=Private	33906	(Other)	401333

element (itemset/transaction) length distribution:

9	10	11	12	13
19	971	2067	15623	30162

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
9.00	12.00	13.00	12.53	13.00	13.00

includes extended item information - examples:

	labels	variables	levels
1	age=Young	age	Young
2	age=Middle-aged	age	Middle-aged

The summary of the transaction data set gives a rough overview showing the most frequent items, the length distribution of the transactions and the extended item information which shows which variable and which value were used to create each binary item. In the first example we see that the item with label `age=Middle-aged` was generated by variable `age` and level `middle-aged`.

To see which items are important in the data set we can use the `itemFrequencyPlot()`. To reduce the number of items, we only plot the item frequency for items with a support greater than 10%. For better readability of the labels, we reduce the label size with the parameter `cex.names`. The plot is shown in Figure 3.

```
> itemFrequencyPlot(Adult[, itemFrequency(Adult) > 0.1],
+   cex.names = 0.8)
```

Next, we call the function `apriori()` to find all rules (the default association type for `apriori()`) with a minimum support of 1% and a confidence of 0.6.

```
> rules <- apriori(Adult, parameter = list(support = 0.01,
+   confidence = 0.6))
```

parameter specification:

confidence	minval	smax	arem	aval	originalSupport	support	minlen
0.6	0.1	1	none	FALSE	TRUE	0.01	1

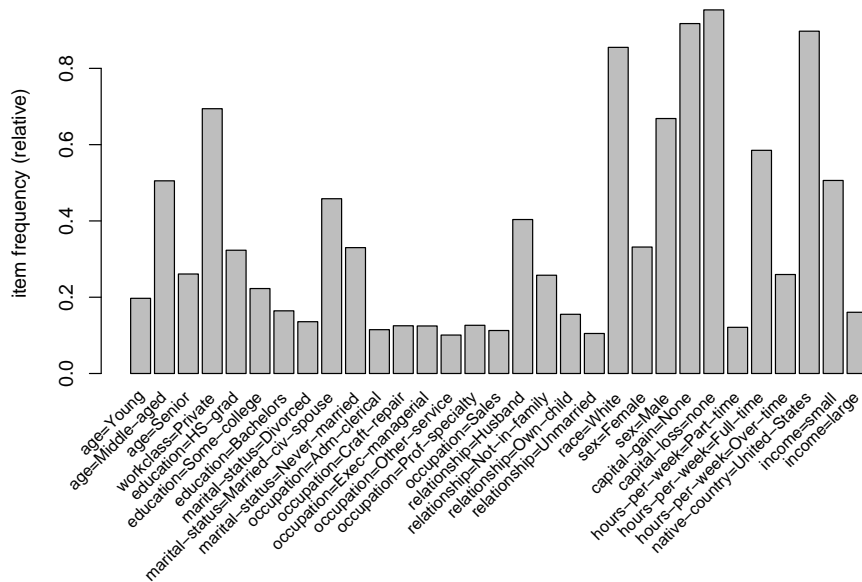


Figure 3: Item frequencies of items in the Adult data set with support greater than 10%.

```
maxlen target ext
5 rules FALSE
```

algorithmic control:

```
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
```

```
apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09) (c) 1996-2004 Christian Borgelt
set item appearances ... [0 item(s)] done [0.00s].
set transactions ... [115 item(s), 48842 transaction(s)] done [0.10s].
sorting and recoding items ... [67 item(s)] done [0.02s].
creating transaction tree ... done [0.14s].
checking subsets of size 1 2 3 4 5 done [0.78s].
writing ... [80215 rule(s)] done [0.04s].
creating S4 object ... done [0.35s].
```

```
> rules
```

```
set of 80215 rules
```

First, the function prints the used parameters. Apart from the specified minimum support and minimum confidence, all parameters have the default values. It is important to note that with parameter `maxlen`, the maximum size of mined frequent itemsets, is by default

restricted to 5. Longer association rules are only mined if `maxlen` is set to a higher value. After the parameter settings, the output of the C implementation of the algorithm with timing information is displayed.

The result of the mining algorithm is a set of 80215 rules. For an overview of the mined rules `summary()` can be used. It shows the number of rules, the most frequent items contained in the left-hand-side and the right-hand-side and their respective length distributions and summary statistics for the quality measures returned by the mining algorithm.

```
> summary(rules)
```

```
set of 80215 rules
```

```
rule length distribution (lhs + rhs):
```

```
  1    2    3    4    5
  6  432 4981 22127 52669
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.000  4.000   5.000   4.584  5.000   5.000
```

```
summary of quality measures:
```

support	confidence	lift
Min. :0.01001	Min. :0.6000	Min. : 0.7201
1st Qu.:0.01364	1st Qu.:0.7542	1st Qu.: 1.0027
Median :0.02090	Median :0.8944	Median : 1.0433
Mean :0.03770	Mean :0.8501	Mean : 1.2476
3rd Qu.:0.03966	3rd Qu.:0.9457	3rd Qu.: 1.2355
Max. :0.95328	Max. :1.0000	Max. :20.6075

As typical for association rule mining, the number of rules found is huge. To analyze these rules, for example, `subset()` can be used to produce separate subsets of rules for each item which resulted from the variable `income` in the right-hand-side of the rule. At the same time we require that the `lift` measure exceeds 1.2.

```
> rulesIncomeSmall <- subset(rules, subset = rhs %in% "income=small" &
+   lift > 1.2)
> rulesIncomeLarge <- subset(rules, subset = rhs %in% "income=large" &
+   lift > 1.2)
```

We now have a set with rules for persons with a small income and a set for persons with a large income. For comparison, we inspect for both sets the three rules with the highest

confidence (using `SORT()`).

```
> inspect(SORT(rulesIncomeSmall, by = "confidence")[1:3])
```

	lhs	rhs	support	confidence	lift
1	{workclass=Private, relationship=Own-child, sex=Male, hours-per-week=Part-time}	=> {income=small}	0.01154744	0.7058824	1.394689
2	{workclass=Private, marital-status=Never-married, sex=Male, hours-per-week=Part-time}	=> {income=small}	0.01517137	0.6951220	1.373428
3	{workclass=Private, occupation=Other-service, relationship=Own-child, capital-gain=None}	=> {income=small}	0.01617460	0.6942004	1.371607

```
> inspect(SORT(rulesIncomeLarge, by = "confidence")[1:3])
```

	lhs	rhs	support	confidence	lift
1	{marital-status=Married-civ-spouse, capital-gain=High, native-country=United-States}	=> {income=large}	0.01562180	0.6849192	4.266398
2	{marital-status=Married-civ-spouse, capital-gain=High, capital-loss=none, native-country=United-States}	=> {income=large}	0.01562180	0.6849192	4.266398
3	{relationship=Husband, race=White, capital-gain=High, native-country=United-States}	=> {income=large}	0.01302158	0.6846071	4.264454

From the rules we see that workers in the private sector working part-time or in the service industry tend to have a small income while persons with high capital gain who are born in the US tend to have a large income. This example shows that using subset selection and sorting a set of mined associations can be analyzed even if it is huge.

4.3. Extending arules with a new interest measure

In this example, we show how easy it is to add a new interest measure, using *all-confidence* as introduced by [Omiecinski \(2003\)](#). The all-confidence of an itemset X is defined as

$$\text{all-confidence}(X) = \frac{\text{supp}(X)}{\max_{I \subset X} \text{supp}(I)} \quad (1)$$

This measure has the property $\text{conf}(I \Rightarrow X \setminus I) \geq \text{all-confidence}(X)$ for all $I \subset X$. This means that all possible rules generated from itemset X must at least have a confidence given by the itemset's all-confidence value. [Omiecinski \(2003\)](#) shows that the support in the denominator of equation 1 must stem from a single item and thus can be simplified to $\max_{i \in X} \text{supp}(\{i\})$.

To obtain an itemset to calculate all-confidence for, we mine frequent itemsets from the previously used Adult data set using the Eclat algorithm.

```
> data("Adult")
> fsets <- eclat(Adult, parameter = list(support = 0.05),
+   control = list(verbose = FALSE))
```

For the denominator of all-confidence we need to find all mined single items and their corresponding support values. In the following we create a named vector where the names are the column numbers of the items and the values are their support.

```
> singleItems <- fsets[size(items(fsets)) == 1]
> singleSupport <- quality(singleItems)$support
> names(singleSupport) <- unlist(LIST(items(singleItems),
+   decode = FALSE))
> head(singleSupport, n = 5)
```

```
          66          63          111          60          8
0.9532779 0.9173867 0.8974243 0.8550428 0.6941976
```

Next, we can calculate the all-confidence using Equation 1 for all itemsets. The single item support needed for the denomination is looked up from the named vector `singleSupport` and the resulting measure is added to the set's quality data frame.

```
> itemsetList <- LIST(items(fsets), decode = FALSE)
> allConfidence <- quality(fsets)$support/sapply(itemsetList,
+   function(x) max(singleSupport[as.character(x)]))
> quality(fsets) <- cbind(quality(fsets), allConfidence)
```

The new quality measure is now part of the set of itemsets.

```
> summary(fsets)
```

```
set of 5908 itemsets
```

```
most frequent items:
```

```
capital-loss=None native-country=United-States
                2301                          2245
capital-gain=None                      race=White
                2236                          2107
workclass=Private                       (Other)
                1784                          13555
```

```
element (itemset/transaction) length distribution:
```

```
  1   2   3   4   5
36 303 1078 2103 2388
```

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.000  4.000   4.000   4.101  5.000   5.000
```

```
summary of quality measures:
      support      allConfidence
Min.   :0.05004   Min.    :0.05249
1st Qu.:0.06230   1st Qu.:0.06986
Median :0.08124   Median :0.09349
Mean   :0.11114   Mean    :0.13111
3rd Qu.:0.12554   3rd Qu.:0.14326
Max.   :0.95328   Max.    :1.00000
```

```
includes transaction ID lists: FALSE
```

It can be used to manipulate the set. For example, we can look at the itemsets which contain an item related to education and sort them by all-confidence (we filter itemsets of length 1 first, since they have per definition an all-confidence of 1).

```
> fsetsEducation <- subset(fsets, subset = items %in% "education")
> inspect(SORT(fsetsEducation[size(fsetsEducation) > 1],
+           by = "allConfidence")[1:3])
```

	items	support	allConfidence
1	{education=HS-grad, hours-per-week=Full-time}	0.2090209	0.3572453
2	{education=HS-grad, income=small}	0.1807051	0.3570388
3	{workclass=Private, education=HS-grad}	0.2391794	0.3445408

The resulting itemsets show that the item high school graduate (but no higher education) is highly associated with working full-time, a small income and working in the private sector. All-confidence is already implemented in **arules** as the function `allConfidence()`.

5. Summary and outlook

With package **arules** we provide the basic infrastructure which enables us to mine associations and analyze and manipulate the results. Previously, in R there was no such infrastructure available. The main features of **arules** are:

- Efficient implementation using sparse matrices.
- Simple and intuitive interface to manipulate and analyze transaction data, sets of itemsets and rules with subset selection and sorting.
- Interface to two fast mining algorithms.
- Flexibility in terms of adding new quality measures, and additional item and transaction descriptions which can be used for selecting transactions and analyzing resulting associations.

- Extensible data structure to allow for easy implementation of new types of associations and interfacing new algorithms.

There are several interesting possibilities to extend **arules**. For example, it would be very useful to interface algorithms which use statistical measures to find “interesting” itemsets (which are not necessarily frequent itemsets as used in an association rule context). Such algorithms include implementations of the χ^2 -test based algorithm by Silverstein, Brin, and Motwani (1998) or the baseline frequency approach by DuMouchel and Pregibon (2001).

Another interesting extension would be to interface synthetic data generators for fast evaluation and comparison of different mining algorithms. The best known generator for transaction data for mining association rules was developed by Agrawal and Srikant (1994). Alternatively data can be generated by simple probabilistic models as done by Hahsler, Hornik, and Reuterer (2005).

Finally, similarity measures between itemsets and rules can be implemented for **arules**. With such measures distance based clustering and visualization of associations is possible (see e.g., Strehl and Ghosh 2003).

Acknowledgments

Part of **arules** was developed during the project “Statistical Computing with R” funded by of the “Jubiläumstiftung der WU Wien.”

The authors of **arules** would like to thank Christian Borgelt for the implementation of Apriori and Eclat.

References

- Agrawal R, Imielinski T, Swami A (1993). “Mining Association Rules Between Sets of Items in Large Databases.” In “Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data,” pp. 207–216. ACM Press. URL <http://doi.acm.org/10.1145/170035.170072>.
- Agrawal R, Srikant R (1994). “Fast Algorithms for Mining Association Rules.” In JB Bocca, M Jarke, C Zaniolo (eds.), “Proc. 20th Int. Conf. Very Large Data Bases, VLDB,” pp. 487–499. Morgan Kaufmann.
- Bates D, Maechler M (2005). **Matrix: A Matrix Package for R**. R package version 0.95-5.
- Berry MJA, Linoff GS (1997). *Data Mining Techniques for Marketing, Sales and Customer Support*. Wiley Computer Publishing.
- Blake CL, Merz CJ (1998). *UCI Repository of Machine Learning Databases*. University of California, Irvine, Dept. of Information and Computer Sciences. URL <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- Borgelt C (2003). “Efficient Implementations of Apriori and Eclat.” In “FIMI’03: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations,” .

- Borgelt C (2004). *Apriori – Finding Association Rules/Hyperedges with the Apriori Algorithm*. Working Group Neural Networks and Fuzzy Systems, Otto-von-Guericke-University of Magdeburg, Universitätsplatz 2, D-39106 Magdeburg, Germany. URL <http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori.html>.
- Borgelt C, Kruse R (2002). “Induction of Association Rules: Apriori Implementation.” In “Proc. 15th Conf. on Computational Statistics (Compstat 2002, Berlin, Germany),” Physika Verlag, Heidelberg, Germany.
- Brin S, Motwani R, Ullman JD, Tsur S (1997). “Dynamic Itemset Counting and Implication Rules for Market Basket Data.” In “SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data,” pp. 255–264. Tucson, Arizona, USA.
- DuMouchel W, Pregibon D (2001). “Empirical Bayes Screening for Multi-Item Associations.” In F Provost, R Srikant (eds.), “Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery in Databases & Data Mining (KDD01),” pp. 67–76. ACM Press.
- Fowler M (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, third edition.
- Goethals B, Zaki MJ (eds.) (2003). *FIMI’03: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*. Sun SITE Central Europe (CEUR).
- Goethals B, Zaki MJ (2004). “Advances in Frequent Itemset Mining Implementations: Report on FIMI’03.” *SIGKDD Explorations*, **6**(1), 109–117.
- Hahsler M, Hornik K, Reutterer T (2005). “Implications of Probabilistic Data Modeling for Rule Mining.” *Report 14*, Department of Statistics and Mathematics, Wirtschaftsuniversität Wien, Research Report Series, Augasse 2-6, 1090 Wien. URL http://epub.wu-wien.ac.at/dyn/openURL?id=oai:epub.wu-wien.ac.at:epub-wu-01_7f0.
- Hastie T, Tibshirani R, Friedman J (2001). *The Elements of Statistical Learning*. Springer-Verlag.
- Hipp J, Güntzer U, Nakhaeizadeh G (2000). “Algorithms for Association Rule Mining – A General Survey and Comparison.” *SIGKDD Explorations*, **2**(2), 1–58.
- Omicinski ER (2003). “Alternative Interest Measures for Mining Associations in Databases.” *IEEE Transactions on Knowledge and Data Engineering*, **15**(1), 57–69.
- Pasquier N, Bastide Y, Taouil R, Lakhal L (1999). “Discovering Frequent Closed Itemsets for Association Rules.” In “Proceeding of the 7th International Conference on Database Theory, Lecture Notes In Computer Science (LNCS 1540),” pp. 398–416. Springer-Verlag.
- Piatetsky-Shapiro G (1991). “Discovery, Analysis, and Presentation of Strong Rules.” In G Piatetsky-Shapiro, WJ Frawley (eds.), “Knowledge Discovery in Databases,” AAAI/MIT Press, Cambridge, MA.
- R Development Core Team (2005). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org/>.

- Silverstein C, Brin S, Motwani R (1998). “Beyond Market Baskets: Generalizing Association Rules to Dependence Rules.” *Data Mining and Knowledge Discovery*, **2**, 39–68.
- Srikant R, Agrawal R (1996). “Mining Quantitative Association Rules in Large Relational Tables.” In HV Jagadish, IS Mumick (eds.), “Int. Conf. on Management of Data, SIGMOD,” pp. 1–12. ACM Press.
- Strehl A, Ghosh J (2003). “Relationship-Based Clustering and Visualization for High-Dimensional Data Mining.” *INFORMS Journal on Computing*, **15**(2), 208–230.
- Zaki MJ (2000). “Scalable Algorithms for Association Mining.” *IEEE Transactions on Knowledge and Data Engineering*, **12**(3), 372–390.
- Zaki MJ (2004). “Mining Non-Redundant Association Rules.” *Data Mining and Knowledge Discovery*, **9**, 223–248.
- Zaki MJ, Parthasarathy S, Ogihara M, Li W (1997). “New Algorithms for Fast Discovery of Association Rules.” *Technical Report 651*, Computer Science Department, University of Rochester, Rochester, NY 14627.

Affiliation:

Michael Hahsler
Institut für Informationswirtschaft
Wirtschaftsuniversität Wien
Augasse 2–6
1090 Wien, Austria
E-mail: Michael.Hahsler@wu-wien.ac.at
URL: <http://www.wu-wien.ac.at/~hahsler/>