Effectiveness of Mining Association Rules for Identifying Trends in Large Health Databases

Tatiana Semenova
Computer Science Laboratory
Australian National University
Canberra ACT 0200, Australia
tatiana@discus.anu.edu.au

Markus Hegland
School of Mathematical Sciences
Australian National University
Canberra ACT 0200, Australia
markus.hegland@anu.edu.au

Warwick Graco
Health Insurance Commission
PO Box 1001
Tuggeranong ACT 2901, Australia
w.graco@hic.gov.au

Graham Williams
Mathematical and Information Sciences
CSIRO, GPO Box 664
Canberra ACT 2601, Australia
graham.williams@cmis.csiro.au

Abstract

Knowledge management focuses on exposing individuals to potentially useful information. An important application of knowledge management is to the coding and sharing of best practice, involving gathering and storing of knowledge, and providing effective search and retrieval mechanisms for locating relevant information. Processing data allows organisations to identify and understand the knowledge held in their databases. Data mining uses dextrous techniques to identify trends and profiles hidden in data. Association rule discovery in particular can be used to identify common or unusual practices in health care. Efficiency is crucial since mining association rules may require repeated scanning of very large datasets which is quite costly. We have developed an association rule algorithm with a focus on delivering knowledge from large administrative databases. Instead of focusing on frequent itemsets the focus is on itemsets that provide knowledge and useful insights.

1 Introduction

Extracting associations in large databases leads to the discovery of potentially useful and previously unknown knowledge from data. Associations between items is important pragmatic knowledge identifying particular relationships in the domain. Insights gained with the new findings can be of great economic value.

For health-care agencies there is a particular need to identify patterns of

practice for purposes such as regulatory management in terms of compliance with government regulations, disease management (best practice) and case management (treatment of patients in different diagnostic categories). One approach to identifying patterns in health data uses association rule mining.

As a consequence of the rapid growth of databases, at least two aspects of data mining algorithms are of increasing importance - processing speed and knowledge novelty. To bias the acquisition of knowledge various measures of statistical significance are used including *support, confidence, conviction, and interest.* Generally speaking, if an association rule is an implication of the form $A \rightarrow B$, where A and B are predicates, then *support* expresses how often A and B are *True* together. The *confidence* is a posteriori probability of B, given A. *Conviction* and *interest* have been introduced by [12] to express better the confirmation of the rule $A \rightarrow B$. All these measurements are based on counting the number of instances involving A and B (or itemsets), which is the most time consuming and expensive task in discovering associations.

The classical association rule discovery algorithms find rules which are well supported [9]. It turns out, however, that many of these rules are well-known and are therefore expected. For example, when we analysed associations of medical services we found that the most frequent associations are the ones which reflect appropriate care as defined in the clinical practice guidelines [20]. The clinical practice guidelines are the knowledge about best medical practice and form the basis of "evidence-based medicine". While there have been attempts to define general interestingness measures [21], the most effective interestingness criteria relate to the common knowledge about the application. For this reason it is essential that the knowledge discovery algorithm is able to access the knowledge in order to separate unknown, therefore, potentially interesting associations from known and, therefore, uninteresting ones. Consequently, the current project is aimed at combining knowledge management and knowledge discovery in order to detect more interesting and relevant structures in the data.

Since interesting rules in health data normally have slightly less support and are more complex, it is necessary to reduce the minimal support in order to find the rules we aim for. We have to be careful, however, not to reduce the minimal support too much as this will increase the number of rules which are not interpretable, typically random or individual fluctuations and, for our purposes, uninteresting as well. Thus the challenge is to find the ight level of support which will generate a sufficient number of potentially interesting associations. One possible method for assessing the interestingness of association rules is to develop *meta*-knowledge of the rules. In the context of this paper, *meta*-knowledge refers to clinical interpretations and evaluations. That is, for each rule the knowledge, insights and views of suitably qualified and experienced medical experts are captured and stored. This *meta*-knowledge is information describing the meaning and implications of the association rule from a clinical perspective. For example, a particular combination of pathology tests may suggest to a clinical expert that it contains services which are unnecessary and therefore wasteful. This knowledge can be stored in the *meta*-knowledge base.

New knowledge can be added to the *meta*-knowledge as a better understanding of the rules and their implications evolves.

The *meta*-knowledge itself can be stored in suitable representations such as lists, networks or tables to facilitate induction learning so that high-order rules or principles can be discerned from the *meta*-knowledge base using both machine learning and human judgement. Rules can be represented as hierarchies with higher-order or general principles at the top, lower-order or specific rules at the bottom and intermediate rules in the middle. The *meta*-knowledge can then be applied for knowledge management purposes such as identifying patterns which are considered best medical practice.

The research is being carried out in a number of stages. These include:

- Developing an efficient algorithm for discovering associations in medical data.

- Developing a meta-knowledge base for storing the expert interpretations and evaluations of the rules.

- Capturing the meta-knowledge from one or more experts and placing it in the meta-knowledge base.

- Applying induction learning to the meta-knowledge to discern high-order rules.

- Using the meta-knowledge for knowledge management purposes.

The research reported in this paper covers the $1st$ stage. Work is in progress with the $2nd$ and $3rd$ stages. The $4th$ and $5th$ stages will be addressed later. The proposed algorithm is capable of discovering all possible association rules of various lengths. Its development is a necessary step towards the integration of knowledge management and data mining for improved knowledge discovery.

The storage, organisation, and retrieval of organisational knowledge constitute an important aspect of effective knowledge management [19]. Advanced computer storage technology and sophisticated searching techniques can be effective tools in enhancing organisational memory. One of the permanent needs in knowledge management is leveraging collected information, transaction data and other sources for continual analysis, which enables organisations to capture knowledge about its needs and directions. In an earlier project we proposed a toolbox that assists in the rapid code development required for such a dynamic scenario [5][7]. The toolbox also provides a framework for access to organisational knowledge as specified in the *Medicare Benefits Schedule Book*[8]. It also allows modeling the relationship between health care providers and patients. This is one of the approaches to integrating knowledge management and data mining, where knowledge management systems (KMS) can be used as a tool to obtain improved results from data mining. A different type of integration of association rule discovery and KMS has been pursued in [23] where association rule discovery was used in combination with natural language processing to discover new knowledge from a knowledge repository like the world wide web.

Rule mining algorithms can be categorised by the dataset scanning strategy. Recent approaches employ either breadth-first search (BFS) or depth-first

search (DFS)[13]. With DFS an algorithm recursively descends the search tree structure. With BFS the supports of all $(k-1)$-itemsets are determined before counting the supports of k-itemsets. Typically, BFS algorithms require k scans through the dataset in order to determine itemset supports. One of the most popular BFS algorithms is the *Apriori Algorithm*[9][10]. In this approach counting the number of occurrences of 2- and 3-itemsets in a database is observed to be the most time consuming step in the whole algorithm.

Effective data selection for data mining requires knowledge about the domain - a *background knowledge*. For example, health care databases contain a huge amount of very broad information about medical services provided, patients, doctors, costs, and many other details, many of which are irrelevant to the particular analysis. In contrast to market basket data *Medicare* transactions are not that diverse. Therefore, it follows that flat-file representation is not optimal for association rule mining. In Section 3.2 we propose a more efficient represention based on hash-tables.

As a way to better represent data, it makes sense to conjunct the contents of some transactions into one sequence according to particular definition relevant to the research purposes and treat this new sequence like transaction later on. If we were to search for associations between health care services, we could assume that medical treatments are scheduled, therefore, not so diverse. For instance, in the *Pathol* database used for our analyses, for 3,617,556 distinct patients only 368,337 unique patient histories were matched. Applying the definition of a health care episode as the group of tests ordered for a patient by the same doctor on the same day, which is in terms of database is the content of all records containing the same *patient identification number*, the same *referring provider*, and the same *date of reference*[8], we represented one of the datasets originally containing 13,192,395 transactions as a set of 2,145,864 sequences (episodes). Amongst them only 62,319 sequences were unique. Our experience in processing administrative health data has shown that unique health care episodes normally occupy less than 10% of the total size of data. So effective pruning of the original data is suggested to be a starting point in handling computations on large datasets. Besides that, the obtained knowledge about diversity and consistency in data is a valuable contribution in understanding the actual meaning of data. This also contributes to the knowledge representation in general.

In the following sections we describe two approaches to finding frequent itemsets in the health data supplied by the *Health Insurance Commission* (Australia). The first is the *Apriori Algorithm* proposed by [Srikant et al, 1993] [9][10], the second (suggested) algorithm *Polydict* is a more straight forward approach implemented in *Python*. Our idea was to apply the *Python's* built-in mapping type *dictionary* to store and update itemsets in order to avoid candidate itemsets generation (like in the *Apriori Algorithm*), apply binary templates to identify frequent itemsets on a single pass of a dataset, and improve therefore the performance without loosing in quality of search. Both algorithms were tested using insightfully preprocessed datasets of the *Pathol* database stored in a hash-table.

2 Datasets Description

Table 1 lists the datasets of the *Pathol* database and their sizes. The database can either be accessed directly by logging into *MySQL* and submitting *SQL* queries on the command line or by submitting *SQL* queries from *Python* scripts. *Python* access to *MySQL* is made using *Data Mining Toolbox's* function **execquery**[5][7].

Table	Records	Size	Episodes	SEp	UniEp	DIt	Ave	LEp
trans-gh	260,225	51Mb	95,255	1.7Mb	9,135	357	2.7	519
trans-sh	1,870,784	269Mb	718,521	12Mb	42,783	417	2.6	6,874
trans-so	8,957,661	1,7Gb	2,671,655	53Mb	142,279	257	2.8	18,307
trans-go97	12,478,278	2.4Gb	2,326,981	44Mb	64,152	232	2.8	9,287
trans-go98	13,192,395	2.5Gb	2,145,864	41Mb	62,319	231	2.8	9,728

Table 1: Tables in the Pathol Database (SEp - size of selection of episodes; UniEp - number of unique episodes; DIt - distinct items; Ave - average length of episode; LEp - large episodes containing more than 10 items).

The tables in the *Pathol* database represent four homogeneous groups with the following identified categories: GH - GPs (general practitioners) referring in-hospital patients; SH - specialists referring in-hospital patients; SO - specialists referring out-patients; GO - GPs referring out-patients. GO dataset is partitioned into two separate tables containing *Medicare* transactions collected during one year time interval each.

To better evaluate the efficiency of our *Python* implementation of both the *Apriori* and *Polydict* algorithms we downloaded Christian Borgelt's *C* program *Apriori* [16] and tested it on our health data. The *BMS WebView-01.dat* set [17] was also used here for comparison as in the *KDD-Cup 2000* competition [18].

Dataset	Records	Size	DIt	Ave	MaxL	LR
BMS WebView-01	59,602	880K	497	2.5	267	1,363

Table 2: Characteristics of BMS WebView-01 dataset (DIt - distinct items; Ave - average record length; MaxL - maximum record length; LR - number of large records with more than 10 items). Note that Size in Table 2 for the BMS WebView-01 corresponds to the size of a separate collection of episodes extracted from a single table in the Pathol database (SEp column in Table 1)

Our testing platform has been the *Sun Enterprise* with 12 *400MHz UltraSPARC* processors each with 8Mb of external cache, 7Gb of main memory,

running *Solaris 2.6.* It has 2 *Sun A5100* disk storage arrays (each 250 Gb capacity) which are fibre channel connected and run in a mirrored configuration. A single process user job would normally have a full use of a single processor, unless all the resources are in a very intensive use.

3 Algorithms

3.1 The Apriori Algorithm

begin
. **Input**: dataset D
. L_1 = (frequent 1-itemsets)
. **while** $C_k \neq \emptyset$ do
. C_k= **join-and-prune**(L_{k-1})
. *counts-C[k]* =**count-candidate-support**(C_k, D, k)
. *indices[k]* = **table-of-indices(counts[k], minsup,** $|D|$)
. *frequent-items[k]* = **generate-L(indices[k],** C_k)
. *counts-L[k]* = **count-support-L(indices[k], counts-C[k])**
. k = k + 1
. **Output**: L - collection of frequent itemsets
end

The algorithm makes k passes over the database. *Python* implementation of the *Apriori* allows to present the database as a collection of unique records with their counts stored in a *dictionary.* Such a presentation benefits to the computational process when counting itemset occurrences. In the first pass the algorithm counts item occurrences to determine the frequent *1*-itemsets (itemsets with 1 item). A subsequent pass, for instance pass k, consists of two steps. Firstly, the frequent itemsets L_{k-1} (the set of all frequent *(k-1)*-itemsets) found in the *(k-1)th* pass are used to generate the candidate itemsets C_k using the **join-and-prune()** function. This function first joins L_{k-1} with L_{k-1}, the joining condition being that the lexicographically ordered first *k-2* items are the same. That is, elements l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1] \wedge (l_1[2] = l_2[2] \wedge ... \wedge (l_1[k-2] = l_2[k-2] \wedge l_1[k-1] < l_2[k-1])$ [3][10]. The result of this operation is a superset C_k with subsets that may or may not be frequent. Secondly, it prunes all those itemsets from the *join* result that have some *(k-1)*-subset that is not in L_{k-1} for the current C_k. Support counts for both candidate and newly generated L_k are stored in separate tables. Function **table-of-indices()** determines and stores indices of those itemsets, which are frequent, it is a link-reference between the table of itemsets and the table of support counts.

To store itemsets and their support counts the sequence type *list* has been chosen because *lists* allow insertion, deletion, substitution, and reversion of elements. Also, this sequence type allows to slice, append, and extend sequences.

The function **count(x)** returns the occurrence of x in a sequence which is extremely convenient when processing large health care episodes with multiply replicated items or when going through records in order to count supports for candidate itemsets and their subsets. Besides that, the *join* step requires multiple use of indexing and slicing operators that the *list*-type provides. As mentioned above, the collection of all episodes initially extracted from the original dataset were stored in a *dictionary*-type structure, where a *key* is a unique episode and a *value* is a number of it's occurrences in the database. This reduces the search space when counting supports for candidate itemsets and certainly saves a great amount of computation. Neveretheless, algorithms like *Apriori* or *Magnum Opus*[22] are found limited for mining interesting associations in health data since they focus on delivering most frequent, thus most expected, associations.

3.2 The Polydict Algorithm

Basically the problem of finding frequent itemsets may be viewed as finding associations between "1" values in a relational table where all the attributes are boolean. Such a table has a column corresponding to each item and a row corresponding to each transaction. The value of an attribute for a given transaction is "1" if the item corresponding to the attribute is present in the transaction, and is "0" if the item is not present. It then represents a more compact version which allows simplified binary structures to be applied in order to detect all associations between items with a single linear scan of the dataset. More importantly, a straight forward linearised technique of applying binary templates aids attainment of all or almost all occurring in the database associations which is necessary when analysing administrative health data.

begin
. **Full linear pass of the database:**
. Update unique episodes in dictionary **Initial**.
. Generate binary templates of lengths *2* to n
. **Full linear pass of Initial dictionary:**
. $\forall$ episode $E \subset$ **Initial** of length **m**, $m \leq n$,
. Get set of binary templates of length **m**.
. Using templates create subsets $e_i \subset E$, i =1,...,m
. and update them in dictionary **Polydict**:
. **if** $polydict.has - key(e_i)$ **then**
. $polydict[e_i] = polydict[e_i] + initial[E]$
. where *initial[E]* is occurrence of E
. **Full linear pass of Polydict dictionary:**
. Delete infrequent e_i.
end

This algorithm makes only one pass over the database during which the

algorithm puts all the selected attributes and then joined into the dictionary *Initial*. The *Initial* dictionary stores all the variety of sequences with their counts, which also makes it more convenient to search for frequent itemsets later on. Having made this step, we get a compact version of our database. This proprocessing phase is the same as in our *Python* implementation of the *Apriori Algorithm*.
The next step is a generation of m binary matrix-templates where in, say the m-th matrix, a row is a template for an episode of length m, and a column represents an attribute. During the following step it scans the dictionary *Initial* in order to linearly apply those templates to all the existing episodes. The *Polydict* dictionary collects all the subsets of various lengths, transaction by transaction, automatically updating their supports. By the end of this step, the *Polydict*-dictionary contains all the associations between *Medicare* items occurring in the database. So in the end, the uninteresting associations get linearly deleted from the *Polydict* dictionary.

4 Performance Considerations

The time for application development and the computational time of the developed application are important dimensions in data analysis. The performance of an algorithm largely depends on the size of data. For databases, two of the most commonly used parameters to describe the size of the input data are the number of records in a database and the number of attributes. To predict computational time one has to distinguish between the worst case (upper bound on the running time) and the average case. We also have to remember that performance time is a hardware-dependent characteristic.

For algorithms doing just simple operations on data like updating or searching, it is very effective to use hash tables, or dictionaries, where the memory location is computed from the key. The *dictionary* is a mapping object containing a collection of objects that are indexed by another collection of constant key values. The basic dictionary operations require only $O(1)$ time on averfge, and a dictionary requires much less storage than a direct access table, in particular, the memory size can be reduced to $\Theta(p)$, where p is a number of keys in a dictionary [1][2].

Scripting languages have been designed specifically to combine different applications and tools. Despite it's great efficiency, *C* presents some serious problems for programmers, including memory management and slower application development. *Python* like other scripting languages has a memory manager built into their runtime configuration and provide a rapid development-cycle turnaround. Thus *Python* codes require typically more memory than equivalent *C* codes, but due to effective memory management they can still be more memory efficient overall [14][15].

In the following section we show that the *Apriori Algorithm* requires $O(2^m n)$ steps, where n is the number of records, m is the number of attributes and the

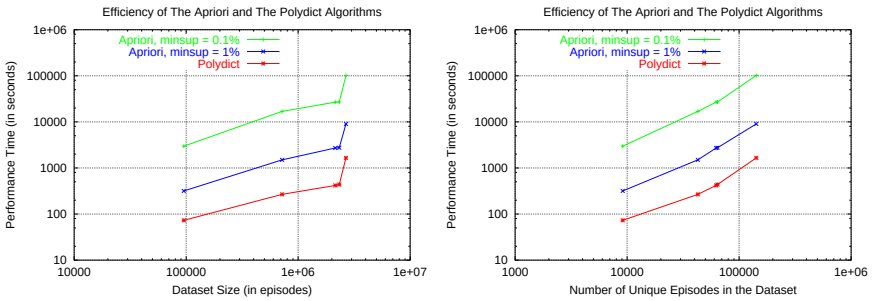

Figure 1: Performance of the Apriori and Polydict algorithms for the Pathol database.

Polydict Algorithm requires $O(2^m p)$, $p \leq n$, steps. So the *Polydict's* complexity does not strongly depend on the number of records in the database, it is rather dependent of similarity/dissimilarity between collections of rendered clinical services. However, it is not really beneficial to use *Polydict* implemented in *Python* in situations where m is comparable with n, for example, in market basket analysis. Also, when m is really large, it causes failures due to *Python Shell* memory restrictions. *Polydict* works significantly faster in situations where m is not too large. In this case, the efficiency of the *Polydict Algorithm* increases with n.

Figure 1 displays the performances of both the *Apriori* and *Polydict* algorithms for two different *minimum support* values. These comparisons are made for the *Pathol* database represented as collections of health care episodes.

All five tables of the *Pathol* database are included in this comparison in the following order (according to the number of extracted episodes): GH; SH; GO98; GO97; SO. Despite the fact that the original SO dataset has smaller size than GO datasets, it contains many more episodes with slightly greater number of distinct items. The proportion of large episodes is also greater for this particular group. The reason for this that the SO dataset represents a more diverse doctor/patient group (specialists referring out-patients) and therefore, more diverse medical treatments.

The computational time plotted over the number of unique episodes looks smoother than the computational time plotted over the total number of episodes for a particular dataset. This comparison shows that both our *Python* implementations of the *Apriori* and *Polydict* are rather sensitive to the similarity between episodes than to the total their number. The performance of the *Apriori* also depends on the size of the transaction. The performance of the *Polydict* algorithm does not depend on *minimum support* but does depend greatly on the number of distinct items and resulting from it the degree of similarity between health care episodes.

Figure 2 compares our *Python* codes with Christian Borgelt's *C* program *Apriori* [16] on medical and non-medical data. *BMS WebView-01* contains

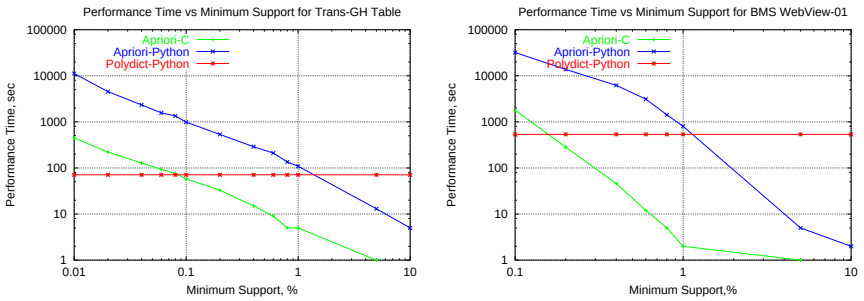

Figure 2: Performance of the Python and C codes for medical and e-commerce data

more larger records and more distinct items in comparison with *trans-gh*. Also, it's maximum record length (267) is significantly greater than the maximum length of a health episode possible, which reaches 40 to 60 items depending on the type of the doctor/patient group. Despite the greater number of transactions in *trans-gh*, the performance of all programs on our health data is much faster than on the *e*-commerce data. Note, that for the *minimum support* smaller than approximately 0.1-0.2%, the efficiency of *Polydict* implemented in *Python* exceeds the efficiency of *Apriori* implemented in *C*.

4.1 Computational Complexity

The input is a sequence of objects $w_1, ..., w_n$, where n is usually a number of records in a dataset [4][11]. Each w_i is a set of type $\{a_1, ..., a_m\}$. The items $\{A_1, ..., A_m\}$ are predicates defined by the incidence:

$$A_j(w) = \begin{cases} 1 & \text{if } a_j \in w \\ 0 & \text{if } a_j \notin w \end{cases}$$

The main computational procedures in the *Apriori Algorithm* are

- count supports for all $A \in C_k$
- find frequent itemsets $L_k \subset C_k$
- join $L_k * L_k$ to get C_{k+1}
- prune C_{k+1} to get L_{k+1}

Let us denote the access time for each step by τ_i, the average size of w_i by l_w, the size of itemsets by $\#L$ or $\#C$, and the size of joined itemset L_k by $\#L_k * L_k$. So the very first step of the algorithm will require
1. $nl_w\tau_1$
2. $m\tau_2$

3. $(\#L_1)^2\tau_3/2$
4. 0 *as nothing yet to prune*,

which in total, if bounded from above, is $nm\tau_1 + m\tau_2 + m^2\tau_3/2$

The second step, analogously, will require

1. $(nl_w^2/2)\#C\tau_4$
2. $(\#C_2\tau_5)$
3. $(\#L_2 * L_2 + \#L_2)\tau_6$
4. $(\#L_2 * L_2)\#L_2\tau_7$,

which in total, if bounded from above, is $(nm^2/2)\tau_4 + m\tau_5 + m\tau_6 + m^2\tau_7$.

Following through some number of steps, we can conclude, that for the apriori technique a performance is influenced mainly by n, l_w (or m for the worst case scenario), the size of the largest frequent itemset k_{max}, and the sizes of candidate and joined itemsets. If the latter ones are stored in hash tables, then procedures *3* and *4* will require constant time, which reaches its largest for the first several iterations. The first procedure strongly depends on the size of the database n and on the average length of a transaction (episode), which makes such a dependence almost exponential.

In sum, if we do not consider local parameters like access time τ_i, we could estimate a performance as [4][11]:

$$S_A \approx n2^{l_w} + k_{max}C_w + k_{max}L_w + k_{max}^2L_w^2$$

where C_w and L_w are the average sizes of candidate and joined itemsets respectively. The order of complexity can roughly be evaluated as $O(2^{l_w}n)$ taking under consideration only the most time consuming procedure.

The *Polydict Algorithm* contains four main procedures, which are not iterative. This makes the evaluations more precise because the performance does not depend on parameters like C_w and L_w, or k_{max}. The very first procedure is creating a hash table for all the transactions. Keeping old notations, the computational time here is proportional to n and l_w. The second procedure takes constant time(and space) and depends mainly on l_w, more exactly, on 2^{l_w}. The third procedure is the most time consuming, it depends on the l_w and p, where p is a length of *Initial*-dictionary containing only unique episodes. The final procedure requires time proportional to the size of *Polydict*. So for the procedures *1 - 4* we have estimates:

1. $S_1 \approx l_w n\tau_1$;
2. $S_2 \approx \sum_1^{l_{max}} 2^{l_w} l_w\tau_2$;
3. $S_3 \approx \sum_1^{p} 2^{l_w} l_w\tau_3$;
4. $S_4 \leq p2^{l_w}\tau_4$.

Therefore, the total sum $S_P = S_1+S_2+S_3+S_4$, which basically is $p2^{l_w}+l_wn$, if we do not include the access time. The order of complexity can be evaluated as $O(2^{l_w}p)$, $p \leq n$. However, $p \leq n$ only in the worst case. Practically, p is much less than n, because the vast majority of shorter episodes contains the same items, therefore, the number of the unique medical treatments (unique episodes) in the database is much smaller than the number of all treatments.

Important to note, that exhaustive search technique (like in *Polydict*) is independent of *minimum support* value, whereas in the *Apriori Algorithm*, a smaller value of *minimum support* affects the algorithm's efficiency almost exponentially.

5 Conclusions

The available administrative health data can be augmented with additional knowledge about the domain. For successful knowledge discovery it is becoming necessary to incorporate efficient data mining techniques into special environment which enables the effective use of knowledge management systems.

Pruning the original data, adequate representation of ordinary transactions as sequences containing specifically selected items, storing and analysing only unique sequences are suggested to be beneficial techniques in handling large health databases.

Utilising either *Python's* built-in features or similar tools saves a great deal of computation and reduces time for application development. Additionally, only the type of storage *dictionary* alone allows to instantly see how diverse or uniform the relationships between attributes within the domain are. Such a type of storing data is an ideal platform to derive and deliver association rules later on. Frequent regularities found in the data can be identified as a common practice. Infrequent regularities may indicate an unusual practice and could consequently become a subject for further exploration. This is an adequate analysis of data to perform in order to identify regularities in the health care system.

The theoretical estimates (section 4.1) of computational times for the *Apriori* and *Polydict* techniques are in agreement with our measurements (section 4). We found that with *minimum support* smaller than certain value, the more straight forward approaches can be a reasonable alternative to the *Apriori* in searching for interesting associations in a large database. This expectation is based on our observations made when analysing transactional health datasets with relatively small number of distinct items. A straight forward technique like *Polydict* implemented in *Python* and supported by *dictionary* type of data storage and retrieval exceeds a *C* program efficiency when *minimum support* decreases beyond a certain value.

It is also expected that if some time consuming computations were replaced with the time-efficient *C*-modules and were embedded in a *Python* script, then the performance of such a mixed-language program would be at least a factor of 10 faster than of an equivalent *Python* program[14]. It will especially benefit to the *Polydict* algorithm making it's *minimum support*- independent performance very efficient for various types of databases with broad data characteristics.

References

[1] Thomas H.Gormen, Charles E.Leiserson, Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press. Cambridge, England, *1991*
[2] Udi Manber. *Introduction to Algorithms: A Creative Approach.* University

of Arizona. Addison-Wesley Publishing Company,*1989*.
[3] J.Han and M.Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, *2001*.
[4] M.Hegland, S.Roberts. *Notes for the 4th year Honours Course in Data Mining*. SMS, ANU. Canberra, *2001*.
[5] P.Christen, O.Nielsen. *Documentation for the Data Mining Toolbox*. CSL, RSISE, ANU. Canberra, *2000*.
[6] David M.Beasley. *Python Essential Reference*. New Riders Publishing. Indianapolis, IN,*2000*.
[7] Ole M.Nielsen, Peter Christen, Markus Hegland, Tatiana Semenova, and Timoty Hancock. *A Toolbox Approach to Flexible and Efficient Data Mining*. ANU, Canberra, *2001*.
[8] Commonwealth Department of Human Services and Health. *Medicare Benefits Schedule Book*. Australian Government Publishing Service, Canberra, *2000*.
[9] Rakesh Agrawal, Tomasz Imielinski, Arun Swami. *Mining Association Rules between Sets of Items in Large Databases*. Proceedings of the 1993 ACM SIGMOD Conference, Washigton DC, USA. *May, 1993*.
[10] Rakesh Agrawal, Ramakrishnan Srikant. *Fast Algorithms for Mining Association Rules*. IBM Almaden Research Center, San Jose, CA, *1994*.
[11] Markus Hegland. *Data Mining Techniques*. ANU, Canberra, *November 6, 2000*.
[12] Yves Kodratoff. *Comparing Machine Learning and Knowledge Discovery: An Application to Knowledge Discovery in Texts*. LNAI-Tutorial-Series. Springer, *2000*.
[13] Jochen Hipp, Ulrich Güntzer, Gholamreza Nakhaezadeh. *Algorithms for Association Rule Mining - A General Survey and Comparison*. SIGKDD Explorations, ACM SIGKDD, *July, 2000*.
[14] Lutz Prechelt. *An Empirical Comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a Search/String Processing Program*. Fakultät für Informatik, Universität Karlsbruhe, Germany. *March, 2000*.
[15] Mark Lutz. *Python: An Object-Oriented Scripting Language*. Indianapolis. *1998*.
[16] Christian Borgelt. http://fuzzy.cs.uni-magdeburg.de/~ borgelt/
[17] http://www.ecn.purdue.edu/KDDCUP/data/BMS-WebView.dat.gz
[18] Ron Kohavi. Blue Martini Software. http://www.bluemartini.com/index.jsp
[19] Maryam Alavi, Dorothy E.Leidner. *Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues*. Management Information Systems. *March, 2001*.
[20] V.Maojo, L.Laita, E.Rones-Lozano, J.Crespo, J. Rodriguez-Pedrosa. *A New Computerized Method to Verify and Disseminate Medical Appropriateness Criteria*. Proc. of the First Int.Symp. ISMDA 2000. Springer *2000*.
[21] R.J.Hilderman, H.J.Hamilton. *Evaluation of Interestingness Measures for Ranking Discovered Knowledge*.p.247-259. LNAI 2035. Springer *2001*.
[22] http://www.rulequest.com/MagnumOpus-info.html
[23] Gerd Stumme, Alexander Maedche. *Ontology Merging for Federated Ontologies on the Semantic Web*. www.aifb.uni-karlsruhe.de/WBS/gst