Basket and Sequence Analysis Benchmarks
XELOPES Java Version

Version 1.1.7

prudsys AG

Bergstraße 61
09113 Chemnitz
Germany

Phone: +49 371 / 27093-0
Fax: +49 371 / 27093-90
E-mail: info@prudsys.com

The information contained in this document is intellectual property of the authors and must not be reused or revealed to thirds without prior written permission by these authors.

Chemnitz
October 12, 2003

The prudsys AG can be owner of patent rights, brands, copyrights, or other rights on intellectual property regarding the contents of this documentation. The provision of this documentation does not automatically mean that you have rights of disposal for these patents, brands, copyrights or other intellectual property.

Without the prior written permission by the prudsys AG, no part of this documentation may be duplicated or transferred for any purposes.

prudsys is a registered trademark of the prudsys AG. All other trademarks or registered trademarks stated in this manual are property of the respective owners.

Authors: Michael Thess, prudsys AG
© 2003 prudsys AG. All rights reserved.

1. Introduction

In this paper we will compare different basket and sequence analysis algorithms of the XELOPES library in order to show their asymptotic behaviour which should help the users to find their right application areas. The algorithms are also available in the prudsys Basket Analyzer tool.

We compare the following basket analysis algorithms:

- *AprioriSimple (XELOPES GPL Version):* simple implementation of Apriori algorithm,
- *AprioriOptimized (XELOPES Extended Version):* fast implementation of Apriori algorithm,
- *AprioriTid (XELOPES Extended Version):* very fast implementation of AprioriTid algorithm,
- *AprioriHybrid (XELOPES Extended Version):* implementation of AprioriHybrid algorithm using *AprioriOptimized* and *AprioriTid*,
- *AssociationRulesDecomposition (XELOPES GPL Version):* decomposition algorithm for basket analysis algorithms, here used with *AprioriTid* as internal algorithm.

Further, the following sequence analysis algorithms are compared:

- *SequentialSimple* (XELOPES GPL Version):* simple implementation of a sequence analysis algorithm, cannot extract cycles,
- *Sequential (XELOPES Extended Version):* very fast implementation of a sequence analysis algorithm (with cycles),
- *SequentialDecomposition (XELOPES Extended Version):* decomposition algorithm for sequence analysis algorithms, here used with *Sequential* as internal algorithm.

Both test data sets, which are used in this document, are real-life data sets.

In the tests we will analyse the influence of both minimum support parameter *minSupp* and size of the data set *#Items* on the calculation times of the different algorithms. When changing the data size, we use the number of items *#Items* and not of the transactions *#Transactions* since the data is stored in transactional format (any item corresponds to one row in the data matrix, the transaction column assigns the items to the transactions). However, the numbers of items and transactions strongly correlate, especially for large data. Hence the tests can be also interpreted in terms of proportionally increased transaction numbers.

All tests were executed on a Windows PC with an AMD Athlon processor of 1 GHz speed and a memory of 512 MB. A JDK 1.3 was used as Java engine. This could be considered as an ordinary computer equipment.

2. Discounter Data

We consider the data of a larger German retailer which is analysed for customer card owners. Here, about 1500 different products (items) belonging to one highest category are included into the analysis. We show the results of the receipt analysis, i.e. all products bought in *one* purchase by one customer are considered to form one transactions. Of course, also other transaction definitions like analysing *all* products bought by one customer would be possible. The average number of items per transaction is 5.5.

2.1 Influence of the Minimum Support Parameter

We start with the smallest sample which contains 20,000 items bought in 3825 transactions and first analyse the influence of the minimum support parameter on different basket analysis algorithms. The results are shown in Table 1.

minSupp	#Rules	AprioriSimple	AprioriOptimized	AprioriTID	AprioriHybrid
3,20	2	3	2	1	2
1,60	4	3	1	2	2
0,80	6	3	2	2	2
0,40	19	5	1	2	2
0,20	257	16	1	2	2
0,10	3599	278	9	3	17
0,05	252478		38	5	40

Table 1: Influence of the minimum support parameter on the calculation times [s] of the basket analysis algorithms.

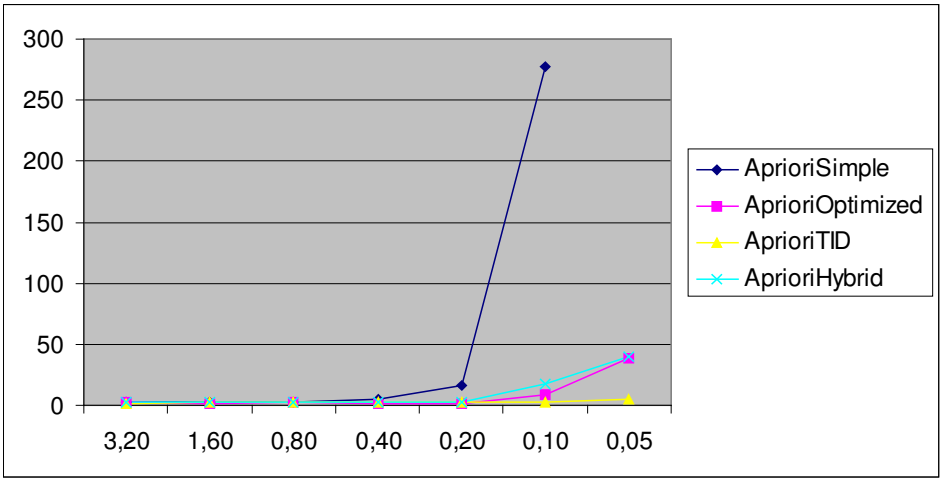

Figure 1: Graphs of the calculation times of Table 1.

The results show that the *AprioriSimple* algorithm much stronger depends on the minim support parameters than the other algorithms. Furthermore, we see that the *AprioriTid* algorithm definitely outperforms all other algorithms indicating an asymptotically optimal runtime behaviour for decreasing support parameters.

Now we do the same tests for sequence analysis algorithms. Notice that we have removed duplicated items in order to compare the *SequentialSimple* algorithm with *Sequential*, since otherwise *SequentialSimple* would only find a subset of all large sequences and the comparison would be unfair. The results are shown in Table 2.

minSupp	#Rules	SequentialSimple	Sequential
3,2	1	2	1
1,6	3	2	2
0,8	4	2	2
0,4	7	*3*	2
0,2	99	15	2
0,1	910	253	5
0,05	17420		11

Table 2: Influence of the minimum support parameter on the calculation times [s] of the sequence analysis algorithms.

The table reveals an exponential growth of calculation time for the *SequentialSimple* algorithm and a nearly optimal behaviour of the *Sequential* algorithm for decreasing support values.

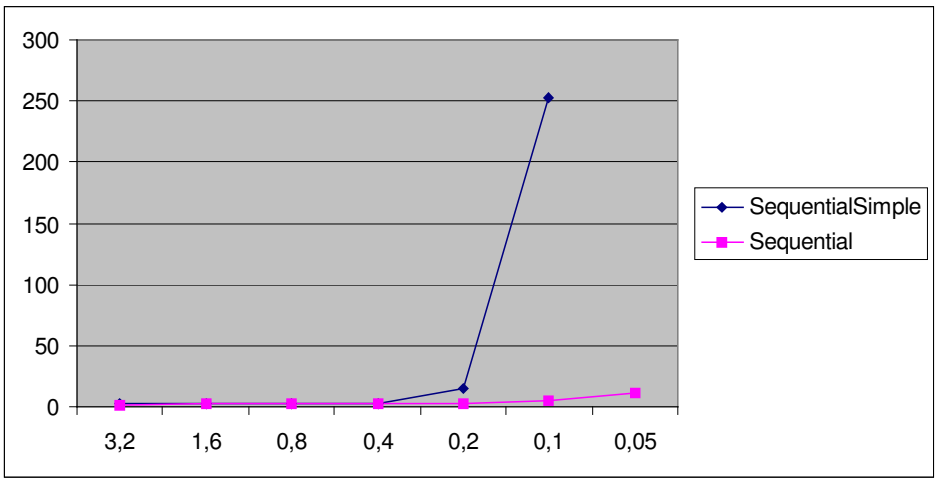

Figure 2: Graphs of the calculation times of Table 2.

2.2 Influence of the Sample Size

In the next step, we are going to investigate the influence of an increasing data size on the calculation times of the different algorithms. We will conduct two series of these tests: In the first series of tests we will fix the minimum support parameter as *minSupp = 0,1%* for all data sets. This will result in a decreasing number of rules for increasing size of the data sets. Therefore, in the second test series we adjust the support parameters for any data set such that between 100 and 200 rules are found.

We start with the basket analysis algorithms with the fixed parameter *minSupp = 0,1%.* The results are given in Table 3.

#Items	#Transactions	#Rules	AprioriSimple	AprioriOptimized	AprioriTID	AprioriHybrid
20000	3825	3599	278	9	3	27
40000	7478	1554	311	5	10	17
80000	14813	1107	331	9	12	12
160000	29428	1861	700	16	9	16
320000	58232	2064		17	11	15
640000	116645	2068		34	25	42
12800000	230792	2090		64	61	61

Table 3: Influence of the sample size on the calculation times [s] of the basket analysis algorithms.

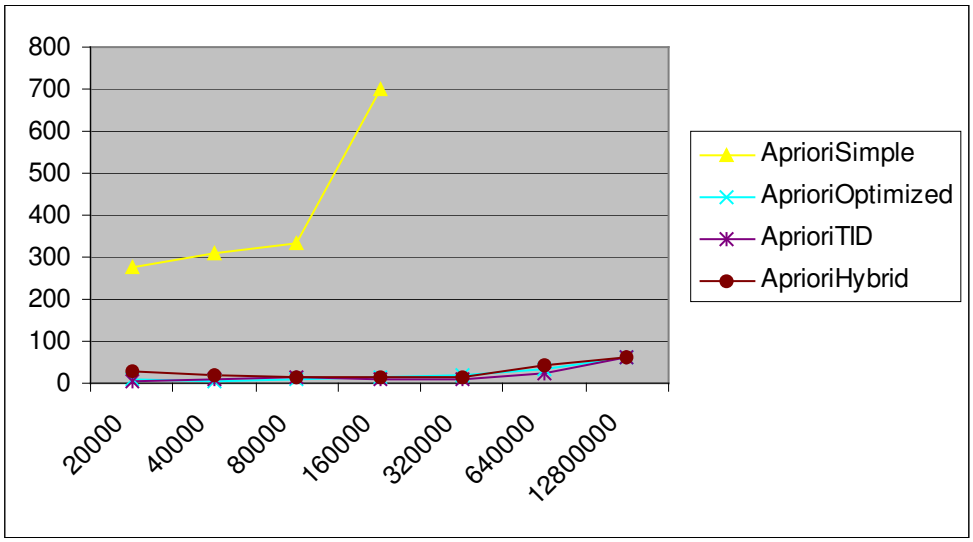

Figure 3: Graphs of the calculation times of Table 3.

The table shows an explosive growth of calculation time of *AprioriSimple* for an increasing sample size. The other algorithms perform very similar and much better, although the growth finally seems to become over-linear.

Similar tests of the sequence analysis algorithms have lead to the results in Table 4.

#Items	#Transactions	#Rules	SequentialSimple	Sequential
20000	3825	910	253	6
40000	7478	431	264	6
80000	14813	334	326	7
160000	29428	445	688	15
320000	58232	512		23
640000	116645	487		41
12800000	230792	490		77

Table 4: Influence of the sample size on the calculation times [s] of the sequence analysis algorithms.

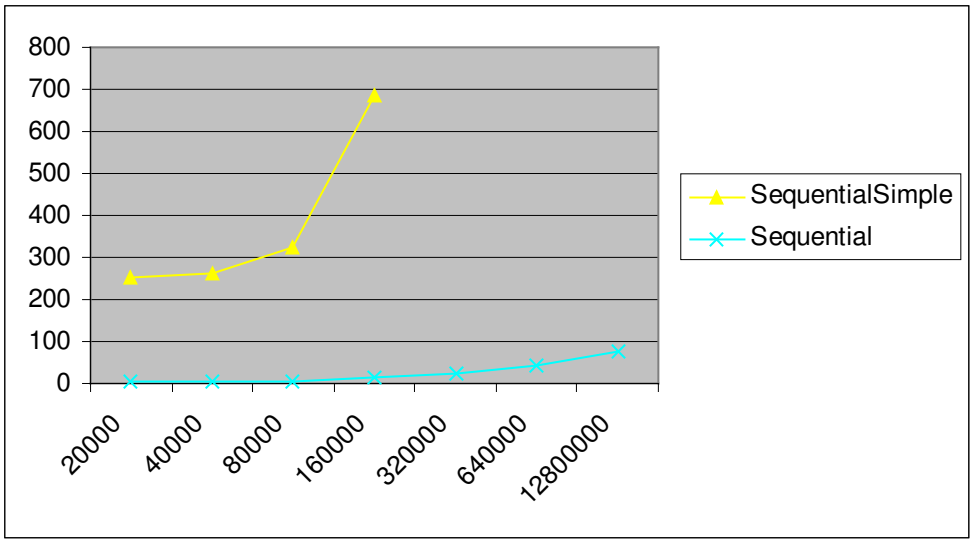

Figure 4: Graphs of the calculation times of Table 4.

The results of the sequence analysis algorithms again show an explosive growth of the simple algorithm whereas the *Sequential* algorithm behaves nearly optimal.

In the second series of tests, we choose the minimum support parameter for any dataset in such a way that we always obtain a number of rules from the interval [100, 200]. The results of the basket analysis algorithms are given in Table 5.

#Items	minSupp	#Rules	AprioriSimple	AprioriOptimized	AprioriTID	AprioriHybrid
20000	0,225	134	10	4	3	4
40000	0,275	143	17	4	3	3
80000	0,3	164	27	7	8	8
160000	0,25	127	91	9	9	8
320000	0,25	182	317	10	8	11
640000	0,25	184		28	29	30
12800000	0,25	148		52	60	51

Table 5: Influence of the sample size on the calculation times [s] of the basket analysis algorithms. The number of transactions is the same as in Tables 3 and 4.

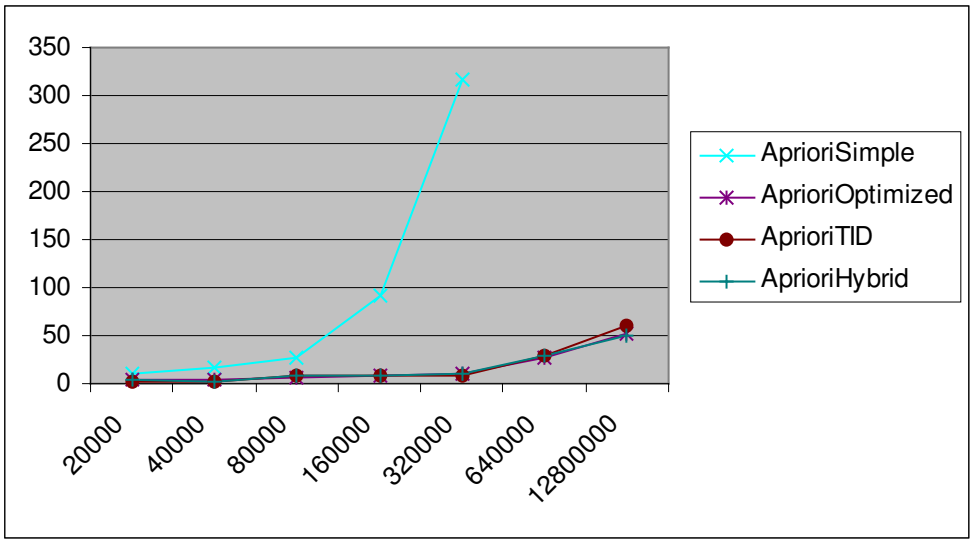

Figure 5: Graphs of the calculation times of Table 5.

The results are basically similar to that of Table 3. One interesting fact is that the *AprioriTid* algorithm for the first time performed slightly worse than the algorithms *AprioriOptimized* and *AprioriHybrid.*

The results of similar tests for sequence analysis algorithms are shown in Table 6.

#Items	#Transactions	minSupp	#Rules	SequentialSimple	Sequential
20000	3825	0,175	167	28	4
40000	7478	0,15	195	56	4
80000	14813	0,2	142	61	8
160000	29428	0,2	127	172	11
320000	58232	0,2	144	528	18
640000	116645	0,2	146		33
12800000	230792	0,2	122		62

Table 6: Influence of the sample size on the calculation times [s] of the sequence analysis algorithms.

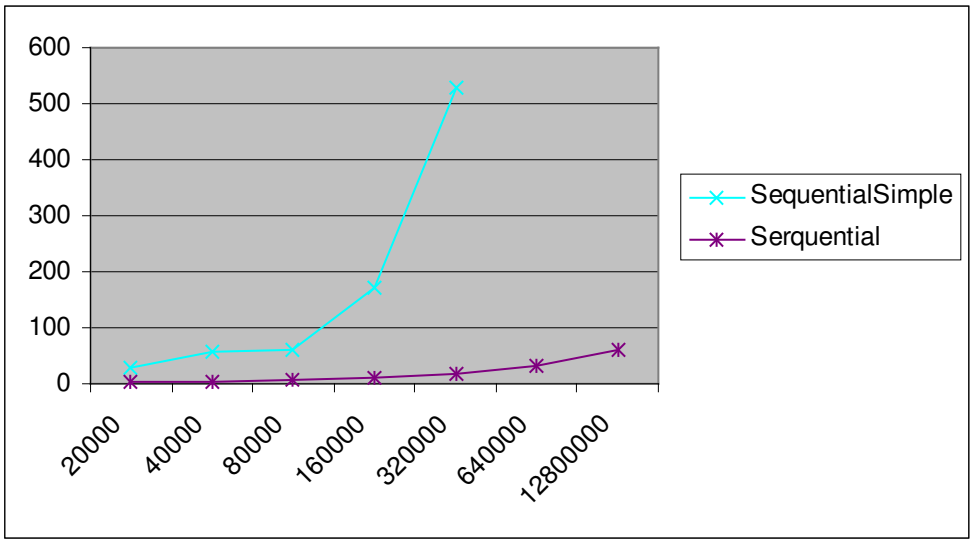

Figure 6: Graphs of the calculation times of Table 6.

Again the results are very similar to that of the first test series (Table 4).

2.3 Tests on Large Sample Size

Now we want to apply association rules algorithms to large data volumes of our discounter example. This is realized through the decomposition methods. We use a fast algorithm inside every decomposition method: *AprioriTid* for *AssociationRulesDecomposition* and *Sequential* for *SequentialDecomposition*. In all examples we use a decomposition size of 50.000 transactions (not items!). We use the same support parameter *minSupp = 0.1%* as in the first series of tests in Section 2.2.

For the basket analysis, the test results are contained in Table 7.

#Items [Mio.]	#Transactions	#Rules	AssociationRulesDecomposition [min]
1,25	919203	2055	2
2,5	1870135	1928	4
5	919203	2013	8
10	1870135	1753	16
20	3726692	1445	35
40	7592484	910	94
57	10378760	826	161

Table 7: Influence of the sample size on the calculation time [min] of the basket analysis decomposition algorithm.

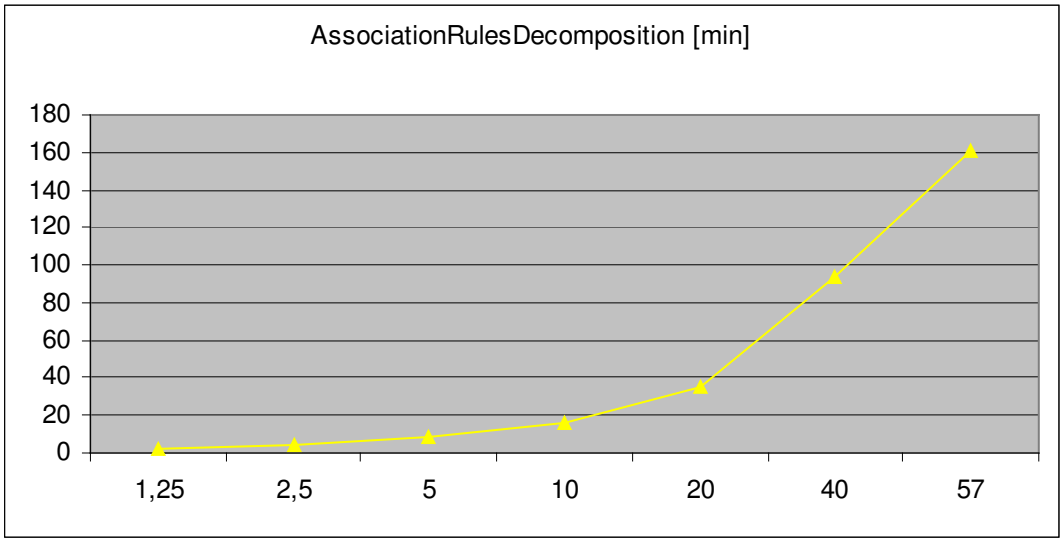

Figure 7: Graph of the calculation times of Table 7.

We see that until 10 million items the algorithm performs optimal but then it starts to deteriorate. The reason is that the *relative* decomposition size is shrinking for an increasing number of items. In fact, the smaller the decomposition size, the lower is the algorithm speed because the *local* algorithm is always faster than the decomposition method. Additionally, smaller decomposition sizes yield greater numbers of rules that are only local but not global optimal what in turn slows down the speed of the pruning phase of the decomposition algorithm. Hence both training and pruning phase of the decomposition algorithm become slower, if the decomposition size is increased.

We conclude: **Always use the highest-possible decomposition size in basket and sequence analysis decomposition algorithms**.

The same test for sequential decomposition is listed in Table 8.

#Items [Mio.]	#Transactions	#Rules	SequentialDecomposition [min]
1,25	919203	481	3
2,5	1870135	457	5
5	919203	469	9
10	1870135	431	20
20	3726692	380	43
40	7592484	252	105

Table 8: Influence of the sample size on the calculation time [min] of the sequence analysis decomposition algorithm.

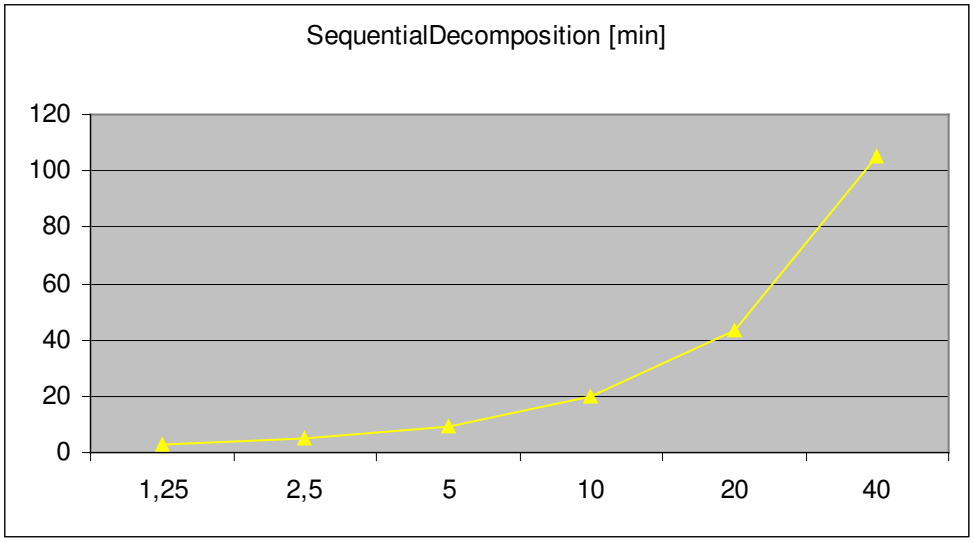

Figure 8: Graph of the calculation times of Table 8.

We see a similar behaviour as for the basket analysis decomposition.

3. E-Commerce Data

In this example, we consider data from a German E-Commerce shop and analyse the orders. The shop contains about 19000 different items. The number of purchased items is 195,338 with a number of 53,649 transactions. Thus, the average number of items per transaction is 3.6.

Though this data set is not very large and especially the number of items / transaction is relatively low, this data set is much more difficult to handle than the previous one. The reason (which cannot be discovered from the few information above) is that this data set contains a high number of very long transactions which are quite similar. This produces a huge search space of long sequences of items. By theory, the *AprioriTid* algorithm should perform very well for this type of data.

We show the results of basket analysis in Table 9.

minSupp	#Rules	AprioriSimple	AprioriOptimized	AprioriTID	AprioriHybrid
0,16	8	65	5	5	5
0,08	42	333	8	5	7
0,04	179		18	6	25
0,02	1242		EoM	12	EoM
0,01	40571			15	
0,005	4874223			84	

Table 9: Influence of the minimum support parameter on the calculation times [s] of the basket analysis algorithms.

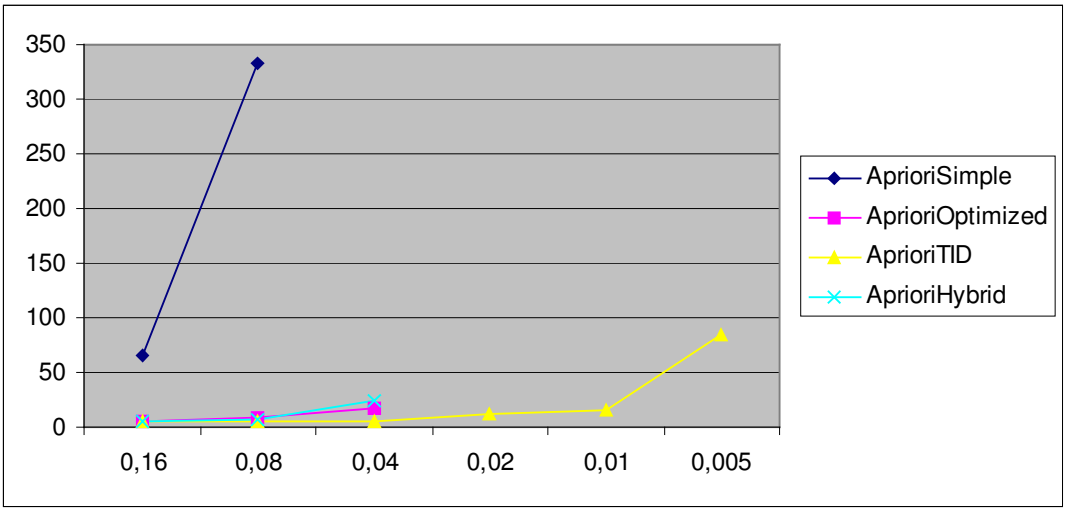

Figure 9: Graphs of the calculation times of Table 9.

Table 9 clearly confirms the assumption about the best performance of the *AprioriTid* algorithm with respect to both speed and memory.

4. Summary

We have seen that the basket and sequence analysis algorithms of XELOPES are very fast and can easily handle huge data volumes.

Not surprising, the commercial algorithms are by magnitudes faster than those of the GPL version. For basket analysis, the *AprioriTid* algorithms usually outperforms the *Apriori* algorithm. Although by theory the *Apriori* algorithm performs better for short item sequences than *AprioriTid,* the point is that the *AprioriTid* algorithm can be more efficiently implemented than *Apriori.* Thus, in practice *AprioriTid* is faster for both short *and* long sequences of items and should be generally prefered. (Only in a few applications the *AprioriOptimized* is slightly faster than *AprioriTid*, see also Table 5.) For the same reason, usually there is no reason to apply *AprioriHybrid.*

For sequence analysis tasks, the *Sequential* algorithm should replace *SequentialSimple* not only because of its higher speed but also for its ability to extract all *cycles.*

The decomposition algorithms *AssociationRulesDecomposition* and *SequentialDecomposition* can easily process large data volumes. However, because of the shrinking relative decomposition size they tend to lose their nearly-optimal asymptotical behaviour for very large datasets. For this reason, the decompositions size should be set as high as possible (depends on the available memory on the machine).

Prudsys is continuously improving the XELOPES algorithms. We are always looking for new benchmark problems – both synthetic and real-life. If you want to submit some datasets or, vice versa, test the commercial XELOPES algorithms, please contact

support@prudsys.com .