STUDIA INFORMATICA
Nr 1(10) Systemy i technologie informacyjne 2008

# Algorithm CFP-SFP with parallel processing

Mariusz Kujawiak[1]
[1] Institute of Computer Science, University of Podlasie,
ul. Sienkiewicza 51, 08-110 Siedlce, Poland

**Abstract**. Existing algorithms for finding association rules do not implement parallel processing. This paper proposes CFP-SFP (Creating Frequent Patterns with Set from Frequent Patterns algorithm with parallel processing. The research involves running CEP-SEP algorithm with one thread and a dozen or so threads that are executed simultaneously. The research was conducted on a computer with one processor and dual-core processor.

**Keywords.** Association rules, data mining, web logs, Apriori, AprioriTID, AprioriHybrid, FP-Tree

## 1 Introduction

The first algorithm for the discovery of association rules was presented in 1993, and another algorithm, SETM, which uses relational operators in the discovery process was also presented in the same year. In 1994, a very important paper by Agrawal and Srikant [1] appeared containing two new algorithms Apriori and AprioriTID for discovering strong binary association rules. Over the years, a lot of new algorithms for discovering association rules were based on these two algorithms. The common feature of all algorithms for discovering association rules is an identical general mechanism in which the algorithm works. This paper compares AprioriTID, AprioriHybrid, FP-Tree and a new proposed model (called CFP-SFP – Creating Frequent Patterns with Set from Frequent Patterns in the paper), which proved to be faster than the models invented so far.

Nowadays, multi-core processors are becoming more and more popular. Hence, it is necessary to create algorithms that make use of the opportunities multi-core processors provide. In most algorithms parallel computation is not included. The aim of this thesis was to propose a model for the CEP-SEP algorithm that allows for parallel processing of the patterns from the previous iteration.

## 2 What are association rules?

Discovering association rules is one of the most frequently used, non-oriented methods of knowledge discovery in web logs. The results this method gives are also the easiest to interpret and they present the information most people image as knowledge discovery. This process means finding associations between occurring groups of elements (attributes or values) in data sets. The associations have the following form: the occurrence of a certain pattern imply the occurrence of another pattern. For the rules that have been found, the values of coefficients defining the strength of a given rule and the probability of its successive occurrence are calculated.

Association rule has the following form: $A \Rightarrow B$. Set A is called the predecessor of the rule and Set B – the successor of the rule. Both left and the right side consist of logical (true or false) statements or sentences. Association rules are defined by the following coefficients:

- support – this is the ratio between the number of transactions from database D, which support a given set and the number of all transactions in database D. Formal definition of the support is as follows:
$$support(A)=|\{T \in D \mid A \subseteq T\}|/|D|$$
- If we make an assumption that A and B are the sets of elements from database D, it is possible to define the following property of support:
$$A \subseteq B \Rightarrow support(A) \geq support(B)$$
- Support for a given rule defines the part of the transaction in the database where a given dependence occurs and it is calculated in the following way:
$$support(A \Rightarrow B) = support(A \cup B)$$
- confidence – defines the probability with which the occurrence of the predecessor in the transaction implies the occurrence of the successor. Confidence is calculated using the following formula:
$$confidence(A \Rightarrow B) = support(A \cup B) / support(A)$$

Discovering association rules consists of the two main stages:

1. Finding all the frequent sets (sets with support that is no lesser than the minimum defined support) on the basis of input data and minimum support defined by the user. There are a lot of algorithms used in this stage, the most popular being Apriori, as well as other algorithms (which often derive from Apriori algorithm, namely AprioriTid, AprioriHybrid.
2. Finding rules that fulfil the defined criteria, based on the sets found in the first step. Association rules are most often used for shopping basket analysis and they allow to make decisions as regards e.g. promotions and discounts, advertisements and marketing activities or product distribution.

# 3 Algorithms CFP-SFP

## 3.1. Algorithms CFP-SFP for making association rules

Previous solutions focused on searching for data in the main set. The new model focuses on finding the association rules in frequent patterns create in the preceding iteration. This approach reduces the area of search to narrowed down data set, which results in shorter time of building association rules.

CFP-SFP algorithm:

```
1: L1 = {all frequent one-element sets together with a list of
transactions }
2: L1 = l∈ L1 | l.support ≥ minSupport}
3: for (k=2;|Lk-1|≠∅; k++) do begin
4: for (i=0; i<|Lk-1|;i++) do begin
5: for (j=1; j<|Lk-1|;j++) do begin
6: if (li∈Lk-1 and lj∈ Lk-1 such as |li∩lj|= k) then begin
7: c = li∪lj
8: if (c does not belong to Lk) then
9: Lk add c
10: end if
11: end if
12: end for
13: end for
14: end for
```

## 3.2. CFP-SFP with parallel processing

CFP-SFP algorithm with parallel processing divides dataset generated in the previous iteration into parts according to the defined parameter (number of threads). Each part is compared in the full set. This solution allows the execution of the algorithm on a few processors at the same time. Each thread searches a fragment of the set of patterns and full set of patterns.

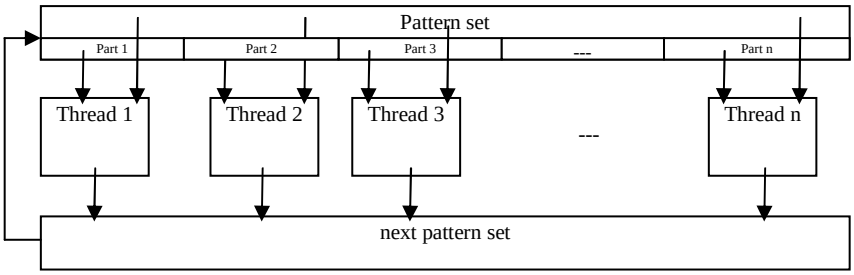



**Fig. 1.** Shows the concept of parallel processing process in CFP-SFP algorithm

# 4 An example showing how CFP-SFP works (min_sup=2)

**Table 1.** Input data

| TID | Elements |
|---|---|
| 1 | 1,3,5,7 |
| 2 | 8,9,2,4 |
| 3 | 6,8,9 |
| 4 | 1,3,5,7 |
| 5 | 8,3,5,6 |

The halt condition for this algorithm is obtaining an empty set for the next step. In the initial phase CFP-SFP algorithm creates a new data structure that is needed for subsequent iterations. Using the data in table 1 we obtain the following data set:

**Table 2.** The result after 1st iteration

| Pattern | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Attributes | 1,4 | 2 | 1,4 | 2 | 1,4,5 | 3,5 | 1,4 | 2,3,5 | 2,3 |
| The number of attributes | 2 | 1 | 2 | 1 | 3 | 2 | 2 | 3 | 2 |

Two elements – 2 and 4 are removed from table 2 as they do not meet the minimum support condition. It is only used in the first iteration. Table 3 represents frequent one-element patterns. As a result, the following data set, determining frequent one element patterns is obtained:

**Table 3.** The result after 1st step

| Pattern | 1 | 3 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| Attributes | 1,4 | 1,4 | 1,4,5 | 3,5 | 1,4 | 2,3,5 | 2,3 |
| The number of attributes | 2 | 2 | 3 | 2 | 2 | 3 | 2 |

Data prepared in this way are used in the next step, in which one element patterns are linked to make two-element patterns on the basis of attributes similarity e.g. element {1} and element {3} from table 3 have the same attributes {1,4} Therefore, it is possible to build a new two-element pattern {1,3} with the support equal 2. A given pattern may be inserted to the nest step only if the minimum support condition will be met. After execution of this step we obtain a list of frequent two-element patterns:

**Table 4.** The result after the 2nd step

| Pattern | 1,3 | 1,5 | 1,7 | 3,5 | 3,7 | 5,7 | 6,8 | 8,9 |
|---|---|---|---|---|---|---|---|---|
| Attributes | 1,4 | 1,4 | 1,4 | 1,4 | 1,4 | 1,4 | 3,5 | 2,3 |
| The number of attributes | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

In the next iteration only three-element patterns are searched for, in this case we look for the shared part of frequent patterns. We take the first pattern {1,3} from the data in table 4 and we compare it with other patterns:
{1,3} and {1,5} -> shared part {1}
It means that it is possible to build a pattern which takes the following form {1,3,5}, with an assumption that the minimum support condition is met. After this iteration we obtain the following frequent patterns:

**Table 5.** the result after the 2nd step

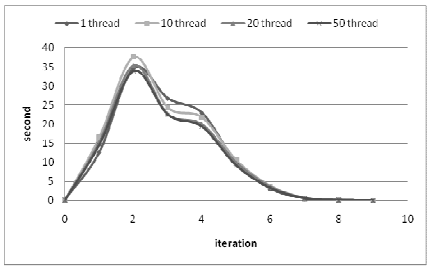
| Pattern | Attributes | The number of attributes |
|---|---|---|
| 1,3,5 | 1,4 | 2 |
| 1,3,7 | 1,4 | 2 |
| 1,5,7 | 1,4 | 2 |
| 3,5,7 | 1,4 | 2 |

The obtained set is further analysed in the search of a four element pattern. In this case it is possible to build one such element {1,3,5,7} with support equal 2 by attributes {1,4}.
The algorithm stops when it is impossible to make any pattern from the previous set.

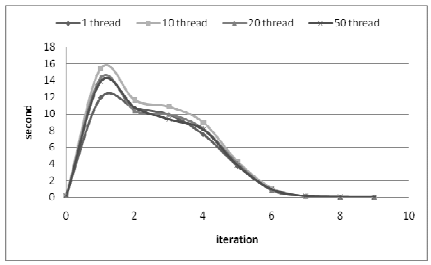
# 5 Research

## 5.1. Research conducted using one core processor

The comparative analysis was conducted using a computer with a processor Intel Celeron Mobile 1.86 MHz z 1024 MB RAM. The research was conducted using a part of the T10I4D100K.dat [1] file. The part of the file starts at the beginning of the file and contains a defined number of lines. A part of the file containing 15000 lines was used for this research.



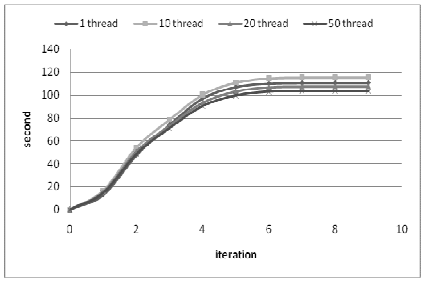

**Graph. 1.** Execution time distribution for CEP-SEP algorithm for support equal 15



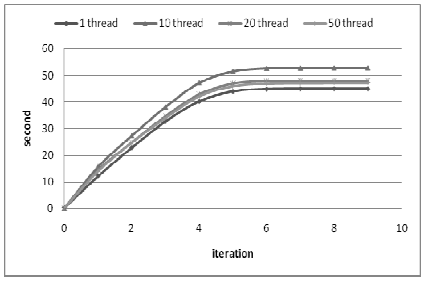

**Graph. 2.** Execution time distribution for CEP-SEP algorithm for support equal 25

[1] The dataset used for the purpose of the research is available at: http://fimi.cs.helsinki.fi/data/

Figures 1 and 2 show the execution of the multi-thread algorithm conducted using one core processor. It may be noticed that there is a slight decrease in efficiency of the algorithm compared with the result for one thread. This situation is caused by the allocation of processor time for each of the threads, which causes the delay in the execution of the algorithm. However, it may also be observed that the more threads the smaller the difference in efficiency.




**Graph. 3.** Execution time for CEP-SEP algorithm for support equal 15

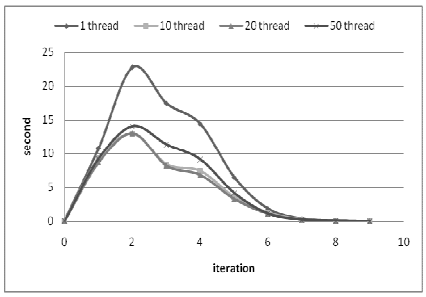



**Graph. 4.** Execution time for CEP-SEP algorithm for support equal 25

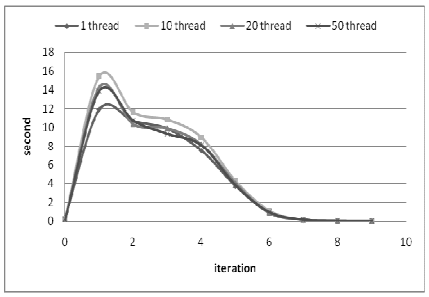
Figures 3 and 4 clearly show the increase in execution time of the algorithm. The differences in the execution become irrelevant if a greater number of threads is defined.

### 5.2. Research using dual-core processor

The comparative analysis was conducted using a computer with a processor Intel Core2Duo 1.86 MHz z 1024 MB RAM. Research was conducted using a part of the T10I4D100K.dat file. The part of the file starts at the beginning of the file and contains a defined number of lines. A part of the file containing 15000 lines was used for this research.



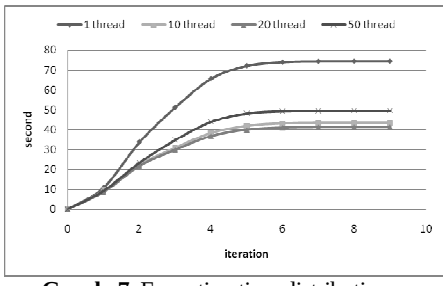

**Graph. 5.** Execution time distribution for CEP-SEP algorithm for support equal 15



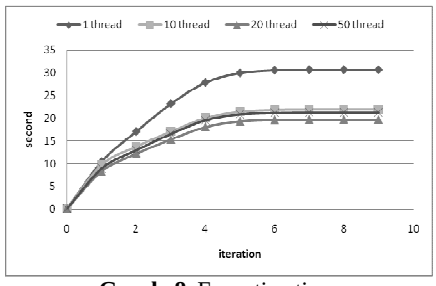

**Graph. 6.** Execution time distribution for CEP-SEP algorithm for support equal 25

Figure 5 shows that if the number of threads equals 50 it does not shorten the execution of the algorithm. To the contrary, the execution takes up more time. The most favourable number of threads is equal to 20. Then, iterations 2,3 and 4 result in the biggest growth in efficiency. This results from the fact that there are a lot of

patterns to search in these iterations, which takes up a lot time in case of single-thread applications.




**Graph. 7.** Execution time distribution for CEP-SEP algorithm for support equal 25




**Graph. 8.** Execution time for CEP-SEP algorithm for support equal 25

Multi-core processors show significant increase in efficiency. However, too big number of threads may slow down the generation of association rules.

## 6 Conclusions

Parallel processing model in CEP-SEP algorithm allows to increase the efficiency using multi-core processors. The research also show the increase in the efficiency using one-core processor. However, this increase in efficiency is not always present. The proposed solution is an improvement of the algorithm for the new generation of processors, namely multi-core processors, which have recently become extremely popular. Further research on the algorithm will focus on the minimization of system memory use during the execution of the algorithm.

## References

[1] Agrawal R., Imielinski T., Swami A.N. "Mining Association Rules between Sets of Items in Large Databases." SIGMOD. June 1993, 22(2):207-16.

[2] Agrawal R., Srikant R. "Fast Algorithms for Mining Association Rules", VLDB. Sep 12-15 1994, Chile, 487-99, ISBN 1-55860-153-8.

[3] Mannila H., Toivonen H., Verkamo A.I. "Efficient algorithms for discovering association rules." AAAI Workshop on Knowledge Discovery in Databases (SIGKDD). July 1994, Seattle, 181-92.

[4] Ezeife C.I., Su Y., Mining Incremental Association Rules with Generalized FP-tree, Proceedings of the Fifteenth Canadian Conference on Artificial Intelligence (AI 2002), Calgary, Canada (May 25-29, 2002), Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin Heidelberg 2002.

[5] Kujawiak M., Kłopotek M.A., Wykrywanie reguł związków w plikach Web Logów, Technologie przetwarzania danych, TPD 2007, September 2007, Wydawnictwo Politachnika Poznańska, ISBN 978-83-7143-349-8.