Partition-Based Frequent Closed Pattern Miner

Anu Soni, Mukta Goel and Rohit Goel

Abstract Frequent closed pattern (FCP) mining has been an important step in data mining research. This paper introduces an algorithm to deal with the problem of finding out (FCP) from a given set of transactions. The miner works on a parallel approach based on compact matrix division to partition the data set. To filter these subtasks two methods are adopted (1) transaction set redundancy removal method and (2) itemset redundancy removal method. Mining of filtered subtasks are done separately. Consolidated result obtained from mining of these filtered independent partitions show all FCPs present in transactions.

Keywords Frequent closed pattern · Mining algorithm · Parallel mining

1 Introduction

Frequent patterns (FP) are itemsets, substructures, or subsequences that appear in a data set with occurrence more than or equal to a user-specified threshold. Frequent patterns play a necessary role in showing interesting relationships among data such associations, correlations, causality as well as it has its own importance in data indexing, clustering, association-based classification, and other data mining tasks as well [1, 2, 3]. However, the number of frequent patterns can be too large to handle and if data is dense or low threshold frequency is used then this complication become vanquish. To overcome this problem, a new term is introduced which is

A. Soni (✉) · M. Goel · R. Goel
The Technological Institute of Textile and Sciences, Bhiwani, India
e-mail: anusunalia@gmail.com

M. Goel
e-mail: rishu.muk@gmail.com

R. Goel
e-mail: rohit_160@rediffmail.com

S.C. Satapathy et al. (eds.), *Proceedings of International Conference on ICT for Sustainable Development*, Advances in Intelligent Systems and Computing 409, DOI 10.1007/978-981-10-0135-2_45

known as frequent closed patterns (FCP) [1, 4]. FCP is a pattern for which no superset exists which has occurrence more than or equal to that pattern [1– 3, 5]. Some notable FCP mining schemes include APRIORI [1], PATTERN–GROWTH APPROACH FOR FCP MINING [1], CLOSET+ [6], C-MINER [4], D-miner [7], and SLIDING WINDOW BASED ALGORITHM. Despite the fact that these algorithm work for small amount of data set well but when it comes to handle large data set, either they work very inefficiently or they do not work. Here a partition-based FCP miner is introduced that mine FCP from dense data effectively and increasingly. This algorithm is different from previous ones because of the following benefits: (1) it handles duplicacy in starting and does not let data go duplicate in further processing, (2) it compresses and divides data based upon compact matrix division strategy, and (3) it divide partitions mined independently thus impose parallelism.

2 Related Work

There are a number of previous proposed actions which deals with FP mining. But numbers of FPs are very large and difficult to handle. Thus procedures for FCP mining came into existence because number of FCPs are less and easy to handle. Few important algorithms proposed so far are PRIORI [1], PATTERN–GROWTH APPROACH FOR FCP MINING [1], and CLOSET+ [6]. These algorithms are good and able to find all FCPs, but these methods cannot work for dense data set. FCP mining algorithm CARPENTER is designed to work with large columns and small rows. CARPENTER combines the depth first with row enumeration strategy with some efficient search techniques, which results in a scheme that performs traditional closed pattern (CP) mining. Another algorithm B-MINER [4] is designed to divide the data with the help of base row projections technique and then mined independently each divided pattern. C-MINER [4] also used same strategy, but division is based upon compact matrix division method. Although these two algorithms are quite good even in case of dense data, they do not handle row duplicacy in starting, thus inefficient. D-miner [7] works for dense data set. However, the efficiency of D-miner highly depends on the minimum number of the data set's rows/columns containing "0." More the zero less is the efficiency. Thus, when the data set has a relatively large number of rows and columns, D-miner loses its beneficiation. Computational cost of data mining is very high, so a lot of attempts are made to design parallel and distributed algorithms. A lot of extensions are made of algorithms like APRIORI, ParEclet, etc., to convert them into parallel algorithm but these are still very costly. Till now attempts to find a good parallel algorithm for FCP mining is going on.

3 Definitions

Key concepts used in frequent pattern mining are:-

Transaction database	If there are n number of transactions and m are number of items available then $T = \{t_1, t_2. \ . \ .t_n\}$ be a transaction database, here each $t_j \in \ T, \forall \ j = \{1 \ \ldots . \ n\}$
Itemset	Each transaction consists of a set of items, say $t_j = \{i_1, i_2, i_3 \ .. \ . \ i_m\}$.
Pattern	A set $P \subseteq \ t_j$ is called a pattern.
Pattern length	Number of items contained in a pattern is called Pattern length.
Support	Number of transactions in which a pattern appears is known as support of pattern.
Frequent pattern	A pattern P is defined as frequent pattern if its support is at least equal to the minimum threshold.

4 Proposed Work

Here an algorithm is introduced which not only mine frequent pattern on the basis of threshold will support, but also on basis of threshold pattern length. This algorithm first takes transaction in which frequent pattern mining is performed. This input is then converted into binary matrix such that transactions are represented by rows, and items are represented by columns. Suppose if a transaction t_j contains item i_k then cell c_{jk} of matrix contains value "1" else it contains "0." All rows of matrix are scanned properly to find out identical rows in matrix; if there are any identical rows present in matrix then they are collectively treated as a single row rather than multiple rows. Now data of this binary matrix is compressed using clustering. This compressed data is divided into various tasks using zero removal principle. Now these divided tasks are scanned properly to check if there is any duplicate subtasks exist, if exist then only one of them is taken. These subtasks are mined independently. Result obtained from mining of every subtask is collected together to produce overall result. Control of flow throughout the process in Flow charts 1 and 2.

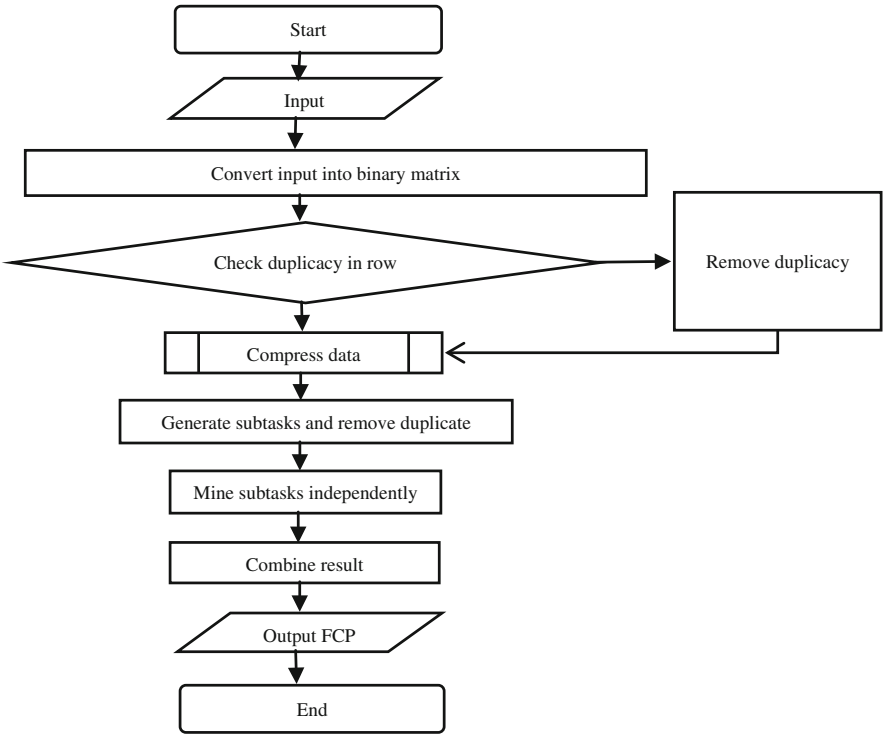
Working of algorithm

Suppose 9 transactions and 5 items are given in Table 1 and we have to find number of FCPs to decide a market strategy for it.

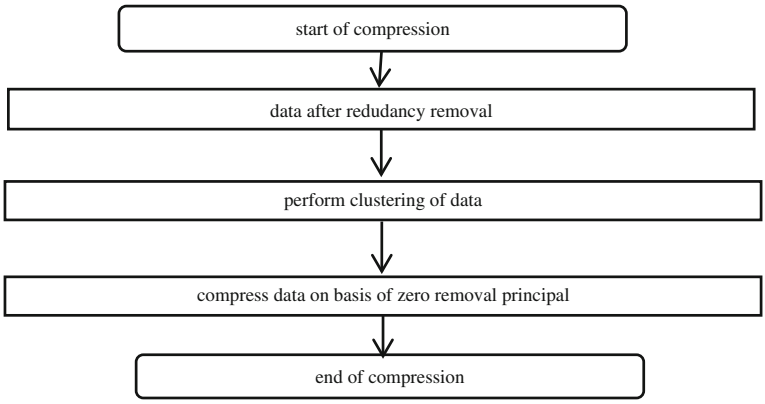
Various steps involved in finding FCPs

Step 1 Conversion into binary matrix

First of all whole data is converted into a Boolean context. Binary conversion of Table 1 is shown in Table 2.



Flowchart 1 Partitioned-based FCP-miners



Flowchart 2 Compression technique used

Table 1 Sample data set

TID	List of items
t_1	i_1, i_2, i_5
t_2	i_2, i_4
t_3	i_2, i_3
t_4	i_1, i_2, i_4
t_5	i_1, i_3
t_6	i_2, i_3
t_7	i_1, i_3
t_8	i_1, i_2, i_3, i_5
t_9	i_1, i_2, i_3

Table 2 Binary data set

r/c	i_1	i_2	i_3	i_4	i_5
t_1	1	1	0	0	1
t_2	0	1	0	1	0
t_3	0	1	1	0	0
t_4	1	1	0	1	0
t_5	1	0	1	0	0
t_6	0	1	1	0	0
t_7	1	0	1	0	0
t_8	1	1	1	0	1
t_9	1	1	1	0	0

Table 3 Removal of duplicate rows

C	i_1	i_2	i_3	i_4	i_5
t_1	1	1	0	0	1
t_2	0	1	0	1	0
t_4	1	1	0	1	0
t_8	1	1	1	0	1
t_9	1	1	1	0	0
t_3, t_6	0	1	1	0	0
t_5, t_7	1	0	1	0	0

Step 2 combine identical rows together

Rows which are exactly same are combined, i.e., multiple rows are considered as one, so there is no need to process them differently throughout the process. Data set after removal of duplicate rows is shown in Table 3.

Step 3 Perform rowwise clustering

Now rowwise clustering is performed. Any of the clustering techniques [8] can applied, here those rows are clustered together in which at least half the number of total items are similar. To represent a cluster, item wise ORing of all rows in a

Table 4 Clustering

r/c	i_1	i_2	i_3	i_4	i_5
$c_1(t_1)$	1	1	0	0	1
$c_2(t_2, t_4)$	1	1	0	1	0
$c_3(t_3, t_5, t_6, t_7)$	1	1	1	0	0
$c_4(t_8, t_9)$	1	1	1	0	1

Table 5 Generators

$G(A, B)$
$G_1(c_1, i_3i_4)$
$G_2(c_2, i_3i_5)$
$G_3(c_3, i_4i_5)$
$G_4(c_4, i_4)$

cluster is taken, i.e., $c(t_a, t_{b,\ldots..}t_z) = t_a \vee t_b \vee \ldots t_z$. After clustering of transactions in Tables 3 and Table 4 is obtained.

Step 4 Generation of generators

If G represents generator, A indicates clusters, and B indicates itemset then $G(A,B)$ is said to be a generator. $G(A,B)$ will take only those values for which A has 0 in Bth column. For Table 3 generators are given by Table 5.

Sorting of generators on the basis of number of items

As generators are used to divide given task by removing useless zeros, we take generator which has largest number of items, by which we are able to remove more zeros in the starting. So sorting of generators is performed in descending order of number of items, generators after sorting is completed.

Step 5 Partition into subtasks

With the help of generators, a binary tree is made. A node is represented by (cluster set, itemset). Root node contains all clusters and items. Now generators are applied one by one to generate left and right child. A left child derived from a node contains all clusters and itemset which are in node except the clusters present in generator. A right child derived from a node contains all clusters and itemset which are in node except the items present in generator. Child having support or pattern length less than threshold support or threshold pattern length are dropped. Minimum threshold support is 2 and minimum pattern length threshold is 2. Tree generated using Tables 4 and 6 is shown in Fig. 1. Here leaf nodes are represented by double

Table 6 Sorted generators

$G(A, B)$
$G_1(c_1, i_3i_4)$
$G_2(c_2, i_3i_5)$
$G_3(c_3, i_4i_5)$
$G_4(c_4, i_4)$

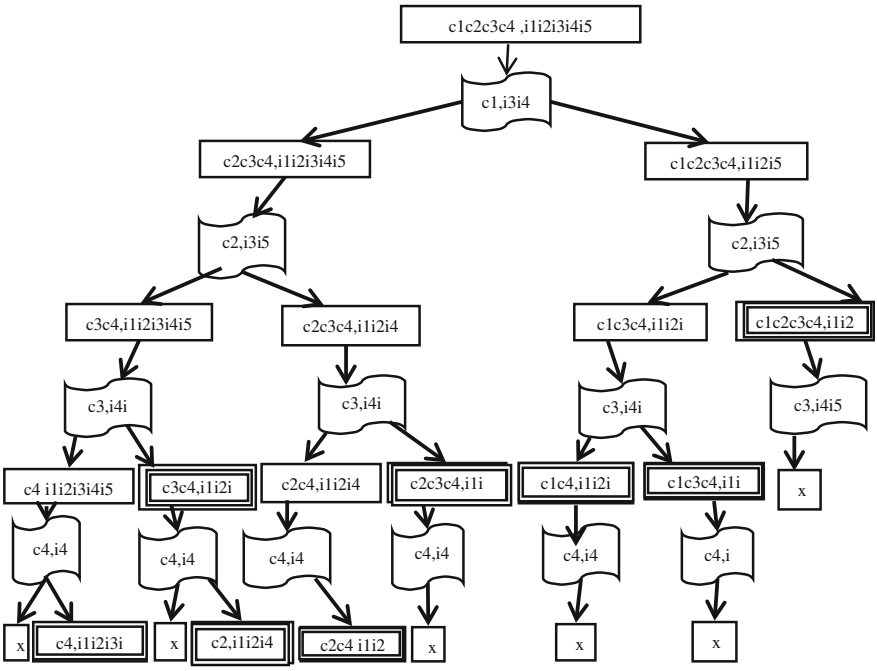


Fig. 1 Subtasks generators

Table 7 Subtasks

Cluster, itemset
c_4, $i_1i_2i_3i_5$
c_3c_4, $i_1i_2i_3$
c_2, $i_1i_2i_4$
c_2c_4, i_1i_2
$c_2c_3c_4$, i_1i_2
c_1c_4, $i_1i_2i_5$
$c_1c_3c_4$, i_1i_2
$c_1c_2c_3c_4$, i_1i_2

boundary rectangles. Subtasks obtained by this process are given in Table 7. Sometimes, more than one subtasks contain the same clusters, but items contained by them can be a superset of other and vice versa. This duplicacy need to be handled in subtasks, otherwise they move through all process and make algorithm inefficient. To make algorithm efficient, two strategies are used, and these are (**1**) **Transaction set redundancy removal method and (2) Itemset redundancy removal method.**

Table 8 Subtasks after TRM

Cluster, itemset
c_4, $i_1i_2i_3i_5$
c_3c_4, $i_1i_2i_3$
c_2, $i_1i_2i_4$
c_2c_4, i_1i_2
$c_2c_3c_4$, i_1i_2
c_1c_4, $i_1i_2i_5$
$c_1c_3c_4$, i_1i_2
$c_1c_2c_3c_4$, i_1i_2

Transaction set redundancy method (TRM) When two or more than two subtasks contains exactly same cluster set but itemset contained by one of them is superset of itemset contained by other subtasks then subtask containing superset of itemset is considered and other are dropped. Table 7 do not contain such subtask, so after applying TRM on Table 7 we get Table 8.

Itemset redundancy method (IRM) When two or more than two subtasks contains exactly same itemset but cluster set contained by one of them is superset of cluster set contained by other subtasks then subtask containing superset of cluster set is considered and other are dropped. After applying IRM on Table 8 we get Table 9.

Table 9 Subtasks considered

Subtasks	Cluster involve	Column set	Item set	Support	No. of items involve
s_1	c_4	$i_1i_2i_3i_5$	t_8t_9	2	4
s_2	c_3c_4	$i_1i_2i_3$	$t_3t_5t_6t_7t_8t_9$	6	3
s_3	c_2	$i_1i_2i_4$	t_2t_4	2	3
s_4	$c_1c_2c_3c_4$	i_1i_2	$t_1t_2t_3t_4t_5t_6t_7t_8t_9$	9	2
s_5	c_1c_4	$i_1i_2i_5$	$t_1t_8t_9$	3	3

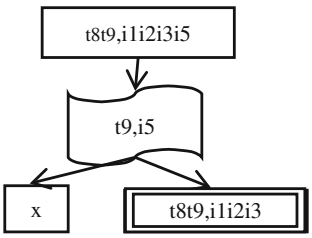


Fig. 2 Mined tree of subtask s_1

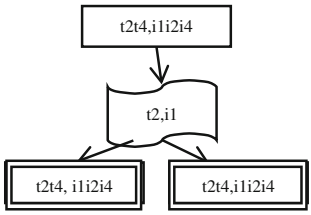


Fig. 3 Mined tree of s_3

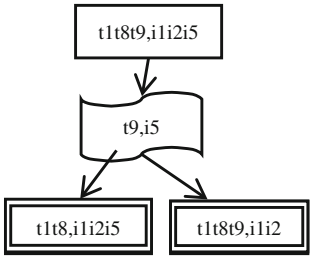


Fig. 4 Mined tree of subtask s_5

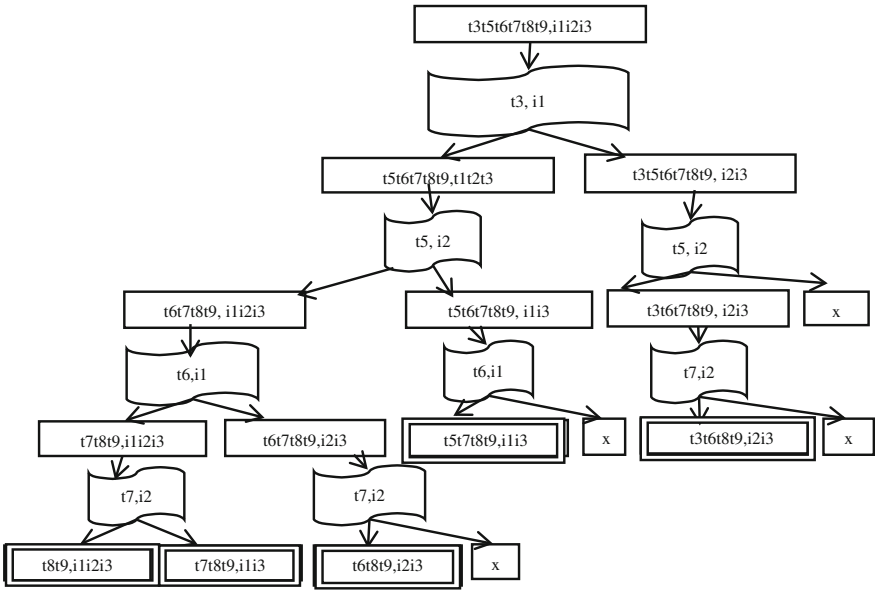


Fig. 5 Mined tree of subtask s_2

Step 6 mining of subtasks

Now these subtasks are mined independently. To mined subtask a binary tree is created. Root node of binary tree is subtask itself. Left and right node of tree is generated with the help of generator. If node is (T,I) such that T is set of transactions and I is set of items, then generator contains those rows and items for which row contain 0 value at corresponding item if rows contained in node has variance in value at that item position. Left child generated from a node contains all rows and items which are in node except row contained in generator. Right child generates from a node contains all rows and items which are in node except items contained in generator. Child having support or pattern length less than threshold support or threshold pattern length are dropped. Mined tree obtained from all subtasks are given by Figs. 2, 3, 4, 5 and 6.

Here leaf nodes of these mined trees represent all FCPs in corresponding transactions. FCPs obtained from these mined trees are given in Table 10.

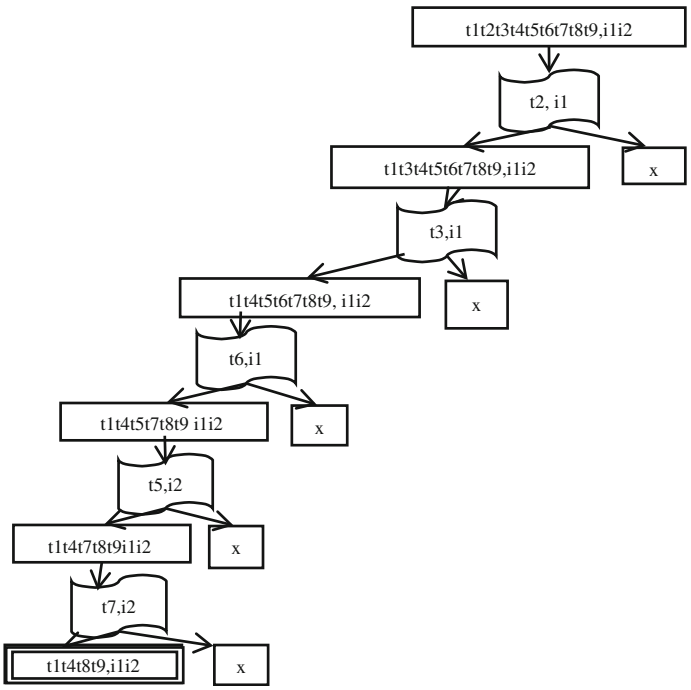


Fig. 6 Mined tree of subtask s_4

Table 10 FCPs generated

Row set	FCP
$t_1t_4t_8t_9$	i_1, i_2
t_8t_9	$i_1i_2i_3$
$t_7t_8t_9$	i_1i_3
$t_6t_8t_9$	i_2i_3
$t_5t_7t_8t_9$	i_1i_3
$t_6t_8t_9$	i_2i_3
t_2t_4	i_2i_4
t_1t_8	$i_1i_2i_5$
$t_1t_8t_9$	i_1i_2
t_8t_9	$i_1i_2i_3$

Table 11 Results

Resultant row set	FCP
t_8t_9	$i_1i_2i_3$
$t_1t_4t_8t_9$	i_1i_2
$t_5t_7t_8t_9$	i_1i_3
$t_6t_8t_9$	i_2i_3
t_2t_4	i_2i_4
t_1t_8	$i_1i_2i_5$

5 Results

Result can be obtained from Table 10 by removing duplicate FCPs. FCPs obtained after removal of duplicate FCPs are given in Table 11.

6 Conclusion

In this paper, partition-based FCP miner is proposed which handle identical rows as one row in the beginning. The key idea is to compress data and partition it into various subtasks. These subtasks are filtered using TRM and IRM. Mining of filtered subtasks is done independently and parallely. This algorithm facilitates parallel mining even in case of dense data set with high efficiency. Transaction wise results are obtained which use efficiently in market strategy designing.

References

1. Han, J., & Kamber, M. *Data mining concept and techniques*, 3rd edition.
2. Mannila, H., Toivonen, H., & Verkamo, A. I. (1994). Efficient algorithms for discovering association rules. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI '94) Workshop Knowledge Discovery in Databases (KDD '94)* (pp. 181–192), 1994.
3. Zaki, M., Parthasarathy, S., Ogihara, M., & Li, W. (1997). New algorithms for fast discovery of association rules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD '97)* (pp. 283–286), 1997.
4. Ji, L., Tan, K. L. Member, IEEE Computer Society, & Tung, A. K. H. (2007). Compressed hierarchical mining of frequent closed patterns from dense data sets. *IEEE Transactions on Knowledge and Data Engineering*, *19*(9), September 2007.
5. Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: Current status and future directions. Received: 22 June 2006/ Accepted: 8 November 2006/ Published online: 27 January 2007, Springer Science + Business Media, LLC 2007.
6. Wang, J., Han, J., & Pei, J. (2003). CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)* (pp. 236–245), 2003.
7. Besson, J., Robardet, C., & Boulicaut, J.-F. (2004). Constraint-based mining of formal concepts in transactional data. In *Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)* (pp. 615–624), 2004.
8. Soni, A., Goel, M., & Goel, R. (2015). Comparative study of various clustering techniques. In *Proceedings of the National Conference on Innovative Trends in Computer Science Engineering (KDD '94)* (pp. 148–150), 2015.