Faculty of Electronics and Information Technology

**Warsaw University of Technology**

**EDAMI Project**

**Subject**

Implementation of “*dEclat” algorithm in Python language*

|  |  |
| --- | --- |
| **Authors** | **Repository** |
| Matysiak Piotr  Tureczek-Zakrzewska Maja | <https://github.com/pmatysiakq/edami-dEclat> |

**Table of contexts**

[Phase 1 2](#_Toc91537455)

[1. Selection of implemented algorithm 2](#_Toc91537456)

[2. Description of selected algorithm 2](#_Toc91537457)

[3. Pros and cons analysis 3](#_Toc91537458)

[4. Pseudo-code of the algorithm 3](#_Toc91537459)

[5. Implement first version of an algorithm 3](#_Toc91537460)

[6. Testing 6](#_Toc91537461)

[6.1 Experiment #1 6](#_Toc91537462)

[6.2 Experiment #2 7](#_Toc91537463)

[Phase 2 12](#_Toc91537464)

[Phase 3 12](#_Toc91537465)

[References 12](#_Toc91537466)

# Phase 1

## Selection of implemented algorithm

Our goal was to implement dEclat algorithm in Python language. In the first phase we had to implement a simple version of the algorithm. Our implementation should be based on the Philippe Fournier-Viger Java implementation [1] and paper written by Mohammed Zaki and Karam Gouda [2].

## Description of selected algorithm

dEclat algorithm was introduced by Mohammed Zaki in 2001 in “Fast Vertical Mining Using Diffsets” paper. Being one of vertial mining alghoritm and using the diffset format (the difference of two sets), dEclat alghoritm has drastically reduced the running time and memory usage of the Eclat algorithm, thus Eclat algorithm using diffset format is called dEclat and it is considered to be more efficient variation of the Eclat algorithm.

Both mentioned algorithms are used for discovering frequent item-sets in a transactional database. The main difference, between them, is a structure which is implemented on top of them - dEclat uses "difflists" to represent differential lists of transaction identifiers, which means that it stores information about transactions which are not present in the set.

This algorithm requires two input parameters:

* data in the form of "transactions"
* minimal support (min-supp) parameter, which indicates a threshold of acceptance.

In simplified terms, we can say that this algorithm is all about computing the support of the item-sets and comparing with min-supp parameter. As output, we expect a group of items, which are called "frequents item-sets".

A group of items is frequent when they are present together in at least "min-supp + 1" transactions, which means that their support is greater than min-supp parameter.

## Pros and cons analysis

The limitations of the algorithm are that the items in transactions are assumed to be sorted by lexicographical order and each transaction in a set is unique- there are no duplicate transactions in a single itemset.

It is neither the only, nor the best algorithm to find frequent item-sets, but it is definitely one of the best. The advantage of dEclat is due to vertical representation of the data - it uses “depth first search” for discovering frequent item-sets.

The main advantage of the vertical format is support for fast frequency counting via intersection operations on transaction ids (tids) and automatic pruning of irrelevant data. The main problem with these approaches is when intermediate results of vertical tid lists become too large for memory, thus affecting the algorithm scalability

The most important advantage of dEclat algorithm is that it's faster than many of other algorithms and at the same time needs less memory comparing with e.g. Apriori or Eclat algorithms.

Looking at disadvantages, we may point out, that dEclat algorithm is suitable for a dense database, but degrades with spare database, and it needs to switch between tidset and diffset for a sparse database

## Pseudo-code of the algorithm

dEclat is a simple algorithm, thus its pseudo-code is also short. Pseudo code was copied from [2]. In the code we define a few parameters:

|  |  |
| --- | --- |
|  | Initial data-set |
|  | Element present in |
|  | Union of and candidate item-sets |
|  | Difflist of element |
|  | Support of element |
|  | New data-set composed of elements which are frequent |
|  | Minimal support indicating if candidate is frequent or not |

Pseudo-code of dEclat algorithm [2]:

|  |
| --- |
|  |

## Implement first version of an algorithm

Initially we have implemented dEclat algorithm only based on the article, but this implementation wasn’t efficient enough and it could be hard for us to improve that implementation in Phase 2 of the project. That’s why we have reimplemented our algorithm. Now it’s mainly based on Philippe Fournier-Viger Java implementation. This new implementation can be found in git repository:

|  |
| --- |
| https://github.com/pmatysiakq/edami-dEclat/tree/master/dEclatV2 |

Our implementation consists of few python files:

* parser/**ParseTTData.py**

This file implements class **Parser**. The goal of this class is to create input file for class **TransactionalDatabase** which is described below.

Before converting raw tweets into transactional database, we had to collect some datasets from Twitter. We have used two different sources of data:

1. **Live Twitter data downloaded using Twitter Developer API** – we have implemented methods which allow us to fetch tweets by users or tags and save as text files. Example raw tweets looks like shown below. One line represent single tweet

|  |
| --- |
| BREAKING NEWS: Australia announces that it may soon begin charging “irresponsible” unvaccinated residents who believe “rubbish” on the internet more for COVID hospital care in order to stop sticking responsible taxpayers with their hospital bills. RT IF YOU SUPPORT THIS!  They lied. I’m so glad I did not get to the “vaccine”, which does NOTHING to stop the spread of Covid. The rest of you are clowns for allowing them to get away with such blatant lies. https://t.co/16AGxzvbWW |

1. **Dataset from Kaggle website** – we downloaded “All Trump’s Twitter insults (2015-2021)” database. Then we extracted the text of the tweets and saved in raw format, just like described above.

To make described data sources possible, we have implemented few static methods. It may be treated as prerequisites for moving on to data Parsing. When we have collected necessary data in raw format we can move on to Parsing.

The instance of class **Parser** takes a file with raw tweets as an input. Then it makes use of python built-in methods and regex to clean up data an divide raw tweets into transactions. Each transaction consists of at least 2 words, we don’t allow duplicate words in the single transaction.

For implementation details see github repository, below we present 4 steps which were performed to generate input data files for dEclat algorithm. Steps 1-3 allow us to create files with raw tweets and step 4 creates file which will be provided for **TransactionDatabase** instance.

|  |
| --- |
| **if \_\_name\_\_ == "\_\_main\_\_":**  **# STEP 1**  ***# Use tweepy to fetch 200 newest "elonmusk" tweets and save* as *tweets\_raw/tweets-elonmusk.txt***  Parser.get\_200\_tweets\_by\_user("elonmusk")  **# STEP 2**  ***# Use tweepy to fetch some tweeter data and save as tweets\_raw/db-covid.txt***  words = ["covid", "vaccine", "covid-19", "quarantine", "restrictions", "phizer", "moderna", "astrazeneca",  "fake covid", "wuhan", "coronavirus", "health", "pandemic", "virus", "corona", "stayhome", "lockdown",  "unvaccinated", "omicron", "sars-cov-2", "death", "antibodies", "plandemic"]  Parser.search\_tweets\_by\_tags(words, "covid")  **# STEP 3**  ***# Parse trump tweets downloaded from Kaggle and save as tweets\_raw/trump-tweets.txt***  Parser.parse\_trump\_dataset()  **# STEP 4**  ***# From raw tweets create databases which will be provided as input for dEclat algorithm***  input\_path = r"tweets\_raw/db-covid.txt"  output\_filename = "../input\_data/covid-transactions.txt"  Parser(input\_file=input\_path, output\_file=output\_filename)  input\_path = r"tweets\_raw/tweets-elonmusk.txt"  output\_filename = "../input\_data/elonmusk-transactions.txt"  Parser(input\_file=input\_path, output\_file=output\_filename)  input\_path = r"tweets\_raw/trump-tweets.txt"  output\_filename = "../input\_data/trump-transactions.txt"  Parser(input\_file=input\_path, output\_file=output\_filename) |

* db/**TransactionDatabase.py**

This file implements class **TransactionDatabase**. Created database is provided as input for dEclat algorithm along with **min\_sup** parameter

We define few methods for ***class TransactionDatabase*** these two are the most important:

def parse\_data\_from\_file(self, file\_path):

# As input we provide text file which consists of 2 parts

# 1st part is transactional database

# 2nd part represents dictionary which will be used to translate integer #

# items into real words

Excerpt from input file is presented below. One line represents single transaction. This input file is generated using **class Parser** which was described earlier.

|  |
| --- |
| [omitted]  2752 170 915 6140  1031 6166 23 6167 6168 6170 6169 157 289 1191 3248 2356 309 182 439 1465 827 2752 986 1375  2752 1057 4967 424 903 6175 26 6171 6172 6173 6174 159  ==== DICTIONARY ====  {"1": "announces", "2": "australia", "3": "begin", "4": "believe", "5": "bills",…omitted} |

def translate\_integers\_into\_words(self, file\_path):

# Used to translate frequent item-sets represented by integers into frequent

# item-sets which consist of real words. Input file is generated by dEclat

# algorithm

Above method allow us to translate found frequent itemsets like shown in table below:

|  |  |  |
| --- | --- | --- |
| 14 #SUP: 1142  21 #SUP: 961  29 #SUP: 1667  62 #SUP: 953  699 #SUP: 1159  2922 #SUP: 1251  699 2922 #SUP: 1042 |  | they #SUP: 1142  have #SUP: 961  that #SUP: 1667  with #SUP: 953  news #SUP: 1159  fake #SUP: 1251  news fake #SUP: 1042 |

* algorithms/DEclat.py

**# dEclat implementation. TODO DESCRIBE dECLAT ALGORITHM**

* data\_visualization/**Ploter.py**

This file implements methods which allow us to make use of data we generated and visualize that data using “matplotlib” and “network” packages.

## Testing

In Phase 1 of this project we would like to perform 2 experiments.

1. The goal of first experiment is to test out implementation of dEclat algorithm on simple input data. We will make use of transactional database, which we will generate on our own. To make sure that our implementation is working properly, we will use already mentioned Java dEclat implementation and we will compare the results.
2. The second experiment will be more sophisticated. We will use dataset mentioned in section 5 - “All Trump’s Twitter insults (2015-2021)” which is available on Kaggle website. The goal is to preprocess that data, prepare transactional database, make few frequent itemsets mining iterations - providing different minimal support, collect some parameters and visualize results. If possible, we will try to infer something directly from frequent itemsets.

In the Phase 2 we will repeat the experiments using improved dEclat implementation and then we will write conclusions.

## Experiment #1

As already said, we want to make sure that our dEclat implementation is working properly. To test algorithm we have used ***./dEclatV2/input\_data/elonmusk-transactions.txt*** input file – which can be found in github repository. Excerpt:

|  |
| --- |
| 1 2 3 4 5 6  7 8 9 10 11  12 13 14 15 16 17 18  19 20 21 22 23 24 25 26 27 28 29 30  32 33 31  34 35 36 37 38 39 40 41  24 41 42 43  8 44 45 46 47 48 49 50 51 52 53  […] |

Here we describe how we performed 1st experiment. It’s listing from dEclat.py file.

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_":  ***# We define path to the input file***  input = "../input\_data/elonmusk-transactions.txt"  ***# We define path to the output file***  output = "../output/output-experiment-1.txt"  ***# We set-up min-supp parameter - 5%***  min\_supp = 0.05  ***# We create TransactionDatabase instance using input file***  database = TransactionDatabase(file\_path=input)  ***# We create dEclat instance and run algorithm for specified parameters***  declat = DEclat()  declat.run\_algorithm(output=output, database=database, minsupp=min\_supp)  ***# We print out statistics***  declat.print\_stats()  ***# Additionally, we translate found frequent item-sets to the actual words***  database.translate\_integers\_into\_words(file\_path=output) |

As output we get two text files – found FIs with items as integers and FIs with actual words.

|  |  |
| --- | --- |
| 10 #SUP: 11  27 #SUP: 13  30 #SUP: 13  47 #SUP: 8  67 #SUP: 7  72 #SUP: 14  90 #SUP: 9  252 #SUP: 8 | with #SUP: 11  that #SUP: 13  will #SUP: 13  like #SUP: 8  still #SUP: 7  this #SUP: 14  tesla #SUP: 9  just #SUP: 8 |

We also printed-out message which describes the run of the algorithm:

|  |
| --- |
| ============= dECLAT Based on SPMF Java implemetation - STATS =============  Transactions count from database: 135  Frequent itemsets count: 8  Total time ~ 0.09980201721191406 ms  Maximum memory usage: 0.159989 mb  =========================================================================== |

When we compare our results with Java implementation we can see that our implementation is working properly. We can say that Experimet #1 PASSED the test.

|  |  |
| --- | --- |
| 10 #SUP: 11  27 #SUP: 13  30 #SUP: 13  47 #SUP: 8  67 #SUP: 7  72 #SUP: 14  90 #SUP: 9  252 #SUP: 8 | Algorithm is running... (08:52:47 PM)  ============= dECLAT v0.96r18 - STATS =============  Transactions count from database : 135  Frequent itemsets count : 8  Total time ~ 4 ms  Maximum memory usage : 7.291709899902344 mb  =================================================== |

## Experiment #2

As we described above, to perform second experiment, we have used dataset mentioned in section 5 - “All Trump’s Twitter insults (2015-2021)” which is available on Kaggle website.

For better understanding what was done, we provide how data preprocessing was done. Initially dataset was a \*.csv file, excerpt:

|  |
| --- |
| "","date","target","insult","tweet"  "1","2014-10-09","thomas-frieden","fool","Can you believe this fool, Dr. Thomas Frieden of CDC, just stated, ""anyone with fever should be asked if they have been in West Africa"" DOPE"  "2","2014-10-09","thomas-frieden","DOPE","Can you believe this fool, Dr. Thomas Frieden of CDC, just stated, ""anyone with fever should be asked if they have been in West Africa"" DOPE"  [omitted] |

This file consists of over 10 000 rows. To make used of that data, we took “tweet” text and saved them to file, which ten was processed using Parser instance, excerpt of newly created file:

|  |
| --- |
| Can you believe this fool, Dr. Thomas Frieden of CDC, just stated, ""anyone with fever should be asked if they have been in West Africa"" DOPE"  Can you believe this fool, Dr. Thomas Frieden of CDC, just stated, ""anyone with fever should be asked if they have been in West Africa"" DOPE"  Big time in U.S. today - MAKE AMERICA GREAT AGAIN! Politicians are all talk and no action - they can never bring us back."  [omitted] |

This “raw tweets” are further preprocessed and as input we get transactional database along with dictionary, which allow us to make translation between items representation – integers and words. Here we present input file for dEclat algorithm, which is a result of data preprocessing.

|  |
| --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18  14 19 20 21 22 23 24 25 26 27 28 29 30  [omitted]  385 1667 2181 701 810 42 396 2861 976 18 8787 470 1751 662 3514 1757  ==== DICTIONARY ====  {"1": "africa", "2": "anyone", "3": "asked", "4": "been", "5": "believe", "6": "dope", "7": "fever", "8": "fool", "9": "frieden", "10": "have", "11": "just", "12": "should",… omitted} |

As experiment we want to perform few iterations of declat algorithm run. Each run will be for different min\_sup parameter. For each iteration we will collect parameters, namely: min\_sup, total\_time, peak\_memory\_usage, database\_size, fis\_count.

For this purpose, we created special method “perform\_experiment()”. This method is described below:

|  |
| --- |
| def perform\_experiment(self):  # Define different min\_sup values  sup\_values = [0.1, 0.08, 0.06, 0.04, 0.03, 0.02, 0.01, 0.009, 0.008, 0.007, 0.006, 0.005]  # Create file to save results  results = open("../results/trump-experiment.csv", "w", encoding="utf-8")  # Indicate input database file and create new database instance  input = "../input\_data/trump-transactions.txt"  database = TransactionDatabase(file\_path=input)  # describe values in csv file  results.write("min\_sup,total\_time,peak\_memory,db\_size,fis\_count")  # For each support value, run declat algorithm and save results to the scv file  for sup in sup\_values:  output = f"../output/output-declat-{sup}.txt"  start\_time, end\_time, memory, db\_size, fi\_count = self.run\_algorithm(output=output, database=database, minsupp=sup)  total\_time = end\_time - start\_time  results.write(f"\n{sup},{total\_time},{memory},{db\_size},{fi\_count}")  database.translate\_integers\_into\_words(file\_path=output)  results.close() |

This is run inside dEclat.py file:

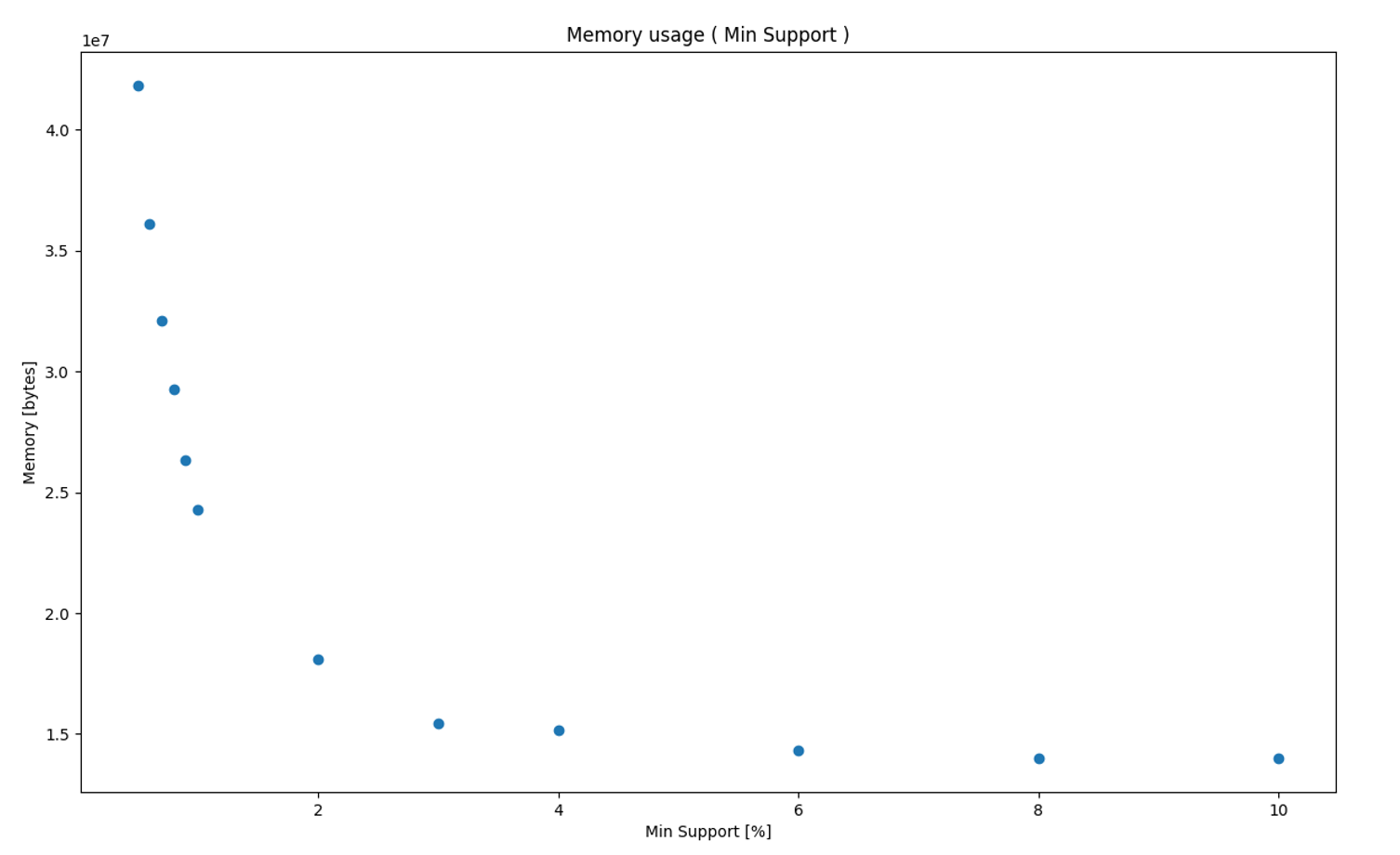
|  |
| --- |
| **if \_\_name\_\_ == "\_\_main\_\_":**  declat = DEclat()  declat.perform\_experiment() |

We saved experiment results to “trump-experiment.csv” file. We will use that file to draw plots.

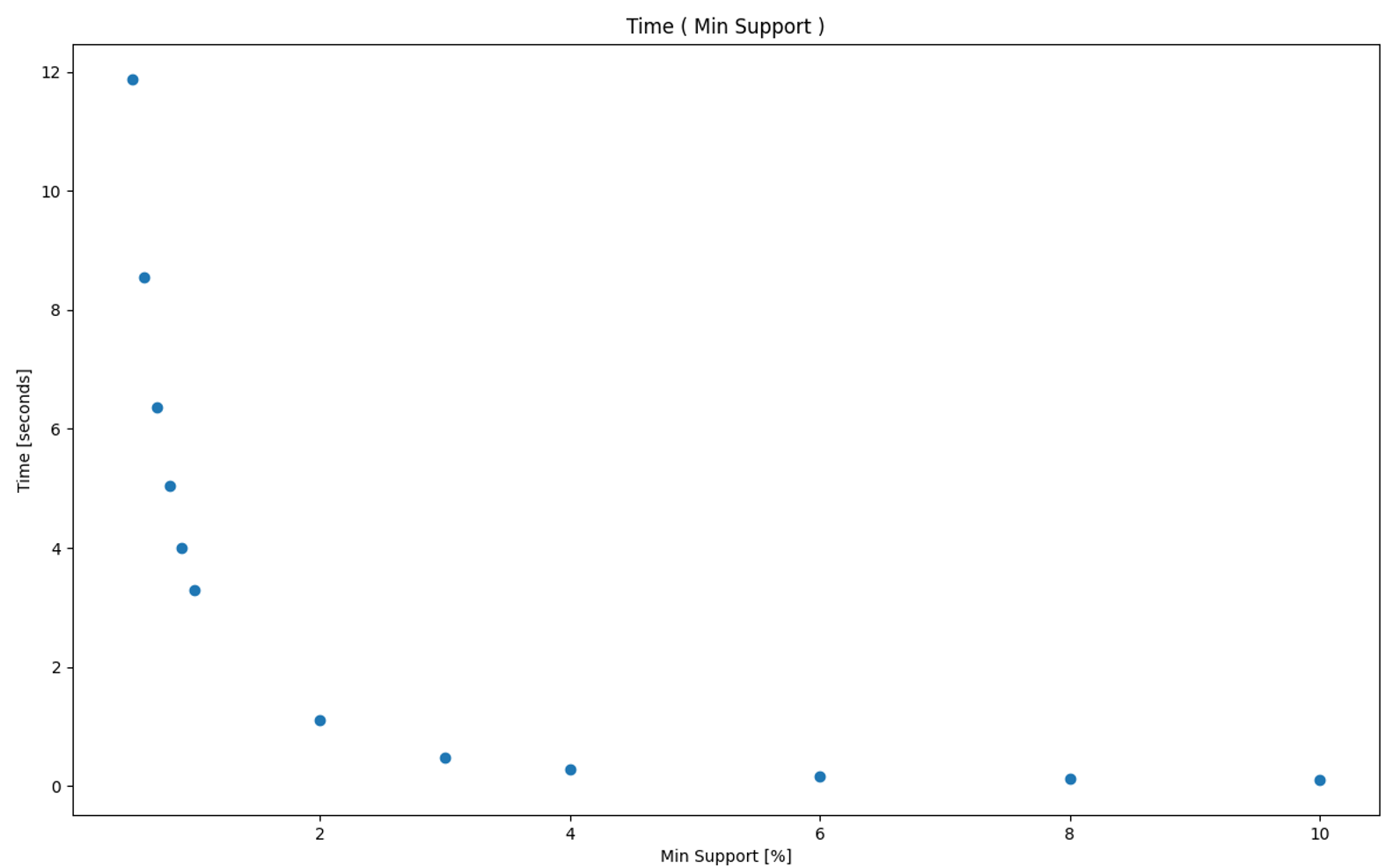
|  |
| --- |
| **min\_sup,total\_time,peak\_memory,db\_size,fis\_count**  0.1,0.10302877426147461,13996236,10336,15  0.08,0.12502837181091309,13995124,10336,25  0.06,0.16403818130493164,14301459,10336,39  0.04,0.2780630588531494,15182210,10336,68  0.03,0.4758181571960449,15450210,10336,107  0.02,1.1078088283538818,18088847,10336,184  0.01,3.2975175380706787,24301738,10336,370  0.009,3.996622085571289,26352385,10336,417  0.008,5.034431457519531,29282705,10336,466  0.007,6.353771448135376,32092638,10336,533  0.006,8.54745626449585,36089233,10336,620  0.005,11.883255004882812,41852078,10336,704 |

We decided to draw three plots based at data obtained during Experiment #2:

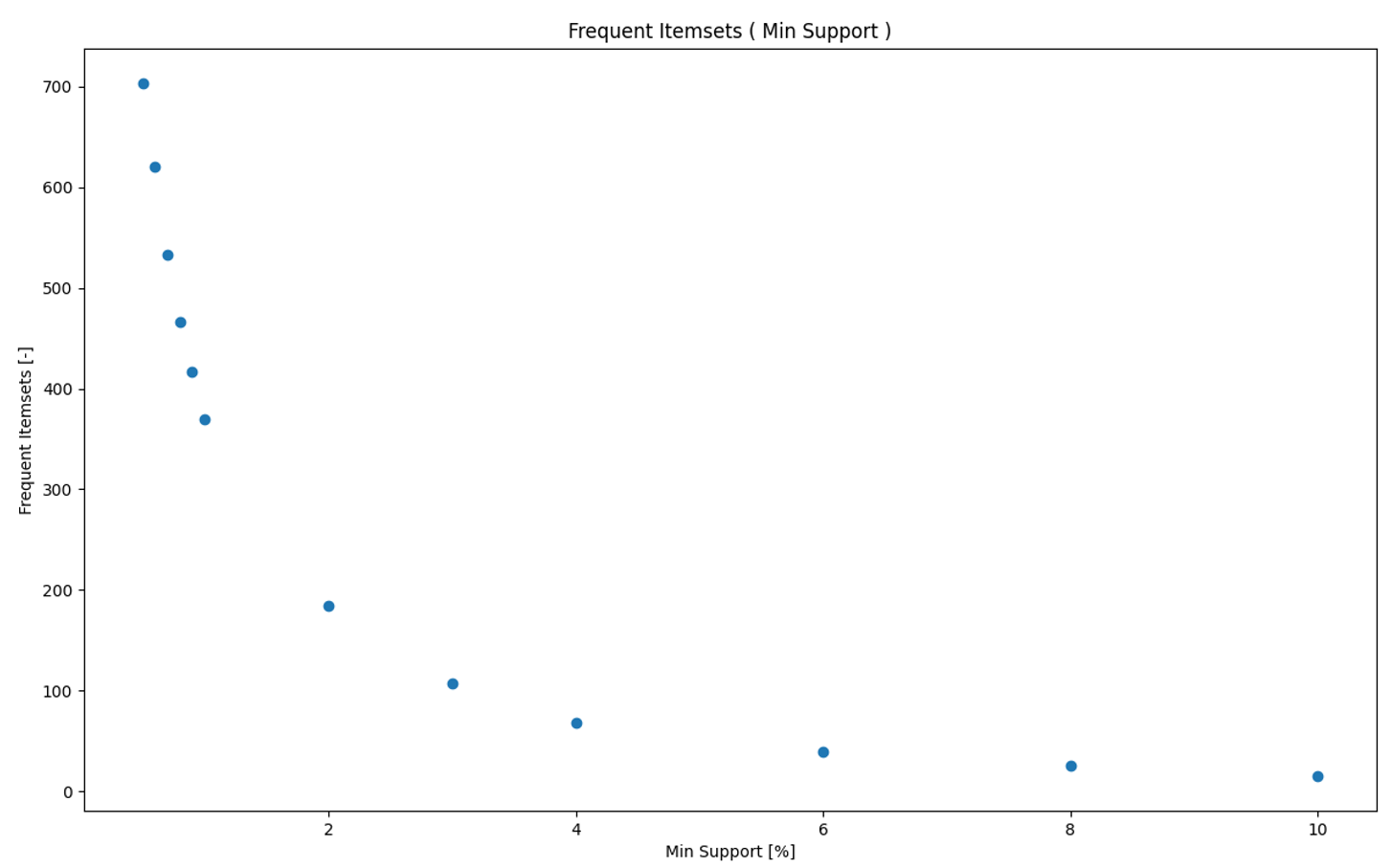
1. Memory consumption as a function of minimum support



1. Experiment duration as a function of minimum support

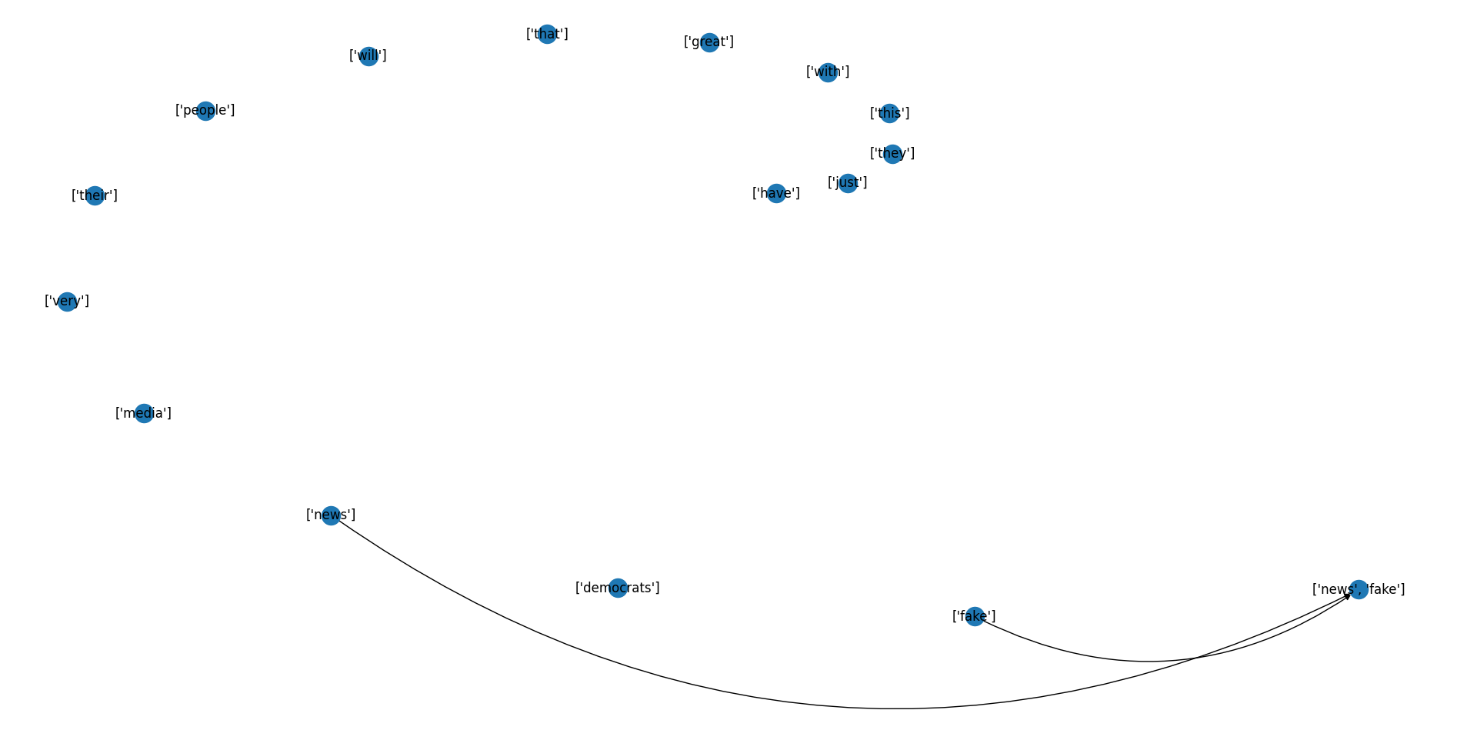


1. Frequent Itemsets count as a function of minimum support

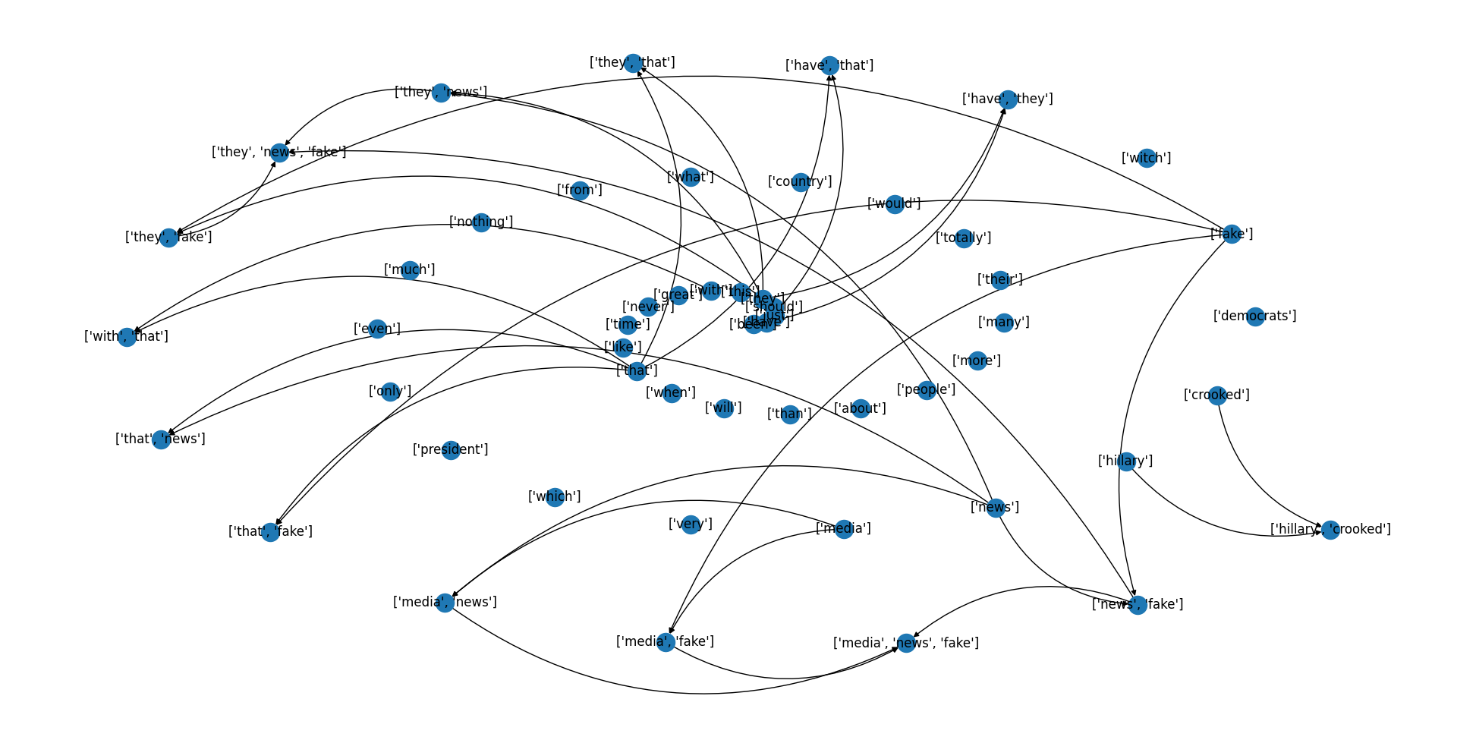


We also prepared hasse-like diagrams for frequent item-sets when min support was:

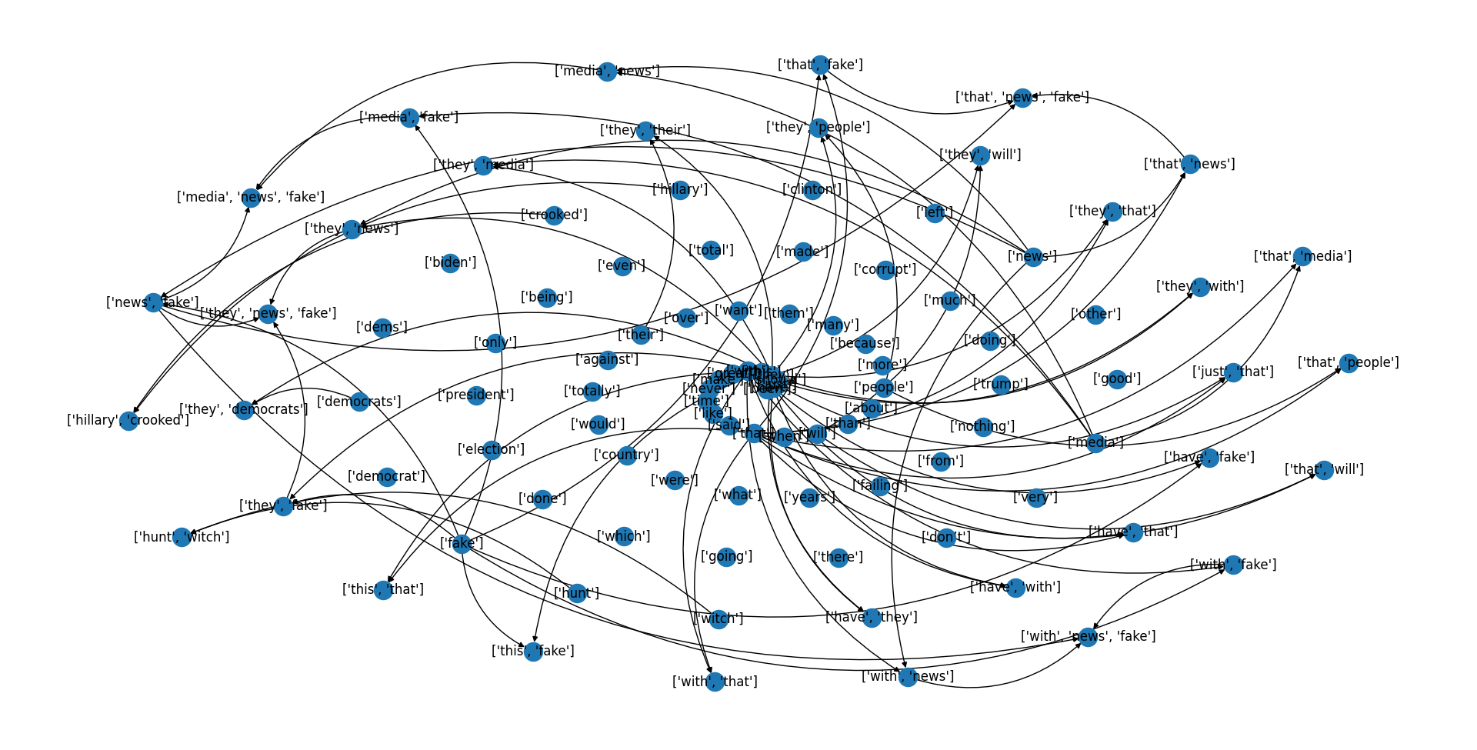
1. min\_sup = 10%



1. min\_sup = 6%



1. min\_sup = 4%



**TODO – conclusions!**

# Phase 2

[Future Work]

# Phase 3

# References

[1] <http://www.philippe-fournier-viger.com/spmf/Eclat_dEclat.php>

[2] <https://www.philippe-fournier-viger.com/spmf/dEclat_dCharm.pdf>