**Logo

Description automatically generated**

**DATA MINING PROJECT REPORT**

**GSP ALGORITHM**

**Submitted by:**

Meng Zhou

Eribake Gabriel Oluwafisayo

Content

[1 – Outline (By Meng Zhou) 4](#_Toc105870346)

[1.1 Problems to Solve 4](#_Toc105870347)

[1.2 Algorithm Description 4](#_Toc105870348)

[1.3 Definition of Test Data 5](#_Toc105870349)

[1.4 Definition of Experimentation Data 5](#_Toc105870350)

[1.4.1 Data Source 5](#_Toc105870351)

[1.4.2 Dataset Size 5](#_Toc105870352)

[1.4.3 Hypothesis 6](#_Toc105870353)

[1.5 Data Pre-Processing and Cleaning 6](#_Toc105870354)

[1.6 Plan the Scope and Method of Visualization 6](#_Toc105870355)

[2 - Implementation of the Algorithm (By Meng Zhou & Gabriel) 6](#_Toc105870356)

[2.1 Introduction 6](#_Toc105870357)

[2.2 Transactions and Sequences 7](#_Toc105870358)

[2.3 k-length Sequence:   7](#_Toc105870359)

[2.4 Support in k-length Sequence: 8](#_Toc105870360)

[2.5 Joining Mechanism 8](#_Toc105870361)

[2.6 Pruning Phase 9](#_Toc105870362)

[3 – Testing (By Gabriel) 9](#_Toc105870363)

[3.1 Testing Defined 9](#_Toc105870364)

[3.2 Testing Result 10](#_Toc105870365)

[4 - Perform Experiments (By Meng Zhou) 10](#_Toc105870366)

[4.1 Retrieve Twitter Data Using Snscrape 10](#_Toc105870367)

[4.2 Pre-Process and Clean the Data 11](#_Toc105870368)

[4.2.1 First View of the Raw Text Data 11](#_Toc105870369)

[4.2.2 Read the data 11](#_Toc105870370)

[4.2.3 Remove the Punctuations 12](#_Toc105870371)

[4.2.4 Tokenization and Stop Words Removal 12](#_Toc105870372)

[4.2.5 Stemming 12](#_Toc105870373)

[4.2.6 lemmatization 12](#_Toc105870374)

[4.2.7 Storing the Clean Dataset 12](#_Toc105870375)

[4.3 Experiment Result 13](#_Toc105870376)

[5 - Visualization of Results (By Meng Zhou) 13](#_Toc105870377)

[6 - Analysis of Results (By Gabriel) 14](#_Toc105870378)

[6.1 Some Findings 14](#_Toc105870379)

[6.2 Performance Analysis 15](#_Toc105870380)

[6.3 Drawbacks of GSP 15](#_Toc105870381)

[7 – Outlook of Second Phase (By Gabriel and Meng Zhou) 15](#_Toc105870382)

[7.1 Potential Performance Improvements (By Meng Zhou) 15](#_Toc105870383)

[7.2 Reference Article 16](#_Toc105870384)

[8 – Phase 2 Introduction (By Meng Zhou) 16](#_Toc105870385)

[9 – Problem Statement 17](#_Toc105870386)

[9.1 -Definitions 17](#_Toc105870387)

[9.2 - Input 17](#_Toc105870388)

[9.3 - Sliding windows 18](#_Toc105870389)

[9.4 - Time Constraints 18](#_Toc105870390)

[9.5 - Problem Definition 18](#_Toc105870391)

[9.6 – Definition of Test and Experimentation Data 18](#_Toc105870392)

[10 - Algorithm Description 19](#_Toc105870393)

[10.1 - Counting Candidates 19](#_Toc105870394)

[10.2 - Reduce the number of candidates by hash-tree 19](#_Toc105870395)

[10.3 - Checking whether a data-sequence contains a specific sequence 20](#_Toc105870396)

[11 - Implementation and Improvement 20](#_Toc105870397)

[11.1 - Representation of Hash-tree 20](#_Toc105870398)

[11.2 - Reducing duplicate searching 22](#_Toc105870399)

[12 - Experiments and Comparison 24](#_Toc105870400)

[13 – Result Visualization 0](#_Toc105870401)

[13.1 -Get some data. 0](#_Toc105870402)

[13.2 - Extracting sequential patterns. 0](#_Toc105870403)

[13.3 - Transforming the output file into GraphViz DOT format. 0](#_Toc105870404)

[13.4 - Generating a graph using GraphViz 1](#_Toc105870405)

[15 - Conclusion 1](#_Toc105870406)

[16 – References 3](#_Toc105870407)

# 1 – Outline (By Meng Zhou)

## 1.1 Problems to Solve

The novel Coronavirus (COVID-19) pandemic, which started in late December 2019, has spread to more than 200 countries. Government and health agencies are taking draconian steps to contain it. This pandemic is also trending on social media, particularly on Twitter. The purpose of this study is to explore and analyze the general public reactions to the COVID-19 outbreak on Twitter. Moreover, GSP algorithm (Generalized Sequential Pattern algorithm), an algorithm used for sequential pattern mining(SPM), is used to ﬁnd frequent words/patterns and their relationship in tweets.

We worked as a team to finish the project. The contribution of each teammate is listed in the content section.

## 1.2 Algorithm Description

GSP is a very important algorithm in data mining. It is used in sequence mining from large databases. Almost all sequence mining algorithms are basically based on apriori algorithm. GSP uses a level-wise paradigm for finding all the sequence patterns in the data. It starts with finding the frequent items of size one then passes that as input to the next iteration of the GSP algorithm. The database is passed multiple times to this algorithm. In each iteration, GSP removes all the non-frequent item sets. This is done based on a threshold frequency which is called support. Only those item sets are kept whose frequency is greater than the support count. After the first pass, GSP finds all the frequent sequences of length-1 which are called 1-sequences. This makes the input to the next pass, it is the candidate for 2-sequences. At the end of this pass, GSP generates all frequent 2-sequences, which makes the input for candidate 3-sequences. The algorithm is recursively called until no more frequent item sets are found.

|  |
| --- |
| Obtain a sequence in form of as length-1 candidates  find F1 (the set of length-1 sequential patterns), after a unique scan of database  Let k=1  **While** Fk is not empty do  - Form Ck+1, the set of length-(k+1) candidates from Fk;  - If Ck+1 is not empty, unique database scan,  find Fk+1 (the set of length(k+1) sequential patterns)  Let k=k+1;  **End While** |

Figure 1. Pseudocode of GSP algorithm[[1]](#footnote-1)

## 1.3 Definition of Test Data

The test data is a sequence of sequences representing items in baskets as shown below.

|  |
| --- |
| transactions = [  ['Bread', 'Milk'],  ['Bread', 'Diaper', 'Beer', 'Eggs'],  ['Milk', 'Diaper', 'Beer', 'Coke'],  ['Bread', 'Milk', 'Diaper', 'Beer'],  ['Bread', 'Milk', 'Diaper', 'Coke']  ] |

Figure 2. Test Data

## 1.4 Definition of Experimentation Data

### 1.4.1 Data Source

**In terms of experimentation data, we will retrieve data from twitter.** We can easily get tweets by using twitter developer(<https://developer.twitter.com/en>) provided by twitter. The corpus of collected tweets was converted into a suitable format to apply SPM techniques on them. Each line in the corpus, called tweets corpus (TC), represents a tweet, which is basically a sequence of words. In SPM, various measures are used to investigate the importance and interestingness of sequences in a sequence database.

In order to get tweets related to our project, we will use the “Advanced search” function in twitter. Here, we can filter tweets that must contain certain keywords like “Covid” and may contain certain keywords like “lockdown” and “mask”. We can also specify tweets within a certain period. It’s no doubt that we need tweets after 2020, because that’s when the Covid19 spread the world. We will also set language to “English”, otherwise we need to do extra text processing later.

We will get a query generated by twitter.

Then we can use it in our python spider script.

### 1.4.2 Dataset Size

Joining and pruning of itemsets and checking subset of each transaction against candidates are very computation intensive process in Apriori algorithm. Also a large number of candidates require large memory during execution of algorithm. Therefore, we decided to set dataset size to 500 tweets.

### 1.4.3 Hypothesis

If one person thinks about something, it becomes an idea. If 1,000 people think the same thing, it becomes an evolution. If one million people think about the same thing, it becomes a revolution. If one billion people think about the same thing, it becomes a law. This is the power of big data. Before we carry our experiments, we can use Google Trend to predict our results.

**For example, here we can see people are searching “covid” and “mask” on google a lot. We can assume that people are interested about whether they should wear mask or not. Similarly, “mask” and “face mask” should also be trends on twitter. We can formulate the hypothesis that we can find frequent words or subsequences similar to what we find on Google Trend.**

## 1.5 **Data Pre-Processing and Cleaning**

In this project, the Twitter API and Python Snscrape library were used for tweet streaming. To collect relevant tweets. After establishing a successful connection to Twitter, a stream of real-time tweets containing the ﬁlter words was received. Data collection was done during random periods of time every day from 2021-06-11 to 2021-11-11. In total, 500 tweets were collected in csv format text ﬁle. After tweet collection, pre-processing was applied to extract meaningful information from tweets using the Python Natural Language Tool Kit(NLTK). Tweets were cleaned by performing the following tasks:

It involves several steps：

* Lowercasing all the letters
* Removing hashtags and mentions
* Removing links
* Removing punctuations(string.punctuation)
* Tokenization(from nltk.tokenize import RegexpTokenizer)
* Stop words removal(nltk.corpus.stopwords.words("english"))
* Stemming(nltk.PorterStemmer; def stemming)
* Lemmatization(nltk.WordNetLemmatizer; def lemmatizer)

After passing through these steps, the cleaned data were saved in a CSV ﬁle. We will start our experiments after pre-processing and cleaning the data.

## 1.6 Plan the Scope and Method of Visualization

We have two tasks in this project. First, we’ll use GSP algorithm to find all the frequent patterns in the dataset. Second, we’ll try to find association rules from these patterns. **We plan to visualize frequent patterns and their support and confidence level**, since they’re necessary for these two tasks.

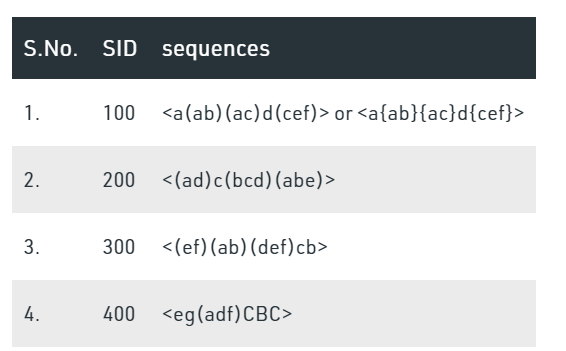
In terms of tools, we will primarily use **matplotlib and pandas** libraries.

# 2 - Implementation of the Algorithm (By Meng Zhou & Gabriel)

## 2.1 Introduction

GSP is a very important algorithm in data mining. It is used in sequence mining from large databases. Almost all sequence mining algorithms are basically based on a prior algorithm. GSP uses a level-wise paradigm for finding all the sequence patterns in the data. It starts with finding the frequent items of size one then passes that as input to the next iteration of the GSP algorithm. The database is passed multiple times to this algorithm. In each iteration, GSP removes all the non-frequent itemsets. This is done based on a threshold frequency which is called support. Only those itemsets are kept whose frequency is greater than the support count. After the first pass, GSP finds all the frequent sequences of length-1 which are called 1-sequences. This makes the input to the next pass, it is the candidate for 2-sequences. At the end of this pass, GSP generates all frequent 2-sequences, which makes the input for candidate 3-sequences. The algorithm is recursively called until no more frequent itemsets are found.

## 2.2 Transactions and Sequences



*Image 1[[2]](#footnote-2)*

* <a(ab)(ac)d(cef)> is a sequence whereas (a), (ab), (ac),
* (d) and (cef) are the elements of the sequence.
* These elements are sometimes referred as transactions.
* An element may contain a set of items. Items within an element are unordered and we list them alphabetically.
* For example, (cef) is the element and it consists of 3 items c, e and f.
* Since, all three items belong to same element, their order does not matter. But we prefer to put them in alphabetical order for convenience.
* The order of the elements of the sequence matters unlike order of items in same transaction.

## 2.3 k-length Sequence:  [[3]](#footnote-3)

The number of items involved in the sequence is denoted by K. A sequence of 2 items is called a 2-len sequence. While finding the 2-length candidate sequence this term comes into use. Example of 2-length sequence is: {ab}, {(ab)}, {bc} and {(bc)}.

{bc} denotes a 2-length sequence where b and c are two different transactions. This can also be written as {(b)(c)}

{(bc)} denotes a 2-length sequence where b and c are the items belonging to the same transaction, therefore enclosed in the same parenthesis. This can also be written as {(cb)}, because the order of items in the same transaction does not matter.

## 2.4 Support in k-length Sequence[[4]](#footnote-4):

Support means the frequency. The number of occurrences of a given k-length sequence into the sequence database is known as the support. While finding the support the order is taken care.

|  |
| --- |
| **Illustration:**  Suppose we have 2 sequences in the database.  s1: <a(bc)b(cd)>  s2: <b(ab)abc(de)>  We need to find the support of {ab} and {(bc)}  **Finding support of {ab}:**  This is present in first sequence.  s1: <a(bc)b(cd)>  Since, a and b belong to different elements, their order matters.  In second sequence {ab} is not found but {ba} is present.  s2: <b(ab)abc(de)> Thus we don’t consider this.  Hence, support of {ab} is 1.  **Finding support of {bc}:**  Since, b and c are present in same element, their order does not matter.  s1: <a(bc)b(cd)>, first occurrence.  s2: <b(ab)abc(de)>, it seeems correct, but is not. b and c are present in different elements here. So, we don’t consider it.  Hence, support of {(bc)} is 1. |

## 2.5 Joining Mechanism [[5]](#footnote-5)

L1 is the final 1-length sequence after pruning. After pruning all the entries left into the set have supported greater than the threshold.

|  |
| --- |
| **Case 1: Join {ab} and {ac}**  s1: {ab}, s2: {ac}  After removing a from s1 and c from s2.  s1’={b}, s2’={a}  s1′ and s2′ are not same, so s1 and s2 can’t be joined.  **Case 2: Join {ab} and {be}**  s1: {ab}, s2: {be}  After removing a from s1 and e from s2.  s1’={b}, s2’={b}  s1′ and s2′ are exactly same, so s1 and s2 be joined.  s1 + s2 = {abe}  **Case 3: Join {(ab)} and {be}**  s1: {(ab)}, s2: {be}  After removing a from s1 and e from s2.  s1’={(b)}, s2’={(b)}  s1′ and s2′ are exactly same, so s1 and s2 be joined.  s1 + s2 = {(ab)e}  s1 and s2 are joined in such a way that items belong to correct elements or transactions. |

## 2.6 Pruning Phase[[6]](#footnote-6)

While building Ck (candidate set of k-length), we delete a candidate sequence that has a contiguous (k-1) subsequence whose support count is less than the minimum support (threshold). Also, delete a candidate sequence that has any subsequence without minimum support. {abg} is a candidate sequence of C3.

|  |
| --- |
| {abg} is a candidate sequence of C3.  To check if {abg} is proper candidate or not, without checking its support, we check the support of its subsets.  Because subsets of 3-length sequence will be 1 and 2 length sequences. We build the candidate sets incremently like 1-length, 2-length and so on.  Subsets of {abg} are: {ab], {bg} and {ag}  Check support of all three subsets. If any of them have support less than minimum support then delete the sequence {abg} from the set C3 otherwise keep it. |

# 3 – Testing (By Gabriel)

## 3.1 Testing Defined

Examples of configuring and running are located in the test folders *GSP*. To use it in a project, import it and use the GSP class.

|  |
| --- |
| from gsppy.gsp import GSP |

For the sake of convenience, our transactions here are a sequence of sequences representing items in baskets.

transactions = [

['Bread', 'Milk'],

['Bread', 'Diaper', 'Beer', 'Eggs'],

['Milk', 'Diaper', 'Beer', 'Coke'],

['Bread', 'Milk', 'Diaper', 'Beer'],

['Bread', 'Milk', 'Diaper', 'Coke']

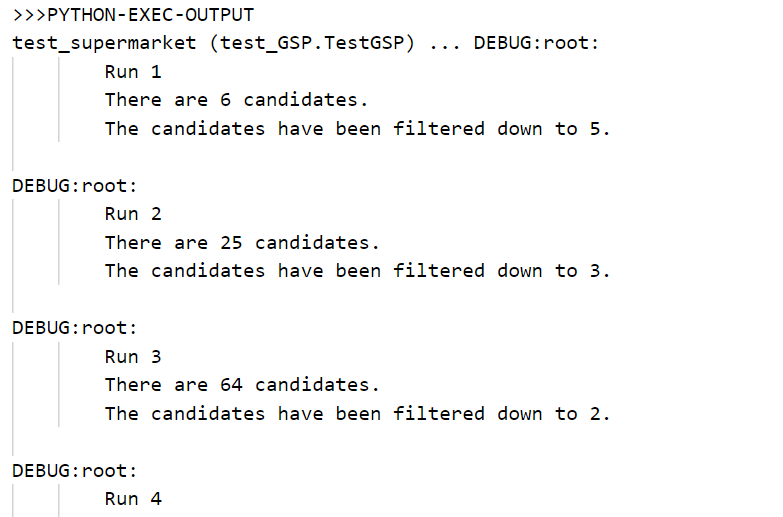
]

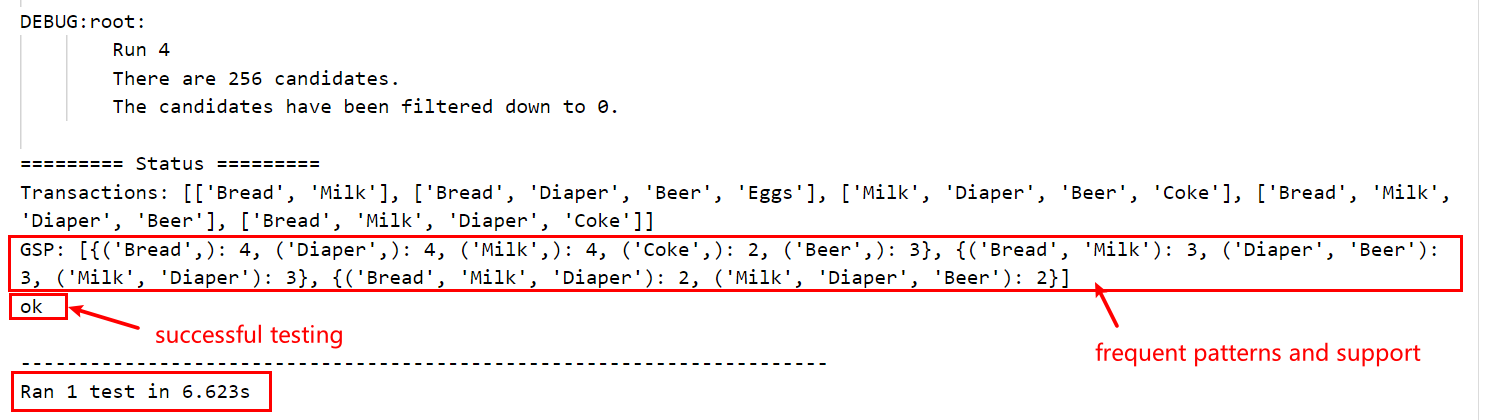
Init the class to prepare the transactions and to find patterns in baskets that occur over the support threshold (count):

|  |
| --- |
| result = GSP(transactions).search(0.3) |

The support count (or simply support) for a sequence is defined as the fraction of total data-sequences that "contain" this sequence. (Although the word "contains" is not strictly accurate once we incorporate taxonomies, it captures the spirt of when a data-sequence contributes to the support of a sequential pattern.)

## 3.2 Testing Result





The test is successful. Now we will try to apply GSP algorithm to our tweet dataset.

# 4 - Perform Experiments (By Meng Zhou)

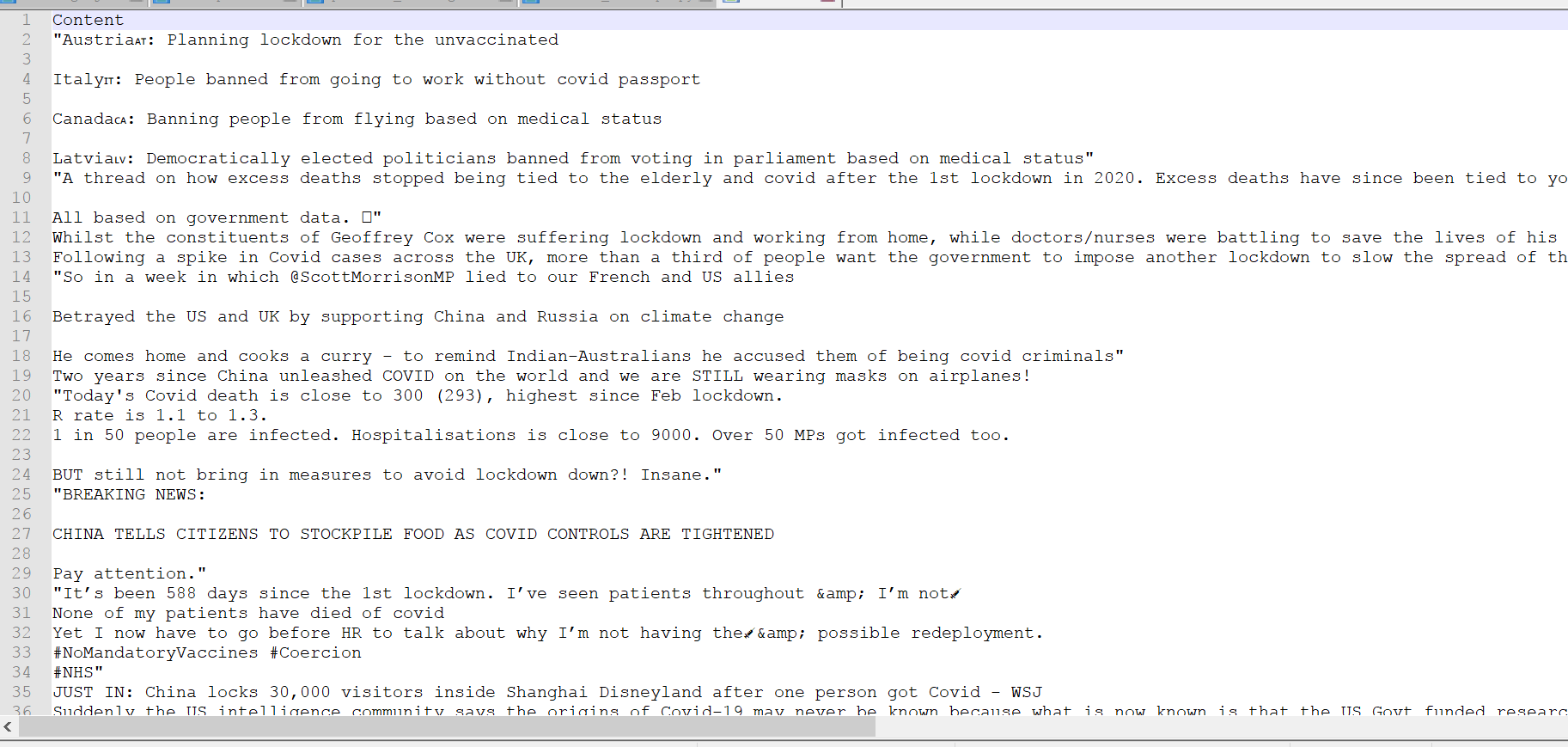
## 4.1 Retrieve Twitter Data Using Snscrape

**Snscrape** is a library that allows anyone to scrape social networking services (SNS) without requiring personal API keys. It can return thousands of user profiles, hashtags, contents, or searches in seconds and has powerful and highly customizable tools. In this project, we used snstwitter in snscrape library to get tweets.



## 4.2 Pre-Process and Clean the Data [[7]](#footnote-7)

### 4.2.1 **First View of the Raw Text Data**



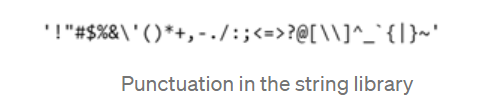
As seen above, we can direcly notice that the data contains a lot of stop words as well as some slangs (such as bf → Boy Friend) that needs to be either replaced or removed completely.

### 4.2.2 Read the data

I use jupyter notebook for this task, because it’s easy to debug and test.

### 4.2.3 Remove the Punctuations[[8]](#footnote-8)

We may want the words, but without the punctuation like commas and quotes. We also want to keep contractions together. One way would be to split the document into words by white space (as in “*2. Split by Whitespace*“), then use string translation to replace all punctuation with nothing (e.g. remove it). Python provides a constant called *string.punctuation* that provides a great list of punctuation characters.



We can see that this has had the desired effect, mostly. Contractions like “*What’s*” have become “*Whats*” but “*armour-like*” has become “*armourlike*“.

### 4.2.4 Tokenization and Stop Words Removal

For faster execution I will only use the first 5 rows to clean the example data and as seen above when we remove all of the stop words (such as is, so, etc….) the sentence becomes more cleaner. One example is**“omg its already 7:30” → “omg already 7:30”**

Tokenizing is the process of splitting strings into a list of words. We will make use of Regular Expressions or regex to do the splitting. Regex can be used to describe a search pattern. Here, “\W+” splits on one or more non-word character

### 4.2.5 Stemming

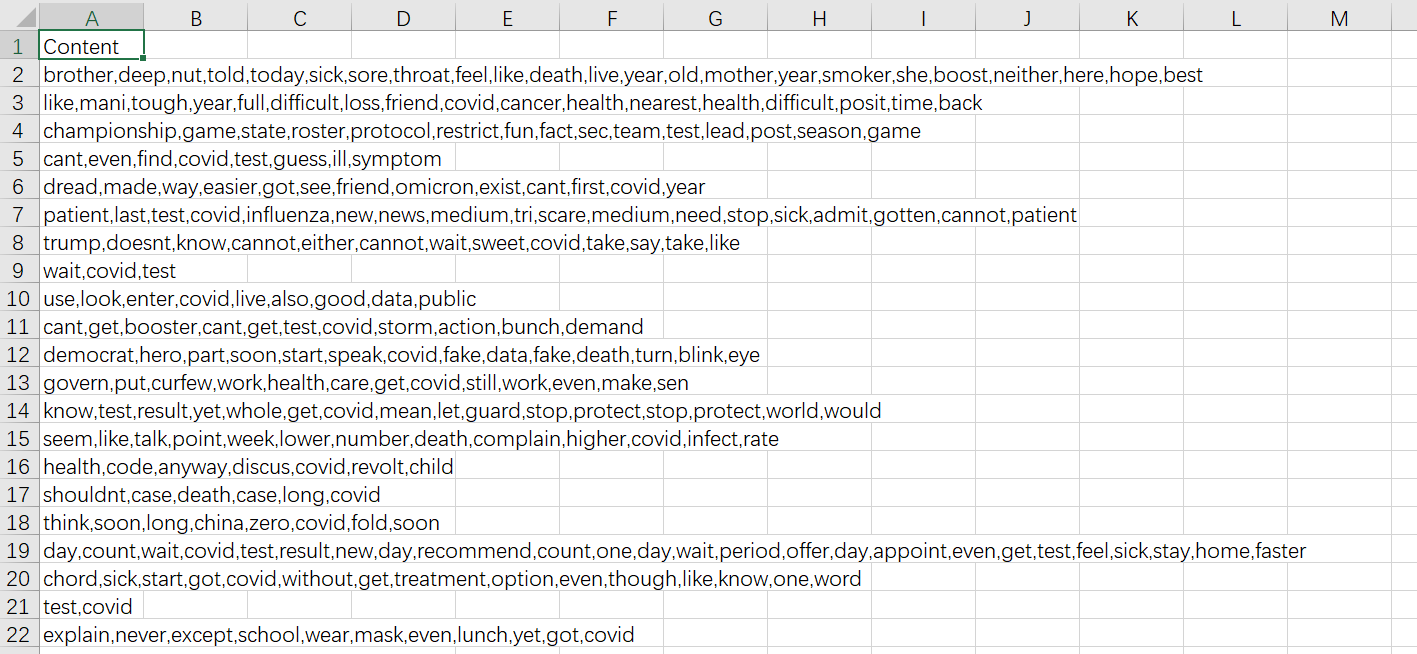
Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form. So as seen above, the word already have changed to alreadi, as well as cheating to cheat.

### 4.2.6 lemmatization

Lemmatization is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word’s lemma, or dictionary form. And this process is very similar to stemming however one difference we can note is the fact that the word already have not changed to alreadi.

### 4.2.7 Storing the Clean Dataset

Here is part of the clean data set.

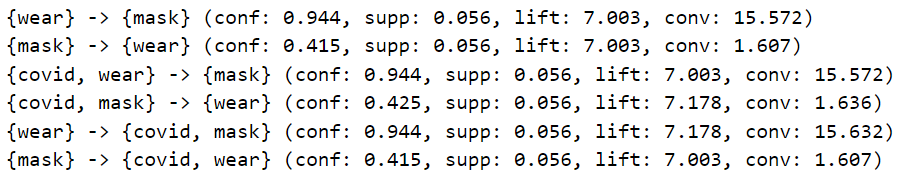


## 4.3 Experiment Result

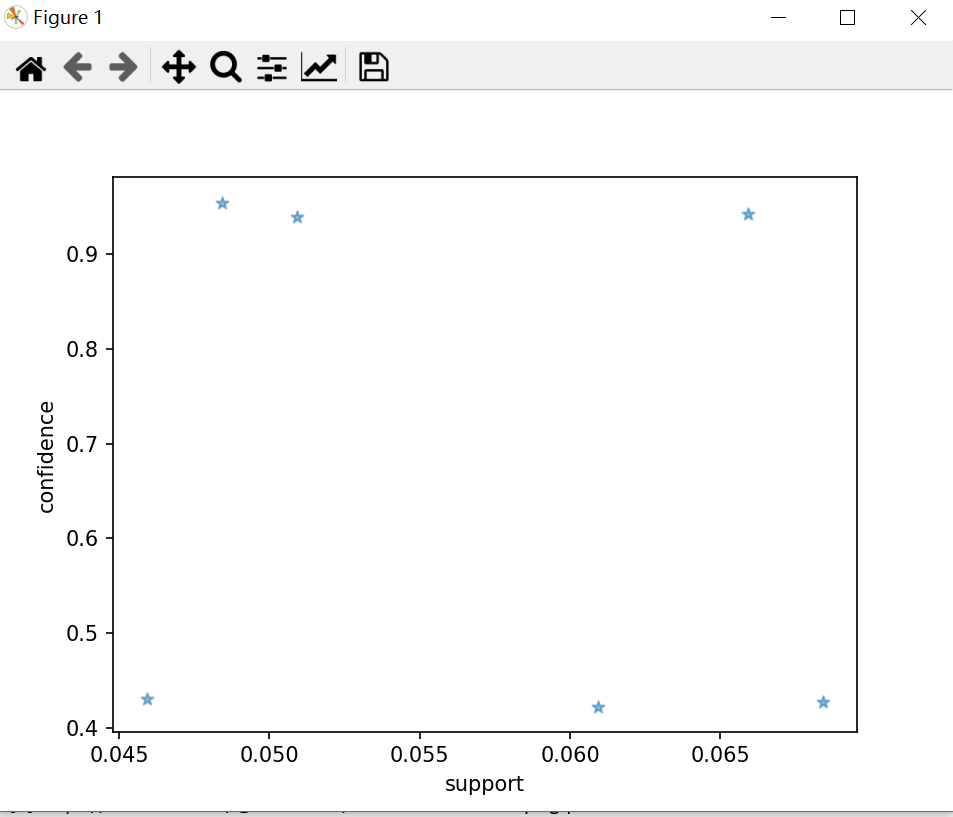
|  |  |  |  |
| --- | --- | --- | --- |
| min\_sup/number of sequences | 200 | 400 | 600 |
| 0.05 | 5.75 | 6.71 | 8.17 |
| 0.03 | 9.06 | 12.58 | 16.32 |
| 0.02 | 18.50 | 24.92 | 30.21 |

# 5 - Visualization of Results (By Meng Zhou)[[9]](#footnote-9)

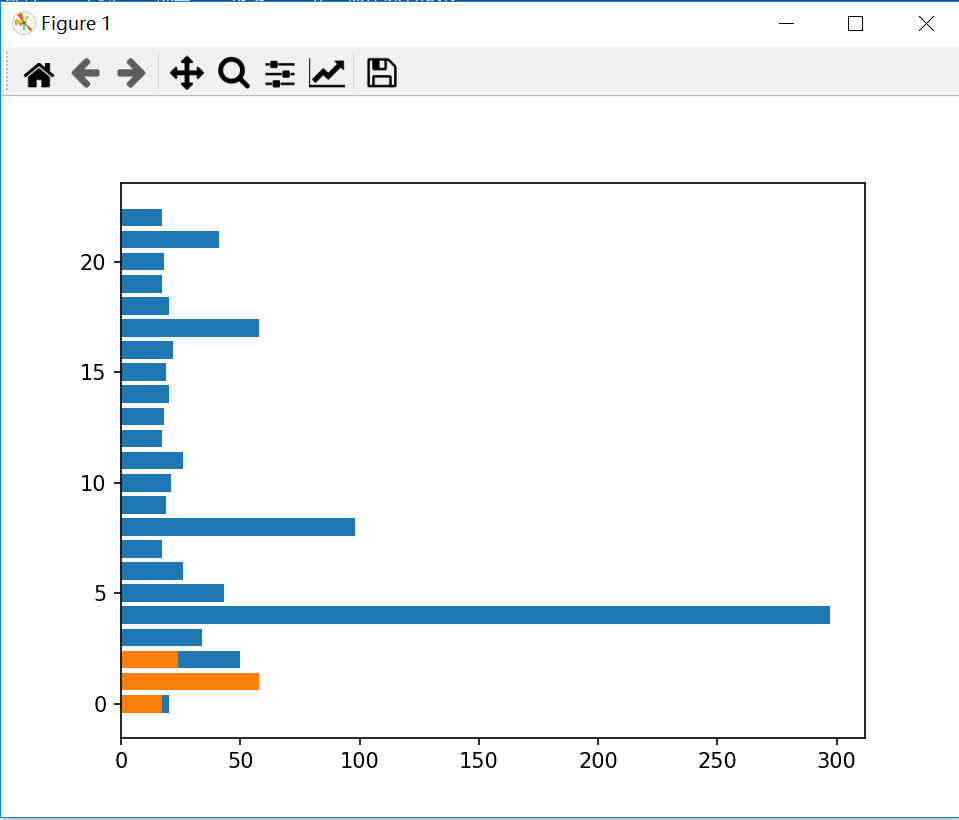
We use matplotlib to visualize the results.



rules



Support & confidence



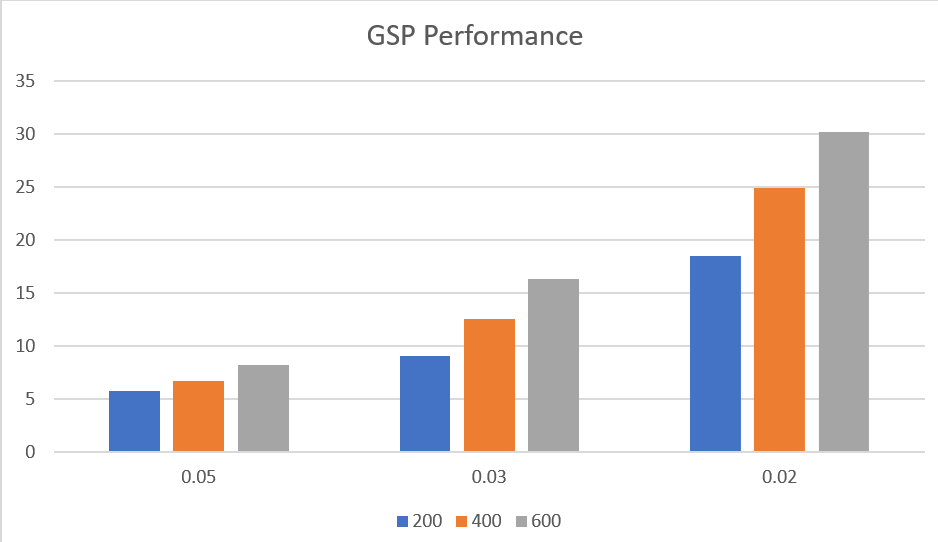
Frequent patterns & support

# 6 - Analysis of Results (By Gabriel)

## 6.1 Some Findings

Having implemented the GSP algorithm on the data set, we were able to see how various items interact with each other. We were also able to see how frequently each items occurred. Using the “df.to\_csv” we were able to write our dataframe in pandas to a csv file. By setting the minimum support, we were able to prune candidate rules by identifying the lowest bound for the support measure of resulting association rules, without setting it, the algorithm would have estimated a level for it which might not have helped us in getting our desired results. Setting the minimum support, minimum confidence and minimum lift, we were able to filter down the candidates and know their relations with each other. Looking at the results, we can conclude that “during covid we have to wear a mask”.

## 6.2 Performance Analysis



As we can see from GSP experiment results, the execution time is proportional to the number of sequences and reversely proportional to the support ratio(min\_support). In other words, the more sequences or the lower min\_support is given, the algorithm will cost more time.

## 6.3 Drawbacks of GSP

The database is passed many times to the algorithm recursively. The computational efforts are more to mine the frequent pattern. When the sequence database is very large and patterns to be mined are long then GSP encounters the problem in doing so effectively.

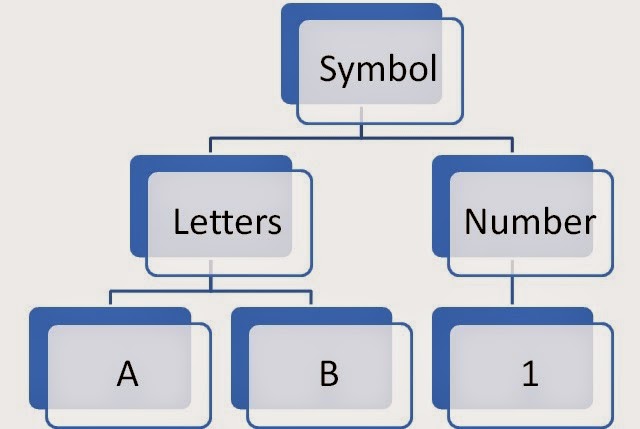
# 7 – Outlook of Second Phase (By Gabriel and Meng Zhou)

## 7.1 Potential Performance Improvements (By Meng Zhou)

We had a brief consultation with Ms.Marzena and she recommended a few articles.

Here's a list of extras that can be added:[[10]](#footnote-10)

1. Taxonomies/Hierarchies: Harry Potter and the Sorcerer's Stone is part of a series by J.K.Rowling, which is part of the children's fantasy genre. Taxonomies like this can be incorporated so that more general sequences can be found if the more specific ones aren't supported. The way that the algorithm handles these taxonomies is to add in the higher level items (e.g. J.K. Rowling is higher level than Harry Potter) to the combined transactions. For example if you had a sequence like this <A(1, B)A> then you would translate that into a sequence like <(A, Letter, Symbol)(1, Number ,B , Letter, Symbol)(A, Letter, Symbol)> if you had a taxonomy like the one here:



1. Window Size: If you've had a customer for years, maybe it isn't that interesting if they support a particular pattern because they have so much buying behaviour on record that they might just support it by chance. Window sizes limit the span of time when transactions are deemed to support a sequence. (e.g. I only want buying patterns for pregnant women during a 9 month window)
2. Max-gap: This parameter is defined by the user to filter out large gaps in data sequences. Let's say a customer buys an iPod, then even though they are within the specified window size, they have a large gap between that purchase and a new set of headphones. To a business owner, this gap might make them "uninteresting" as a customer to market to, so the GSP algorithm can filter this out as it is running and checking for support
3. Min-Gap: Think of this as answering the question "how much time would I let pass between transactions before I would consider 2 purchases separate elements in my sequences?" An example of this would be me at Home Depot. I go to start a project on a Friday evening and buy what I think I need. It isn't until Saturday about noon that I realize I had no idea what I REALLY needed and go back for more stuff. Some store owners may want to treat these as one "transaction" or event. If the min-gap was set to 24 hours in this case, all of my purchases would be grouped together to look something like this <(Friday night stuff, Saturday noon stuff)>.

## 7.2 Reference Article

Ramakrishnan Srikant, Rakesh Agrawal: Mining Sequential Patterns: Generalizations and Performance Improvements. EDBT 1996: 3-17

# 8 – Phase 2 Introduction (By Meng Zhou)

Ramakrishnan and Rakesh proposed a sequence mining algorithm that is faster than previous algorithms. The GSP(Generalized Sequential Patterns)[1] algo- rithm finds all sequential patterns that have a user-specified minimum support. Besides, they declare that GSP is much faster than the AprioriAll algorithm. Though the later algorithms like Spade[2] and Prefixspan[3] has achieved better performance than GSP, we discover that there’s lot to improve from the original algorithm. We present a strategy called *state pruning* in hash tree to eliminate extra comparison. Furthermore, we adjust parameters before building hash-tree in each iteration, which reduces a great amount of search space. Since we almost reinvent the original GSP algorithm, we call our algorithm *Hash*2 GSP, which means using the quality of the hash-tree to save time. In our experiment, the *Hash*2 GSP runs faster than Prefixspan algorithm.

# 9 – Problem Statement

## 9.1 -Definitions

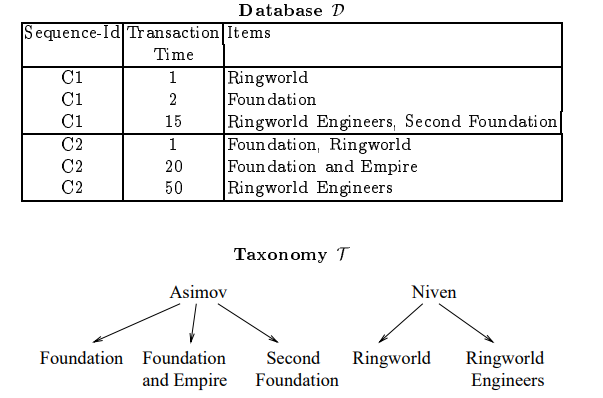
An itemset is a collection of items that is not empty. A sequence is a list of itemsets in a particular order. A sequence s is denoted by <s1s2s3sn>, where sj is an itemset. Sj is also referred to as a sequence element. (x1; x2;::; xm), where xj is an item, denotes a sequence element. An object can only appear once in a sequence element, but it can appear several times in distinct sequence elements. An itemset is a single-element sequence. We presume that elements in a sequence element are in lexicographic order without sacrificing generality.

If there are integers i1 i2 ::: in such that a1 bi1, a2 bi2,..., a bin, a sequence <a1a2:: an> is a subsequence of another sequence <b1b2::: bm>.

For example, because (3) (3, 8), (4, 5) (4, 5, 6) and (8) are subsequences of (7)(3, 8)(9)(4, 5, 6)(8), the sequence (3)(45)(8) is a subsequence of (7)(3, 8)(9)(4, 5, 6)(8). (8). The sequence (3)(5), on the other hand, is not a subsequence of (3, 5). (and vice versa).

## 9.2 - Input

We are given a database D of sequences called data-sequences. Each data-sequence is a list of transactions, ordered by increasing transaction-time. A transaction has the following elds: sequence-id, transaction-id, transactiontime, and the items present in the transaction. While we expect the items in a transaction to be leaves in T, we do not require this.



## 9.3 - Sliding windows

The sliding window helps to sustain a series by loosening the definition of a data-sequence. Formally, a data-sequence d = d1... dm contains a sequence s = s1... sn if there are integers l1 u1 l2 u2... ln un such that

1. si is contained in ui k=li dk, 1 I n, and

2. transaction-time(dui )-transaction-time(dli ) window-size, 1 I

We do not discuss the specific implementation of the attribute, despite the fact that it is defined in the original paper.

## 9.4 - Time Constraints

The time gap between sets of transactions containing consecutive parts of the sequence is limited by time restrictions. A data-sequence d = d1... dm contains a sequence s = s1... sn, given a user-specified window-size, maxgap, and min-gap. if there are integers l1 u1 l2 u2... ln un that are such that

1. Si is contained in ∪ui k=li dk, 1 ≤ i ≤ n,
2. transaction-time(dui )-transaction-time(dli ) ≤ window-size, 1 ≤ i ≤ n,
3. transaction-time(dli )-transaction-time(dui−1 ) > min-gap, 2 ≤ i ≤ n, and
4. transaction-time(dui )-transaction-time(dli−1 ) ≤ max-gap, 2 ≤ i ≤ n

## 9.5 - Problem Definition

The task of mining sequential patterns is to locate all sequences whose support is greater than the user-specified minimum support, given a database D of data-sequences, user-specified min-gap and max-gap time limitations, and a user-specified sliding-window size. A sequential pattern, also known as a frequent sequence, is represented by each of these sequences.

It is sometimes useful to know the "support relationship" between the sequence's elements when given a frequent sequence s = s1... sn. That is, what percentage of data sequences that support s1... si also support the whole sequence s. This relationship is simple to compute because s1... si must also be a frequent sequence.

## 9.6 – Definition of Test and Experimentation Data

1. There are two types of input data, you can switch by specifying the -file\_type argument, where 0 represents the common input data, and 1 represents input data the same as the spmf project. We provide some of the test data in the data directory.

**common input data:**

Examples are 100.txt, seq.txt and gen.data, the format is as follow:

sequence\_id number\_of\_items item\_id item\_id ...

sequence\_id number\_of\_items item\_id item\_id ...

sequence\_id number\_of\_items item\_id item\_id ...

Where each line represents an itemset, consist of distinct items.

**spmf input data:**

Exapmles are BMS1\_spmf.txt, kosarak10k.txt, kosarak25k.txt, and small.txt, the format is as follow:

item\_id -1 item\_id item\_id -1 ... -1 item\_id -2

item\_id -1 item\_id item\_id -1 ... -1 item\_id -2

item\_id -1 item\_id item\_id -1 ... -1 item\_id -2

Where each item\_set is seperated by -1, and -2 indicates the end of a sequence.

1. There is an experimental data\_provider in the source code, but the features are not fully developed. Currently it supports the *Gaussian distribution* and *Even distribution*. You must modify the parameters in the file if you want to use it, and recompiliation is also required.

##Usage:

*./gsp -i [file\_name] -t [support: float] -sequNUM [unsigned int32] -min [unsigned int32] -max [unsigned int32] -eventNUM [unsigned int32] -file\_type [0:common, 1:spmf]*

For instance, if you want to use 100.txt in ../data/ directory, you should use the following command:

*./gsp -i ../data/100.txt -t 0.5 -sequNUM 100 -min 2 -max 4 -eventNUM 100 -file\_type 0*

# 10 - Algorithm Description

To make the report concise, we’ll only list features that are different from phase1.

## 10.1 - Counting Candidates

We read one data-sequence at a time, increasing the number of candidates whose support is contained in that data-sequence. To tackle this difficulty, we employ two methods.

1. Using a hash-tree data structure to reduce the number of candidates in C.

2. We efficiently determine whether a special candidate is a subsequence of d by altering the data-sequence d's representation.

## 10.2 - Reduce the number of candidates by hash-tree

A hash-tree node either holds a list of sequences (left node) or a hash table (right node) (interior node). Each nonempty bucket of the hash table in an internal node points to another node. An inner node at depth p points to nodes at depth p + 1, while the root node is specified as being at depth 1.

**Candidate sequences are being added to the hash-tree**. To add candidate sequences to the hash-tree, we employ a top-down approach. We pick which branches to pursue at an inner node at depth p by applying the hash function to the p-th item in succession, starting from the root. A leaf node becomes an interior node when the number of sequences in it exceeds a certain threshold.

**Identifying the candidates in a data-sequence**. To locate all candidates contained in the data-sequences d, we use different algorithms to different types of nodes.

1. Interior node, if it is the root: Apply the hash function to each item in d, then repeat the process for each node in the appropriate bucket. We don't need to examine the max-gap or min-gap because this is a candidate's prospective start.
2. Interior node, if it is not the root: Let t represent the transaction time of the node we arrived at by hashing on item x. Apply the hash function to each item in d whose transaction time is between [t window size, t + max(window size, max gap)] and [t window size, t + max(window size, max gap)], apply this produce to each node in the corresponding bucket in a recursive manner.
3. Leaf node: We simply add s to the answer set for each sequence s in the leaf if it is contained by d by the algorithm in the next chapter.

## 10.3 - Checking whether a data-sequence contains a specific sequence

Let d represent a data sequence and s represent a candidate sequence.

**Contain test algorithm**: The algorithm tests if s is contained in d and alternates between two phases, starting with the initial components in the forward phase. The method is repeated until all of the elements have been discovered.

* Forward phase: If the end-time of the newly discovered elements is less than max-gap plus the start-time of the prior elements, the algorithm will proceed to the next elements in s. If this is not the case, the algorithm will revert to the previous phase.
* Backward phase: Let si represent the current element in s, and t represent the end-of-time (si). By finding the first set of transactions containing si1 whose end time is greater than t-max-gap, the method can "pull up" the prior element. If necessary to maintain the constraint satisfied, draw up the prior items, which is done until the constraint the max-gap between the element recently pushed up and the previous elements is satisfied or the constraint cannot be satisfied. If the condition is met, the algorithm proceeds to the forward phase, where it searches for the next element of si in s. If this is not the case, the algorithm will be terminated.

**Finding a single element:** By transforming the representation of d as follows, the algorithm speeds up the rate of locating the first occurrence of an element in a data-sequence.

**Creating an array using the items in the sequence as its elements.** A list of transaction times for the associated item is stored for each element in the array. To discover the first occurrence of an item after time t, we must traverse the item's list until we find an item whose transaction time exceeds t. The algorithm performs one pass through the items in the element and finds the first occurrence of each item after time t to locate the first occurrence of an element in a data-sequence. This method is superior to basic matching. Because in this time-list matching technique, the sequence is only accessed once, but in the simple approach, the sequence is accessed several times. The algorithm is described in detail in the following section.

# 11 - Implementation and Improvement

## 11.1 - Representation of Hash-tree

At first we implemented the whole original GSP algorithm, including the hash- tree, whereas the performance is not as good as we expected. The number of candidates are not reduced by hash-tree when encountering a new sequence from database. We check each candidate with the sequence from the database and increase their support count when they are contained by the sequence. But the checking step may cost much time since the number of candidate is very large. For example, if the number of frequent item is L1 = 300, then the number of 2-item candidates is L1 (L1 1)/2 + L1 L1 = 134850, which is a huge number. If we compare every candidate with the every sequence from database, the algorithm may run over 10 minutes just return a small number of frequent 2-item sequence.

The huge gap between the number of candidates and real frequent patterns motivates us to improve the algorithm. The key step is reducing the number of comparison for each sequence from database. Since each sequence might match only a small number of candidate patterns, we needn’t check every candidates with the sequence. A proper solution is first filtering the candidate patterns that mustn’t be supported by the given sequence and then checking the rest candidate patterns one by one.We first apply the hash-tree method proposed by the author.

Each interior tree node contains a list of pointer pointing to their children. Leaf nodes do not have children but contain a list of candidate patterns. For a new coming sequence, we use the method mentioned above to search the hash-tree from the tree root.

**Algorithm 1** Search along hash-tree

**Require:** Sequence *s*, hash-tree node *v* and last hashed *idx*

1: **function** SearchNode(*v, s, idx*)

2: **if** *v* is leaf node **And** *v* has not been visited **then**

3: CheckPatterns(*v.patterns*);

4: **else**

5: **for** *i* = *idx* + 1 *s.length* **do**

6: **if** *idx* 0 **And** *s*[*i*]*.time s*[*idx*]*.time <*= *max gap* **then**

7: *child idx* = Hash((*s*[*i*]));

8: *child* = *v.children*[*child idx*];

9: SearchNode(*child, s, i*);

10: **end if**

11: **end for**

12: **end if**

13: **end function**

We hash every remained items in the sequence, then move to the corre- sponding child and search with remained sequence. The searching process is recursive, thus for one node duplicate searching is possible. For instance, given a sequence (1*,* 2)(3*,* 4)(5) and hash-tree with hash function *item*%3, part of the hash searching process are drawn below

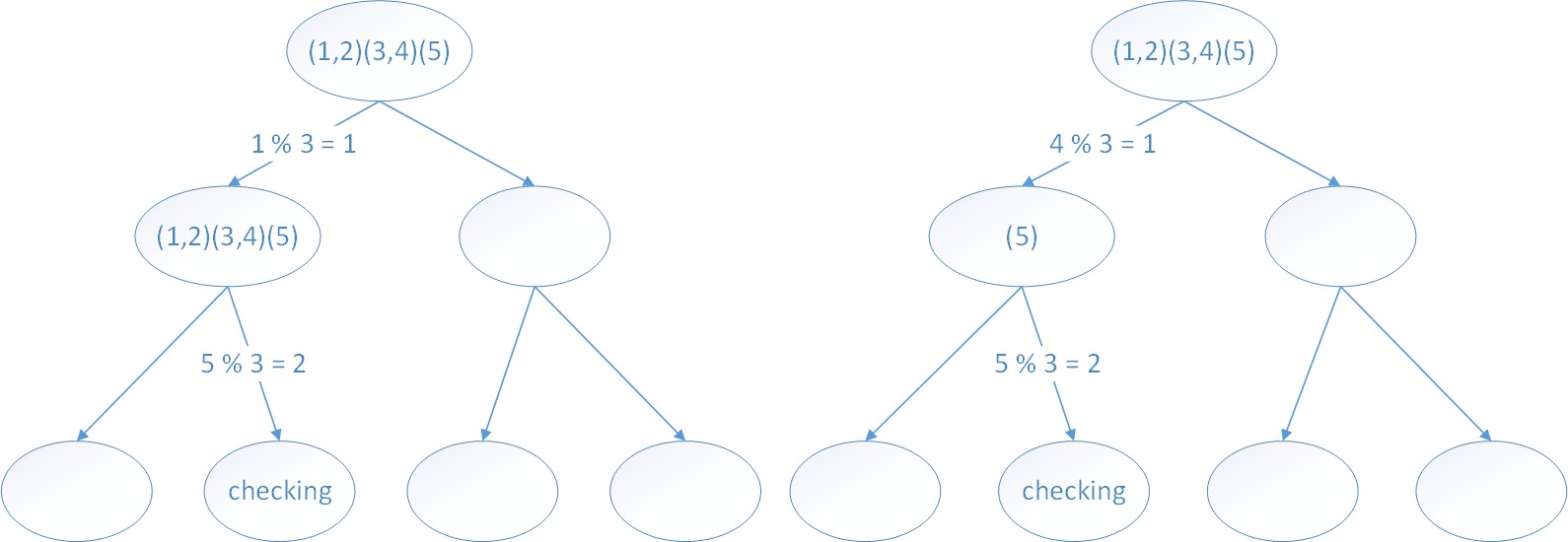


Figure 1: Hash searching

## 11.2 - Reducing duplicate searching

From the figure we can find that though sequence (1, 2)(3, 4)(5) was hashed in different ways, the final leaf node to be checked is the same. Indeed, not only the leaf nodes, the interior nodes may encounter duplicate visit. The duplicate visit cost much time, since each visit is a recursive function. So we want to eliminate the duplicate visit of nodes. That means we will not visit one interior node twice with the same state, in the meanwhile visit every potential leaf nodes. Here we define the condition that we shall stop further visiting.

item index : idx For on node v in hash-tree, now the sequence’s idxth item is hashed and the algorithm try further to apply recursive searching function on v.

Since for node v, we do not care the hashed items before reaching v. We just care which item was hashed last time. If the last hashed items of two distinct visit were hashed into the same position of the sequence, the following process of the searching is the same too. So for each sequence, we save all the different visiting record of each node when the searching function visiting them. Before searching along this node, we check whether this node has been visited with current sate. If so, we will stop this searching function. In order to record the visiting state of each node, we design the data structure as below.

From the ﬁgure we can ﬁnd that though sequence (1,2)(3,4)(5) was hashed in diﬀerent ways, the ﬁnal leaf node to be checked is the same. Indeed, not only the leaf nodes, the interior nodes may encounter duplicate visit. The duplicate visit cost much time, since each visit is a recursive function. So we want to eliminate the duplicate visit of nodes. That means we will not visit one interior node twice with the same state, in the meanwhile visit every potential leaf nodes.

Here we deﬁne the condition that we shall stop further visiting.

* item index : idx For on node v in hash-tree, now the sequence’s idxth item is hashed and the algorithm try further to apply recursive searching function on v.

Since for node v, we do not care the hashed items before reaching v. We just care which item was hashed last time. If the last hashed items of two distinct visit were hashed into the same position of the sequence, the following process of the searching is the same too. So for each sequence, we save all the diﬀerent visiting record of each node when the searching function visiting them. Before searching along this node, we check whether this node has been visited with current sate. If so, we will stop this searching function.

**Algorithm 2** Search along hash-tree

**Require:** Sequence *s*, hash-tree node *v* and last hashed *idx*

**function** SearchWithSate(*v, s, idx*)

2: **if** *v* is leaf node **then**

CheckPatterns(*v.patterns*);

4: **else**

**for** *i* = *idx* + 1 *s.length* **do**

6: **if** *idx* 0 **And** *s*[*i*]*.time s*[*idx*]*.time <*= *max gap* **then**

*child idx* = Hash((*s*[*i*]));

8: *child* = *v.children*[*child idx*];

**if** *child.visted*[*i*] = *false* **then**

10: *child.visted*[*i*] = *true*;

SearchNode(*child, s, i*);

12: **end if**

end if

14: **end for end if**

16: **end function**

After adding vector visited for every node, the recursive function will not visit one node with the same item index twice. Take the search case below for example,

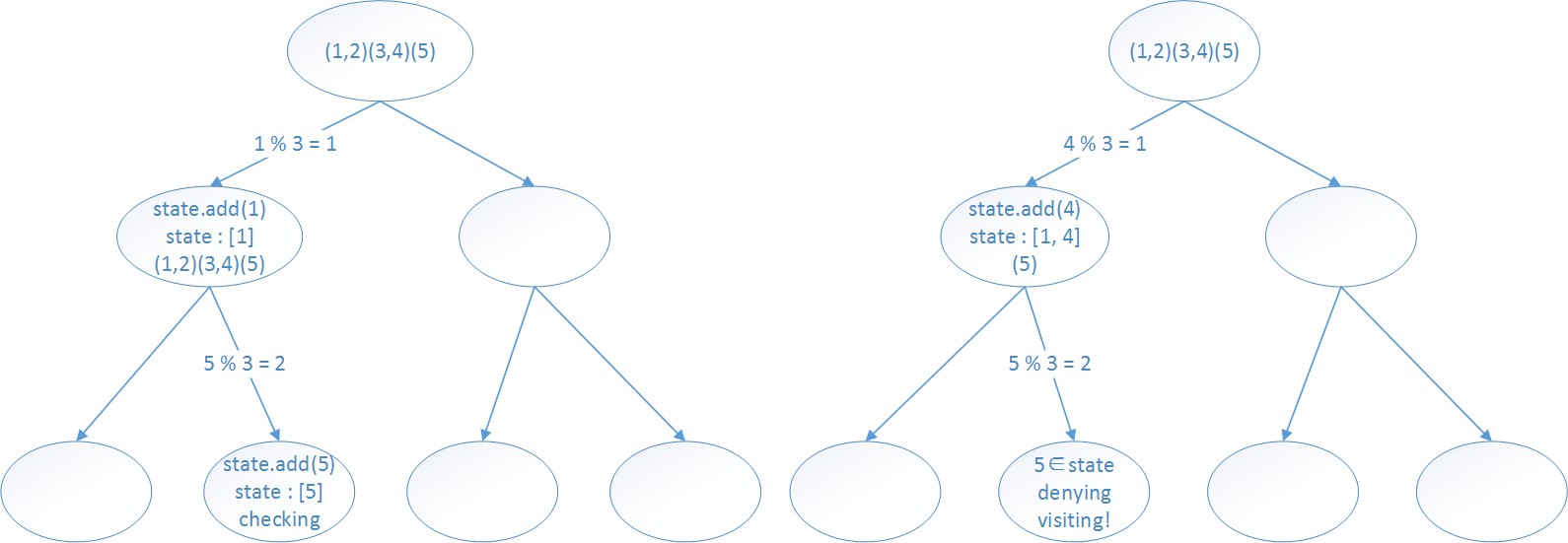


Figure 2: Hash searching with state

The figure in the left present one of the execution of recursive searching function. Two nodes add new state in this execution, including one leaf node. When another searching process try to visit the leaf node with the same state, it is denied. With the state vector of every node, our algorithm is much faster than the original one.

The following pseudo-code illustrates the procedure of checking a specific can- didate in a sequence

**Algorithm 3** Check with time-list

**Require:** Candidate *c*, Sequence *s*

**function** SearchWithSate(*c, s*)

*start idx* = 0*, pos*[] = *NULL*;

3: **while** true **do**

*item set* = *c*[*start idx*];

**while** *item set*! = *c.end*() **do**

6: Find a point where *items* in *itemset* has interval WindowsSize

≤

end while

Increase *pos*[] for each position in *item set*;

9: **if** do not find any *item set* **then**

return *false*;

end if

12: **if** *pos*[*start idx*]*.time* **match** *min gap* **and** *max gap* **then**

+ + *start idx*;

**else if** *pos*[*start idx*]*.time* **exceed** *max gap* **then**

15: *start idx*;

— −

end if

**if** *start idx* == *c.size*() **then**

18: return *true*.

end if end while

21: **end function**

In Line 8, we increase the position of matching index in the sequence. Since the indexed will never decrease, we only have to traverse the sequence once.

# 12 – Result Visualization

## **12.1 -** Get some data.

To discover sequential patterns, we will use the SPMF software. Therefore the data has to be in SPMF format.  In this blog post, I will just use the BMS1\_spmf.txt dataset webpage of the SPMF website. It is already in SPMF format.

## **12.2 - Extracting sequential patterns.**

Then using the **SPMF.jar** file downloaded from the SPMF website, I have applied the **PrefixSpan** algorithm to discover frequent sequences of webpages visited by the users.  I have set  the minsup  parameter of PrefixSpan to 0.15, which means that a sequence of webpages is frequent if it is visited by at least 15% of the users.

The result is an output file containing 5123 sequential patterns in a text file. For example, here are three patterns from the output file:

|  |
| --- |
| 1 -1 #SUP: 1959  2 -1 #SUP: 467  3 -1 #SUP: 4569  4 -1 #SUP: 793  6 -1 #SUP: 6058  7 -1 #SUP: 843  11 -1 #SUP: 3647  27 -1 #SUP: 684 |

The first line indicates that this patterns has a support of 1959. The result may be hard to understand in a text file, so we will next visualize them using GraphViz.

## **12.3 - Transforming the output file into GraphViz DOT format.**

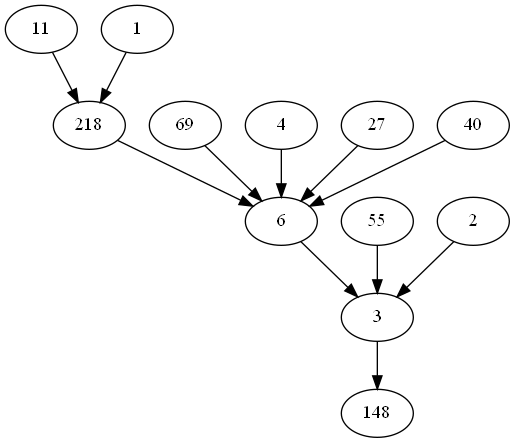
I use a very simple piece of Java code to transform the sequential patterns found by SPMF to the GraphViz DOT format.

## **12.4 - Generating a graph using GraphViz**

Then I installed GraphViz on my computer running Windows 10. GraphViz is a great software for the visualization of graphs and it is not very hard to use. The idea is that you feed GraphViz with a text file describing a graph and then he will automatically draw it. Of course there are many options that can be selected and here I just use the basic options. I use the command:  dot -Tpng patterns.dot > output.png"

This convert my DOT file to a graph in PNG format.

The result is the following.

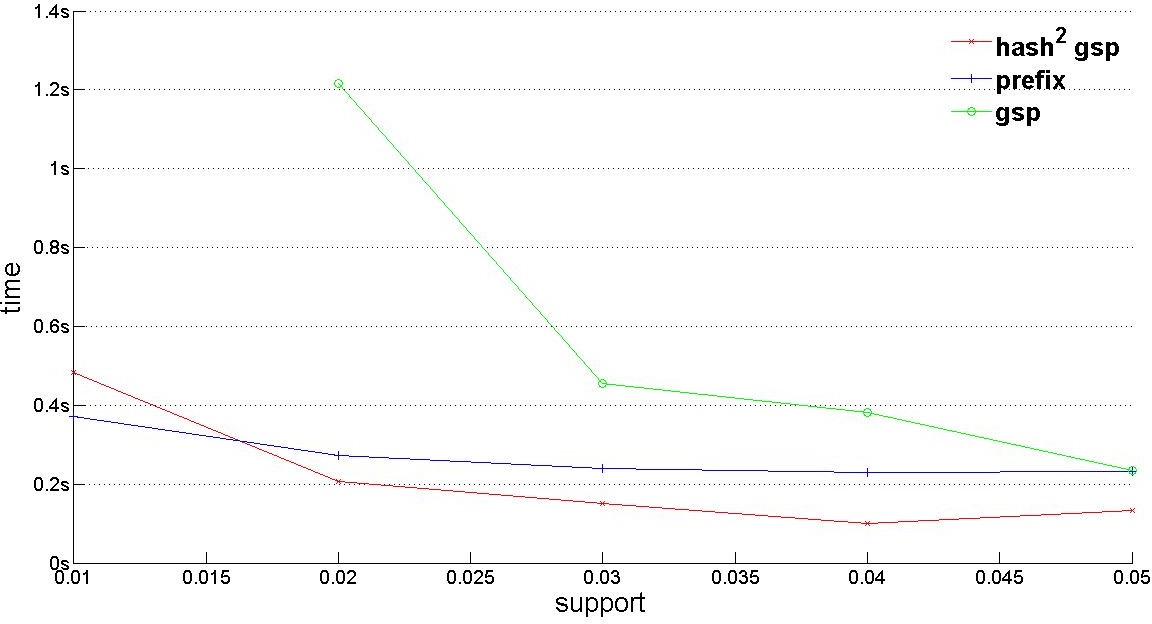


The result is pretty interesting.  It summarizes the 5123 patterns into an easy to understand graph. By looking at the graph we can easily see that many sequential patterns passes by the nodes 3 and 6, which must be very important webpages on the website.

Note that using this technique, we lose a little bit information. For example, the support of edges is not indicated on this graph.  It may be possible to improve upon this visualization for example by using various colors to indicate the support or the number of patterns that pass by a node.

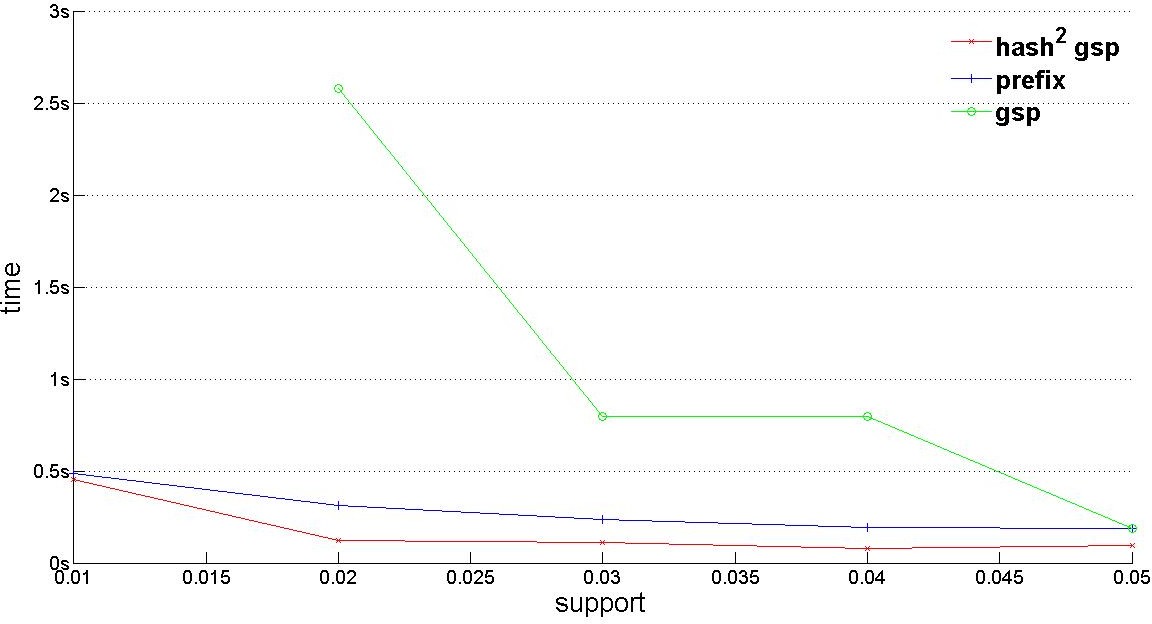
# 13 - Experiments and Comparison

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **algorithm** | **dataset** | **support** | **time(s)** | | --- | --- | --- | --- | | hash^2 gsp | kosarak10k | 0.05 | 0.133 | | gsp | kosarak10k | 0.05 | 0.235 | | prefix | kosarak10k | 0.05 | 0.232 | | hash^2 gsp | kosarak10k | 0.04 | 0.099 | | gsp | kosarak10k | 0.04 | 0.382 | | prefix | kosarak10k | 0.04 | 0.23 | | hash^2 gsp | kosarak10k | 0.03 | 0.15 | | gsp | kosarak10k | 0.03 | 0.454 | | prefix | kosarak10k | 0.03 | 0.24 | | hash^2 gsp | kosarak10k | 0.02 | 0.207 | | gsp | kosarak10k | 0.02 | 1.217 | | prefix | kosarak10k | 0.02 | 0.272 | | hash^2 gsp | kosarak10k | 0.01 | 0.484 | | gsp | kosarak10k | 0.01 | 4.373 | | prefix | kosarak10k | 0.01 | 0.372 | |



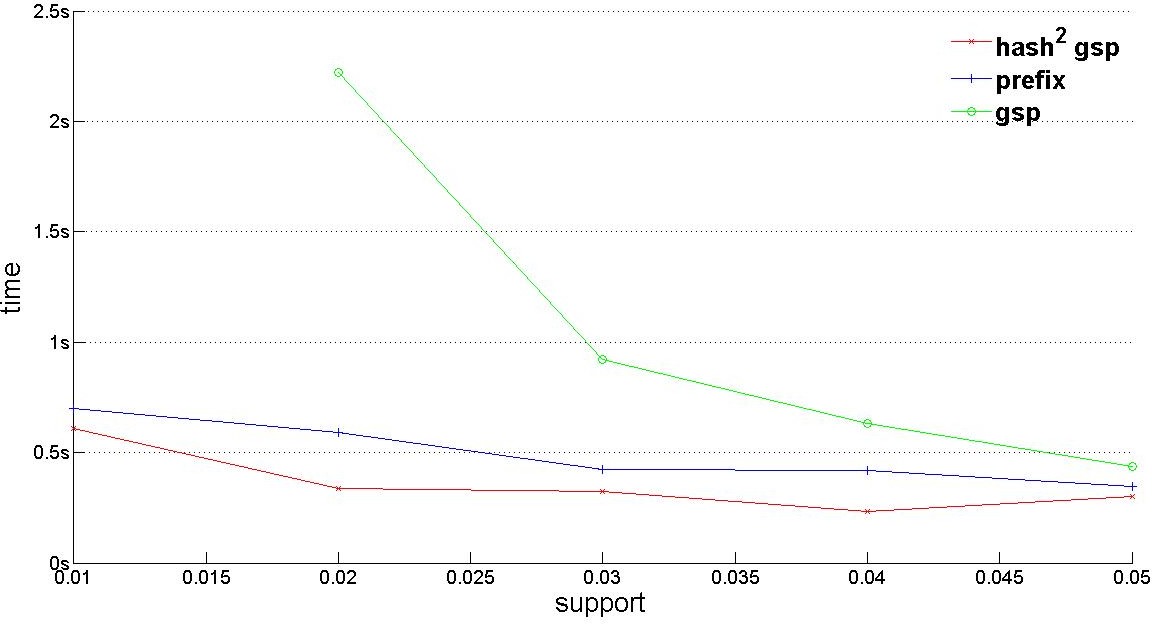
1. kosarak10k

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **algorithm** | **dataset** | **support** | **time(s)** | | --- | --- | --- | --- | | hash^2 gsp | kosarak25k | 0.05 | 0.299 | | gsp | kosarak25k | 0.05 | 0.436 | | prefix | kosarak25k | 0.05 | 0.345 | | hash^2 gsp | kosarak25k | 0.04 | 0.233 | | gsp | kosarak25k | 0.04 | 0.631 | | prefix | kosarak25k | 0.04 | 0.42 | | hash^2 gsp | kosarak25k | 0.03 | 0.323 | | gsp | kosarak25k | 0.03 | 0.921 | | prefix | kosarak25k | 0.03 | 0.425 | | hash^2 gsp | kosarak25k | 0.02 | 0.339 | | gsp | kosarak25k | 0.02 | 2.22 | | prefix | kosarak25k | 0.02 | 0.591 | | hash^2 gsp | kosarak25k | 0.01 | 0.611 | | gsp | kosarak25k | 0.01 | 8.95 | | prefix | kosarak25k | 0.01 | 0.7 | |



1. kosarak10k

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **algorithm** | **dataset** | **support** | **time(s)** | | --- | --- | --- | --- | | hash^2 gsp | kosarak25k | 0.05 | 0.299 | | gsp | kosarak25k | 0.05 | 0.436 | | prefix | kosarak25k | 0.05 | 0.345 | | hash^2 gsp | kosarak25k | 0.04 | 0.233 | | gsp | kosarak25k | 0.04 | 0.631 | | prefix | kosarak25k | 0.04 | 0.42 | | hash^2 gsp | kosarak25k | 0.03 | 0.323 | | gsp | kosarak25k | 0.03 | 0.921 | | prefix | kosarak25k | 0.03 | 0.425 | | hash^2 gsp | kosarak25k | 0.02 | 0.339 | | gsp | kosarak25k | 0.02 | 2.22 | | prefix | kosarak25k | 0.02 | 0.591 | | hash^2 gsp | kosarak25k | 0.01 | 0.611 | | gsp | kosarak25k | 0.01 | 8.95 | | prefix | kosarak25k | 0.01 | 0.7 | |



(c)BMS1 spmf60k

Our experiments are on the environment of linux core 3.13.0-30-generic, with distribution of Ubuntu 14.04 LTS 64bit. The hardware platform is Intel Core i7-4600U CPU @ 2.10GHz.

In our experiments, we compare three different algorithms by running data from [5], which are collected from real life. The traditional GSP algorithm and Prefixspan algorithm are implemented in the spmf [4]. Because the data are sparse, the support is set from 0.01 to 0.05. From the following results, we figure out that our Hash2 GSP is very much faster than the original one. In the front of large data, the Hash2 GSP provides a 5 10 times speed-up against the original one. Furthermore, it also renders a better average performance than the Prefixspan algorithm.

The support vector is (0.1, 0.2, 0.3, 0.4, 0.5) for hash2gsp and prefix, while the support vector of gsp is (0.2, 0.3, 0.4, 0.5) because the time of support 0.1 of gsp is too large to be shown in these figures. In figure 3(a), the time vec- tor of hash2gsp is (0.299, 0.233, 0.323, 0.339, 0.611), the time vector of prefix is (0.345, 0.42, 0.425, 0.591, 0.7) and the time vector of gsp is (0.436, 0.631, 0.921, 2.22).

In figure 3(b), the time vector of hash2gsp is (0.133, 0.099, 0.15, 0.207, 0.484), the time vector of prefix is (0.232, 0.23, 0.24, 0.272, 0.372) and the time vector of gsp is (0.235, 0.382, 0.454, 1.217). In figure 3(c), the time vector of hash2gsp is (0.093, 0.078, 0.109, 0.124, 0.453), the time vector of prefix is (0.188, 0.191, 0.235, 0.312, 0.485) and the time vector of gsp is (0.188, 0.796, 0.797, 2.58).

# 14 - Conclusion

In this project, we implement the GSP Algorithm (Generalized Sequential Pattern algorithm) and improve it.

The original version of the GSP[1] just reduced the number of checked candidates with hash-tree. We further study the mechanism of recursive searching and find that the execution of recursive function may visit one node for many times with the same state. This results duplicate search and the calling of recursive function costs much time.

We find the rule of calling recursive function and use vector in each hash- tree node to record the visit state. In this way, we sharply reduce the duplicate searching space. We compare the running time of our algorithm with several other algorithms and find that our algorithms is much faster than the original GSP. Moreover, our algorithm beats the prefix in the databases from [5]. Currently, our algorithm’s efficiency heavily depends on the number of hashed branches and the max size that a hash-tree node can contain. We think there parameters are related with the distribution of items. So we may further study the relationship between parameters and the distribution of items. We expect this heuristics algorithm will performance better. If possible we will provide the upper bound of this heuristics algorithm based on the distribution of real data

# 15 – References

[1] https://medium.com/@nicharuch/collocations-identifying-phrases-that-act-like-individual-words-in-nlp-f58a93a2f84a

[2] https://www.kaggle.com/code/ragnisah/text-data-cleaning-tweets-analysis/notebook

[3]http://tecdat.cn/python%E5%9C%A8%E7%BA%BF%E9%9B%B6%E5%94%AE%E6%95%B0%E6%8D%AE%E5%85%B3%E8%81%94%E8%A7%84%E5%88%99%E6%8C%96%E6%8E%98apriori%E7%AE%97%E6%B3%95%E6%95%B0%E6%8D%AE%E5%8F%AF%E8%A7%86%E5%8C%96/

[4] http://simpledatamining.blogspot.com/2015/03/generalized-sequential-pattern-gsp.html

[5] https://www.emerald.com/insight/content/doi/10.1108/K-05-2020-0258/full/html

[6] https://github.com/chen0040/java-sequential-pattern-mining  
[7] https://flexiple.com/python/python-timestamp#section3

[8] https://towardsdatascience.com/nlp-in-python-data-cleaning-6313a404a470

1. https://www.emerald.com/insight/content/doi/10.1108/K-05-2020-0258/full/html [↑](#footnote-ref-1)
2. https://www.geeksforgeeks.org/generalized-sequential-pattern-gsp-mining-in-data-mining/ [↑](#footnote-ref-2)
3. https://github.com/chen0040/java-sequential-pattern-mining [↑](#footnote-ref-3)
4. https://www.youtube.com/watch?v=SbVhtoO5Is4 [↑](#footnote-ref-4)
5. https://www.youtube.com/watch?v=SbVhtoO5Is4 [↑](#footnote-ref-5)
6. https://link.springer.com/chapter/10.1007/BFb0014140 [↑](#footnote-ref-6)
7. https://www.kaggle.com/code/ragnisah/text-data-cleaning-tweets-analysis/notebook [↑](#footnote-ref-7)
8. https://towardsdatascience.com/nlp-in-python-data-cleaning-6313a404a470 [↑](#footnote-ref-8)
9. https://intelligentonlinetools.com/blog/2018/02/10/how-to-create-data-visualization-for-association-rules-in-data-mining/ [↑](#footnote-ref-9)
10. Marzena Kryszkiewicz, Lukasz Skonieczny: Fast Discovery of Generalized Sequential Patterns. Intelligent Methods and Big Data in Industrial Applications 2019: 155-170 [↑](#footnote-ref-10)