TD3: Declarative Itemset Mining

Pablo Mollá Chárlez

February 4, 2025

Contents

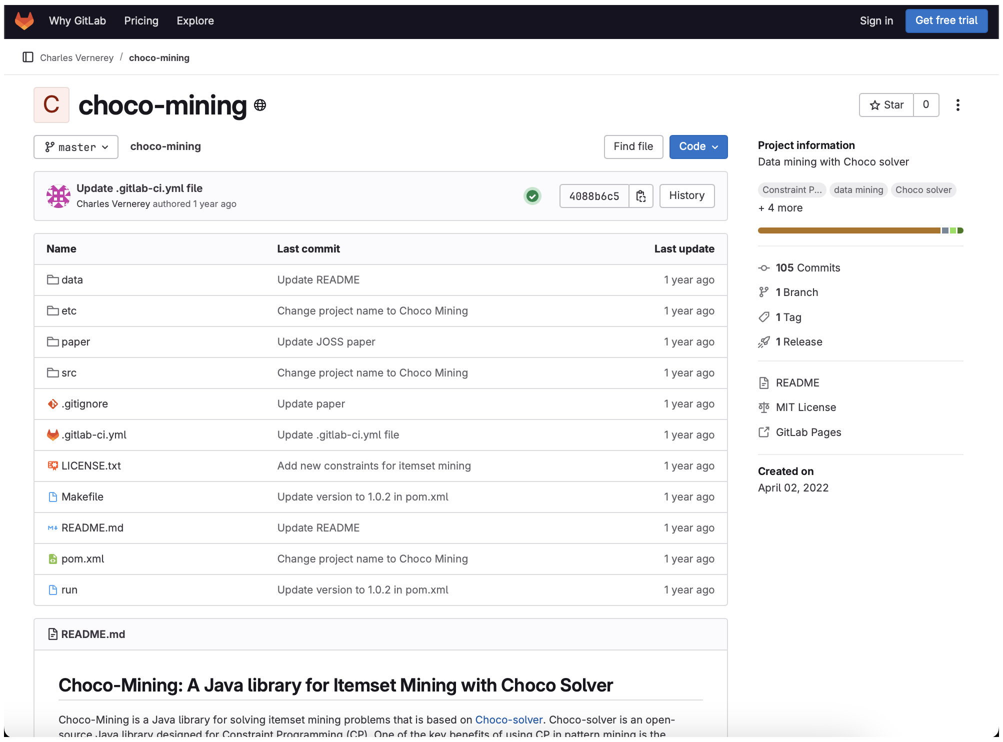
1 Exercise 1

In this exercise, you will work with the following tools:

- **Choco-Mining:** A Java library designed for solving itemset mining problems, built on the Choco-solver framework.
- **The SPMF library:** An open-source Java-based software and data mining library specializing in pattern mining (SPMF).

1.1 Question 1

Clone the GitHub repository of Choco-Mining (link)



Update .gitlab-ci.yml file
Charles Vernerey authored 1 year ago

Name	Last commit	Last update
data	Update README	1 year ago
etc	Change project name to Choco Mining	1 year ago
paper	Update JOSS paper	1 year ago
src	Change project name to Choco Mining	1 year ago
.gitignore	Update paper	1 year ago
.gitlab-ci.yml	Update .gitlab-ci.yml file	1 year ago
LICENSE.txt	Add new constraints for itemset mining	1 year ago
Makefile	Update version to 1.0.2 in pom.xml	1 year ago
README.md	Update README	1 year ago
pom.xml	Change project name to Choco Mining	1 year ago
run	Update version to 1.0.2 in pom.xml	1 year ago

Project information
Data mining with Choco solver
Constraint P... data mining Choco solver
+ 4 more
105 Commits
1 Branch
1 Tag
1 Release
README
MIT License
GitLab Pages
Created on
April 02, 2022

README.md

Choco-Mining: A Java library for Itemset Mining with Choco Solver

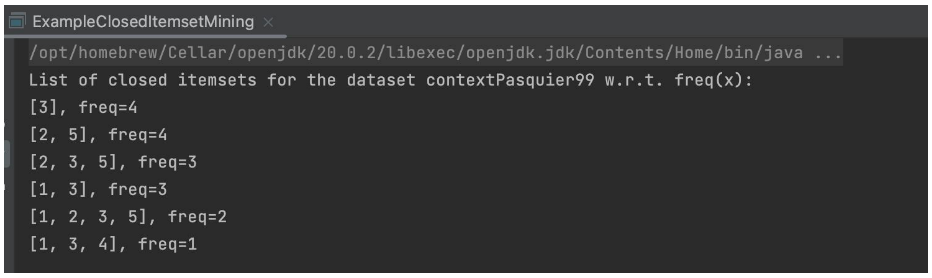
Choco-Mining is a Java library for solving itemset mining problems that is based on Choco-solver. Choco-solver is an open-source Java library designed for Constraint Programming (CP). One of the key benefits of using CP in pattern mining is the

Figure 1: Choco-Mining Repository

1.2 Question 2

Open the file ExampleClosedItemsetMining.java and perform the following tasks:

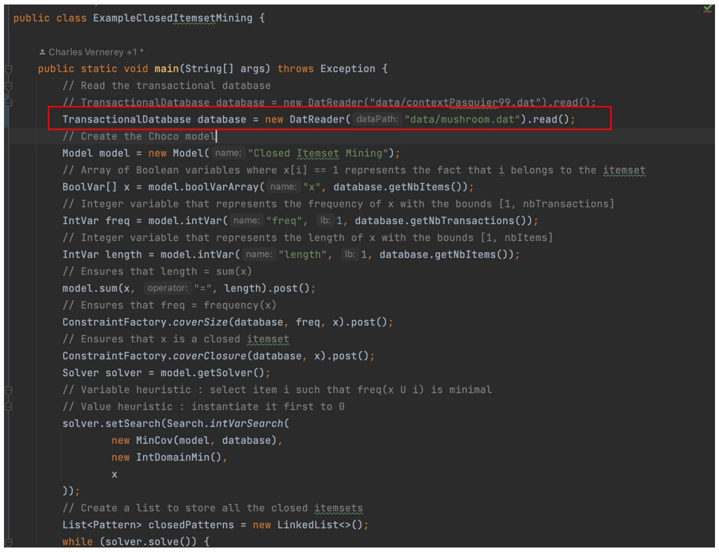
1. Review the code in detail.
2. Run the main method.



```
/opt/homebrew/Cellar/openjdk/20.0.2/libexec/openjdk.jdk/Contents/Home/bin/java ...
List of closed itemsets for the dataset contextPasquier99 w.r.t. freq(x):
[3], freq=4
[2, 5], freq=4
[2, 3, 5], freq=3
[1, 3], freq=3
[1, 2, 3, 5], freq=2
[1, 3, 4], freq=1
```

Figure 2: Main Method Result

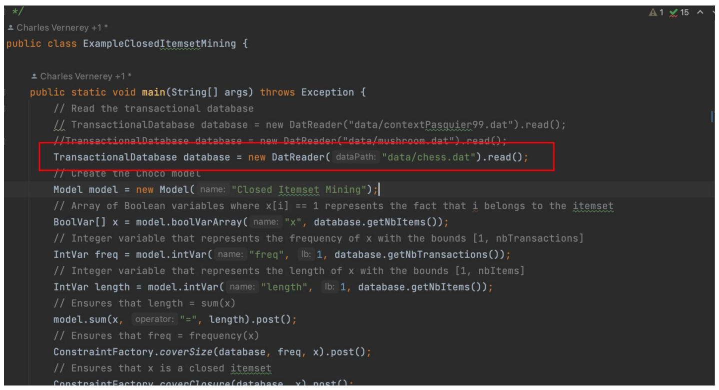
3. Run it on other datasets such as mushroom or chess.



```
public class ExampleClosedItemsetMining {

    ≗ Charles Vernerey +1 *
    public static void main(String[] args) throws Exception {
        // Read the transactional database
        // TransactionalDatabase database = new DatReader("data/contextPasquier99.dat").read();
        TransactionalDatabase database = new DatReader( dataPath: "data/mushroom.dat").read();
        // Create the Choco model
        Model model = new Model( name: "Closed Itemset Mining");
        // Array of Boolean variables where x[i] == 1 represents the fact that i belongs to the itemset
        BoolVar[] x = model.boolVarArray( name: "x", database.getNbItems());
        // Integer variable that represents the frequency of x with the bounds [1, nbTransactions]
        IntVar freq = model.intVar( name: "freq",  lb: 1, database.getNbTransactions());
        // Integer variable that represents the length of x with the bounds [1, nbItems]
        IntVar length = model.intVar( name: "length",  lb: 1, database.getNbItems());
        // Ensures that length = sum(x)
        model.sum(x,  operator: "=", length).post();
        // Ensures that freq = frequency(x)
        ConstraintFactory.coverSize(database, freq, x).post();
        // Ensures that x is a closed itemset
        ConstraintFactory.coverClosure(database, x).post();
        Solver solver = model.getSolver();
        // Variable heuristic : select item i such that freq(x U i) is minimal
        // Value heuristic : instantiate it first to 0
        solver.setSearch(Search.intVarSearch(
                new MinCov(model, database),
                new IntDomainMin(),
                x
        ));
        // Create a list to store all the closed itemsets
        List<Pattern> closedPatterns = new LinkedList<>();
        while (solver.solve()) {
```

Figure 3: Mushroom Modification Code



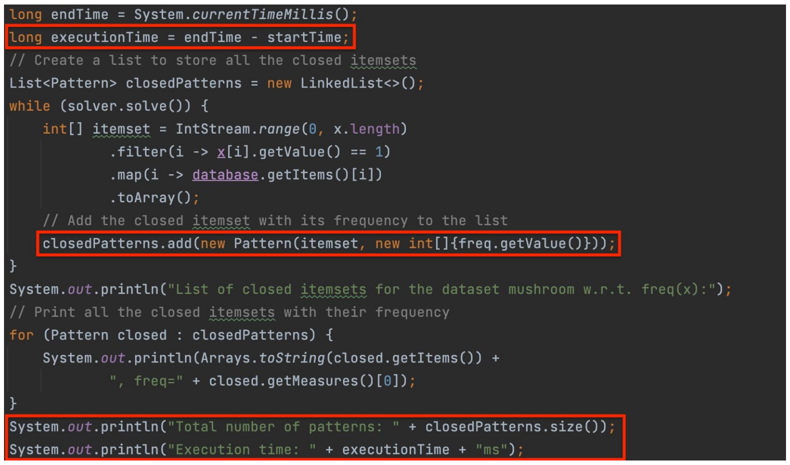
```
*/
≗ Charles Vernerey +1 *
public class ExampleClosedItemsetMining {

    ≗ Charles Vernerey +1 *
    public static void main(String[] args) throws Exception {
        // Read the transactional database
        // TransactionalDatabase database = new DatReader("data/contextPasquier99.dat").read();
        //TransactionalDatabase database = new DatReader("data/mushroom.dat").read();
        TransactionalDatabase database = new DatReader( dataPath: "data/chess.dat").read();
        // Create the Choco model
        Model model = new Model( name: "Closed Itemset Mining");
        // Array of Boolean variables where x[i] == 1 represents the fact that i belongs to the itemset
        BoolVar[] x = model.boolVarArray( name: "x", database.getNbItems());
        // Integer variable that represents the frequency of x with the bounds [1, nbTransactions]
        IntVar freq = model.intVar( name: "freq",  lb: 1, database.getNbTransactions());
        // Integer variable that represents the length of x with the bounds [1, nbItems]
        IntVar length = model.intVar( name: "length",  lb: 1, database.getNbItems());
        // Ensures that length = sum(x)
        model.sum(x,  operator: "=", length).post();
        // Ensures that freq = frequency(x)
        ConstraintFactory.coverSize(database, freq, x).post();
        // Ensures that x is a closed itemset
        ConstraintFactory.coverClosure(database, x).post();
```

Figure 4: Chess Modification Code

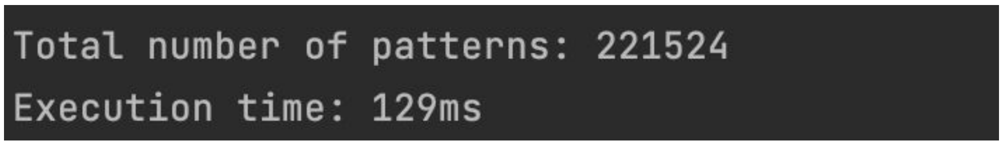
4. Display the number of resulting patterns.

5. Display the execution time.



```
long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;
// Create a list to store all the closed itemsets
List<Pattern> closedPatterns = new LinkedList<>();
while (solver.solve()) {
    int[] itemset = IntStream.range(0, x.length)
            .filter(i -> x[i].getValue() == 1)
            .map(i -> database.getItems()[i])
            .toArray();
    // Add the closed itemset with its frequency to the list
    closedPatterns.add(new Pattern(itemset, new int[]{freq.getValue()}));
}
System.out.println("List of closed itemsets for the dataset mushroom w.r.t. freq(x):");
// Print all the closed itemsets with their frequency
for (Pattern closed : closedPatterns) {
    System.out.println(Arrays.toString(closed.getItems()) +
            ", freq=" + closed.getMeasures()[0]);
}
System.out.println("Total number of patterns: " + closedPatterns.size());
System.out.println("Execution time: " + executionTime + "ms");
```

Figure 5: Patterns and Time Execution Modification



```
Total number of patterns: 221524
Execution time: 129ms
```

Figure 6: Patterns and Time Execution for Mushroom Dataset

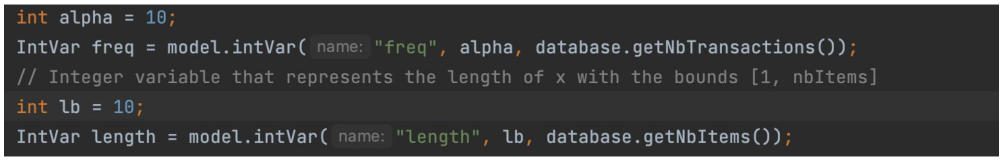
1.3 Question 3

Add the frequency constraint: $freq(P) \geq \alpha$

In next section, the image answers the question.

1.4 Question 4

Add a constraint on the size of the returned patterns: $size(P) \geq lb$.

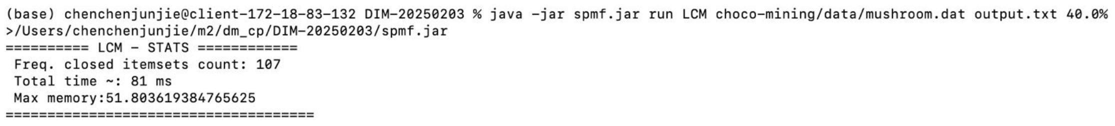


```
int alpha = 10;
IntVar freq = model.intVar( name: "freq", alpha, database.getNbTransactions());
// Integer variable that represents the length of x with the bounds [1, nbItems]
int lb = 10;
IntVar length = model.intVar( name: "length", lb, database.getNbItems());
```

Figure 7: Constraint on frequency and size of returned patterns

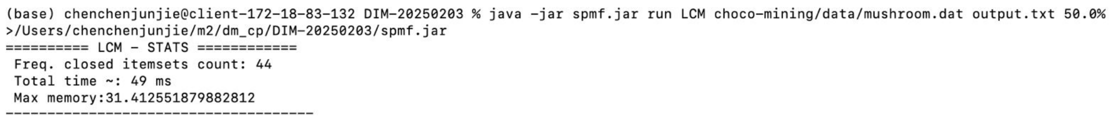
1.5 Question 5

Now, replicate the tasks using SPMF. Run the **.jar** file available in your local repository. The goal is to run LCM for closed itemset enumeration, relaunch with different thresholds for frequency, and also for pattern size.



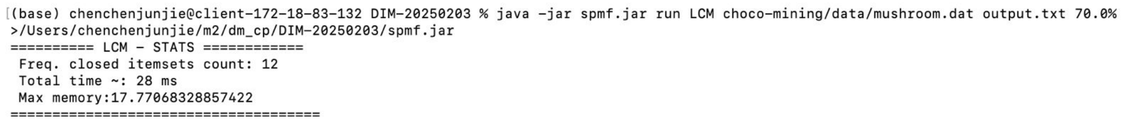
```
(base) chenchenjunjie@client-172-18-83-132 DIM-20250203 % java -jar spmf.jar run LCM choco-mining/data/mushroom.dat output.txt 40.0%
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCM - STATS ============
 Freq. closed itemsets count: 107
 Total time ~: 81 ms
 Max memory:51.803619384765625
=====================================
```

Figure 8: SPMF: Mushroom Dataset with 40% Frequency



```
(base) chenchenjunjie@client-172-18-83-132 DIM-20250203 % java -jar spmf.jar run LCM choco-mining/data/mushroom.dat output.txt 50.0%
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCM - STATS ============
 Freq. closed itemsets count: 44
 Total time ~: 49 ms
 Max memory:31.412551879882812
--------------------------------------
```

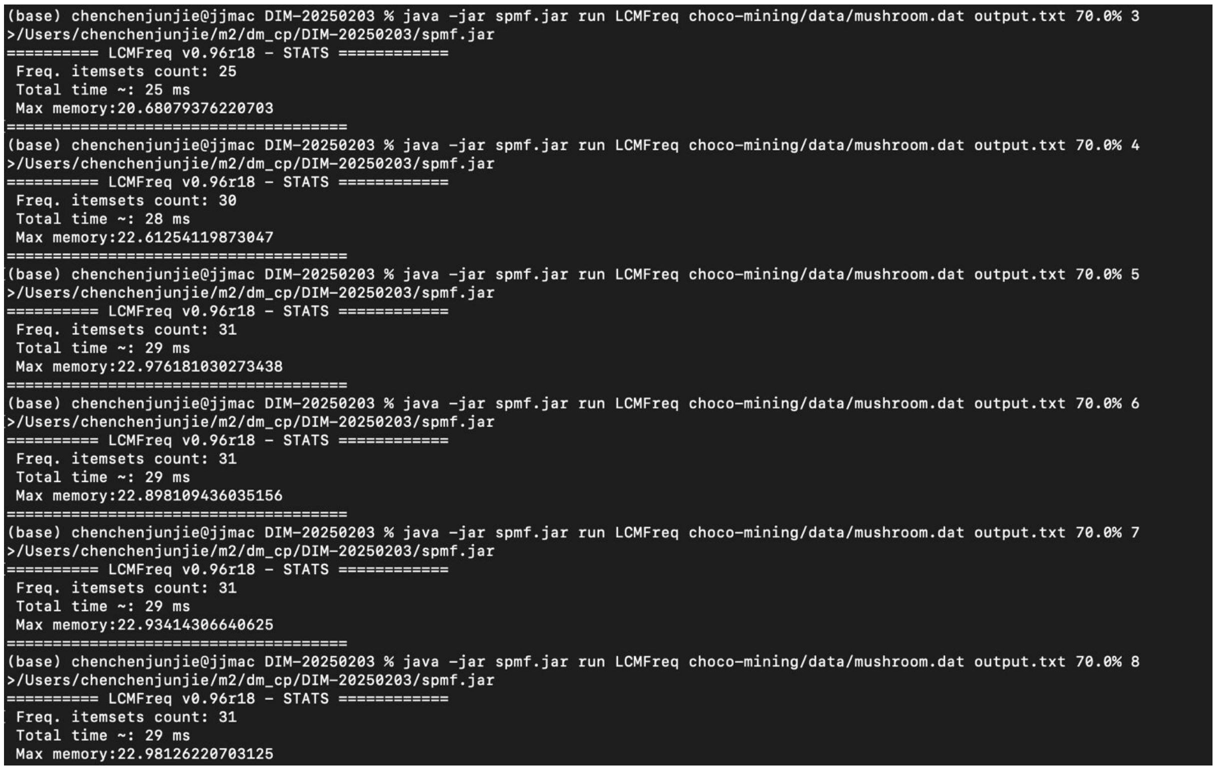
Figure 9: SPMF: Mushroom Dataset with 50% Frequency



```
(base) chenchenjunjie@client-172-18-83-132 DIM-20250203 % java -jar spmf.jar run LCM choco-mining/data/mushroom.dat output.txt 70.0%
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCM - STATS ============
 Freq. closed itemsets count: 12
 Total time ~: 28 ms
 Max memory:17.77068328857422
=====================================
```

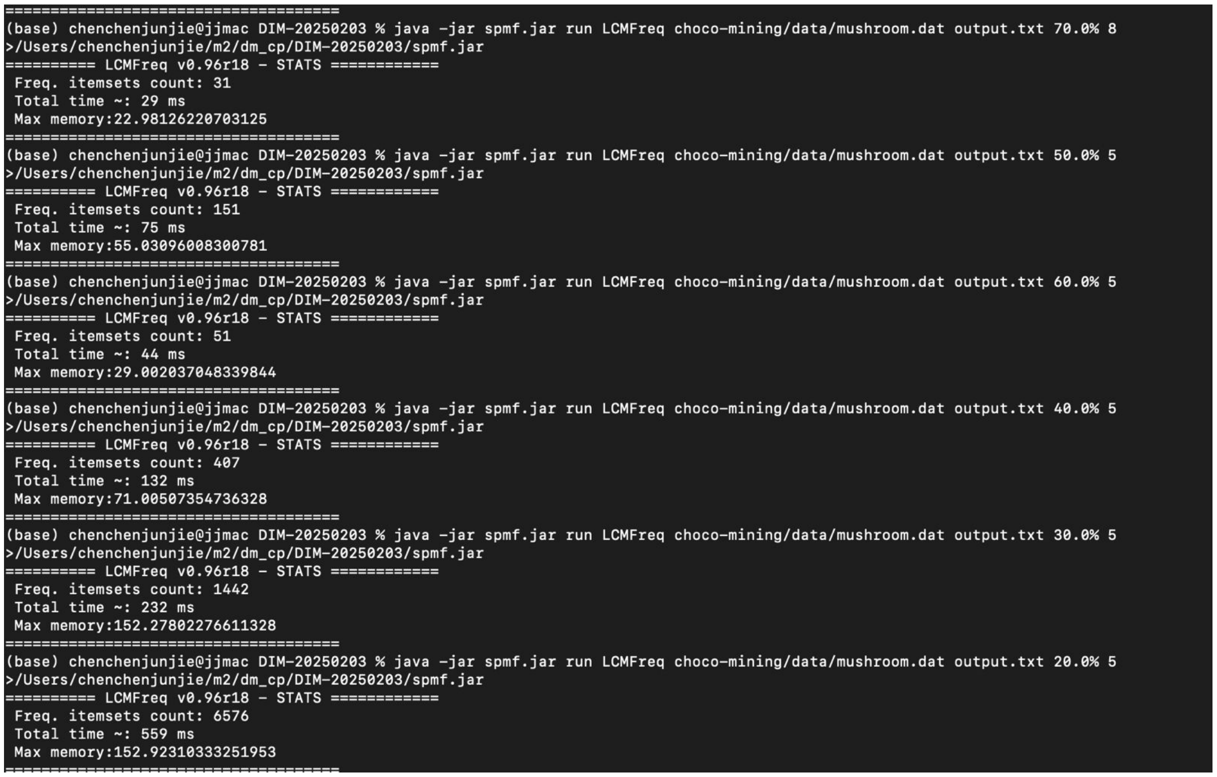
Figure 10: SPMF: Mushroom Dataset with 70% Frequency

Now, we apply modifications at both levels, **frequency** and **pattern size**.



```
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 70.0% 3
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 25
 Total time ~: 25 ms
 Max memory:20.68079376220703
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 70.0% 4
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 30
 Total time ~: 28 ms
 Max memory:22.61254119873047
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 70.0% 5
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 31
 Total time ~: 29 ms
 Max memory:22.976181030273438
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 70.0% 6
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 31
 Total time ~: 29 ms
 Max memory:22.898109436035156
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 70.0% 7
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 31
 Total time ~: 29 ms
 Max memory:22.93414306640625
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 70.0% 8
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 31
 Total time ~: 29 ms
 Max memory:22.98126220703125
```

Figure 11: SPMF: Mushroom Dataset with 70% Frequency and different pattern sizes



```
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 70.0% 8
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 31
 Total time ~: 29 ms
 Max memory:22.98126220703125
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 50.0% 5
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 151
 Total time ~: 75 ms
 Max memory:55.03096008300781
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 60.0% 5
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 51
 Total time ~: 44 ms
 Max memory:29.002037048339844
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 40.0% 5
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 407
 Total time ~: 132 ms
 Max memory:71.00507354736328
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 30.0% 5
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 1442
 Total time ~: 232 ms
 Max memory:152.27802276611328
===================================
(base) chenchenjunjie@jjmac DIM-20250203 % java -jar spmf.jar run LCMFreq choco-mining/data/mushroom.dat output.txt 20.0% 5
>/Users/chenchenjunjie/m2/dm_cp/DIM-20250203/spmf.jar
========== LCMFreq v0.96r18 - STATS ============
 Freq. itemsets count: 6576
 Total time ~: 559 ms
 Max memory:152.92310333251953
===================================
```

Figure 12: SPMF: Mushroom Dataset with different frequencies and pattern size (= 5)

1.6 Question 6

Add a constraint, called CategoryConstraint, to the file **ExampleClosedItem-setMining.java** to model the following problem : Consider a dataset with n items, organized into categories of size **catSize** (e.g., household products, appliances, etc.). The dataset is divided into $nbCat = \frac{n}{catSize}$ categories, with items that do not belong to any category (but do not exceed the size of **catSize**). Figure **??** shows an example with 8 items, 2 categories of size 3, and 2 items that do not belong to any category. The task is to create a constraint model that enumerates all closed itemsets composed of items belonging to at least m categories:

$$\text{CategoryConstraint(P)} = \sum_{i=1}^{\text{nbCat}} \prod_{j=1}^{\text{catSize}} P_i \geq m$$

For example, in the dataset shown in Figure **??**, with $m = 2$, the following pattern is produced: **BEF**.

1.7 Question 7

How can this **CategoryConstraint** be taken into account in SPMF ?