SplitterGUI

GUI Application of Mining Fine-Grained Sequential Patterns in Semantic Trajectories

Tanjung Dion, Fawwaz Dzaky Zakiyal
Department of Electrical and Computer Engineering
Pusan National University
Busan, Republic of Korea
{tanjung.dn, dzakybd}@gmail.com

***Abstract*—Semantic trajectory is a sequence of time-stamped places wherein each place has information about spatial location and a semantic label. By mining fine-grained pattern that satisfy semantic consistency, spatial compactness and temporal continuity, it will give benefit such as urban planning and targeted advertising. One of the state-the-art method to mine the fine-grained pattern is Splitter algorithm. Firstly, it find set of coarse patterns and their snippets by using PrefixSpan, and then find the set of fine-grained patterns using mean shift method. In this project we build, SplitterGUI, GUI application to visualize the process of the Splitter algorithm. We also discuss the key techniques in Splitter and conduct an experiment using 4SQ database to demonstrate the result of SplitterGUI.**

***Index Terms*—fine-grained pattern, semantic trajectories, GUI application.**

I. Introduction

The sequential patterns can be found in a trajectory database, if it properly extracted. It become benefit for targeted advertising, urban planning, location prediction, etc. In other hand, classic sequential pattern mining algorithms cannot work effectively in semantic trajectories, a sequence of time-stamped places wherein each place has information about spatial location and a semantic label. Because of the places in the continuous space cannot be regarded as independent items, so independent place cannot be used to find the frequent sequences. Instead, semantic label need to be grouped to form frequent sequential patterns. But, there is also a challenge to get fine-grained sequential patterns, that it must fulfill requirements of spatial compactness (compact area for each place category), semantic consistency (consistent place ID in each place category) and temporal continuity (limited time constraint).

The process of mining fine-grained pattern have been proposed by Zang et. al [1]. They provide two-step approach to obtain fine-grained sequential patterns. First-step, obtain the coarse patterns from the semantic trajectory data using the full projection method to modify PrefixSpan. The result patterns will fulfill the semantic and temporal constraints. Second-step, splitting each coarse patterns into fine-grained patterns that fulfill the spatial constraint through top-down pattern discovery process using weighted snippet shift by Epanechnikov kernel.

The SplitterGUI project files can be downloaded at $github.com/dzakybd/SplitterGUI$

In this project, aiming at implementing the mining fine-grained pattern method proposed by Zang et. al [1] and build the GUI application to visualize the process of the Splitter algorithm, called SplitterGUI. It make the database, course patterns and fine-grained patterns into a visualized map by determined Splitter algorithm's parameters. Afterwards, we discuss the key techniques in Splitter and conduct an experiment using 4SQ database to demonstrate the result of SplitterGUI.

II. Implementation

A. System Design

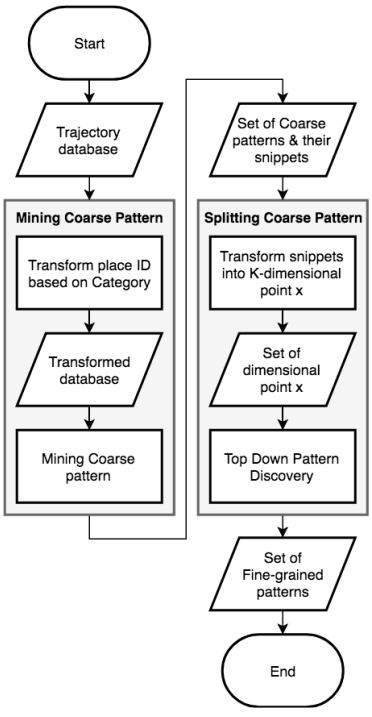


Fig. 1: System Design

Fig. 1 show the process of mining fine-grained pattern in semantic trajectories. This method consist of two part: mining coarse pattern and splitting coarse pattern. Mining coarse pattern is a process to find sequential pattern that satisfy semantic consistency and temporal continuity. In the

first step, we add trajectory database that have information about place id, movement time, group id, and spatial location to the system. This trajectory database will be transformed into semantic trajectory database by grouping place id based on category. However, we still keep the original place id in each category to prevent one trajectory from being counted repeatedly. By doing this process, our trajectory database will satisfy semantic consistency.

Then, PrefixSpan algorithm will be used to mining the coarse pattern. In the PrefixSpan algorithm, we include all post-fixes to avoid missing pattern under the given time constrain. Output from mining coarse pattern is set of coarse pattern that will be used in the next process. The next process is finding fine-grained pattern that satisfy spatial compactness in by splitting a coarse pattern in top down manner. Set of coarse pattern from previous process will be transformed into K-dimensional point x.

Then, we employ mean shift clustering to extract the dense and compact snippet cluster based on the support and variance threshold. To speed up the process, the unqualified snippet cluster will be organized into several disjoint communities. By only clustering each communities, we can gradually reduce the kernel windows in clustering method to speed up the clustering process.

B. Key techniques in Splitter

1) Mining Coarse Pattern: Before mining coarse pattern, trajectory database D should be transformed to semantic trajectory by grouping each place based on its category. The purpose is to find any frequent sequences that have similar categories. For example, Fig. 2 shows database transformation with office $= G_1$, shop $= G_2$, restaurant $= G_3$.

Obj.	Semantic Trajectory
O_1	<($G_{1<P3>}$, 0),($G_{1<P1>}$, 10), ($G_{2<P7>}$, 30), ($G_{3<P9>}$, 40)>
O_2	<($G_{1<P2>}$, 0) ,($G_{2<P7>}$, 30), ($G_{1<p5>}$, 130), ($G_{2<P4>}$, 160), ($G_{3<P9>}$, 170)>
O_3	<($G_{1<P3>}$, 0),($G_{2<P4>}$, 30), ($G_{3<P6>}$, 60)>

Fig. 2: Semantic trajectory database

After transformation, we mining sequential pattern that satisfy support threshold σ and time constraint Δt using PrefixSPan algorithm. PrefixSpan algorithm use frequent item as short pattern and prefixes to build projected database then grow the short pattern by searching local frequent item in projected database. However, PrefixSpan algorithm should be modified to satisfy our coarse pattern. 3 shows the modified prefix span algorithm. First, we extract all frequent item at least σ in semantic trajectory D. Then for each item we build item-projected database as S using full projection. Full projection means that we extract all pattern that occurred in database. For example, object 0 in 2 has 2 pattern $P_3 \rightarrow P_7 \rightarrow P_9$ and $P_1 \rightarrow P_7 \rightarrow P_9$.

Output this item as our short pattern and then grow this pattern by calling PrefixSpan function. This function has similar procedure with initial projection function. However, we check time constraint when searching frequent item in projected database. We also call this PrefixSpan function recursively to grow the pattern until the pattern cannot be grown anymore. Output of this algorithm is list of coarse pattern with its snippet (i.e. $P_1 \rightarrow P_7 \rightarrow P_9$), group (i.e. $G_1 \rightarrow G_2 \rightarrow G_3$), and pattern length (i.e. 3).

```
Algorithm 1 : Mining Coarse Pattern
Input : transformed semantic database D, support threshold σ, temporal
constraint Δt
1  Procedure InitialProjection(D, σ, Δt)
2      L ← Frequent items in D at least σ;
3      foreach item i in L do
4          S ← {} ;
5          foreach trajectory o in D
6              R ← postfixes all occurrences of i in o;
7              S ← S U R
8          Output i and its snippets;
9          PrefixSpan(i,1,S| i, Δt);
10 Function PrefixSpan(α, l, S| α, Δt)
11     L ← Frequent item in S| α at least σ and Δt;
12     foreach item i in S| α do
13         α' ← append i to α;
14         S ← {} ;
15         foreach trajectory o in D
16             R ← postfixes all occurrences of i in o;
17             S ← S U R
18         Output α' and its snippets;
20         PrefixSpan(α',1+1,S| α', Δt);
```

Fig. 3: Mining coarse pattern algorithm [1]

2) Finding Fine-Grained Pattern: The result of coarse pattern has satisfy support threshold σ and time constraint Δt. Next step is checking the spatial variance of place in each group category. For example, graph representation of coarse pattern is shown in Fig. 4. Spatial location of place P_5 with other place in group G_1 is to far. Therefore, we have to prune pattern $P_5 \rightarrow P_4 \rightarrow P_6$ from fine-grained pattern.

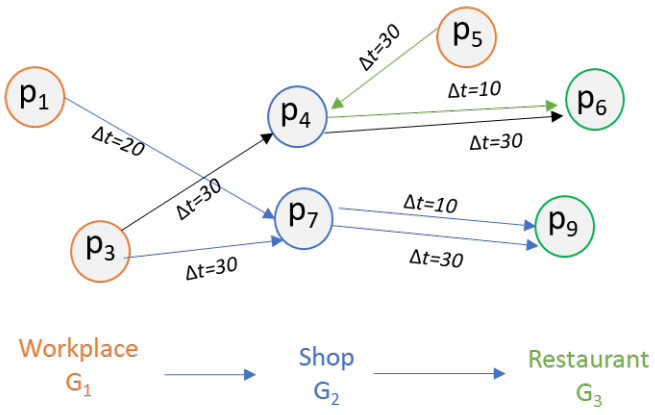


Fig. 4: Graph representation of coarse pattern result

For finding fine-grained pattern, mean shift clustering algorithm is used because it does not need any prior knowledge and only use one parameter called bandwidth. Bandwidth parameter is used to limit the area of clustering. The main idea of mean shift algorithm is choosing random point as

the center and use bandwidth to limit the coverage area of clustering. Then calculate mean shift vector to shift the center point toward new point that more dense than previous point. For windows kernel, we use Epanechnikov kernel.

Before clustering, we transform snippet in each coarse pattern into a 2K-dimensional point x by assembling the coordinates of each snippet. Then, the number of visitors is added to x as weight. Algorithm 2 shows how to cluster point using mean shift algorithm. Output of this algorithm is set of cluster.

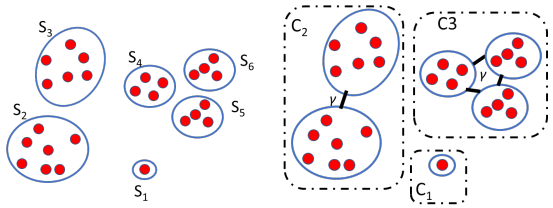
Algorithm 2 : Weighted snippet shift

Input : a set of S of weighted snippets, bandwidth h

foreach $\mathbf{x} \in S$ **do**
  $k \leftarrow 0$, $\mathbf{y}^{(0)} \leftarrow \mathbf{x}$;
  while True do
    $N_k \leftarrow \{X\}_{ii=1}^{m} s.t. \forall i, \left|\left|Y^{(k)} - Xi\right|\right| \leq h$;
    $y^{(k+1)} \leftarrow (\sum_{i=1}^{m} wixi)/(\sum_{i=1}^{m} wi)$;
  If $||\, y^{(k+1)} - y^{(k)} \,|| \leq \epsilon$ **then**
    return ($\mathbf{x}$; $\mathbf{y}^{(k+1)}$);
  $k \leftarrow k + 1$;

Fig. 5: Weighted snippet shift algorithm [1]

Specify the best bandwidth size is not an easy task. If bandwidth is small, there will be many small snippet cluster that not exceed support threshold σ. On the other hand, large bandwidth make the large cluster that cannot satisfy variance threshold rho. Therefore, we have to start with large bandwidth then dampen it until no more pattern exist. For this problem, notion of γ-community is presented to reduce the cost of repeating the process in large cluster. γ-community can be defined as connected component between two cluster. Two cluster can be connected if distance between two cluster no more than γ. Fig. 6 shows the example of γ-community.



(a) Snippet cluster (b) γ-community

Fig. 6: Example of γ-community

Algorithm 3 shows refine coarse pattern algorithm. First, we cluster snippet using weighted snippet shift with bandwidth h. For each cluster we check if the support is at least $sigma$ and the variance is no more than rho then report as fine grained pattern and remove from cluster set. Set γ as root 2 multiply by dampening factor and current bandwidth. Then construct set of γ-community from cluster S that are not reported as fine grained pattern. For each γ-community, check if the support is at least $sigma$ then repeat the process of refine coarse pattern algorithm.

Algorithm 3 : Refine Coarse Pattern

Input: snippet set S, support threshold σ, variance threshold ρ, bandwidth h, dampening factor τ

Procedure *SplitPattern*(S, σ ,ρ , h, τ)
  ΓS cluster S via weighted snippet shift with h;
  foreach $Si \in \Gamma S$ **do**
    if $Sup(Si) \geq \sigma$ *and* $Var(Si) \leq \rho$ **then**
      Report Si as a fine-grained pattern;
      Remove Si from ΓS;
  $\gamma \leftarrow \sqrt{2}\tau h$;
  $\Omega_C \leftarrow \gamma$-communities constructed from ΓS;
  foreach Ci *2* C **do**
    if $Sup(Ci) \geq \sigma$ **then**
      SplitPattern(Ci; σ; ρ; τh; τ);

Fig. 7: Refine coarse pattern algorithm [1]

III. Demonstration

A. Database

This project will use the real database, 4SQ that collected from Foursquare check-in sequence in New York. It consists of 48564 places that divided into 417 place with list of categories and it stored semantic trajectories from 14909 users. 4SQ database provides 3 files:

1) *Sequences*, it contains users semantic trajectories including attributes of check-in time-stamp and place ID.
2) *Places*, it contains information about the places (place ID, latitude, longitude, list of category ID).
3) *Category*, it contains information of place category ID corresponding with its place category name.

B. Implementation detail

We will re-implement the process of mining fine-grained pattern using Java programming language. For improving GUI performance, we use G4P library (lagers.org.uk/g4p) that based on OpenGL and JavaFX rendering engines. The other purpose is to support geographical map visualization from unfolding library (unfoldingmaps.org) that also use OpenGL rendering engines. We also using other libraries such as Apache Commons (commons.apache.org) to support our program.

Input of our program is database file that will be used to build graph representation as our database in the system. This graph representation is based on adjacency list that is shown in 8. There are 4 object: database, trajectory, place, and category. Database object has list of trajectory that have id to represent people and have list of sequence to represent the movement. Each movement is a list of time-stamped place that has information of place id, position, and category. In the next step, we implement mining coarse pattern algorithm and refine coarse pattern algorithm for finding fine-grained pattern from the database.

The user interface of program will be provided as shown in the Fig. 9. The figure shows the illustration of our program. There are 3 database input in .csv format for category and place and .txt format for the sequence data. There are also 6

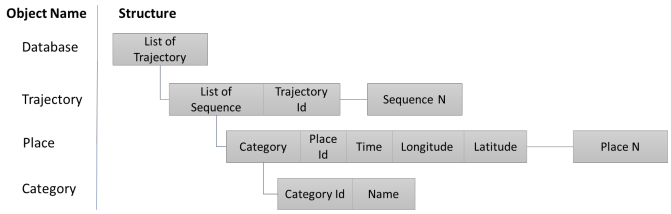


Fig. 8: Structure of database in adjacency list representation

input parameters in the configuration field. Support threshold, pattern length, and time limit for mining coarse pattern, and variance threshold, dampening factor for refine the coarse pattern. In the first step, we have to create the database. After that,we can mine coarse pattern first then perform refine coarse pattern or do both coarse pattern and refine coarse pattern respectively.

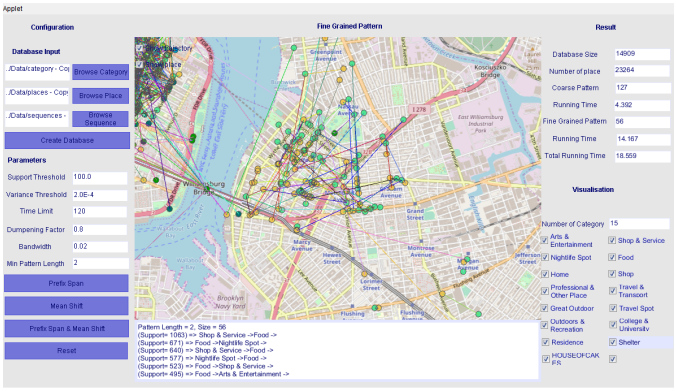


Fig. 9: Visualization

C. Experiment

Experiment is conducted in windows 10 Pro 64-bit running on Processor Intel Core I5 7500 @3.40GHz, 4GB of RAM, and Intel HD Graphics 630. First, we choose database file and fill the input parameters support threshold 100, variance threshold 0.0004, time constraint 120, dampening factor 0.8, Bandwidth 0.02, and minimum pattern length 2. In this application, minimum pattern length is used to set the minimum length of pattern when mining coarse pattern. Secondly, We create the database by pushing create database button. Before, pushing the button, we decided to unchecked show trajectory radio button because it decrease the performance when creating line marker in map visualization. In this function, it also search the number of category that exist in the database. From 48.564 of distinct places, we get 15 categories. Fig. 10 shows the original number of place in the database.

After creating database, we run prefix span algorithm for mining coarse pattern. In this process, places that occurs in the coarse pattern is 23264. It means that the process reduce places up to half of number of place occurred in the original database. The running time of mining coarse pattern is 3.685 seconds.

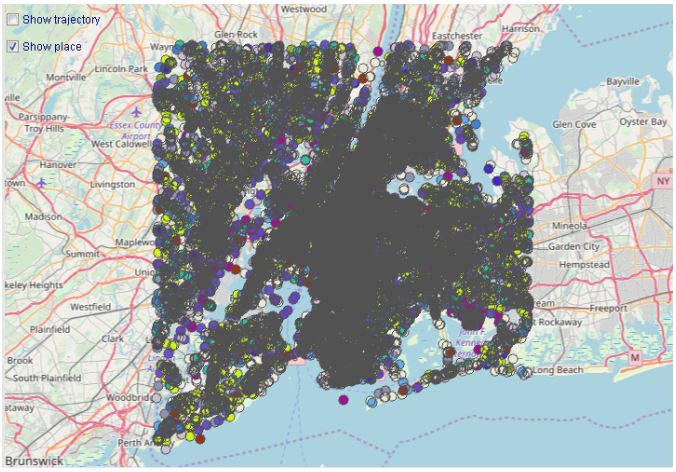


Fig. 10: Visualization of place found in from database

Therefore, PrefixSpan algorithm is fast enough to finding a pattern in a database. Coarse pattern found from the database is 127 which consist of pattern length of 2 (109 patterns) and pattern length of 3 (18 patterns). The highest support or the total number of trajectory that using this pattern in the length of 2 is shop $\rightarrow$ food which contains 2905 of trajectories. For the pattern length of 3, the highest support is food $\rightarrow$ shop $\rightarrow$ food which used by 299 of trajectories. From this result we can assume that a lot of people like to go the food area then go shopping then go to the another food area. Fig. 11 shows place reduction after mining coarse pattern.

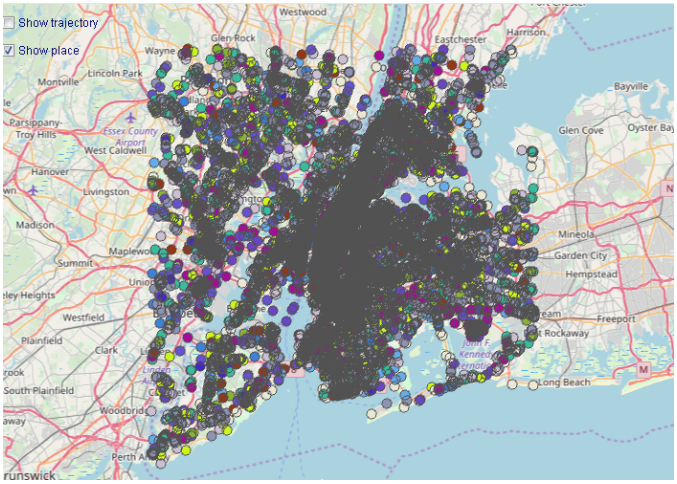


Fig. 11: Visualization of place found after mining coarse pattern

The next step is refine coarse pattern to get fine grained pattern from the database. After we push mean shift button in the left side, the system will begin to refine coarse pattern. Fine grained pattern found in the database using the inputted parameter are 57 patterns. Using the inputted parameters, we cannot find fine-grained pattern that have length 3 occurred in the coarse pattern. It means that our coarse pattern that have length 3 is not satisfy the spatial closeness based on the

parameters. Most trajectory follows pattern shop → food that have support 1063. Fig 12 shows that places are clustered in a dense location only.

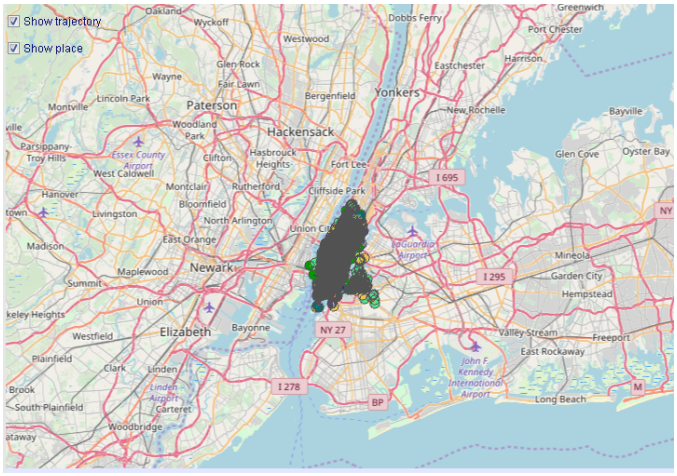


Fig. 12: Visualization of place found after mining coarse pattern

IV. Conclusions

After doing the implementation of Splitter: Mining Fine-Grained Pattern, it shows that it capable to find pattern that are frequent, in limited time movement, and have spatial closeness. In this paper, it also shown that the running time of Splitter to find fine-grained pattern only take total running time below 20 seconds from database that consist of 14909 trajectory and 48564 of places. In the first step, mining coarse pattern algorithm is used to searching pattern that satisfy support threshold σ and time constraint Δt. In the next step, coarse pattern will be used as the input in refining pattern that satisfy spatial variance.

Acknowledgment

This paper is submitted to fulfill term project task from Stream Database class (Fall 2018), Pusan National University.

References

[1] C. Zhang, J. Han, L. Shou, J. Lu, and T. La Porta, "Splitter: Mining fine-grained sequential patterns in semantic trajectories," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 769–780, 2014.