



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

Algorithms for Spatial Colocation Pattern Mining

by
Srikanth Baride
(PhD15014)

Under the Supervision of
Prof. Vikram Goyal

Computer Science And Engineering
Indraprastha Institute of Information Technology Delhi
New Delhi - 110020
December, 2023



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

Algorithms for Spatial Colocation Pattern Mining

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

by

SRIKANTH BARIDE

`srikanthb@iiitd.ac.in`

The Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi (IIIT-Delhi)
New Delhi, India - 110020

Advisor:

Prof. Vikram Goyal

December 7, 2023

Certificate

This is to certify that the dissertation titled **Algorithms for Spatial Colocation Pattern Mining** being submitted by **Srikanth Baride** to the Indraprastha Institute of Information Technology-Delhi, for the award of the degree of **Doctor of Philosophy**, is an original research work carried out by him under my supervision. In my opinion, the dissertation has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this dissertation have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

December, 2023



Prof. Vikram Goyal

Faculty,

Department of Computer Science and Engineering,
Indraprastha Institute of Information Technology Delhi,
Okhla Industrial Estate, Phase III, New Delhi, Delhi 110020.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: B. Srikanth DATE: 8/12/2023

Dedication

This dissertation is dedicated to my parents for believing in me.

Acknowledgement

I would like to express my sincere gratitude to the numerous individuals who have contributed to the completion of this dissertation. Without their help and support, this work would not have been possible.

First and foremost, I extend my heartfelt appreciation to my advisor, Prof. Vikram Goyal, for his unwavering support throughout my PhD journey. Despite his busy schedule, he consistently made time for detailed technical discussions and provided invaluable feedback on my papers. His guidance encompassed both theoretical and practical aspects, and I thoroughly enjoyed our lengthy technical conversations. I am truly grateful for his continuous assistance, feedback, and encouragement, which played a significant role in enabling me to complete this dissertation.

I would also like to express my gratitude to my senior colleague, Dr. Anuj Saxena, for his invaluable time and constant support. His enthusiasm, wisdom, knowledge, and unwavering commitment to the highest standards served as a great source of inspiration and motivation for me. His tailor-made guidance and limitless patience provided me with hope during the challenging times of this journey. I sincerely acknowledge and thank him for his inspirational mentorship, and I cherish the enriching experience of working with him. It is his motivational attitude that paved the way for the successful completion of my dissertation.

Additionally, I would like to acknowledge the Visvesaraya Ph.D. Scheme for Electronics and IT for providing the funding necessary for my research.

I am truly blessed to have the unwavering support and prayers of my parents. I would like to offer a special thanks to my brother for his relentless encouragement and support throughout this journey.

Most importantly, I would like to express my deepest gratitude to my wife Sneha and her parents for their unwavering love, sacrifices, prayers, and constant encouragement. Without their support, this dissertation would not have been possible.

Finally, I am immensely grateful to my little baby boy, Bhavyansh Baride, for his boundless joy. His presence was my motivation and a constant reminder of what truly matters in life.

I would also like to acknowledge and give thanks to God Almighty for the abundant blessings bestowed upon me throughout this journey.


Srikanth Baride

Abstract

Spatial data mining is a specialized field that focuses on extracting meaningful insights and patterns from geographical or spatial data. One particular area of interest in spatial data mining is colocation pattern mining. Colocation patterns refer to objects or entities that tend to occur frequently in close spatial proximity to each other. These patterns can provide valuable insights into spatial relationships and dependencies.

Traditional colocation mining algorithms typically operate on static data and require a predefined single distance threshold to determine spatial proximity. However, deciding on a suitable threshold can be challenging and may not capture the full range of interesting patterns. Moreover, processing the graph representation of spatial data and handling dynamic or evolving datasets present additional challenges in colocation pattern mining.

To address these challenges, our work introduces several novel approaches. Firstly, we propose a new colocation query called Range colocation mining. This query enables the computation of colocation patterns over a range of distances, rather than relying on a single threshold value. This provides greater flexibility to analysts when the determination of a specific distance threshold is difficult or uncertain. Unlike classical algorithms that compute patterns separately for each distance threshold, our method efficiently computes patterns in a single scan over the spatial data, ensuring scalability.

In addition, we extend the traditional notion of colocation patterns beyond cliques to any subgraph representation. This notion allows for a broader exploration of patterns and considers the edges' labels and the degree of affinity between objects. We analyze the complexity of mining subgraph colocation patterns and propose a novel query for high-utility subgraph (colocation) pattern mining. The problem turns out to be more complex than the classical colocation pattern mining. Leveraging the power of Apache Spark, our solution employs a set of heuristics to traverse the pattern space efficiently, utilizing an anti-monotonic relationship over utility values. Our proposed approach is scalable and aids in discovering interesting subgraph patterns prevalent across a set of disjoint regions.

Furthermore, we investigate the dynamic nature of spatial datasets and their impact on computational challenges while mining colocation patterns. We introduce the concept of dynamic colocation pattern mining, which focuses on mining patterns in an incremental manner. Instead of recomputing patterns from scratch for each window, our approach utilizes patterns computed in previous windows to compute patterns in future time windows efficiently. This incremental approach significantly reduces computation time and resources while capturing the temporal evolution of colocation patterns.

Overall, our work contributes to the advancement of colocation pattern mining in spatial data mining. The introduction of Range colocation mining, high-utility sub-graph pattern mining, and dynamic colocation pattern mining expand the scope and capabilities of colocation pattern analysis. These approaches offer greater flexibility, scalability, and efficiency in discovering spatial relationships and dependencies within datasets, enhancing our understanding of spatial phenomena.

Table of Contents

	Page
1 Introduction	1
1.1 Colocation Pattern Mining	2
1.2 Colocation Pattern Mining Applications	3
1.3 A General Approach for Colocation Pattern Mining	3
1.4 Fundamentals of Mining Co-Location Patterns	5
1.4.1 Spatial Neighbor Relationships	6
1.4.2 Clique Instance	7
1.4.3 Participation Ratio	7
1.4.4 Participation Index	8
1.4.5 Fraction Score	8
1.5 Join-less co-location mining algorithm	10
1.6 Key Challenges	12
1.7 Thesis Contributions	13
1.7.1 Mining Colocation Patterns over a Range Query	13
1.7.2 Colocation Subgraph Pattern Mining	14
1.7.3 Mining Co-location Patterns on Dynamic Data	15
1.8 Outline of the Thesis	16
2 Literature Review	17
2.1 Background	17
2.2 Computational techniques	18
2.3 Prevalence thresholds	19
2.4 Support measures	20
2.5 Distance Measures	22
2.6 Fuzzy proximity-based techniques	22
2.7 Incremental Mining	24

2.8	Dynamic spatial co-location patterns	25
2.9	Exploration of Spatio-Temporal Colocation Patterns	27
2.10	Interactive and Visual Analytics	28
2.11	Summary	29
3	Mining Colocation Patterns for a Range Query	31
3.1	Introduction	31
3.2	Preliminaries, Problem Definition, and Mathematical formulation	34
3.3	Related Work	38
3.4	Methodologies for Range Colocation Mining	39
3.4.1	Naïve Approach	39
3.4.2	<i>RangeInc – Mining</i>	41
3.4.3	<i>Range – CoMine</i>	42
3.5	Algorithm for <i>Range – CoMine</i>	48
3.6	Experimental Analysis	53
3.6.1	Experimental Settings	53
3.6.2	Effect of Threshold Settings	54
3.6.3	Effect on number of candidates	57
3.6.4	Scalability Experiments	58
3.6.5	Explainability of real data set output:	60
3.7	Conclusion	60
4	Colocation Subgraph Pattern Mining	61
4.1	Introduction	61
4.2	Preliminaries, Problem Definition, and Mathematical formulation	63
4.3	Related Work	66
4.4	Proposed Methods	67
4.4.1	Baseline Algorithm	68
4.4.2	Distributed WSM Algorithm	70
4.5	Optimizations	72
4.5.1	Use of bloom filter for Pruning of Non-candidates	72
4.5.2	Schimmy Approach	75
4.5.3	Lightweight Object Approach	75

4.6	Experimental Study	76
4.6.1	Effect of Utility Thresholds	77
4.6.2	Effect of Cluster Size	78
4.6.3	Effect of Partition Size	80
4.6.4	Effect of Optimization Strategies in FSM	80
4.7	Conclusion	81
5	Mining Co-location Patterns on Dynamic Data	82
5.1	Introduction	82
5.2	Related work	84
5.2.1	Classical Colocation Mining	85
5.2.2	Incremental Colocation Mining	85
5.2.3	Dynamic Co-location Mining	86
5.3	Preliminaries and Problem Formalization	86
5.3.1	Dynamic data update framework	87
5.3.2	Overview of Fraction-Score	88
5.3.3	Formal Definition of Fraction-Score	89
5.3.4	Problem Formalization	92
5.4	Framework for Updating Fraction score	93
5.4.1	Addition of an object	97
5.4.2	Deletion of object	99
5.5	Algorithms for Mining Co-locations on Dynamic Data	100
5.5.1	Fraction_Update_List	101
5.5.2	Update_Neigh_Deletion	102
5.5.3	Update_Neigh_Addition	103
5.5.4	Update Fraction Score	104
5.5.5	Implementations	106
5.6	Experiments and Results	106
5.6.1	Experimental Setup	106
5.6.2	Influence of the distance threshold on performance	107
5.6.3	Influence of the window size on performance	108
5.6.4	Performance with window change rate	109

5.6.5	Influence of data change rate on performance	111
5.6.6	Performance with Minimum Support Threshold	113
5.6.7	Performance with Damped Window	113
5.6.8	Effect of $m_{overlap}$ on synthetic datasets	114
5.6.9	Effect of m_{clump} on synthetic Datasets	114
5.6.10	Effect of λ_2 on synthetic Datasets	115
5.7	Performance Evaluation and Analysis	116
5.7.1	Dynamic Mining (DM):	116
5.7.2	Dynamic Mining with NBD-list (DM NBD-list):	116
5.7.3	Dynamic Mining with NBD-Hash map (DM NBD-HM):	116
5.7.4	Overall Considerations:	117
5.8	Conclusion	117
6	Conclusion	119

List of Tables

Table	Page
1.1 Feature Types and Instances	5
1.2 Example patterns and their Participation Indexes	6
1.3 Star neighborhood	11
2.1 Finding Efficient Computational Techniques for Mining Colocation Pat- terns	19
2.2 Support Measures for Mining Co-location Patterns	21
2.3 Colocation Mining Literature based on Distance	23
2.4 Literature on Fuzzy-based Colocation Mining	24
2.5 Literature on Incremental Colocation Mining	26
2.6 Literature on Mining Co-locations with Dynamic Constraints	27
3.1 Star Neighborhood List for $d = 8$	44
3.2 Clique Instances with Diameters for $d = 8$ and $min_prev = 7$	44
3.3 Map of objects to distance (CAND) for colocation $\{A, B\}$	47
3.4 Object instance union (Updated Mapping)	47
3.5 Feature weight and PI for candidate distances	48
3.6 <i>ColList</i> : Critical distance to list of Colocations	48
3.7 Relevant information about Real Data Sets	54
3.8 Synthetic data parameters and their values in experiments.	55
3.9 Relevant information Synthetic Data Sets	55
3.10 Parameter Settings for experiments	55
4.1 Statistics of datasets	76
4.2 Parameters for experimental-study	77
4.3 Effect of Bloom filter based pruning	79
4.4 Effect of partition size on run time and % pruning	80

5.1	Neighborhood, Neighborhood Count & fraction scores for W^2	90
5.2	Snapshot After Addition	105
5.3	Experimental parameters and their values in experiments.	107

List of Figures

Figure		Page
1.1	A general approach for colocation mining	4
1.2	A real-time spatial dataset	5
1.3	Example neighborhood graph of a spatial dataset at distance $d=8$	6
1.4	Star neighborhood partition method	10
2.1	A survey of research space	17
3.1	Lattice Visualisation of Colocations Over Range Query $D = [3, 8]$	33
3.2	Neighbourhood Relation R_d for pattern $\{A, B\}$ for distances $d = 7, 5, 4$	35
3.3	Participation Index vs. Distance Threshold for colocation $\{A, B\}$	37
3.4	Graph representation of Neighborhoods Relation at distance $d = 8$	43
3.5	Procedure for finding critical distances	46
3.6	Performance with <i>min_prev</i>	56
3.7	Performance with Range	56
3.8	Effect on the number of candidates in <i>Synthetic_Data_1</i>	57
3.9	Scalability with number of objects	58
3.10	Scalability with number of features	59
3.11	Effect of density	59
4.1	An example showing various weights for pattern P	65
4.2	Addition of size-2 frequent patterns to the bloom filter	73
4.3	Illustration of membership test	74
4.4	Effect of utility threshold on run time	78
4.5	Effect of Cluster Size on Runtime	79
4.6	FSM: Effect of support threshold on run time	80
5.1	Framework for Mining Co-locations from Dynamic Data	84

5.2	Snapshot of dynamic spatial data	87
5.3	Temporal moment from W^2 to W^3	88
5.4	Fraction score Computation for W^2 (at time $t = 2$)	90
5.5	Effect of addition	98
5.6	Effect of deletion	98
5.7	Deletion of Objects	101
5.8	Addition of Objects	102
5.9	Performance with Distance Threshold for Dataset-1	108
5.10	Performance with Distance Threshold for Dataset-2	109
5.11	Performance with window Size for Dataset 1	109
5.12	Performance with window Size for Dataset 2	110
5.13	Performance with window change rate for Dataset-1	110
5.14	Performance with window change rate for Dataset-2	111
5.15	Performance with λ for Dataset-1	111
5.16	Performance with λ for Dataset-2	112
5.17	Performance with Minimum Support Threshold for Dataset -1	112
5.18	Performance with Minimum Support Threshold for Dataset-2	112
5.19	Performance with No. of objects added for Dataset -1	113
5.20	Performance with No. of objects added for Dataset-2	114
5.21	Effect of $m_{overlap}$ on synthetic Datasets	114
5.22	Effect of m_{clump} on synthetic Datasets	115
5.23	Effect of λ_2 on synthetic Datasets	115

List of Symbols

E	Set of edges
V	Set of vertices
Δ_{label}	Fraction score
λ	Data change rate
λ_2	The parameter used in Poisson distribution to construct the instances for each maximal co-location
D	Distance range
d	Distance
D_{pair}	The set of distances between any two objects in R_d
f_i	Feature
G	Graph
I^p	Clique instance of the pattern p
$I_{R_d}^p$	Collection of all the clique instances of a pattern p for the neighborhood relation R_d
m_{clump}	Number of feature instances for each co-location in it's neighborhood
$m_{overlap}$	Number of maximal co-locations generated by appending more features into co-locations
O^a	Added objects
O^d	Deleted objects
p	Candidate Colocation Pattern
R_d	Neighborhood relation for a distance d
W^t	Window at time t
F	Feature-set
O	The collection of all objects in the dataset

o_i	Object of a unique feature type f_i
$ t $	Window change rate
$ W $	Window size
Fraction value	Fraction value
S	Minimum support thresholds
t	time

Introduction

The rapid expansion in the size of spatial databases has emphasized the importance of utilizing spatial data mining techniques to extract interesting and useful spatial patterns from these large datasets [29]. Spatial data has a geographic or spatial component, such as maps, satellite imagery, and GPS coordinates. A lot of location-specific significant and valuable information is hidden in such data, like the co-existence of groups of objects, clusters with similar characteristics, spatial outlier objects, etc. Discovering patterns from the extensive spatial datasets is crucial in numerous application domains, including urban planning, ecology, environmental management, public safety, transportation, public health, business, travel, and tourism [36] [17] [11]. For instance, spatial data mining is instrumental in detecting regions with a high prevalence of disease incidents in epidemiology to contain outbreaks.

Due to the massive and rapid flow of spatial data, computational techniques are indispensable for uncovering spatial patterns that exceed the analytical capacity of human experts. To detect spatial patterns, there are three crucial stages:

1) Data preparation: This stage involves correcting noise, errors, and missing information in the spatial data. Additionally, spatial distribution analysis is conducted to gain insights into the underlying patterns.

2) Applying spatial data mining algorithms: In this stage, appropriate spatial data mining algorithms are applied to the pre-processed data to generate pattern output. These algorithms utilize various techniques such as clustering, classification, association rule mining, and outlier detection.

3) Refining output patterns: The generated pattern output is further refined through post-processing techniques. This allows domain experts to examine the output, validate the patterns, and gain insights into the underlying spatial phenomena.

The different families of spatial patterns include hotspot detection, colocation detection, spatial prediction, and spatial outlier detection. Colocation detection methods focus on identifying objects that are frequently located near each other in space, indicating spatial associations. On the other hand, spatial outlier detection methods identify data points that deviate significantly from neighboring points, representing unique or unusual spatial phenomena.

In this work, we propose efficient methodologies to address challenges associated with colocation pattern mining techniques. Colocation pattern mining aims to uncover sets of features that frequently co-occur in specific spatial areas. By analyzing the spatial proximity of these features, valuable insights can be gained across various domains. In the next section, we will explain the idea of colocation pattern mining and provide an example to illustrate its significance in uncovering meaningful patterns within spatial data.

1.1 Colocation Pattern Mining

Spatial colocation patterns are sets of features whose instances are frequently located near each other in space [54]. To illustrate this concept, let's consider a dataset of crime incidents in a city, where each incident is recorded as a point on a map. The objective is to identify sets of crime types that co-occur frequently in specific areas of the city. To achieve this, a colocation pattern mining algorithm can be employed, which identifies sets of crime types that appear together more often than expected by chance. The algorithm searches for colocations of crime incidents that are in closer proximity to each other than would be randomly anticipated.

For instance, the algorithm might uncover a colocation pattern consisting of theft, burglary, and vandalism in a particular neighborhood. This pattern suggests that these specific types of crimes tend to occur together more frequently in that neighborhood compared to other parts of the city. Armed with this information, law enforcement officials can strategically allocate more resources to that neighborhood, aiming to prevent such crimes. Furthermore, they can investigate the underlying factors contributing to the co-occurrence of these crimes and take necessary steps to address any issues, such as improving lighting or implementing enhanced security measures. This example highlights how colocation pattern mining serves as a valuable tool for discovering meaningful patterns within spatial data.

Colocation pattern mining has extensive applications in various domains. In urban planning and resource allocation, it can optimize the placement of facilities and amenities based on their co-occurrence patterns. Retail and marketing benefit from colocation pattern mining by identifying associations between different businesses, allowing for collaboration opportunities and optimal store locations. Environmental studies leverage colocation pattern mining to understand ecological interactions and guide conservation efforts.

In the next section, we will look into the detailed applications of colocation pattern mining, exploring its potential in urban planning, retail, marketing, and environmental studies. By understanding these applications, we can further appreciate the significance and utility of colocation pattern mining in real-world scenarios.

1.2 Colocation Pattern Mining Applications

Colocation pattern mining finds extensive applications in various domains, including urban planning, ecology, environmental management, public safety, transportation, public health, and business [11, 17, 36]. Below are some notable examples of colocation pattern mining applications:

Urban Planning: Colocation pattern mining plays a vital role in urban planning, particularly in the analysis of land use patterns [11]. By examining the spatial distribution of different land uses, such as residential, commercial, and industrial areas, colocation pattern mining can identify areas where certain land uses are more concentrated. This information empowers urban planners to make informed decisions regarding zoning regulations and development strategies, leading to more efficient land use.

Public Safety: The analysis of crime patterns is an important application of colocation pattern mining [17]. By studying the spatial locations of criminal incidents, colocation pattern mining enables the identification of high-crime areas, allowing law enforcement agencies to implement targeted interventions and preventive measures.

Public Health: Colocation pattern mining finds utility in public health by identifying spatial relationships between disease cases and contributing locations [36]. By analyzing the locations visited by infected individuals, public health officials can pinpoint high-risk areas and implement targeted measures to mitigate disease transmission in those locations.

Business: Colocation pattern mining is also applied in the business domain to uncover spatial relationships between different types of establishments, such as retail stores, restaurants, and hotels [77]. By examining the co-locations of these establishments, businesses can gain valuable insights into consumer behavior, optimize their operations, and develop effective marketing strategies.

In summary, colocation pattern mining serves as a powerful tool for discovering valuable patterns within spatial data, with diverse applications across various fields and industries. The ability to identify spatial relationships and co-occurrences enables professionals in urban planning, public safety, public health, and business to make data-driven decisions and improve their strategies. In the next section, we will present a general approach for colocation pattern mining, providing insights into the techniques and methods used in the process.

1.3 A General Approach for Colocation Pattern Mining

A general approach for colocation pattern mining is presented in Figure 1.1, which outlines the steps involved in mining colocation patterns given a spatial dataset, neighborhood relation, and minimum prevalence criteria.

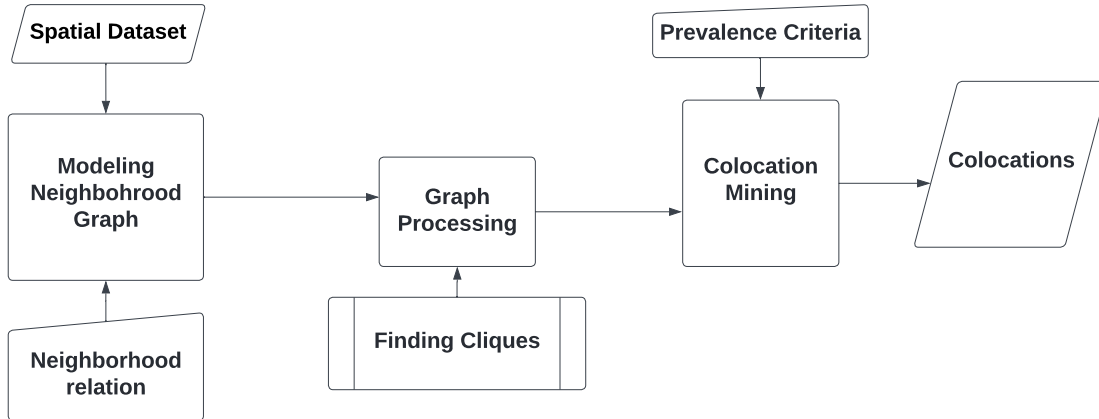


Figure 1.1: A general approach for colocation mining

Modeling a Neighborhood Graph: One common approach to mine colocation patterns from a spatial dataset is to model a neighborhood graph that captures the spatial relations between the objects in the dataset. In this approach, each object is represented as a node in the graph, and the edges between nodes denote their spatial relationships.

Graph Processing: The neighborhood graph is then processed to enumerate all possible cliques, which are fully connected subgraphs within the neighborhood graph. Identifying cliques is essential for discovering potential colocation patterns in the data. However, it is crucial to consider the computational complexity of this task, as the size of the neighborhood graph can grow rapidly with an increasing number of spatial units. Efficient algorithms and appropriate computational resources should be employed for processing large neighborhood graphs.

Colocation Mining: Once the cliques have been identified, they are evaluated to discover interesting colocation patterns. The prevalence of each clique is calculated and compared against a predefined prevalence threshold. Cliques that meet the prevalence criteria are selected as colocation patterns, indicating sets of objects that exhibit significant spatial co-occurrence.

In summary, the general approach for colocation mining involves modeling a neighborhood graph, processing the graph to identify cliques, and subsequently evaluating the cliques to discover colocation patterns that meet the predefined prevalence criteria. Two popular prevalence measures used in the literature are the participation index (explained in Section 1.4.4) and the fraction score (explained in Section 1.4.5). These measures help determine the significance of colocation patterns based on their frequency and spatial proximity.

For graph processing, the Join-less colocation mining algorithm (explained in Section 1.5) is commonly used in state-of-the-art approaches. This algorithm efficiently identifies colocation patterns by avoiding unnecessary join operations, reducing compu-

tational overhead, and improving performance. By leveraging the properties of colocation patterns and optimizing the graph processing techniques, Join-less algorithms provide effective solutions for colocation mining tasks.

In the section 1.4, we will delve deeper into the fundamentals of mining co-location patterns. We will explore the participation index and the fraction score as prevalent measures for evaluating colocation patterns. Additionally, we will discuss the Join-less co-location mining algorithm, which is widely used in state-of-the-art approaches for efficient graph processing. By understanding these foundational concepts, we will lay the groundwork for further exploration and propose novel methodologies to enhance the efficiency and effectiveness of co-location pattern mining.

1.4 Fundamentals of Mining Co-Location Patterns



Figure 1.2: A real-time spatial dataset

Point of Interest	Feature Types	Spatial Instances
Apartment	A	A.1, A.2, A.3
Billiard	B	B.1, B.2, B.3
Community centre	C	C.1, C.2, C.3
Dancing hall	D	D.1, D.2, D.3

Table 1.1: Feature Types and Instances

In the context of mining co-location patterns, we consider a spatial dataset consisting of objects with unique features from the feature-set $F = \{f_1, f_2, \dots, f_n\}$. The collection of all objects in the dataset is denoted by O , which can be expressed as $O = O_1 \cup \dots \cup O_n$, where each O_i (with $1 \leq i \leq n$) represents objects of a unique feature type f_i . Each object $o \in O$ is represented as a tuple (id, f, l) , where id is the instance ID, $f \in F$ denotes its feature type, and l indicates its location (x, y) .

Consider a real-time spatial data shown in Figure 1.2 where apartment, billiard, community centre and dancing hall are different point of interest features. For example, in Figure 1.3, the instance $A.1$ with ID 1 belongs to feature type A. The spatial dataset illustrated in Figure 1.3 includes instances from four feature types: A, B, C, and D. Table 1.1 summarizes the number of instances for each feature type. Table 1.1 provides an overview of the feature types and spatial instances present in the dataset.

1.4.1 Spatial Neighbor Relationships

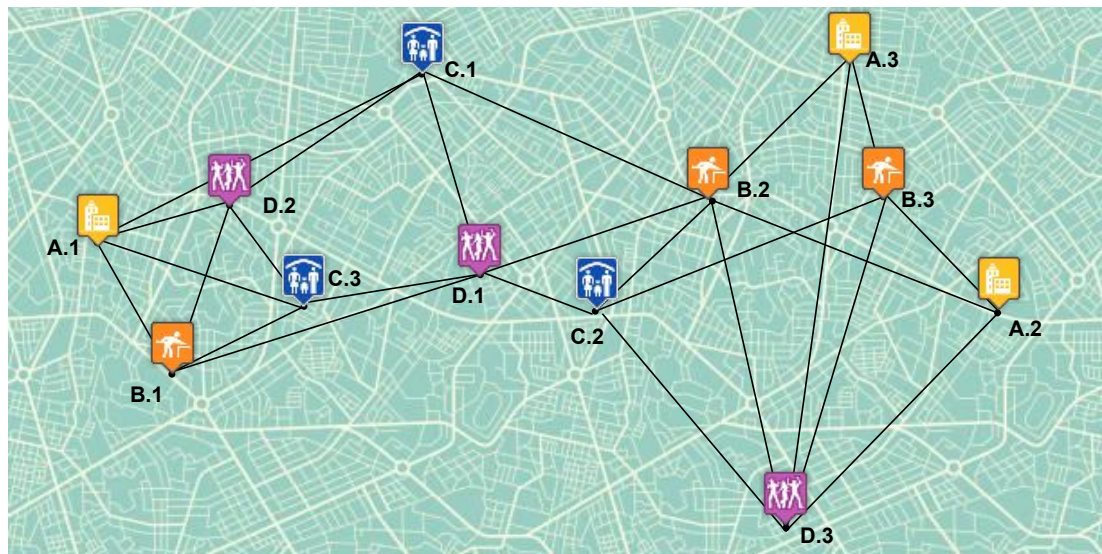


Figure 1.3: Example neighborhood graph of a spatial dataset at distance $d=8$

AB :3/3	AD: 2/3	BD: 3/3	ABD: 2/3
A.2, B.2	A.3, D.3	B.1, D.1	A.3, B.2, D.3
A.3, B.2	A.2, D.3	B.2, D.1	A.3, B.3, D.3
A.2, B.3	A.1, D.2	B.1, D.2	A.2, B.2, D.3
A.1, B.1		B.3, D.3	A.2, B.3, D.3
A.3, B.3		B.2, D.3	A.1, B.1, D.2

Table 1.2: Example patterns and their Participation Indexes

A spatial neighbor relationships can be established between two spatial instances based on their Euclidean distance, which is measured using a threshold value d . The neighbor relationship between $A.1$ and $B.1$ in Figure 1.3 can be denoted as $R(A.1, B.1) \leftrightarrow (\text{distance}(A.1, B.1) \leq d)$, where d represents the minimum threshold value. Solid lines in the figure connect instances that satisfy the neighbor relationship condition. For example, $A.1$ and $B.2$ meet this condition since they are within a distance of d from each other. Spatial instances are grouped into clique instances as follows:

1.4.2 Clique Instance

A clique is a group of objects in which each pair of objects is co-located with each other. For example, in Figure 1.3, $A.3$, $B.3$, and $D.3$ form a clique.

Definition 1.1 (Clique Instance). Let $p = \{f_1, \dots, f_k\} \subseteq F$ be a pattern, and let $\{o_1, \dots, o_k\} \subseteq O$ be its object instances, where $o_i.f = f_i$ for all i . If the distance between any pair of objects in $\{o_1, \dots, o_k\}$ satisfies the neighborhood relationship R with a distance threshold d , then $CI = \{o_1, \dots, o_k\}$ is considered a clique instance of the pattern p . The collection of all clique instances of pattern p for the neighborhood relation R , denoted by I^p , is defined as follows:

$$I^p = \{CI = \{o_1, \dots, o_k\} \mid \forall i, o_i.f = f_i \wedge CI \text{ is a clique in } R\}$$

Table 1.2 displays the clique instances for patterns AB, AC, AD, BD, etc., considering a distance threshold of $d = 8$. For each of these clique instances participation index is calculated. To explain the participation index we will first introduce participation ratio.

1.4.3 Participation Ratio

The participation ratio [54] of a feature type f_i in a colocation pattern p for a given distance threshold d is defined as the ratio of the number of unique instances of f_i participating in any clique instances of p to the total number of instances of f_i in O .

Definition 1.2 (Participation Ratio). Let $p = \{f_1, \dots, f_k\}$ be a colocation pattern with clique instances I^p . The participation ratio of feature $f_i \in p$, denoted by $Pr(f_i, p, I^p)$, is calculated as:

$$Pr(f_i, p, I^p) = \frac{|\{o_i \in O_i \mid \{o_1, \dots, o_k\} \in I^p\}|}{|O_i|} \quad (1.1)$$

where O_i represents the set of instances of feature type f_i in the spatial dataset O .

For example, in Table 1.2 for Figure 1.3, the participation ratio of feature type B in the pattern $p = \{A, B\}$ is $3/3$ for $d = 8$.

1.4.4 Participation Index

In statistics, the cross-K function is an extension of Ripley’s K function that is commonly used to identify colocation patterns between multiple spatial features. It is a spatial statistical method specifically designed for analyzing point events. The cross-K function, denoted as $K_{ij}(h)$, is defined for binary spatial features and can be expressed as follows: $K_{ij}(h) = \lambda_j^{-1} \mathbf{E}[\text{number of } j \text{ instances within distance } h \text{ of a randomly-chosen type } i \text{ instance}]$ where λ_j is the density (number per unit area) of type j instances. Higher cross-K values indicate a higher likelihood of the features being located in close proximity to each other, while lower values suggest that they tend to be located further apart. The participation index estimates the cross-K value and is efficiently computable.

The participation index [54] is the minimum among all the participation ratios of distinct features f_i in a colocation pattern p .

Definition 1.3 (Participation Index). The participation index of a colocation pattern $p = \{f_1, \dots, f_k\}$ with clique instances I^p under the spatial neighborhood relation R , denoted by $Pi(p, I^p)$, is defined as:

$$Pi(p, I^p) = \min_{f_i \in p} Pr(f_i, p, I^p) \quad (1.2)$$

For example, in Table 1.2 for Figure 1.3, the participation index of the pattern $p = \{A, B\}$ is 1 for $d = 8$ since all instances of feature types A and B participate in the clique instances.

The participation index is an upper bound of the cross-K function and is widely used as a measure of colocation due to its computational efficiency. However, it is important to note that the participation index may overcount the number of instances involved in colocation patterns. To address the issue of overcounting, the fraction score measure was introduced in [9].

1.4.5 Fraction Score

Fraction score provides a refined support computation for a pattern p by considering the fraction of shared objects in different groups [9]. Let’s consider a pattern $p = \{A, B\}$ with clique instances $\{A.2, B.2\}$, $\{A.3, B.2\}$, and $\{A.2, B.3\}$ as shown in Table 1.2. To compute the fraction score, we first fix a feature $f \in p$ to group the clique instances of p . The set of all objects o in some clique instance of p that have the fixed feature f is denoted by $obj(f, p)$. For example, if we fix $f = A$, then $obj(A, p)$ includes the objects $\{A.1, A.2, A.3\}$.

Since some objects may be shared among different instances, overcounting can occur in participation-based scores. To address this, a fraction value is assigned to each object based on its participation in the grouping.

Let o be an object in $obj(f, p)$ and o' be an object from an instance of p with a feature different from f . The fraction value $\Delta_{obj}(o, o')$ is defined as the reciprocal of the number of neighbors of o' that have feature f within a distance threshold d :

$$\Delta_{obj}(o, o') = \frac{1}{|\text{Neigh}(o', o.f, d)|} \quad (1.3)$$

By assigning the appropriate fraction values to the objects in $obj(f, p)$, the fraction score measure effectively handles overcounting and provides a more accurate assessment of colocation patterns.

Consider $o = A.3$ and $o' = B.3$. Clearly, $|\text{Neigh}(B.3, A, d)| = 2$. That is, the two objects in the neighborhood of $B.3$, namely $A.2$ and $A.3$, share the object $B.3$. As a result, an equal fraction value is assigned to the two objects $A.2$ and $A.3$ as their participation:

$$\Delta_{obj}(A.2, B.3) = \frac{1}{|\text{Neigh}(B.3, A, d)|} = \frac{1}{2} = \Delta_{obj}(A.3, B.3)$$

The support of p for the feature f (denoted as $sup(p|f)$) is calculated by summing the fraction-scores of objects $o \in obj(f, p)$:

$$sup(p|f) = \sum_{o \in obj(f, p)} \Delta_{pattern}(o, p) \quad (1.4)$$

The overall support of p (denoted as $sup(p)$) is then determined by dividing the minimum support value among all features in p by the maximum number of objects that have a specific feature in the dataset:

$$sup(p) = \frac{\min_{f \in p} sup(p|f)}{\max_{f \in F} |\{o.f = f | o \in O\}|} \quad (1.5)$$

This calculation allows for the determination of the support of a given feature set p based on the fraction-score measure.

Spatial co-location patterns are subsets of features that are frequently located together in geographic space, indicating a potential spatial relationship [20, 75]. In Figure 1.3, the set of features $\{A, B, D\}$ represents a co-location pattern.

Definition 1.4 (Colocation Patterns). For a given $\langle O, F, d, min_prev \rangle$, where O is a spatial object with spatial features from F , d is a distance, and min_prev is a prevalence threshold, a pattern p is a *colocation pattern* c if the participation index (or support) for $p \subseteq F$ crosses the min_prev threshold for the distance d .

For example, in Table 1.2 for Figure 1.3, considering the min_prev threshold as $2/3$, pattern AB qualifies as a colocation pattern, while pattern AC does not meet the criteria to be considered a colocation pattern.

1.5 Join-less co-location mining algorithm

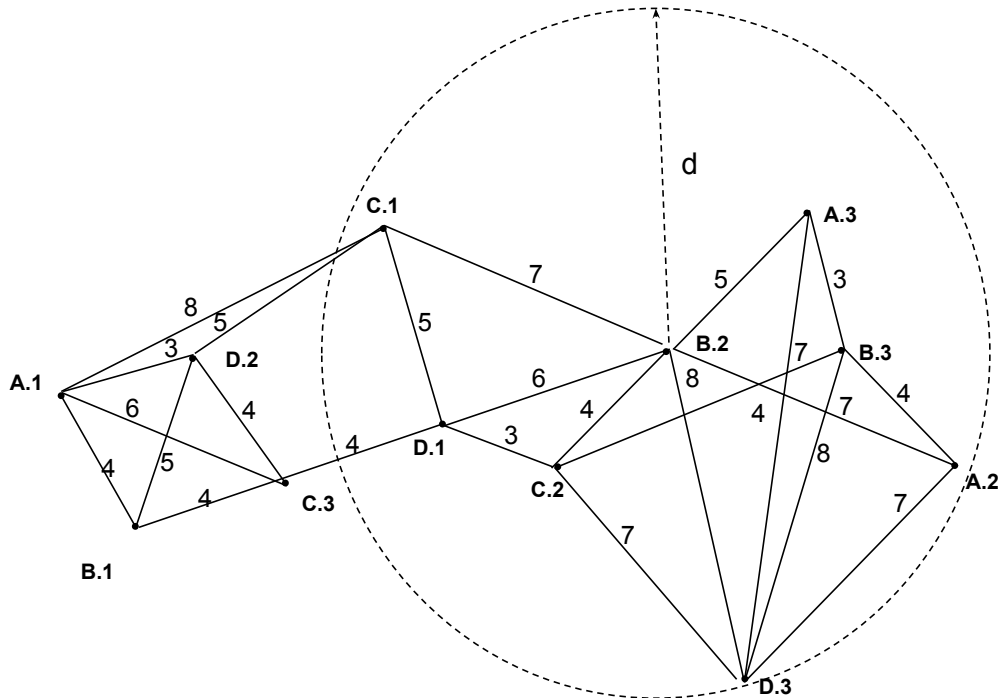


Figure 1.4: Star neighborhood partition method

The join-less algorithm is designed to identify co-location patterns in spatial data without the need for explicit join operations. Instead, it relies on the concept of a star neighborhood, which involves creating a circular area around each spatial instance. Within each star neighborhood, all neighbor relationships among the spatial instances are recorded, allowing for efficient identification of cliques that form co-location patterns. By using the star neighborhood approach, the join-less algorithm can avoid expensive join operations and improve the speed of co-location pattern mining in spatial data sets.

Star neighborhood approach: Figure 1.4 describes the star neighborhood partition method. The dashed circle in the figure represents the star neighborhood of spatial instance $B.2$ (i.e., the star neighborhood with $B.2$ as the center point). Instances within each star neighborhood are listed in Table 1.3. It is important to note that although $A.3$ and $B.3$ satisfy the neighborhood relationship, there is no $A.3$ in the star neighborhood of $C.3$. This is because the star neighborhood is defined as a set of central instances and instances that satisfy the neighborhood relationship with the central instance, where the spatial feature types of the instances listed in Table 1.3 are in an order greater than those of the central instance. For example, $A.1$ belongs to feature type A, which has an order greater than $C.3$ belonging to feature type C.

Centre		Star neighborhood
Feature	Object	
A	A.1	A.1, B.1, C.1, C.3, D.2
	A.2	A.2, B.2, B.3, D.3
	A.3	A.3, B.2, B.3, D.3
B	B.1	B.1, C.3, D.1, D.2
	B.2	B.2, C.1, C.2, D.1, D.3
	B.3	B.3, C.2, D.3
C	C.1	C.1, D.1, D.2
	C.2	C.2, D.1, D.3
	C.3	C.3, D.1, D.2
D	D.1	D.1
	D.2	D.2
	D.3	D.3

Table 1.3: Star neighborhood

The basic steps of joinless algorithm are as follows:

Step 1: Transformation— To transform the spatial dataset into a non-intersecting star neighborhood set, the first step is to find all adjacent object pairs using a geometric method based on the provided neighborhood relationship. Then, for each object, its neighbor objects can be grouped together to form a star neighborhood.

Step 2: Candidate Generation—The process of generating candidate co-location patterns involves initializing all features to determine size 1 prevalent co-location patterns based on the participation index definition. Then, in the process of neighborhood materialization, size k candidate co-location patterns are generated from size $k-1$ prevalent co-location patterns. These patterns are filtered at the feature level. If any subset of the candidate co-location pattern is non-prevalent, the candidate pattern is pruned.

Step 3: Star Instance Filtering—The star instances of the candidate co-locations are filtered from the star neighborhood sets based on their feature types. The star neighborhood corresponding to the feature type of the first feature in each candidate co-location pattern is examined, and the instances of all features in the candidate pattern are collected from that star neighborhood.

Step 4: Prevalent Co-location Selection—Approximately select prevalent co-location patterns by collecting cluster instances for co-locations of size 2 from the star instances of both neighbor features. For co-locations with a size larger than 3, check whether a star instance is a cluster instance. Prior to this step, the candidate co-locations are roughly filtered by computing their participation indexes and discarding those whose value is below the minimum popularity threshold set by the user.

Step 5: Row Instance Filtering—Filtering row instances of co-locations involves

using an instance lookup scheme to identify and filter instances of co-locations from the star instances of candidate co-location patterns.

Step 6: Prevalent Co-location Generation—The generation of prevalent co-locations is achieved by computing the actual participation index based on the instances of co-locations. This involves an optimized filtering process where candidate co-locations are examined for their actual participation index, and only those that meet a predetermined threshold are considered as prevalent co-locations.

1.6 Key Challenges

The research in co-location pattern mining involves tackling several challenges that hinder the flexibility, efficiency, and effectiveness of the existing approaches. In this section, we discuss the key challenges and limitations that this thesis aims to address:

Challenge 1: Determining an Appropriate Distance Threshold — Existing co-location mining approaches rely on a single distance threshold, which can be challenging and time-consuming for domain experts to determine. The selection of this threshold is highly dependent on the data and requires expertise in the specific domain. One way to solve this problem is by incorporating a range query into existing co-location mining approaches, and is non-trivial. Existing methods are designed to work with a fixed distance threshold, and adapting them to handle a range of distances poses several challenges. When using a range query approach, it is crucial to reduce the computation by selecting a few distance values, we call as the critical distances, within the given range. These critical distances represent the points at which the set of co-location patterns changes, and efficiently identifying them is necessary to avoid unnecessary computation. Furthermore, recomputing clique instances for each distance within the range can be computationally expensive and should be minimized. Finding an efficient approach to avoid multiple runs of the co-location mining algorithm is necessary to improve performance.

Challenge 2: Considering the Importance of Nodes and Edges for colocation Pattern Mining — There exists a lot of scientific work for subgraph pattern mining which supports finding more general patterns compared to clique instances in colocation pattern mining. This work may be leveraged to mine patterns on neighbourhood graphs constructed for a set of regions to find common patterns. Traditional subgraph pattern mining approaches, such as Frequent Subgraph Mining (FSM), do not consider the relative importance of instances and their relationship with other instances in a pattern. However, real-life scenarios may have different importance for different instances. Leveraging graph pattern mining and incorporating the importance notion requires the development of new approaches. Furthermore, mining patterns from large graph databases is computationally expensive and presents a challenge in terms of efficiency. Although distributed frameworks like Map-Reduce/Apache Spark can handle the iterative nature of pattern mining tasks, optimizing the process on these platforms is crucial to improve efficiency.

Challenge 3: Handling Dynamic Updates in Co-location Pattern Mining

— Traditional co-location mining techniques are designed for static data and are not efficient in handling dynamic updates. Updating co-location patterns from the previous state while considering the previously computed patterns, old data state, and changes in the data poses computational challenges. When mining co-location patterns from dynamic data, the addition or deletion of objects affects the patterns. Determining how these changes impact the patterns and efficiently updating their relevant scores adds to the complexity of the problem. To handle dynamic data, efficient algorithms and frameworks are required to minimize the need for re-computation and optimize the update process. This involves identifying candidate objects whose scores are likely to change and implementing lazy computation techniques to avoid redundant updates.

By acknowledging and addressing these challenges, the research aims to overcome limitations in co-location mining, provide practical benefits, and advance the field by proposing innovative solutions and algorithms.

1.7 Thesis Contributions

The dissertation explores three tasks: First, mining colocation patterns over a range query. Second, Colocation Subgraph Pattern Mining. Third, mining co-location patterns on dynamic data. A brief introduction and contributions of the problems addressed are as follows:

1.7.1 Mining Colocation Patterns over a Range Query

Existing approaches in co-location mining typically rely on a single distance threshold (d) and a minimum prevalent score (min_prev) to identify co-locations with a PI higher than the specified min_prev . However, determining an appropriate d value can be challenging and time-consuming for domain experts, as it is highly dependent on the data. To address this challenge and provide more flexibility to users, we propose a range query approach for co-location pattern mining. Instead of a single distance threshold, users can input a distance interval ($D = [d_1, d_2]$) that represents their desired range of spatial proximity. This range query allows users to capture patterns that occur within a specific distance range, enabling them to explore different levels of proximity.

There are several reasons why a range query can be beneficial. Firstly, it alleviates the difficulty of choosing a single distance threshold by allowing users to specify a range that is more intuitive and easier to determine. Secondly, the distribution of co-location patterns within the range interval provides a more expressive representation of the result space, allowing users to observe the prevalence of patterns at different distances. Finally, the additional distance information associated with each co-location pattern assists in post-mining decision-making, as users can analyze how patterns evolve and appear/disappear with changes in the distance value.

However, incorporating a range query into existing co-location mining approaches is non-trivial. Existing methods are designed to work with a fixed distance threshold, and extending them to handle a range of distances requires addressing several challenges. One of the main challenges is determining the critical distances within the given range, i.e., the distances at which the set of co-location patterns changes. These critical distances play a crucial role in determining the result set and need to be efficiently identified to avoid unnecessary computation. Additionally, recomputing clique instances for each distance within the range is computationally expensive and should be avoided.

In our work, we propose a novel solution to address these challenges and enable range colocation pattern mining. We introduce a range colocation query type, develop an efficient algorithm called *Range – CoMine*, and leverage structural properties of colocation patterns to minimize computational costs. The algorithm operates in a single pass and utilizes a space-efficient data structure to avoid multiple runs of the co-location mining algorithm. Extensive experiments on real and synthetic datasets demonstrate the effectiveness and efficiency of our proposed approach, outperforming adapted versions of existing algorithms in terms of time and space requirements.

1.7.2 Colocation Subgraph Pattern Mining

The approach to colocation pattern mining can be viewed as a subgraph pattern mining task wherein we relax the constraint of clique formation over the instances. It involves representing objects as nodes in a graph and the relationships between objects as edges with distance features. The subgraph (colocation) patterns can be mined using standard approaches like Frequent Subgraph mining (FSM) over a graph database, wherein each graph in the database is a neighbourhood graph of a specific region. The algorithm would be able to find out common spatial patterns over a set of regions.

The FSM is a well-studied problem that aims to discover subgraph patterns that occur frequently in a given graph database. However, FSM does not consider the relative importance of nodes and edges in a pattern. To address this, we introduce high-utility subgraph pattern mining (WSM), which considers the importance of participating nodes and edges to compute a pattern’s relevance (utility).

Both FSM and WSM are computationally expensive, and mining patterns from a large graph database can be challenging. Distributed frameworks like Map-Reduce/Apache Spark are suitable for handling the iterative nature of pattern mining tasks. However, straightforward extensions of Map-Reduce solutions for FSM to Spark do not offer significant advantages. Therefore, optimization strategies are needed to improve the efficiency of pattern mining on Spark.

The contributions of the research include defining the high-utility subgraph pattern mining problem, developing a distributed solution for both WSM and FSM on big-data platforms, designing effective optimization strategies to reduce data communication and

unnecessary computations, and conducting experiments to demonstrate the efficiency of the proposed approach.

Overall, the research aims to enhance the efficiency and effectiveness of colocation pattern mining by considering the importance of nodes and edges and by leveraging distributed computing platforms for large-scale graph databases.

1.7.3 Mining Co-location Patterns on Dynamic Data

The focus of this work is on mining co-location patterns from dynamic data, where the data changes over time with the addition and deletion of objects. Traditional co-location mining techniques are designed for static data and do not handle dynamic updates efficiently. The proposed computational framework aims to update co-location patterns from the previous state by considering the previously computed patterns, the old data state, and the changes that occurred in the data.

The fraction-score measure is used as the support measure for co-location patterns, as it addresses the issue of overcounting and undercounting of support measures in overlapped participating instances. The framework takes into account the topological relationships between objects and the effects of object addition or deletion on the fraction scores of participating objects. Multiple score updates may be required for affected participating objects, and the same objects may participate in multiple feature-sets, adding to the computational challenges.

The temporal-window framework is employed, where the data changes are realized at fixed time intervals. The current co-location patterns are updated from the previous window's patterns by identifying candidate objects whose scores are likely to change and efficiently updating their fraction scores using their previous score and the changes in the dataset. This approach minimizes the need for re-computation and optimizes the update process.

The contributions of the work include the proposed computational framework for online updating of fraction-score-based co-location patterns, an efficient algorithm for updating fraction scores that avoids redundant updates and utilizes lazy computation, and the design of an efficient algorithm named UpFS for mining co-locations in the current window based on the proposed update technique. Experimental results demonstrate the effectiveness and efficiency of the algorithm compared to the baseline approach.

Overall, the research aims to enhance the productivity of analysts, reduce computational overhead, and improve the utilization of computing resources in mining co-location patterns from dynamic data.

1.8 Outline of the Thesis

In Chapter 2, a comprehensive literature review of colocation pattern mining is presented. The review provides the necessary background and understanding to follow the discussions in the subsequent chapters. Various techniques, algorithms, and measures related to colocation pattern mining are explored, setting the foundation for the research presented in the thesis.

Chapter 3 focuses on mining colocation patterns over a range query. A framework for mining range colocation is proposed, and detailed algorithms are presented. The algorithms aim to efficiently identify patterns where objects co-occur within a specified range. The chapter discusses the design, implementation, and evaluation of these algorithms, highlighting their effectiveness and applicability in different domains.

In Chapter 4, the concept of colocation subgraph pattern mining is introduced. The chapter discusses the importance of considering the utility or significance of individual nodes and edges within subgraph patterns. An efficient solution for high-utility subgraph pattern mining using Spark, a popular distributed computing framework, is presented. The chapter presents the algorithm's design and implementation details, along with experimental evaluations demonstrating its effectiveness and computational advantages.

Chapter 5 focuses on mining colocation patterns on dynamic data. A window-based model is introduced to handle the evolving nature of the data. The chapter proposes two operators for efficiently performing the computation of dynamic colocations. The algorithms and techniques presented in this chapter address the challenges associated with updating colocation patterns as the data changes over time. Experimental results and evaluations are provided to validate the efficiency and effectiveness of the proposed approach.

The final chapter concludes the thesis by summarizing the main findings, contributions, and implications of the research presented. The limitations of the proposed methods are discussed, and potential avenues for future research in colocation pattern mining are outlined. This chapter provides closure to the thesis and highlights its significance in advancing the field of colocation pattern mining.

Literature Review

2.1 Background

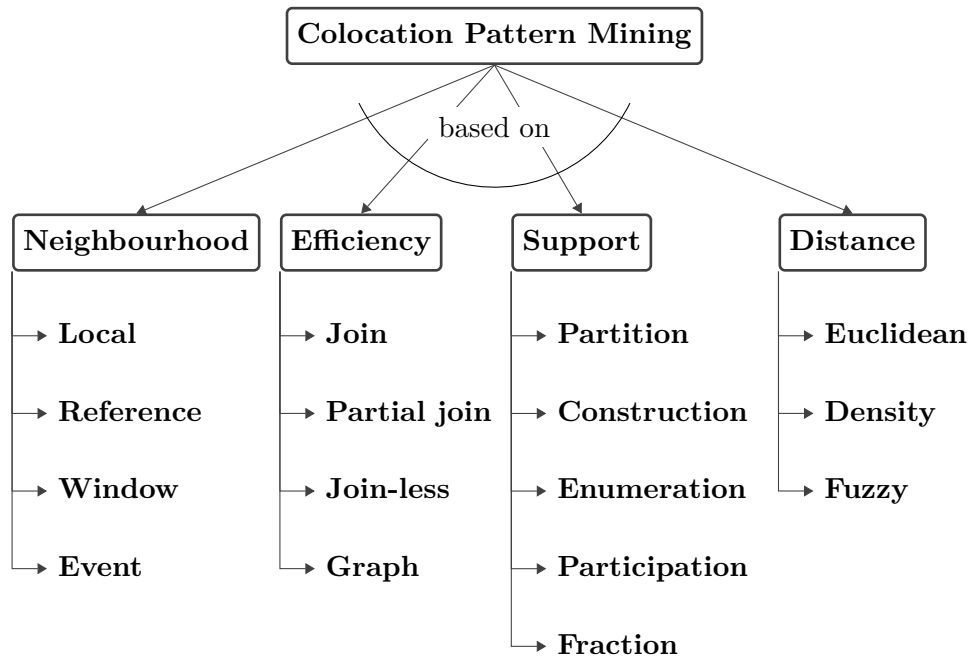


Figure 2.1: A survey of research space

The concept of spatial colocation pattern mining having applications in various application domains [43, 68, 76] was first introduced by Shashi Shekhar *et.al.* [54]. In this work, a join-based approach was used to first create transactions by enumerating neighborhood objects based on predefined spatial proximity. For mining colocation patterns from transactions, they identified candidates using generalized-apriori and pruned them using apriori property for the chosen prevalence threshold. Some of the challenges of this approach are- 1) Finding efficient computational techniques for mining colocation patterns; 2) choosing the appropriate prevalence threshold for pruning candidates and

filtering relevant patterns; 3) defining measures that quantify the relevance of a pattern, and 4) choosing appropriate distance threshold for spatial proximity that identify transactions. To address these challenges, several techniques have been proposed in literature. These techniques aim to improve the efficiency, accuracy, and scalability of spatial colocation pattern mining. Figure 2.1 provides an overview of the research space in spatial colocation pattern mining, highlighting the different techniques, challenges, and recent advancements.

2.2 Computational techniques

The state-of-the-art join-based approach for spatial colocation pattern mining is known to be computationally expensive due to the join step itself. In order to improve efficiency, several algorithms [70–72] have been proposed.

Yoo et al. [72] addressed the issue of computational complexity by introducing a partial join-based approach. This approach improves performance by considering only a subset of the join operations. However, its efficiency heavily depends on the distribution of spatial datasets, and in the worst-case scenario, the time complexity remains equivalent to the join-based approach.

To further enhance efficiency, the joinless approach was introduced by Yoo et al. [70, 71]. This approach employs an instance look-up scheme known as star instances, which eliminates the need for actual join operations. By utilizing this scheme, the time complexity is significantly reduced compared to the join-based and partial join-based approaches.

Another approach to address computational complexity is through the exploration of alternate computational frameworks. Mehta et al. [48] proposed the use of a distributed graph database for spatial colocation pattern mining, leveraging the distributed nature of the database to improve efficiency.

In addition, Huang et al. [24] proposed a multi-resolution filter that reduces the search space for colocation pattern mining, thereby addressing the issue of computational complexity. This filter utilizes different levels of spatial granularity to efficiently filter out irrelevant patterns. Chan *et al.* developed an apriori like algorithm for mining colocation patterns based on fraction score and gave a filter and verification approach to decide whether an object is part of a candidate colocation or not [9].

Table 2.1 provides an overview of the reviewed literature in terms of the proposed computational techniques and their respective references.

Study	Research Objective	Contributions
Yoo et al. (2004)[71]	Address the issue of computational complexity in join-based algorithms.	Proposed a partial join-based approach to improve efficiency. The performance depends on the distribution of spatial datasets, with worst-case time complexity equivalent to the join-based approach.
Yoo et al. (2006) [70]	Introduce the joinless approach with a star instance look-up scheme to further improve time complexity.	Reduced computational complexity by introducing star instances for efficient instance look-up.
Huang et al. (2006) [23]	Reduce the search space with a multi-resolution filter.	Addressed computational complexity by reducing the search space through the implementation of a multi-resolution filter.
Mehta et al. (2018) [48]	Propose an alternate computational framework using a distributed graph database.	Addressed computational complexity by utilizing a distributed graph database as an alternate computational framework.
Chan et al. (2019) [9]	Develop an Apriori-like algorithm based on fraction score for mining colocation patterns.	Introduced a filter and verification approach to determine object inclusion in candidate colocations.

Table 2.1: Finding Efficient Computational Techniques for Mining Colocation Patterns

2.3 Prevalence thresholds

To address the issue of suitable prevalence thresholds and prevalence measures in colocation mining, several techniques have been proposed, as outlined below.

Yan et al. introduced the maximal participation index measure, which does not require a support threshold [25]. This measure focuses on capturing the participation of objects in colocations without setting a specific threshold value, providing more flexibility in identifying significant patterns.

Tran et al. proposed an overlapping clique-based spatial co-location pattern mining framework to tackle the issue of prevalence thresholds [59]. This framework reduces the recomputing cost of the participation index by utilizing a hash map structure, improving the efficiency of prevalence-based pruning techniques.

Barua et al. proposed a statistical approach to overcome the need for a worst-case threshold [7]. By leveraging statistical measures, this approach provides a more adaptive and data-driven method for determining prevalence thresholds, taking into account the inherent variability in spatial data.

Shekhar et al. introduced the participation index as a prevalence measure in colocation mining [54]. The participation index counts the frequency of clique instances and has been widely studied in the field. It serves as a baseline measure for assessing the prevalence of colocation patterns.

Chan et al. proposed a fraction score as a prevalence measure, which offers a tight bound over the participation index and addresses the issue of overcounting due to instance sharing [9]. This measure provides a more accurate estimation of pattern significance by considering the fractional contributions of objects in colocations.

Wenkai et al. presented an alternate prevalence measure based on the likelihood ratio statistic [43]. This measure, suitable for combinatorial optimization problems, offers a different perspective on assessing the prevalence of colocation patterns, taking into account the statistical significance of observed patterns.

Wenhao et al. proposed a network-constrained model for mining colocations by considering distance decay effects [74]. This model incorporates the impact of distance on the prevalence of colocation patterns, allowing for the identification of spatial relationships that exhibit distance-dependent behavior.

These techniques provide various approaches to address the challenge of suitable prevalence thresholds and prevalence measures in colocation mining, enabling researchers and practitioners to better capture and analyze significant spatial patterns.

2.4 Support measures

In colocation mining, various support measures have been proposed to quantify the significance of co-location patterns. The most studied measures include partition-based, construction-based, enumeration-based, and fraction-based measures. Each of these measures has its advantages and limitations, as discussed below.

Partition-based approaches divide the geographical region into smaller grids, and the count of co-location patterns in each grid is aggregated to determine support [54]. This approach is straightforward to apply and can efficiently mine frequent co-locations. However, it has a limitation of missing instances that overlap between different grid cells, leading to potential underestimation of support.

Construction-based methods use heuristics, such as Voronoi partitioning, to find instances of a label set, and the count of these instances is used as support [50]. The limitation of this approach is that the heuristic used to group instances may not always result in an optimal grouping, leading to potential undercounting of support for patterns.

Study	Research Objective	Key Findings
Partition-based [47]	Divide the geographical region into small grids and use the aggregate of co-location patterns in the grids as support.	Easy to apply but misses instances that overlap between different grid cells.
Construction-based [44]	Find instances of a label set using heuristics like Voronoi partitioning and use the instance count as support.	Heuristics may result in undercounting support by not providing optimal grouping.
Enumeration-based [22]	Use the number of row instances where objects are in each other's neighborhood as support.	Can overcount co-location frequency due to overlapping instances.
Fraction-based [8]	Assign fractions to objects based on the overlap of potential co-location pattern instances and aggregate fractions as support.	Addresses overcounting and undercounting issues, providing support closest to the ground truth.

Table 2.2: Support Measures for Mining Co-location Patterns

Enumeration-based approaches determine support based on the number of row instances, where objects form a row instance if they are in each other's neighborhood [79]. While this approach is efficient and straightforward, it can result in overcounting co-location frequency due to the presence of overlapping instances.

The fraction-based approach, proposed by Chan et al. [9], assigns fractions to objects based on the overlap of instances in potential co-location patterns and aggregates these fractions to calculate support. This approach addresses the issues of both overcounting and undercounting, making it a suitable support measure for colocation pattern mining. Studies have shown that the supports obtained from the fraction-based approach are closest to the ground truth compared to other support measures [9].

It is important to consider the strengths and limitations of different support measures when mining colocation patterns, as the choice of the measure can impact the accuracy and reliability of the discovered patterns. The fraction-based approach has emerged as a promising measure that mitigates some of the limitations associated with other approaches.

2.5 Distance Measures

In earlier works [9, 54, 70–72], the choice of a suitable distance threshold for colocation pattern mining was crucial as a slight change in the distance could significantly impact the quality of the results. However, researchers have explored alternative approaches to address this challenge, as summarized in Table 2.3.

Yao et al. proposed a density-weighted distance threshold approach [67], where the proximity and direction of instances are considered to determine the neighborhood relation. By incorporating density information, this approach adapts the distance threshold based on the local density of instances, resulting in more flexible and context-aware colocation mining.

Xiaoqing et al. utilized Voronoi diagrams to establish the network for colocation mining [66]. Instead of relying on a fixed distance threshold, Voronoi diagrams divide the space into regions based on the proximity to predefined objects, enabling the identification of colocations without explicitly specifying a distance threshold.

Vanha et al. employed statistical measures to overcome the need for distance threshold parameters [58]. By leveraging statistical techniques, such as confidence intervals or standard deviations, the approach determines the colocation patterns based on the underlying distribution of data rather than a predefined distance threshold.

Barua et al. proposed an algorithm to find colocation patterns at multiple distances [6]. Instead of a single distance threshold, this approach explores colocations at different distances, allowing for the identification of patterns with varying levels of spatial proximity.

Huang et al. mapped the colocation mining problem into the clustering problem and applied clustering techniques for mining colocations [23]. By treating colocation mining as a clustering task, distance thresholds are implicitly determined based on the clustering algorithm, providing more flexibility in capturing spatial relationships.

Fang et al. used a density-based clustering approach to identify spatial relationships and mine colocations from these clusters [14]. By considering density information, this approach identifies dense regions in the dataset and extracts colocation patterns that satisfy a given notion of density, reducing the dependency on a fixed distance threshold.

These alternative approaches alleviate the challenge of selecting a precise distance threshold and offer more flexible and adaptive mechanisms for colocation pattern mining.

2.6 Fuzzy proximity-based techniques

The second set of problems in spatial colocation pattern mining focuses on incorporating fuzzy proximity metrics, which are summarized in Table 2.4.

Study	Approach	Key Findings
Yao <i>et al.</i> [67]	Density-weighted distance threshold, proximity, and direction for colocation mining	Proposed a density-weighted distance threshold approach for colocation mining that takes into account the density of the points
Xiaojing <i>et al.</i> [66]	Utilization of Voronoi diagrams to find the network	Used Voronoi diagrams to identify co-location patterns based on the relative distances between points
Vanha <i>et al.</i> [58]	Statistical measures used to overcome the need for distance threshold parameters	Developed a method based on statistical measures to avoid the need for distance threshold parameters
Barua <i>et al.</i> [6]	Algorithm for finding colocation patterns at multiple distances	Proposed an algorithm to find co-location patterns at multiple distances
Huang <i>et al.</i> [23]	Mapping of colocation mining problem to clustering problem, applying clustering techniques	Mapped the colocation mining problem to a clustering problem and applied various clustering techniques to identify co-location patterns
Fang <i>et al.</i> [14]	Density-based clustering approach for identifying spatial relationships and mining colocations	Developed a density-based clustering approach to identify spatial relationships and mine co-location patterns

Table 2.3: Colocation Mining Literature based on Distance

Lei et al. proposed a fuzzy-based algorithm that utilizes a fuzzy proximity measure to analyze similarities between spatial features [35]. By incorporating fuzziness in the proximity measure, this approach considers the degree of similarity between objects, allowing for more flexible and nuanced analysis of colocation patterns.

Wang et al. addressed the instance sharing problem by developing a fuzzy proximity metric based on the fuzziness of spatial neighbor relations [63]. This approach enables the identification of colocation patterns by considering the varying degrees of proximity between objects and capturing the uncertainty in spatial relationships.

Li et al. proposed a method to find the top-k spatial colocations without explicitly calculating distances [28]. Instead of relying on distance thresholds, this approach ranks the colocations based on their relevance and importance, providing a fuzzy ranking mechanism for colocation pattern discovery.

Wang et al. also explored the use of fuzzy neighbor relationships in colocation mining

Study	Approach	Key Findings
Lei <i>et al.</i> [35]	Fuzzy-based algorithm utilizing fuzzy proximity measure for analyzing similarities between spatial features	Proposed a fuzzy-based algorithm that uses fuzzy proximity measure to analyze the similarities between spatial features
Wang <i>et al.</i> [63]	Fuzziness of spatial neighbor relation, developing fuzzy proximity metric to address instance sharing problem	Utilized the fuzziness of spatial neighbor relation and developed a fuzzy proximity metric to address the instance sharing problem
Li <i>et al.</i> [28]	Method to find top-k spatial colocations without calculating distances	Proposed a method to find the top-k spatial colocations without the need to calculate distances
Wang <i>et al.</i> [62]	Mining colocations using fuzzy neighbor relationship with a distance range as input	Presented an approach to mine colocations using a fuzzy neighbor relationship, taking a distance range as input rather than a single distance threshold

Table 2.4: Literature on Fuzzy-based Colocation Mining

by taking a distance range as input instead of a single distance threshold [62]. By incorporating a range of distances, this approach considers the uncertainty in spatial proximity and provides more flexibility in capturing colocation patterns.

These fuzzy proximity-based techniques enhance the quality of results in spatial colocation pattern mining by incorporating variation in the distance threshold at the neighborhood relations level. By considering the fuzzy nature of spatial relationships, these approaches offer more robust and adaptive mechanisms for colocation pattern discovery.

2.7 Incremental Mining

In addition to the techniques discussed earlier, there have been studies focusing on mining co-locations in incremental databases. Table 2.5 provides an overview of these incremental mining algorithms. He *et al.* proposed the ICMP algorithm for the incremental maintenance of discovered spatial co-location patterns when a new object is added to the database [16]. This algorithm efficiently updates the existing co-location patterns based on the addition of new points.

Yoo et al. introduced the EUCOLOC algorithm, which efficiently mines co-location patterns in evolving spatial databases [3]. EUCOLOC employs the concept of borders to avoid unnecessary candidate generation, resulting in improved efficiency compared to ICMP. However, both ICMP and EUCOLOC focus only on handling the addition of new points and do not consider point deletion. To address the issue of point deletion, Lu et al. developed the IMPCA algorithm for incremental mining of prevalent co-locations [44]. IMPCA handles both new and deleted points and incorporates a pruning strategy to further enhance efficiency. Lee et al. proposed the incremental topology miner (*Inc_TMiner*) algorithm, specifically designed for incrementally updating topological patterns in spatial-temporal databases [34]. This algorithm performs database projections and searches for patterns using a depth-first search approach. Wang et al. introduced the UUOC (Utility Update of Co-locations) algorithm, which incrementally mines high utility co-locations in a database by considering both the addition and deletion of data points [64]. UUOC takes into account the utility of co-locations and efficiently updates the patterns with changing data.

For efficient mining of co-locations in large datasets, Andrzejewski et al. proposed a parallel approach using the iCPI-Tree structure [5]. This approach leverages the processing power of parallel computing to handle the mining of co-locations effectively. The INC-MGPUCPM algorithm takes advantage of GPU processing power to mine co-locations when the database is updated with new data points. By utilizing the computational capabilities of GPUs, INC-MGPUCPM achieves efficient incremental mining of co-locations. Wang et al. introduced the incremental fuzzy participation index for measuring the prevalence of changed co-locations in updated datasets and designed the IMPCP-FNR algorithm for incremental mining of prevalent co-location patterns [61]. Chang et al. proposed a novel approach that rearranges neighborhood relations to reduce storage requirements and avoid generating non-incremental candidate instances [10]. These incremental mining algorithms provide efficient ways to update and maintain co-location patterns as databases evolve over time. They address the challenges of handling new and deleted points, as well as achieving scalability for large datasets.

2.8 Dynamic spatial co-location patterns

On the other hand, to handle the dynamic nature of spatial objects, research has been conducted on mining dynamic co-locations, as shown in Table 2.6.

Qian et al. proposed an algorithm for mining co-locations with dynamic neighborhood constraints [51]. This algorithm formulates the co-location mining problem as an optimization problem and solves it using a greedy approach. By considering the dynamic changes in neighborhood constraints, this algorithm captures the evolving relationships among spatial objects.

Hu et al. introduced the concept of dynamic spatial co-location patterns, which takes into account the dynamic relationships among spatial features [21]. Instead of mining all

Study	Approach	Key Findings
He <i>et al.</i> [16]	ICMP algorithm for incremental maintenance of discovered spatial co-location patterns when a new object is added	The concept of cross is introduced to reuse already-known knowledge, improving the efficiency of the incremental maintenance algorithm.
Yoo <i>et al.</i> [3]	EUCOLOC algorithm for efficient mining of co-location patterns in evolving spatial databases	EUCOLOC is more efficient than ICMP, but both algorithms only consider adding new points and cannot handle point deletion
Lu <i>et al.</i> [44]	IMPCA algorithm for incremental mining of prevalent co-locations, handling new and deleted points and providing a pruning strategy	IMPCA handles both new and deleted points, and the pruning strategy enhances efficiency
Lee <i>et al.</i> [34]	Inc_TMiner algorithm for incremental updating of topological patterns in spatial-temporal databases	Inc_TMiner performs database projections and uses a depth-first search for pattern search
Wang <i>et al.</i> [64]	UUOC algorithm for incremental mining of high utility co-locations in a database	UUOC considers both addition and deletion of data points for mining high utility co-locations
Andrzejewski <i>et al.</i> [5]	Parallel approach using iCPI-Tree structure for efficient mining of co-locations in large data	iCPI-Tree structure enables efficient mining of co-locations in large datasets
Wang <i>et al.</i> [61]	Incremental mining of prevalent co-location patterns based on FNR (the IMPCP-FNR algorithm)	Incremental fuzzy participation index measures the prevalence of changed co-location, enhancing the mining algorithm
Chang <i>et al.</i> [10]	Approach to rearrange neighborhood relations for efficient storage and non-incremental candidate instance generation	Efficient storage and avoidance of non-incremental candidate instances

Table 2.5: Literature on Incremental Colocation Mining

possible co-locations, this approach focuses on mining only maximal co-locations, which can derive all prevalent co-locations. By considering the dynamics of spatial objects, this method captures the evolving patterns over time.

Ma *et al.* proposed a two-step framework for discovering evolving spatial co-location

patterns [46]. In this framework, an extend-and-evaluate scheme is proposed to form evolving spatial co-locations by selecting appropriate evolvers from the top-k spatial co-location patterns at each time slot. This approach allows for the identification of evolving patterns and provides insights into the changes and trends in co-locations over time.

These studies on dynamic co-location mining address the need to capture the evolving relationships and patterns among spatial objects. By considering dynamic neighborhood constraints and temporal changes, these approaches provide valuable insights into the dynamics and evolution of co-locations.

Authors	Approach	Key Findings
Qian <i>et al.</i> [51]	Algorithm for mining co-locations with dynamic neighborhood constraints, treating it as an optimization problem and solving it using a greedy approach	Dynamic neighborhood constraints can be incorporated in co-location mining using a greedy optimization approach
Hu <i>et al.</i> [21]	Concept of dynamic spatial co-location pattern to consider dynamic relationships among spatial features, focusing on mining only maximal co-locations to derive all prevalent co-locations	Dynamic spatial co-location patterns capture dynamic relationships among spatial features, and mining only maximal co-locations can derive all prevalent co-locations
Ma <i>et al.</i> [46]	Two-step framework for discovering evolving spatial co-location patterns, including an extend-and-evaluate scheme to form evolving spatial co-location by selecting appropriate evolvers from top-k spatial co-location patterns at each time slot	Evolving spatial co-location patterns can be discovered using a two-step framework with an extend-and-evaluate scheme, selecting appropriate evolvers from top-k spatial co-location patterns

Table 2.6: Literature on Mining Co-locations with Dynamic Constraints

2.9 Exploration of Spatio-Temporal Colocation Patterns

The thesis now delves into the evolving landscape of spatio-temporal colocation patterns, a pivotal area highlighted in recent research.

Li *et al.* [42] present an innovative methodology, the Geographically and Temporally

Weighted Co-location Quotient, offering a profound analysis of spatio-temporal crime patterns across Greater Manchester. Their approach tackles the challenges posed by spatial data pooled over time, unraveling symmetrical spatio-temporal co-location patterns and mapping local clusters. This method not only identifies patterns but also addresses the intricacies of varying temporal scales, contributing significantly to understanding the relationships between crime and the urban environment.

Additionally, Ma et al. [46] contribute to this field by proposing a two-step framework for discovering Evolving Spatial Co-location Patterns (ESCs) from spatio-temporal databases. Their methodology, validated using real and synthetic datasets, showcases the identification of ESCs in the Shilin nature preservation zone of Yunnan Province over a decade. This work illuminates the variation trends, diversity, and relationships between SCPs over time, providing crucial insights into evolving spatial co-location patterns.

Furthermore, the study by Tang et al. [56] examines the spatio-temporal evolution patterns and influencing factors of the attractiveness of residential areas to restaurants in the central urban area. The research introduces the concept of the ARTR (the attractiveness of residential areas to restaurants) and measures its value, along with its spatial and temporal evolutionary patterns using global and local colocation quotients. Their analysis unveils the impact of urban expansion and regeneration on the clustering of catering establishments, shedding light on the changing dynamics between urban elements and catering businesses.

These contributions collectively deepen our understanding of spatio-temporal colocation patterns, offering insights into diverse domains such as crime analysis, ecological preservation zones, and the intricate relationship between urban development and catering establishments.

2.10 Interactive and Visual Analytics

In recent years, the landscape of spatial data exploration has seen a significant rise in the development of interactive and visual analytics tools designed to empower analysts and domain experts. Notably, Seebacher *et al.* contributed to this field by developing methods for the interactive analysis of spatio-temporal data abstractions [52]. Their work focuses on enhancing the interpretability and usability of spatial data exploration.

Wang *et al.* introduced a spatial-temporal visual analytics approach for interpretable object detection in autonomous driving [60]. Their work addresses the challenges of understanding when, where, and how object detection may fail in autonomous driving scenarios, providing valuable insights for the development of safer autonomous systems.

While there is existing work on the identification and visualization of spatial and temporal trends [73], there has been limited exploration in the realm of colocation pattern mining. Zhou *et al.* proposed a visualization method for discovering colocation patterns constrained by a road network [78]. Their approach involves network kernel density

estimation and the construction of colocation rule maps, providing a spatially intuitive representation of colocation patterns.

Mengjie *et al.* contributed to colocation pattern mining by introducing a visualization approach for discovering colocation patterns for two independent point distributions [49]. Their method utilizes human color perception to generate colocation rule maps, enhancing the intuitive understanding of spatial relationships.

In a study by Kuo and Lord, the authors applied the color mixing theorem to define the colocation pattern of multiple variables related to crashes, crimes, and alcohol retailers [30]. Their work explores the spatio-temporal evolution patterns and influencing factors of attractiveness in residential areas to restaurants, shedding light on the spatial dynamics of these interactions.

However, these approaches exhibit limitations when applied to a large number of features and colocation sizes. The scalability of these methods becomes challenging, particularly in cases involving a high number of features and big-size colocation patterns, necessitating the integration of machine learning techniques for efficient decoding. Addressing these challenges is essential to unlock the full potential of colocation pattern mining in spatial data analytics.

2.11 Summary

The literature highlights several challenges involved in selecting an appropriate distance threshold for colocation mining. Scholars emphasize the significance of domain knowledge in understanding the spatial relationships relevant to the analysis. Additionally, analyzing the data distribution, aligning with analysis objectives, considering data variability, adopting an iterative approach, and applying validation and evaluation techniques are crucial factors that researchers have investigated to optimize the selection of a distance threshold.

The efficient processing of the neighborhood graph presents unique challenges in colocation mining. Graph representation, encompassing the choice of graph model and determining edge types and properties, has been explored in the literature. Furthermore, handling large-scale graphs with considerations for scalability, capturing indirect relationships, developing graph-specific algorithms, and ensuring effective graph partitioning and distribution are key challenges that have been addressed to improve the processing of the neighborhood graph.

Dynamic data poses specific challenges in colocation mining. Literature has highlighted the importance of managing data updates and maintenance while preserving colocation patterns. Scholars have proposed techniques for incremental processing to efficiently update patterns when new data is added or existing data changes. Additionally, accounting for temporal variations, conducting temporal analysis, and designing efficient

storage and retrieval mechanisms for dynamic colocation patterns have been addressed in the literature.

By reviewing the existing literature, it is evident that researchers have made substantial progress in addressing the challenges of choosing a distance threshold, processing the neighborhood graph, and handling dynamic data in colocation mining. The insights gained from these studies provide a foundation for further research and advancements in the field. Future work can focus on developing novel approaches, algorithms, and frameworks to overcome these challenges and enhance the effectiveness and applicability of colocation mining in various domains.

Overall, this literature review emphasizes the significance of addressing the challenges related to choosing a distance threshold, processing the neighborhood graph, and handling dynamic data in colocation mining. The reviewed studies provide valuable insights into the advancements made thus far and serve as a roadmap for future research endeavors in the field.

Mining Colocation Patterns for a Range Query

This chapter is based on the following journal paper:

Srikanth Baride, Anuj S. Saxena, Vikram Goyal, "Efficiently Mining Colocation Patterns for Range Query", *Big Data Research*, Volume 31, 2023, 100369, ISSN 2214-5796.

3.1 Introduction

One of the most preferred measures for finding colocation patterns is the *Participation Index (PI)* [54]. The PI of a feature set is defined using the count of different objects in spatially close clique instances of the feature set. For a user-provided distance threshold d and a minimum prevalent score min_prev , the existing approaches report all those sets of features as colocations for which the PI is more than the min_prev . The distance threshold d determines spatially close clique instances of the feature set, and thus, the quality of the results depends upon the chosen d value. However, determining a d value is difficult for any domain expert as it is data-dependent.

This causes the domain expert to try different distance values to find out interesting patterns thus forming relevant hypotheses. It can be a very time-consuming effort. In practice, the user will be interested in having the flexibility to input parameters. For example, a user who desires patterns that are around 100 meters apart may consider objects at 100 meter distance and 105 meter distance both nearby. This scenario indicates that in most of the situations the understanding of spatially nearby can reasonably be modeled as an interval over the distance threshold, i.e., $D = [d_1, d_2]$, where $d_1, d_2 \in \mathbb{R}$, and $d_1 < d_2$. In the running example, the user may provide an interval $D = [95, 105]$ for mining colocations that are around 100 meters apart.

Motivated by the challenges, we propose a range query for colocation pattern mining that would accept a distance interval instead of a single distance threshold value. The patterns mined over the distance interval are organized and presented in a way

that reduces a domain expert’s efforts and time to find interesting patterns. The proposed solution would also help a user to know how the pattern changes with distance. There are three reasons why a range query can be useful. 1) It is difficult to decide on a single distance threshold for a task, whereas, intervals are rather easier to choose. 2) The distribution of colocation patterns in a range interval as shown in the lattice visualization of result space in figure 3.1 makes the result space more expressive. 3) The extra distance information with every colocation pattern is useful for post-mining decision-making. The user might be interested in knowing at what distance a particular colocation gets appeared. Moreover, the appearance and disappearance of patterns with changes in the distance value can help understand the trade-off between colocations and the distance threshold for the data at hand. For example, using our technique for the range query $[100m, 400m]$ on *Real_Data_1*, we discover that pattern {Social Services, Religious Institution} gets introduced at distance 351m and pattern {Residential, Social Services} gets introduced at distance 396m. Similarly, for *Real_Data_2* in the range $[50m, 250m]$ we discover pattern {bar, restaurant} at distance of 71m and pattern {recreation, supermarket} at distance 203m.

The existing approaches neither provide support for range query nor can be extended trivially to mine colocations for distance interval. It is because, in mining with a distance threshold $d = 100$, objects with a distance of more than 100 meters are not considered nearby and, therefore, are not a part of any clique instances. Finding new clique instances for the distance value higher than 100 requires the recomputation of nearby objects for those d values.

All distances higher than 100 meters may not add new clique instances. Even those distances that add new clique instances may not contribute enough to the participation index of candidate patterns and therefore may not add new colocations. We call those distances in the range for which there is a change in the colocation pattern set as *critical distances*. One of the major challenges in extending the existing colocation mining approaches for distance range is the lack of information about critical distances in the given range. Somehow even if we know the critical distances, re-computation of (additional) clique instances can not be avoided completely. It makes colocation mining over the distance range query computationally challenging.

We articulate the following research questions solving a range colocation pattern query :

- There are uncountable many points in the given distance range. How do we find a computationally feasible candidate critical distance set out of these uncountable choices?
- Given colossal search space, how do we efficiently find the distances at which a colocation pattern is no longer valid?
- While mining patterns for candidate critical distances, do we need to recompute colocations at each of the critical distances from scratch?, or can there be a better

way out? The recomputation of colocations at each candidate distance is computationally expensive and should be avoided.

To overcome these challenges, our contributions in this work are the following:

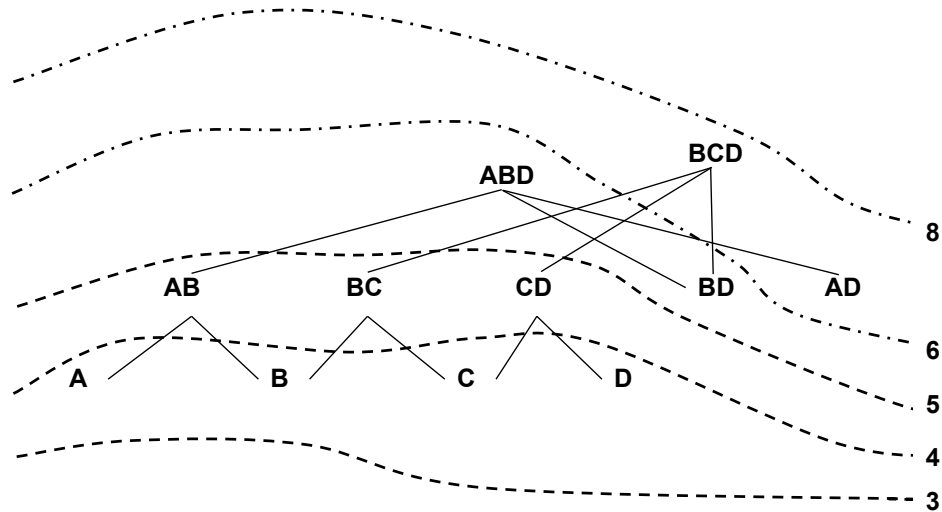


Figure 3.1: Lattice Visualisation of Colocations Over Range Query $D = [3, 8]$

1. We introduce a new query type, called range colocation pattern mining, that allows a domain expert to extract colocation patterns over a distance interval, hence facilitating the exploration of patterns over a large space of neighborhood notion.
2. We give a method to organize and present the patterns over a distance interval as shown in figure 3.1. The presentation method would allow efficient exploration of spatial colocations patterns and form relevant hypotheses with ease.
3. We propose an efficient single-pass algorithm for range colocation mining called *Range – CoMine* that exploits the structural properties of the colocation patterns. It also uses a space-efficient data structure to mine the patterns efficiently by avoiding multiple runs of a colocation mining algorithm. The memory space requirement of the proposed algorithm is minimal.
4. We perform extensive experiments on both real and synthetic datasets to show the efficiency of our proposed strategy. Our proposed algorithm outperforms the two adapted versions of the join-less colocation pattern mining algorithms in terms of space and time requirements.

3.2 Preliminaries, Problem Definition, and Mathematical formulation

In this section, we formally introduce the problem of colocation mining for the distance range query.

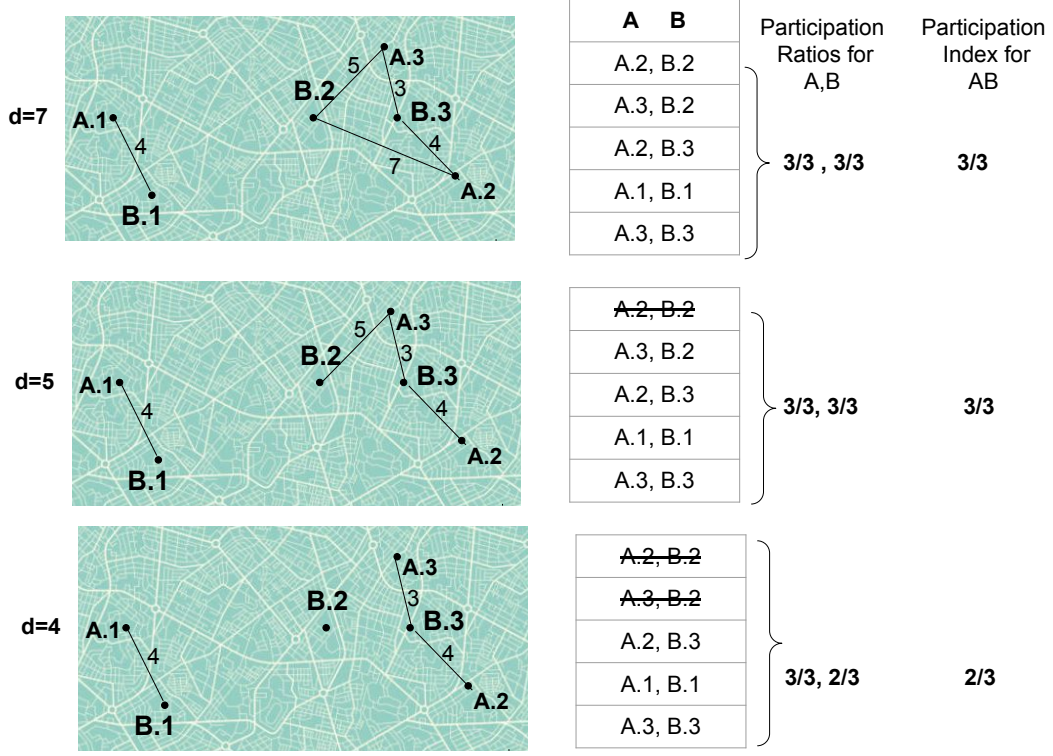
We use the participation index [54] measure to find prevalent colocation patterns that we briefly discuss next, with the necessary changes as required for the distance range query scenario.

We consider that the spatial dataset consists of objects having a unique feature from the feature-set $F = \{f_1, f_2, \dots, f_n\}$. The collection of all the objects in the dataset is denoted by O that can be seen as $O = O_1 \cup \dots \cup O_n$, where each $O_i (1 \leq i \leq n)$ denote objects of a unique feature type f_i . The objects $o \in O$ are represented as a tuple (id, f, l) , where id is the instance id, $f \in F$ is its feature type, and l is its location (x, y) . The problem of co-location pattern mining for the distance range $D = [d_1, d_2]$, with $d_1 < d_2$, is to find a subset p of the spatial features F whose object instances in O are in the spatial proximity for the given range D with high frequency. More formally, a collection of objects $\{o_1, \dots, o_k\} \subseteq O$ is said to be an *object instance* of $p = \{f_1, \dots, f_k\} \subseteq F$ if $o_i.f = f_i$ for all i . We call those subsets p of F for which object instance exists in O a *pattern* or a *candidate colocation*. Thus, we intend to find those patterns (called prevalent colocations) whose object instances are frequently nearby in the spatial region. One of the measures to find how frequently the object instances are spatially nearby is Participation Index (PI). We use the participation Index measure to formally define the prevalent colocation for distance range query $D = [d_1, d_2]$. If no confusion arises, we call the distance range query just a range query in the subsequent discussion.

Two objects o and o' in O satisfy a neighborhood relation R_d for a distance $d \in D$ if the Euclidean distance between them is less than or equal to d . We denote it by $R_d(o, o')$. In figure 3.2, the neighborhood of object instances $B.1$, $B.2$, and $B.3$ of feature type B are shown in a graphical representation for the distance $d = 7$ and the neighborhood relation for the feature types A and B are shown in the list form for distances $d = 7, 5, 4$. From the example, it is clear that objects $A.2$ and $B.2$ satisfy neighborhood relation $R_7(A.2, B.2)$ but not $R_5(A.2, B.2)$. The change in the distance d changes the neighborhood relationship R_d . We will interchangeably use ‘distance d ’ or ‘neighborhood relation R_d ’ to highlight changes due to distance $d \in D$.

We now discuss the definition of clique instances of a pattern $p = \{f_1, \dots, f_k\} \subseteq F$. An object instance $\{o_1, \dots, o_k\}$ of a pattern p is said to satisfy the *mutual neighbourhood relationship* R_d for a distance $d \in D$ if $R_d(o_i, o_j)$ for all $i, j = 1, \dots, k; i \neq j$.

Definition 3.1 (Clique Instance). Let $p = \{f_1, \dots, f_k\} \subseteq F$ be a pattern and $\{o_1, \dots, o_k\} \subseteq O$ be its object instance, i.e., $o_i.f = f_i$ for all i . If for a distance $d \in D$ objects in $\{o_1, \dots, o_k\}$ satisfy the mutual neighbourhood relationship R_d , then $CI = \{o_1, \dots, o_k\}$ is said to be a clique instance of the pattern p for the distance d . The collection of all

Figure 3.2: Neighbourhood Relation R_d for pattern $\{A, B\}$ for distances $d = 7, 5, 4$

the clique instances of a pattern p for the neighborhood relation R_d , denoted by $I_{R_d}^p$, is,

$$I_{R_d}^p = \{CI = \{o_1, \dots, o_k\} \mid \forall i, o_i.f = f_i \wedge CI \text{ is clique in } R_d\}$$

A pattern for which clique instances exist is called a *colocation*. We denote colocations by the symbol c . For every colocation there exists a family of clique instance set $I_{R_d}^p$ for $d \in D$ that satisfy $I_{R_{d_1}}^p \subset I_{R_{d_2}}^p$ for $d_1, d_2 \in D$ with $d_1 < d_2$. The cliques of a two-size pattern are its neighbourhood relations. For example, in figure 3.2, $R_7(A.2, B.2)$ is a clique of a colocation $c = \{A, B\}$ but not $R_5(A.2, B.2)$ is not. The neighbourhood relation lists of a colocation $c = \{A, B\}$ in figure 3.2 also shows the family of clique instances (i.e., $I_{R_d}^{AB}$) for distances $d = 7, 5, 4$.

Next, we discuss the participation ratio of a feature type f_i in a colocation c for $d \in D$ that is defined as the ratio of the number of unique instances of f_i participating in any clique instances of c to the total number of instances of f_i in O .

Definition 3.2 (Participation Ratio). Let $c = \{f_1, \dots, f_k\}$ be a colocation having clique instances $I_{R_d}^c$ under the spatial neighborhood relation R_d . The participation ratio of

feature $f_i \in c$ at distance d , denoted by $Pr(f_i, c, I_{R_d}^c)$, is given as:

$$Pr(f_i, c, I_{R_d}^c) = \frac{|\{o_i \in O_i \mid \{o_1, \dots, o_k\} \in I_{R_d}^c\}|}{|O_i|} \quad (3.1)$$

For example, in figure 3.2, the participation ratio of B in the colocation $c = \{A, B\}$ is $2/3$ for $d = 4$, and is 1 for $d = 5$ or 7. Now we discuss the participation index which is the minimum amongst all the participation ratios of distinct features f_i in the colocation c .

Definition 3.3 (Participation Index). The participation index of a colocation $c = \{f_1, \dots, f_k\}$ having clique instances $I_{R_d}^c$ under the spatial neighborhood relation R_d , denoted by $Pi(c, I_{R_d}^c)$, is defined as:

$$Pi(c, I_{R_d}^c) = \min_{f_i \in c} Pr(f_i, c, I_{R_d}^c) \quad (3.2)$$

For example, in figure 3.2, the participation index of $c = \{A, B\}$ is 1 for $d = 7$ since all the instances of A and B are in the clique instances and is $2/3$ for $d = 4$ as participation ratio of B in $c = \{A, B\}$ is $2/3$. On changing the distance, the participation ratio and participation index also change. We now discuss this relationship between the participation index and the distance $d \in D$.

Lemma 3.4. (*Participation Index Monotonicity Property*) *The participation ratio and the participation index decrease monotonically with the decrease in the distance.*

Proof. The proof of the lemma is direct from the fact that the decrease in the distance threshold reduces the neighborhood relation and clique instances monotonically. \square

The lemma allows not to evaluate a pattern at a lower distance value once it becomes irrelevant at a higher distance value. In figure 3.3, the changes in the participation index of the colocation $c = \{A, B\}$ are shown for the neighbourhood relation as in figure 3.2.

For a user-provided prevalence threshold min_prev , colocation is a *prevalent colocation* for a distance $d \in D$ if its participation index under spatial neighborhood relation R_d is greater or equal to the min_prev . A colocation is said to be a *prevalent colocation in the range D* if it is prevalent for some distance $d \in D$. Thus the objective of range colocation mining is to find all prevalent colocations in the given range D .

Definition 3.5 (Range Colocation Mining). For a given $\langle O, F, D, min_prev \rangle$, where O is a special object having spatial features from F , $D = [d_1, d_2]$ is a distance range, and min_prev is a prevalence threshold, a distance-colocation pair (d, c) is a *prevalent colocation for the range query D* if $d_1 \leq d \leq d_2$ and $c \subseteq F$ is a prevalent colocation for the distance d .

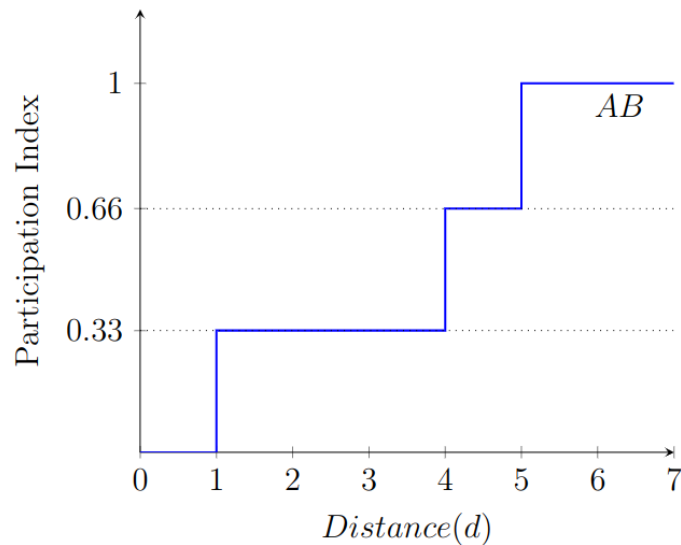


Figure 3.3: Participation Index vs. Distance Threshold for colocation $\{A, B\}$

The extra distance information associated with prevalent colocations for the range query makes the mined patterns more expressive and thus useful for post-mining analysis in decision problems. However, the computation of prevalent colocations for the range D is computationally expensive. It is not clear how to select distances from D if we apply multiple applications of the existing colocation mining approaches for the single distance value. Also, if (d, c) is a prevalent colocation in the range D , then (d', c) is also prevalent for all $d' \geq d$ in the range D and fixed min_prev . Reporting such redundant patterns in the result will reduce its readability and usefulness. To address these issues we define the *critical distance* of a prevalent colocation.

For a prevalent colocation in the range D , a distance in D at which the colocation becomes prevalent is called its *critical distance*. For example, in figure 3.3 for the $min_prev = 0.7$, the critical distance for the colocation $c = \{A, B\}$ is 5 because it is prevalent for $d \geq 5$ but not for $d < 5$. Similarly, for $min_prev = 0.6$, the critical distance for $c = \{A, B\}$ is 4. We denote a list of all the prevalent colocations C having critical distance $d \in D$ by the pair $\langle d, C \rangle$, and by $ColList$, the collection of all such pairs $\langle d, C \rangle$ in the range D . The $ColList$ is the desired output of a range colocation pattern mining algorithm. Since every prevalent colocation in the range D is reported once in $ColList$ by associating it with its critical distance, it resolves the issue of redundant patterns. Also, all critical distances in D are the values that can extend the existing single-distance colocation mining techniques through multiple executions. However, there remain several challenges that we discuss in subsequent sections.

One of the challenges in finding prevalent colocations for the range query $D = [d_1, d_2]$ is- how to efficiently compute all critical distances in the range D for a fixed min_prev ? Also, updating prevalent colocations at those distances is required. Thus, the second challenge is - how to efficiently find prevalent colocations at critical distances in D ? The recomputation of prevalent colocations for every critical distance is an expensive operation. In the next section, we introduce our main approach *Range – CoMine* which efficiently resolves these issues in range colocation mining. We start our discussion by introducing two baseline approaches, namely the Naïve approach and the Range Inc-Mining, which extend the join-less colocation pattern mining [54] by recomputing prevalent colocations. We compare the efficiency of the *Range – CoMine* with these baseline approaches in the experiment section.

3.3 Related Work

In our framework, we extend the computational framework of the join-less approach [71], use the participation index as a prevalence measure, and address the issue of choosing a suitable distance threshold. Many of the earlier works [9, 54, 70–72] require the user to specify a distance threshold that is an exact value such as 5 meters, 10 meters, etc. A slight change in the distance may result in a substantial change in the quality of the result, and thus the choice of a suitable distance threshold is important. In the literature, two types of approaches address this challenge. The first set of problems avoids the need for a distance threshold and develops the neighborhood relation through other means.

Yao *et al.* [67] proposed a density-weighted distance threshold to mine colocations based on the proximity and direction of the instances. Xiaojing *et al.* [66] used Voronoi diagrams to find the network. Vanha *et al.* [58] uses statistical measures to overcome the need for distance threshold parameters. Barua *et al.* [6] proposed an algorithm to find colocation patterns at multiple distances. Huang *et al.* [23] map the colocation mining problem into the clustering problem and show that the clustering techniques can be applied to mine colocations. Fang *et al.* [14] used a density-based clustering approach to identify the spatial relationship and then mine colocations from these clusters. Density-based colocation pattern mining techniques report patterns that satisfy a given notion of density. On the other hand, the proposed Range query captures a family of density (neighborhoods) defined using distance thresholds in the given distance range.

The second set of problems incorporates fuzzy proximity metrics in spatial colocation pattern discovery. Lei *et al.* [35] proposed a fuzzy-based algorithm that uses fuzzy proximity measure to analyze the similarities between the spatial features. Wang *et al.* [63] uses fuzziness of spatial neighbor relation to addressing the instance sharing problem by developing fuzzy proximity metric. Li *et al.* [28] proposed a method to find the top-k spatial colocations without calculating distances. Wang *et al.* [62] also worked on mining colocations using fuzzy neighbor relationship by taking a distance range as input rather than a single distance threshold. The fuzzy proximity-based techniques

enhance the quality of results by incorporating variation in the distance threshold at the neighbourhood relations level. However, this information about the distance change and its effect on the patterns is not associated with the result that may be desirable for post-mining analyses. Thus we are motivated to find an efficient technique that mines spatial colocation patterns for distance range query and enhances the expressive power of patterns for post-mining analysis by associating a pattern visibility distance with the patterns in the result set.

To the best of our knowledge, there exists no prior work that discusses the colocation pattern mining for the distance range query. The existing algorithms for colocation pattern mining are not directly applicable due to computational overhead as multiple applications over the distance range are required. Also, in some of the existing approaches for colocation mining, the spatial data was first converted into the transaction data, and there exist techniques for range-based association rule mining [38, 47, 53] over transaction data. However, these approaches are also not helpful for getting a meaningful solution for range colocation mining. It is because the spatial data is converted into transaction data by realizing the neighbourhood relation that depends upon the distance threshold. Therefore, for comparison purpose, we have developed a Naïve Approach that apply joinless approach [71] multiple times by choosing multiple distances in the distance range.

3.4 Methodologies for Range Colocation Mining

In range colocation Mining, for a user-provided min_prev threshold and a range query $D = [d_1, d_2]$, the objective is to find $ColList$ consists of pairs $\langle d, C \rangle \mid d \in D$ where C is a list of prevalent colocations having critical distance $d \in D$.

We now discuss our first baseline, the Naïve Approach, that would find candidate critical distances in D and then compute prevalent colocations for those distances. It uses a traditional colocation mining algorithm [54] as a basic routine to recompute prevalent colocations for the chosen candidate distances. From the result of the candidate distances, the desired $ColList$ is generated.

3.4.1 Naïve Approach

For distance $d \in D$ and its neighbourhood relation R_d , we denote by D_{pair} the set of distances between any two objects in R_d that lies in the range D . Distances in D_{pair} are the edge distances in the neighbourhood graph of R_d . The next result discusses the relationship between the critical distance of the prevalent colocations and D_{pair} .

Lemma 3.6. *The set of all the critical distances of the prevalent colocations corresponding to the neighbourhood relation R_d is a subset of D_{pair} for the same R_d .*

Proof. The proof is direct from the fact that the critical distance of a prevalent colocation corresponds to some of its clique instance deletions. The clique instance deletion is due to an edge deletion in the neighbourhood graph. Thus the collection of critical distances is a subset of D_{pair} . However, every clique instance deletion may not decrease the participation index below the min_prev . Hence every edge distance in D_{pair} need not be a critical distance. \square

We use D_{pair} as candidate critical distances in the Naïve Approach. For computational ease, the distances in D_{pair} are stored in decreasing order. We denote by C_d the collection of all prevalent colocations at a distance d . The next result explains the relationship of the prevalent colocation set with a change in candidate distance in D_{pair} .

Lemma 3.7. *For consecutive distances in D_{pair} , the prevalent colocation sets are monotonically decreasing, i.e., for d_{i-1}, d_i in D_{pair} , we have, $C_{d_{i-1}} \supseteq C_{d_i}$, where C_d denotes a set of prevalent colocations at distance d .*

Proof. The result is direct from that edge-distances in D_{pair} are in decreasing order, i.e., $d_{i-1} > d_i$, and the neighborhood graph for d_i is a sub-graph of neighbourhood graph for d_{i-1} . \square

Since every distance in D_{pair} need not be a critical distance, the prevalent colocations set C_d may not change for every $d \in D_{pair}$. Let there be a change in C_{d_i} for $d_i \in D_{pair}$. From lemma 3.7, we have $C_{d_{i-1}} \supset C_{d_i}$. Let us denote by $C_{changed}$ all the prevalent colocations that are in $C_{d_{i-1}}$ but not in C_{d_i} . In the next lemma, we discuss that d_{i-1} is a critical distance for all the prevalent colocations in $C_{changed}$.

Lemma 3.8. *For any two consecutive distances d_{i-1}, d_i in D_{pair} such that $C_{d_{i-1}} \supset C_{d_i}$, d_{i-1} is a critical distance for colocations in $C_{changed} = C_{d_{i-1}} \setminus C_{d_i}$.*

Proof. The neighbourhood relation changes only for distances in D_{pair} . Since d_{i-1} and d_i are consecutive distances in D_{pair} , no distance in the interval (d_i, d_{i-1}) can be the critical distance for $C_{changed}$. Hence the result. \square

We now discuss the Naïve Approach for the range query $D = [d_1, d_2]$ in the Algorithm 1. It uses D_{pair} for distance d_2 as candidate distances to compute $ColList$ consisting of pairs $\langle d_{i-1}, C_{changed} \rangle$ as stated in lemma 3.8.

Procedure 1 Naïve Approach ($O, F, [d_1, d_2], \text{min_prev}$)

```

1: Compute neighbourhood relation  $R_{d_2}$  at distance  $d_2$ 
2: Compute  $D_{pair}$  for  $R_{d_2}$ , and maintain it in decreasing order
3: for each  $d_i \in D_{pair}$  do
4:   Compute neighbourhood  $R_{d_i}$  and clique instances at  $d_i$ 
5:   Compute colocation set  $C_{d_i}$  for  $R_{d_i}$  using colocation algorithm
6:   if  $C_{d_i} \neq C_{d_{i-1}}$  then
7:     Compute  $C_{changed} = C_{d_{i-1}} \setminus C_{d_i}$ 
8:     Update ColList with pair  $\langle d_{i-1}, C_{changed} \rangle$ 
9:   end if
10: end for
11: return  $ColList$ 

```

The correctness of the Naïve Approach is justified by Lemma 3.8. It is easy to apply but is computationally expensive. There are two issues with this approach.

1. The candidate distances in D_{pair} are much more as compared to the actual critical distances within the range D . This results in unnecessary computations of prevalent colocation set several times.
2. It is expensive to recompute prevalent colocations for each distance in D_{pair} as in line 4 and line 5 of Algorithm 1. From lemma 3.7, the colocation patterns at different distances in D_{pair} are monotonically decreasing. Updating them from the previous state is advantageous.

In the next section, we discuss the second baseline approach *RangeInc – Mining* that updates the prevalent colocations for the distances in D_{pair} .

3.4.2 *RangeInc – Mining*

The *RangeInc – mining* algorithm differs from Algorithm 1 in line 4 and line 5 only. We compute D_{pair} for d_2 for the given range query $D = [d_1, d_2]$, and maintain it in decreasing order, as in line 1 and line 2 of Algorithm 1. For the first distance value in D_{pair} , the prevalent colocations are computed using a traditional colocation mining algorithm. All the prevalent colocations with their clique instances are stored for further computation. In the next stage, with a change in the distance $d \in D_{pair}$, we discard those clique instances from the previously stored clique instances that disappear. It requires an update operation in line 4 of Algorithm 1. The process is repeated for every distance in D_{pair} by storing colocations and clique instances of the previous state. Thus with extra storage space, we can avoid the recomputation of clique instances, a costly operation.

3.4.3 Range – CoMine

The limitation of the baseline approaches is that the candidate critical distances in D_{pair} are too many, and multiple passes of traditional colocation mining techniques over D_{pair} make them highly inefficient. To overcome this issue, in this section, we first define the critical distance formally and propose an efficient technique to compute it. Next, we propose a single pass technique *Range – CoMine* for the range query that significantly reduces the computation cost and memory requirement.

A *critical distance* of a prevalent colocation is the smallest distance in the range $D = [d_1, d_2]$ at which the colocation is prevalent for the first time. More formally,

Definition 3.9 (Critical Distance). For a given min_prev threshold, a distance d is the critical distance of colocation c if the pattern c is prevalent at distance d but not prevalent for any distance smaller than d . More formally, for a colocation c that is prevalent for distance $d \in D$, its critical distance denoted by $cr.dist(c, min_prev)$ is

$$cr.dist(c, min_prev) = \min_{d' \in [d_1, d]} \{d' \mid Pi(c, I_{R_{d'}}^c) > min_prev\}$$

The above definition does not help in computing critical distance due to infinitely many candidate distances in the interval $[d_1, d]$. We denote the *candidate critical distances* for a prevalent colocation c by $CAND(c)$. It is to observe that only those distances that correspond to the distance between the pair of the objects in some clique instances of c may become critical distances. Therefore, for a prevalent colocation c having collection of clique instances $I_{R_d}^c$ for distance d ,

$$CAND(c) = \{dist(o_i, o_j) \mid CI \in I_{R_d}^c, CI = \{o_1, \dots, o_k\} \subseteq O\}$$

The number of distances in $CAND(c)$ is still too many. We use clique diameter that reduces $CAND(c)$ substantially. The clique diameter of a clique $CI \in I_{R_d}^c$, denoted by $dia(CI)$, is the maximum pairwise distance of the objects in CI , i.e.,

$$dia(CI) = \max\{dist(o_i, o_j) \mid o_i, o_j \in CI\}$$

A clique instance $CI \in I_{R_d}^c$ exists if the distance threshold is greater or equal to the clique diameter. This reduces the candidate distance of a prevalent colocation c to only the diameters of the clique instances CI in $I_{R_d}^c$, one for each clique instance. More formally,

$$CAND(c) = \{dia(CI) \mid CI \in I_{R_d}^c\} \quad (3.3)$$

Let us consider a toy example as in Figure 3.4 that shows the graph representation of neighbourhood relation for the distance threshold $d = 8$. The distance between the objects is shown as an edge weight. For computational purposes, we store the neighbourhood relation as a star neighbourhood [70]. Extra distance information with each object is stored in the star neighborhood, which is its distance from the center.

Definition 3.10 (Star Neighborhood). Given a spatial object $o_i \in O$ whose feature type is $f_i \in F$, the star neighborhood of o_i is defined as a set of pairs consisting of spatial objects together with its distance from o_i , given by the following equation:

$$SN_d = \{(o_j, dist(o_i, o_j)) | o_j = o_i \vee (f_i > f_j \wedge R_d(o_i, o_j))\} \quad (3.4)$$

where $f_j \in F$ is the feature type of o_j and R_d is neighbourhood relation for distance d .

The star neighbourhood of the relation R_8 is shown in Table 3.1. The edge weight with the neighbourhood objects, i.e., (*Neighbour, distance*), in the star neighbourhood list supports the computation of the diameter. The clique diameters are computed while validating the clique from the star instances using the distance information stored in the star neighbourhood list. The collection of all the clique instances with their diameters is shown in Table 3.2, below the respective colocation patterns.

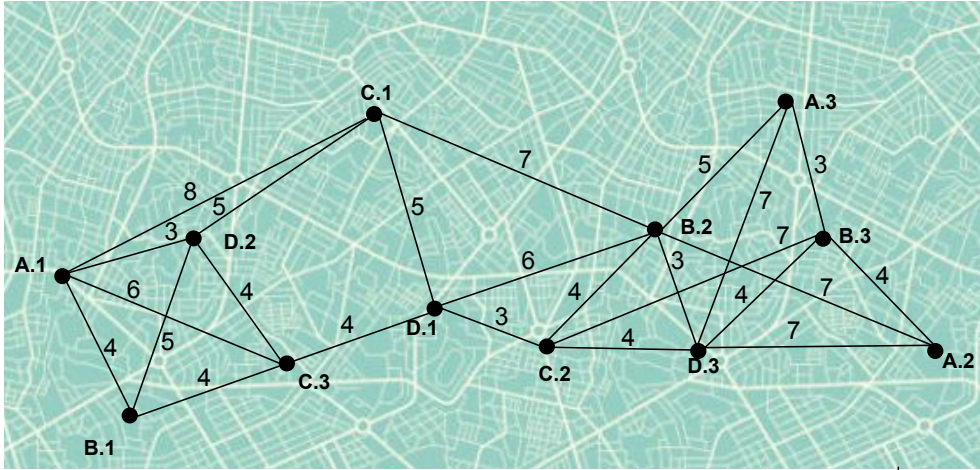


Figure 3.4: Graph representation of Neighborhoods Relation at distance $d = 8$

From Table 3.2, the candidate critical distances for pattern $\{A, B\}$ are $CAND(\{A, B\}) = \{3, 4, 5, 7\}$, which are the diameters of their clique instances. Though, the diameters of the clique instances of prevalent colocations in the range $[d_1, d_2]$ reduce candidate critical distances substantially. The next property, namely *critical distance monotone property (CDMP)* further reduces the size of the candidate critical distance set. We first discuss the property.

Lemma 3.11. (*Critical Distance Monotonicity Property (CDMP)*) Let d and d' be the critical distances of prevalent colocations c and c' respectively such that $c \subset c'$. Then the critical distance of c' would be larger or equal than the critical distance of c , i.e., $d \leq d'$.

Proof. On the contrary, assume that the statement is not correct, i.e., $d > d'$. Since d is the critical distance of c , we get that the pattern c would not be prevalent at d' .

Centre		(Star Neighbour, distance) List
F	O	
A	A.1	(A.1, 0), (B.1, 4), (C.1, 8), (C.3, 6), (D.2, 3)
	A.2	(A.2, 0), (B.2, 7), (B.3, 4), (D.3, 3)
	A.3	(A.3, 0), (B.2, 5), (B.3, 3), (D.3, 8)
B	B.1	(B.1, 0), (C.3, 4), (D.1, 8), (D.2, 5)
	B.2	(B.2, 0), (C.1, 7), (C.2, 4), (D.1, 6), (D.3, 4)
	B.3	(B.3, 0), (C.2, 7), (D.3, 3)
C	C.1	(C.1, 0), (D.1, 5), (D.2, 5)
	C.2	(C.2, 0), (D.1, 3), (D.3, 6)
	C.3	(C.3, 0), (D.1, 4), (D.2, 4)
D	D.1	(D.1, 0)
	D.2	(D.2, 0)
	D.3	(D.3, 0)

Table 3.1: Star Neighborhood List for $d = 8$

AB :3/3	AD: 2/3	BC: 3/3	BD: 3/3	CD: 3/3
A.1, B.1 - 4	A.1, D.2 - 3	B.1, C.3 - 4	B.1, D.1 - 8	C.1, D.1 - 5
A.2, B.2 - 7	A.2, D.3 - 3	B.2, C.1 - 7	B.1, D.2 - 5	C.1, D.2 - 5
A.2, B.3 - 4	A.3, D.3 - 7	B.2, C.2 - 4	B.2, D.1 - 6	C.2, D.1 - 3
A.3, B.2 - 5		B.3, C.2 - 7	B.2, D.3 - 4	C.2, D.3 - 6
A.3, B.3 - 3			B.3, D.3 - 4	C.3, D.1 - 4
				C.3, D.2 - 4
	ABD: 2/3	BCD: 3/3		
	A.1, B.1, D.2 - 5	B.1, C.3, D.2 - 5		
	A.2, B.2, D.3 - 7	B.2, C.1, D.1 - 7		
	A.2, B.3, D.3 - 4	B.2, C.2, D.1 - 6		
	A.3, B.2, D.3 - 7	B.2, C.2, D.3 - 6		
	A.3, B.3, D.3 - 7	B.3, C.2, D.3 - 7		

Table 3.2: Clique Instances with Diameters for $d = 8$ and $min_prev = 7$

Consider all clique instances $I_{R_{d'}}^{c'}$ of colocation c' for the neighbourhood relationship $R_{d'}$ at distance d' . Let $F = c' \setminus c$ be a collection of all features in c' that are not in c . Clearly, the projection of $I_{R_{d'}}^{c'}$ to $I_{R_{d'}}^c$ by removing objects of features F will give us collection of all clique instances of c at distance d' . We argue that c would be prevalent at distance d' .

As c' is prevalent at d' , let $f_i \in c'$ is the feature with the minimum participation ratio in $I_{R_{d'}}^{c'}$. There are two cases:

1. When $f_i \in c$: For all the features $f \in F$, the objects of f in $I_{R_{d'}}^{c'}$ are in $I_{R_{d'}}^c$ also. This implies that the participation ratios of $f \in F$ remain the same in $I_{R_{d'}}^c$, i.e., f_i would be the feature with the minimum participation ratio in $I_{R_{d'}}^c$. Therefore c is prevalent at distance d' also.
2. When $f_i \notin c$: Let $f_j \in (c' \cap c)$ is feature in pattern c with the minimum participation ratio in $I_{R_{d'}}^{c'}$. Since f_j is in c' also, the participation ratio of f_j is greater or equal to that of f_i . Thus the participation index of c will be greater or equal to the participation index of c' at distance d' , and therefore c will remain prevalent at d' .

This is a contradiction to our initial assumption that implies c cannot be prevalent at d' . Therefore the critical distance of a super pattern is greater or equal to the critical distance of a subpattern i.e., $d \leq d'$. \square

For example, from Table 3.6 colocations $\{A, B\}$ and $\{A, B, D\}$ having critical distances 4 and 5 respectively satisfy the property in lemma 3.11. Similarly, the relationship is true for colocations $\{B, C\}$ and the $\{B, C, D\}$ also. Since the participation index satisfies anti-monotonicity property[24], we use the Apriori strategy for enumerating candidate colocations and computing their critical distances. The CDMP property reduces the candidate critical distances of colocations of size k that are generated from colocation of size $k - 1$. While mining colocations of size k , all colocations of size $k - 1$ with their critical distances are already computed. Let c_{k-1} is a prevalent colocations of size $k - 1$ having the critical distance cr_{k-1} , and c_k be a pattern of size k such that $c_{k-1} \subset c_k$. From the CDMP property, we have

$$CAND(c_k) = \{dia(CI) \mid CI \in I_{R_d}^{c_k} \text{ and } dia(CI) \geq cr_{k-1}\} \quad (3.5)$$

We now discuss an efficient algorithm that on the fly compute the critical distances for all the colocations for a fixed distance threshold d and given min_prev threshold.

Procedure for Computing Critical Distances

The critical distance computation is a three-step procedure as shown in Figure 3.5. For computing the critical distance of a prevalent colocation c having cliques instances $I_{R_d}^c$ and candidate critical distances $CAND(c)$, we perform the following steps. For

computation, we consider $CAND(c)$ consists of diameters of the clique instances as in equation 3.3.

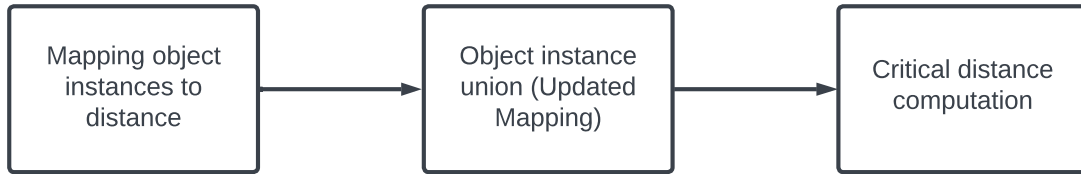


Figure 3.5: Procedure for finding critical distances

- **Step 1: (Mapping object instances to distance).** In this step, we maintain a *map of different object instances that participate in clique instances of a colocation c to its diameter*.

For each clique instance $CI \in I_{R_d}^c$ having diameter dia , the clique instance CI exist for any distance $d > dia$. Thus, the objects in CI can contribute to the participation index for distances greater or equal to dia . We record this information about the objects by mapping different feature objects in CI to the corresponding dia . The data structure for maintaining this record is shown in Table 3.5 that demonstrates the record of object instances of colocation $\{A, B\}$ as in Table 3.2.

The different clique instances of a colocation may have the same diameter. The objects of such clique instances are appended to the object list of the respective features of the same diameter. For example, for colocation $\{A, B\}$ as in Table 3.2, there are two clique instances $A.1, B.1$ and $A.2, B.3$ with diameter 4. The map contains $\{A.1, A.2\}$ objects of feature A and $\{B.1, B.3\}$ objects of feature B with distance 4 corresponding to these colocations.

An object may be a part of multiple clique instances having different diameters. Such objects would be in the object list of respective features at multiple locations. For example, in Table 3.5, object $A.2$ is available at distance 7 and distance 4 both as it is a part in clique instance $A.2, B.2$ and $A.2, B.3$ with diameters 7 and 4 respectively.

- **Step 2 : (Object instance union).** In this step, we take the union of the object instances of the map maintained in the previous step starting from the smallest candidate distance to the largest candidate distance in $CAND(c)$.

All the object instances that are first available at a distance d remain available for all $d' \geq d$. This is because $I_{R_d}^c \subseteq I_{R_{d'}}^c$ for all $d' \geq d$. This requires updating the map to mark the availability of objects for all the distances in $CAND(c)$ that is greater than the object's critical distance. Table 3.5 shows the updated mapping by incorporating object instance union of objects in Table 3.5.

- **Step 3 : (Computing Critical distance of a colocation)** For each candidate distance $d \in CAND(c)$, we associate the feature weight as the number of distinct objects of that feature in the object instance union list. For a feature $f \in F$ we denote its feature weight as W_f . Using the feature weight, we can compute the participation index (PI) for each candidate distance for the prevalent colocation. The feature weights of features A and B and the participation index (PI) of the colocation $\{A, B\}$ for the candidate distances are shown in Table 3.5.

For the given min_prev threshold, by comparing PI value with the threshold value the critical distance of a colocation can be computed. The distance value corresponding to a PI value that is just bigger than the min_prev is the critical distance of a colocation. For example, from Table 3.5, for $min_prev = 0.66$, the critical distance of $\{A, B\}$ is 4 whereas for $min_prev = 0.7$, the critical distance of $\{A, B\}$ is 5.

<i>CAND</i>	A	B
7	{A.2}	{B.2}
5	{A.3}	{B.2}
4	{A.1, A.2}	{B.1, B.3}
3	{A.3}	{B.3}

Table 3.3: Map of objects to distance (CAND) for colocation $\{A, B\}$

While validating the clique instances from the star instances of a pattern, we maintain the mapping of candidate distances to object instances (as in Table 3.5), and use it to compute the mapping of candidate distances to PI value (as in Table 3.5). It helps compute the critical distance of colocations on the fly while mining them for the first time.

We also maintain *ColList* consisting of critical distances associated with the list of colocations having that critical distance. The *ColList* of the critical distances to the list of colocations for the neighbourhood relation R_8 in figure 3.4 is shown in Table 3.6. The lattice visualisation of *ColList* for Range query $D = [3, 8]$ as in Table 3.6 is shown in figure 3.1.

<i>CAND</i>	A	B
7	{A.1, A.2, A.3}	{B.1, B.2, B.3}
5	{A.1, A.2, A.3}	{B.1, B.2, B.3}
4	{A.1, A.2, A.3}	{B.1, B.3}
3	{A.3}	{B.3}

Table 3.4: Object instance union (Updated Mapping)

<i>CAND</i>	W_A	W_B	PI
7	3	3	1
5	3	3	1
4	3	2	$\frac{2}{3}$
3	1	1	$\frac{1}{3}$

Table 3.5: Feature weight and PI for candidate distances

Critical Distance	List of Colocations
3	A, B, C, D
4	AB, BC, CD
5	BD, ABD
6	AD, BCD

Table 3.6: *ColList*: Critical distance to list of Colocations

The *ColList* can be computed while mining the prevalent colocation for the first time and is instrumental in providing the result of the range query on the fly. To further reduce the candidate critical distance set, as proposed in Equation 3.5, we need to maintain a temporary list of different features' objects in Table 3.5. This temporary list stores objects of all the clique instances whose diameter is less than the critical distance of the sub-pattern that grows the current pattern. For all those clique instances for which the diameter is more than or equal to the critical distance, a map is maintained similarly to the previous case. The union of the objects of this temporary list is taken with the lowest distance objects in the map before performing the object instance union in step 2. Thus, we only maintain a record of the candidate distances in Table 3.5 and Table 3.6 that are greater or equal to the critical distance of the sub-patterns.

3.5 Algorithm for *Range – CoMine*

In this section, we delve into the details of the *Range – CoMine* procedure, which is the core component of our research for mining colocation patterns within a specified range query $D = [d_1, d_2]$, as outlined in procedure 2. Additionally, we conduct a comprehensive complexity analysis of the *Range – CoMine* algorithm, shedding light on its computational efficiency and performance characteristics.

The initial phase of our colocation pattern mining process involves the transformation of a spatial dataset (in line 2) and initialization of candidate colocations P_1 and prevalent colocations *ColList* (in line 3).

In line 2, the spatial data consisting of spatial objects O having features F is transformed into a set of distinct star neighborhoods for the distance d_2 . Each star neighborhood is composed of a central object and its associated neighboring objects together

with its distance from the central object as defined in equation 3.4. We store only those objects in its neighborhood whose features are greater than the feature of the center object, arranged in a lexical order. A pivotal step in this transformation is the identification of neighboring objects. To accomplish this efficiently, we employ a spatial data structure known as the IR-tree [37]. The IR-tree assists in the efficient identification of neighboring objects within d_2 distance to each other. The resulting set of star neighborhoods effectively partitions the spatial dataset. Star neighborhood avoids duplication without any loss of neighbor relationships. Further details and examples of the star neighborhood structure can be found in Table 3.1.

In line 3, we generate size-1 Candidate Colocations. By definition of participation index, equation 3.2, all the features of size-1 are prevalent colocation. We, therefore, initialize the *ColList* with size-1 subset of features as prevalent colocations with $d_1 \in D$ as their critical distance.

In lines 4-24, we incrementally iterate over candidate pattern size to find colocations with their critical distances over the range $[d_1, d_2]$. In line 4, we initialize k to size-2, to find candidate colocations $C[k]$ (starting with $C[2]$) from the prevalent colocations of size $k-1$ (i.e., P_{k-1}).

In line 6, we undertake the task of generating candidate colocations. This involves determining the number of instances associated with each feature during the neighborhood materialization process. Our primary objective is to produce candidate colocations with a size greater than one ($k > 1$) using prevalent colocations of size $k-1$ as a starting point. We initiate this process with a set of prevalent colocations of size $k-1$ and subsequently generate new candidate colocations of size- k by combining features that are prevalent. To refine these candidates, we employ feature-level filtering, effectively eliminating any candidate colocation where even a single subset of features is not prevalent.

In line 7, we filter the star instances. The star instances of a candidate colocation are collected from the star neighborhoods that have a center object with a feature type matching the first feature of the colocation. For instance, if we're dealing with the colocation B, D, we would gather instances from the star neighborhoods associated with feature B. Similarly, for colocation A, B, D, we collect instances from the star neighborhoods linked to feature A. For example for candidate colocation ABD the start instance would consists of A1,B1,D2 along with maximum of distances of the instances from the centre.

The list of clique instances CI_k contains elements (ci, dia) where ci is the clique having diameter dia . First, we check if the size of the star instance is 2 (line 8) and directly assign 2-size star instances as clique instances(line 9).

In line 11, we first verify if they exhibit clique-like characteristics. Before initiating this process, we apply a coarse filter to the colocations, where pruning candidate colocations is done based on the participation index of their associated star instances.

In line 12, from the star instances of a candidate colocations we filter the clique instances based on whether they form a clique under neighborhood relationship R_{d_2} or

not. We do this using an instance look-up scheme. For example, the star instance A.1, B.1, D.2 is evaluated for cliqueness by examining if the subinstance B.1, D.2 (excluding A.1) belongs to the set of clique instances of colocation B, D. While validating ci is a clique we also compute its diameter which is maximum of pair wise distances. An example clique instances with diameters are shown in Table 3.2.

Overall in lines 6-12, we generate clique instances with the corresponding diameters using the apriori rule. The steps for clique generation are similar to the earlier approaches [41],

Procedure 2 Algorithm for *Range – CoMine*

```

1: procedure RANGE-COMINE( $[d_1, d_2]$ )
2:    $R_{d_2} = \text{gen\_Star\_Nbd}(F, O, d_2)$ 
3:    $P_1 = F; \text{ColList}[d_1] \leftarrow P_1$ 
4:    $k = 2$ 
5:   while (not empty  $P_{k-1}$ ) do
6:      $C[k] = \text{gen\_candidate\_colocations}(P_{k-1})$ 
7:      $SI_k \leftarrow \text{filter\_Star\_Instance}(C_k, R_{d_2})$ 
8:     if  $k = 2$  then
9:        $CI_k = SI_k$ 
10:    else
11:       $C[k] = \text{select\_coarse\_prevalent\_colocations}(C[k], SI_k, \text{min\_prev})$ 
12:       $CI_k \leftarrow \text{filter\_Clique\_Instances\_with\_Diameter}(C[k], SI_k)$ 
13:    end if
14:    for  $c_k \in C[k]$  do
15:      if  $c_k$  prevalent then
16:         $cr \leftarrow \text{compute\_CrDistances\_colocations}(c_k, CI_k, \text{min\_prev})$ 
17:         $P_k.append(c_k)$ 
18:         $\text{ColList}[cr].append(c_k)$ 
19:      end if
20:    end for
21:     $k = k + 1$ 
22:  end while
23:  return  $\text{ColList}$ 
24: end procedure

```

In lines 14-23, we compute critical distances for each of the candidate colocation patterns and maintain the *ColList* containing list of colocations along with their critical distance. The detailed procedure of computation of critical distance is discussed in section 3.5. If a candidate colocation c_k is not prevalent, it is discarded; otherwise, c_k is appended in *ColList* along with its critical distance cr (line 18). The structure of *ColList* is presented in Table 3.6 for the colocations of spatial objects in Figure 3.4 for $d = 8$. The colocation c_k is also stored in the list of prevalent colocations P_k to enumerate its super set (line 17). Finally, we return the list of colocations with their

critical distance *ColList* in line 23 when there are no more colocation patterns to explore (i.e., P_k is empty for some k in line 5).

Implementation Issues

The primary challenge in implementing colocation pattern mining is defining an appropriate distance threshold. There are two common methods to determine values for d_1 and d_2 :

A. Statistical Analysis (Mean and Standard Deviation)

- Start by analyzing the dataset to calculate the mean and standard deviation of distances between data points.
- Choose d_1 by adding a multiple of the standard deviation to the mean (e.g., mean + k * standard deviation).
- Choose d_2 by subtracting the same multiple of the standard deviation from the mean (e.g., mean - k * standard deviation).
- The specific value of ' k ' should be adjusted based on your dataset's characteristics and the requirements of your analysis.

B. Dataset Properties

- Another approach is to consider the unique properties of your dataset, especially those related to colocation patterns.
- Based on these dataset-specific properties, you can determine suitable values for d_1 and d_2 .

These methods help ensure that the chosen distance threshold values are appropriate for your specific dataset and analytical goals.

Complexity Analysis

This section analyses the computational complexity of the *Range – CoMine* algorithm. We first define the cost to compute critical distance in *Range – CoMine* (line 14 in Algorithm 2) for each prevalent colocation as discussed in Section 3.5. Let us assume that the maximum size colocation pattern in the result set is of length N , and the upper bound over the number of clique instances for any colocation pattern is K . Then the cost of computing critical distances (refer Table 3.5) is

1. The cost of mapping object instances to the clique diameter, thus constructing the map as in Table 3.3 is $O(N \cdot K)$. This is equal to seeing all the object instances in the clique instance once and creating the map.
2. The cost of object instance union as discussed in Table 3.4 is $O(N \cdot K^2)$. In the worst case, all the clique diameters can be different, and object instances of a feature in the clique instances can be all distinct. Thus the union operation cost of a feature is $O(K^2)$, and there are a total of N features.
3. The cost of computing feature weight for all the clique diameters, and computation of critical distance is $O(N \cdot K + \log K)$. This is due to finding weight for N features for K candidate distances and then a lookup of PI below the min_prev for K size list.

Therefore, the total time complexity of critical distance computation for a prevalent colocation in $O(N \cdot K^2)$.

If the total number of prevalent colocations in the result set is P and the running time of the join-less colocation mining is J , then the running time of *Range – CoMine* is $O(J + P \cdot N \cdot K^2)$.

The following equation shows the total cost function of J as per analysis given by Yoo *et al.* in [70]:

$$J(k) = T_{\text{star_neighborhoods}}(S) + J(2) + \sum_{k>2} J(k) \quad (3.6)$$

Here, S represents the input spatial dataset. $J(2)$ represents the cost for finding size 2 colocation patterns in the procedure. The following equation represents the costs of finding size k ($k > 2$) colocation patterns:

$$\begin{aligned} J(k) &= T_{\text{generate_candidates}}(P_{k-1}) + T_{\text{filter_star_instances}}(C_k) + T_{\text{filter_coarse_colocation}}(C_k) \\ &\quad + T_{\text{filter_clique_instances}}(C'_k) + T_{\text{filter_previous_colocation}}(C'_k) \\ &\approx T_{\text{filter_star_instances}}(C_k) + T_{\text{filter_clique_instances}}(C'_k) \end{aligned} \quad (3.7)$$

Here, P_{k-1} is a size $k-1$ prevalent colocation set. C_k is a size k candidate colocation set. Each C'_k is a size k candidate colocation set filtered by the coarse filtering process in the joinless algorithm [70]. The overall cost is expected to be slightly larger than the cost to generate the star neighborhood set, so $T_{\text{star_neighborhoods}}$ can be ignored. Similarly, $T_{\text{generate_candidates}}$, $T_{\text{filter_coarse_colocation}}$, and $T_{\text{filter_previous_colocation}}$ can be ignored when compared with other computation factors.

Therefore,

$$\begin{aligned}
J(k) &\approx T_{\text{filter_star_instances}}(C_k) + T_{\text{filter_clique_instances}}(C'_k) \\
&\approx |C_k| \times t_{\text{scan_cost}} + p_{ij} \times |C_k| \times t_{\text{lookup_cost}} \\
&\approx t_{\text{scan_cost}} + p_{ij} \times t_{\text{lookup_cost}}
\end{aligned} \tag{3.8}$$

Here, $t_{\text{scan_cost}}$ is the average cost to collect the star instances of a candidate colocation by scanning the materialized neighborhood set. p_{ij} is the coarse filtering ratio. $t_{\text{lookup_cost}}$ is the average cost to check the cliqueness of its star instances per colocation.

3.6 Experimental Analysis

In this section, we study the performance of our proposed approach *Range – CoMine* by comparing it with two baseline approaches on synthetic and real data sets. Both the baseline approaches, namely the Naïve Approach and *RangeInc – Mining*, are an adaptation of the popular join-less colocation pattern mining algorithm [70] for the case of the range query.

All the algorithms are implemented in C/C++ and run on a computer with a 2.30 GHz Processor with 96GB memory.

3.6.1 Experimental Settings

The experiments are conducted on two real and eleven synthetic datasets. The datasets are chosen for the study as per the practice in the research community [9, 24]. Table 3.7 shows the characteristics of real datasets. The real data sets have points of interest in China and New York, respectively.

Synthetic Data Generation Process

Now, we describe the process for generating synthetic datasets, which follows the methodology outlined in previous research studies [54][24].

Step 1: Feature Set Generation

The first step in generating synthetic data is the creation of feature sets. We begin by generating $N_{\text{co_loc}}$ subsets of features systematically. Each subset is constructed by sampling a specific number of features randomly. The number of features in each subset follows a Poisson distribution with a mean denoted as λ_1 . To enhance the complexity of our synthetic dataset, we then create m_{overlap} maximal co-location patterns or feature sets from each subset of features. This is achieved by adding one additional random feature to each subset, making the resulting feature sets representative of real-world data patterns.

Step 2: Instance Construction

Once we have defined our feature sets, the next step is to generate instances for these sets. For each maximal co-location pattern created in Step 1, we construct a certain number of instances. The number of instances is determined using a Poisson distribution with a mean denoted as λ_2 . These instances are the fundamental building blocks of our synthetic dataset. Each instance consists of creating m_{clump} multiple objects corresponding to the features within the co-location pattern. These objects are then placed inside random grid cells. The grid cells are characterized by their size, denoted as $d' \times d'$, and are part of a spatial frame with dimensions of $D \times D$. This step simulates the spatial distribution of objects related to the features in our dataset.

Step 3: Noise Injection

To make our synthetic dataset more realistic, we introduce noise into the data. This step involves the generation of noisy features and instances. First, we generate $(r_{\text{noisy_feature}} \times n_1)$ noisy features, where n_1 corresponds to the number of non-noisy features generated in Step 1. These noisy features introduce variability and randomness into the dataset. Next, we construct $(r_{\text{noisy_num}} \times n_2)$ noisy instances based on the noisy features, following a process similar to that employed for non-noisy instances in Step 2. Each noisy instance is placed within a random grid cell. The number of noisy instances, denoted as n_2 , is equal to the number of non-noisy instances generated in Step 2.

We set $N_{\text{co_loc}}$, λ_1 , m_{overlap} , λ_2 , m_{clump} , d' , D , $r_{\text{noisy_feature}}$ and $r_{\text{noisy_num}}$ as per the values given in Table 3.8. The characteristics of synthetic datasets are shown in Table 3.9. By default, we use *Synthetic_Data_1* to study the performance.

Data Set	Number of Objects	Number of Features
Real_Data_1	20558	13
Real_Data_2	182334	36

Table 3.7: Relevant information about Real Data Sets

Parameters: Parameters like minimum prevalence (min_prev), Range $[d1, d2]$ are set as per Table 3.10. The default values used are shown in bold.

3.6.2 Effect of Threshold Settings

Effect of min_prev threshold: This section analyses the effect of min_prev on the runtime of algorithms. Experiments conducted on *Synthetic_Data_1* are shown in Figure 3.6(a). In this experiment, *Range – CoMine* takes significantly less time than both *RangeInc – Mining*, and Naïve approaches. Naïve takes more time because it computes star neighborhood, star instances, and clique instances etc. from scratch

Parameter	Meaning	Values
N_{co_loc}	Number of co-location	20
λ_1	Size of each co-location	5
$m_{overlap}$	Number of maximal co-locations generated by appending more features into co-locations	1, 5, 10 , 15
λ_2	The parameter used in Poisson distribution to construct the instances for each maximal co-location	40, 50 , 60, 70, 80
m_{clump}	Number of feature instances for each co-location in it's neighborhood	1 , 2, 3, 4, 5
D	Spatial framework size	10^6
d'	Size of grid cell	10
$r_{noisy_feature}$	Noise feature ratio	0.5
r_{noisy_num}	Noise instance ratio	0.5

Table 3.8: Synthetic data parameters and their values in experiments.

Data Set	Number of Objects	Number of Features	m_{clump}	$m_{overlap}$	λ_2
Synthetic_Data_1	9372	192	1	1	50
Synthetic_Data_2	94028	462	1	10	50
Synthetic_Data_3	46748	312	1	5	50
Synthetic_Data_4	282083	462	3	10	50
Synthetic_Data_5	470138	462	5	10	50
Synthetic_Data_6	142925	612	1	15	50
Synthetic_Data_7	191370	762	1	20	50
Synthetic_Data_8	75108	462	1	20	40
Synthetic_Data_9	113439	462	1	20	60
Synthetic_Data_10	132405	462	1	20	70
Synthetic_Data_11	151487	462	1	20	80

Table 3.9: Relevant information Synthetic Data Sets

Dataset	min_prev	$[d1, d2]$
Real_Data_1	[0.1, 0.3]	[100m,150m], [100m,200m], [100m,250m],[100m,300m], [100m,350m], [100m,400m]
Real_Data_2	[0.1, 0.6]	[100m,150m], [100m,200m], [100m,250m],[100m,300m], [100m,350m], [100m,400m]
Synthetic Datasets	[0.01, 0.06]	[1,2], [1,3], [1,4], [1,5], [1,6], [1,7]

Table 3.10: Parameter Settings for experiments

for each candidate distance. However, the *RangeInc – Mining* takes less time than the Naïve Approach approach because it reuses the clique instances computed for the previous candidate distance.

On the real datasets, in Figure 3.6(b) and Figure 3.6(c), we observe that with an increase in the *min_prev* there is a decrease in running time. This is due to the decrease in the number of candidate patterns. Further, we report only those results that complete in 10 hrs. As a result, in Figure 3.6(c), plots of Naïve Approach and *RangeInc – Mining* approach are not included as they do not complete the experiment within the time limit for the *Real_Data_2*. The *Range – CoMine* also took more than 10hrs at 0.1 and 0.2 due to a huge number of candidate patterns being generated at these thresholds.

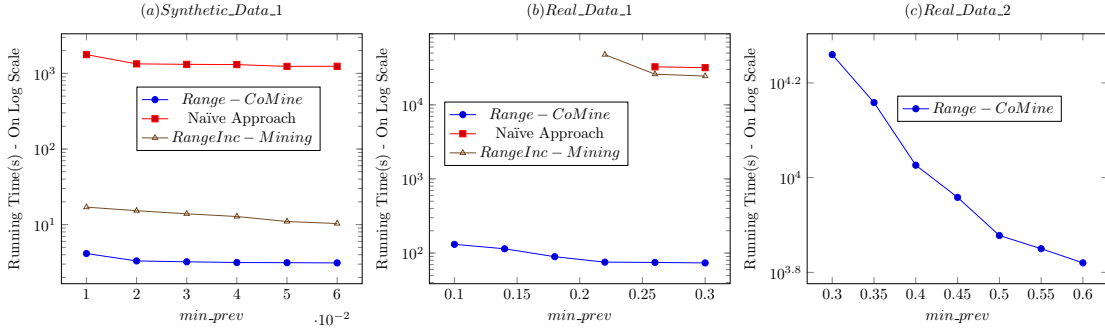
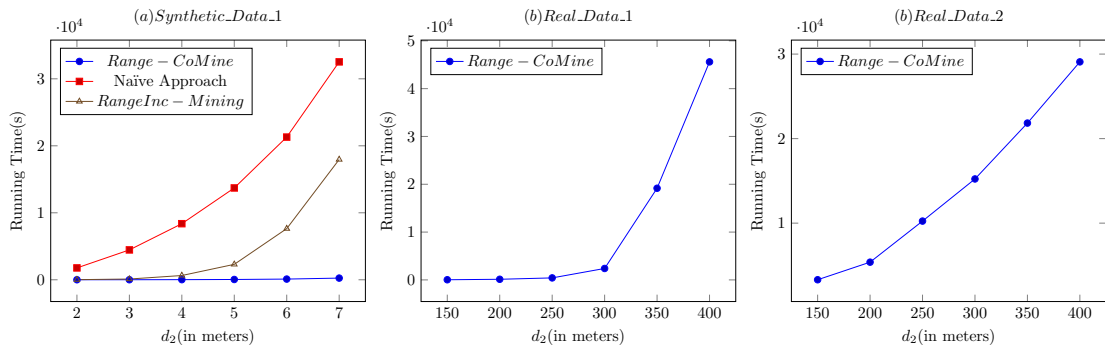
Figure 3.6: Performance with min_prev 

Figure 3.7: Performance with Range

Effect of Range threshold: Experiments conducted on *Synthetic_Data_1* are shown in Figure 3.7(a). Here, d_1 is set to 1 and d_2 is varied from 2 to 7 to increase the range interval. In this experiment, our approach takes very little time than both *RangeInc – Mining*, and Naïve approaches. The number of neighborhood objects increases with an increase in range, thus increasing the number of candidate distances. The Naïve approach takes more time than the other two approaches as it needs all the computations at each candidate distance. In comparison, the *RangeInc – Mining* approach takes less time than Naïve as it reuses previous iterations’ information at each iteration. *Range – CoMine* takes lesser time than the other two as it computes critical distances in one iteration itself. Experiments conducted on *Real_Data_1* and *Real_Data_2* are shown in Figure 3.7(b) and 3.7(c) respectively. Here R1 is set to 100m, and d_2 is varied from 200m to 700m. In *Real_Data_1* the run time increases linearly as we increase d_2 to 300m as there are very few objects that get added in the neighborhood. After 300 meters, the run time increases exponentially as it encounters many neighborhood objects, thus increasing the number of clique diameters it needs to validate the critical distance test. Plots on other approaches are missing because they ran for more than 10hrs without producing any results. Also, in the case of *Real_Data_2*, the run time increases exponentially as we increase the range. This is because neighborhood objects

start increasing from the beginning itself as we increase d_2 . Plots on other approaches are missing as they did not complete.

3.6.3 Effect on number of candidates

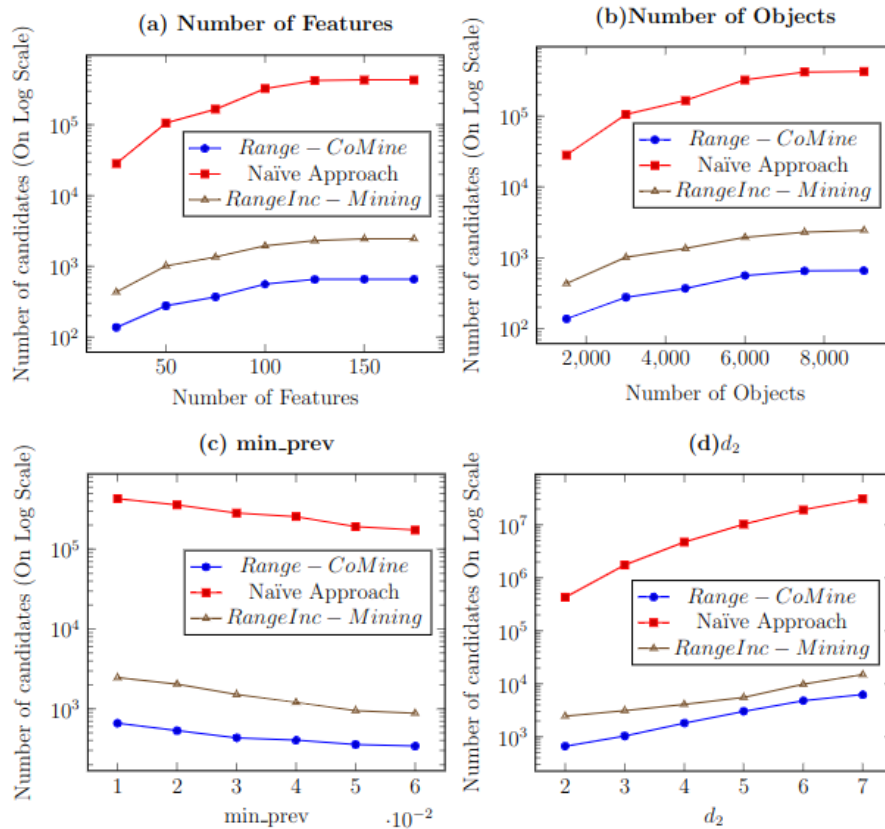


Figure 3.8: Effect on the number of candidates in *Synthetic_Data_1*

In this section, we analyze the number of candidates generated in *Range - CoMine*, *RangeInc - Mining* and Naïve approaches on *Synthetic_Data_1*. Experiments conducted by varying Number of Features, Number of Objects, `min_prev` and d_2 are shown in Figure 3.8(a), Figure 3.8(b), Figure 3.8(c), and Figure 3.8(d), respectively. Figure 3.8(a) shows that in the Naïve approach the number of candidates increases exponentially with the increase in the number of features. This is clearly due to the candidate enumeration made at each and every distance value. In Figure 3.8(a), *RangeInc - Mining* plot shows that the number of candidates generated increases linearly because candidates are generated in the first iteration and in addition to that only colocations created in the previous iteration are added as candidates for the next iteration. Figure 3.8(a)

shows that *Range – CoMine* generates very less candidates compared to the other two approaches, this is because the candidates are generated only once. Similarly, Figure 3.8(b) also shows that *Range – CoMine* generates very less candidates compared to the other two approaches. Figure 3.8(c) shows that the number of candidates decreases with the increase in min_prev value for all the three techniques. This is because, in a pattern-growing technique, as a lesser number of patterns qualify to be a prevalent collocation on increasing the minimum prevalence threshold, they generate lesser candidates for the next step. Figure 3.8(d) shows that the number of candidates increases with the increase in d_2 value. This is because, on increasing the distance, more objects are added to the neighborhood. However, *Range – CoMine* generates fewer candidates compared to the other two approaches as it enumerates candidates only once.

3.6.4 Scalability Experiments

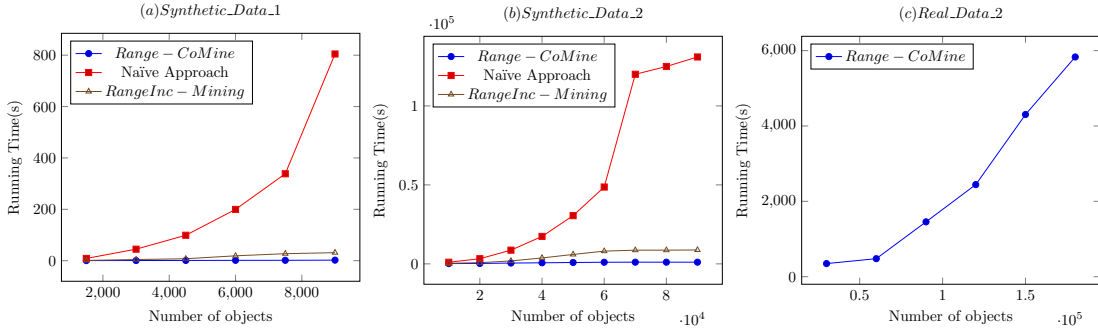


Figure 3.9: Scalability with number of objects

For scalability analysis, we have selected objects from the respective dataset of a particular size through random sampling with a uniform probability distribution. We have followed a similar method for feature selection.

Scalability test with a number of objects: In this section, we study the scalability tests by varying the number of objects. Experiments conducted on *Synthetic_Data_1* and *Synthetic_Data_2* are shown in Figure 3.9(a) and Figure 3.9(b). In these experiments *Range – CoMine* takes very less time than both *RangeInc – Mining* and Naïve approaches. As we increase the number of objects the running time of Naïve Approach increases exponentially. This is because the number of candidate instances increases exponentially with the increase in the number of objects. Experiments conducted on *Real_Data_2* are shown in Figure 3.9(c). The experiments show that *Range – CoMine* is scalable, whereas other approaches do not scale well. The runtime of *Range – CoMine* increases linearly.

Scalability test with a number of features: In this section, we do scalability tests by varying the number of features. Experiments conducted on *Synthetic_Data_1* and *Synthetic_Data_2* are shown in Figure 3.10(a) and Figure 3.10(b). In these experi-

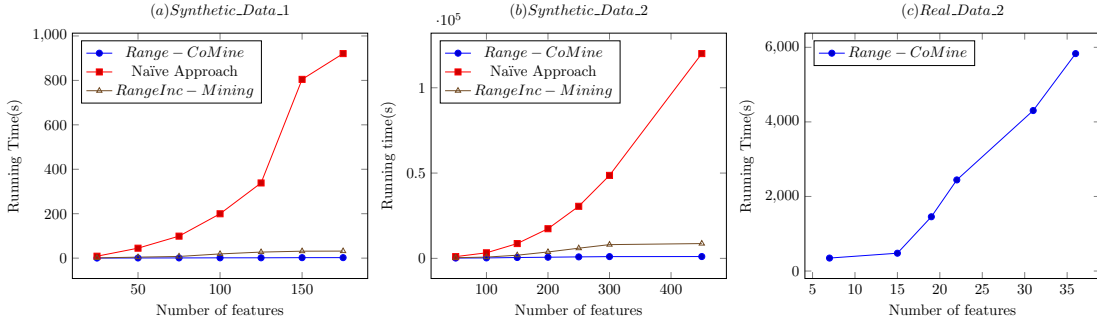


Figure 3.10: Scalability with number of features

ments, our approach takes very little time than both *RangeInc – Mining* and Naïve approaches. Our approach is scalable as its running time increases linearly. Whereas the Naïve approach is not scalable. As we increase the number of features, the running time of Naïve approach increases exponentially. We chose to perform the same experiment on *Real_Data_2* as it has a large number of objects with a large number of features. The results are shown in Figure 3.10(c). In these experiments, as the number of features increases, the running time of other algorithms increases exponentially. Plots on other approaches are missing because they ran for more than 10hrs without producing any results.

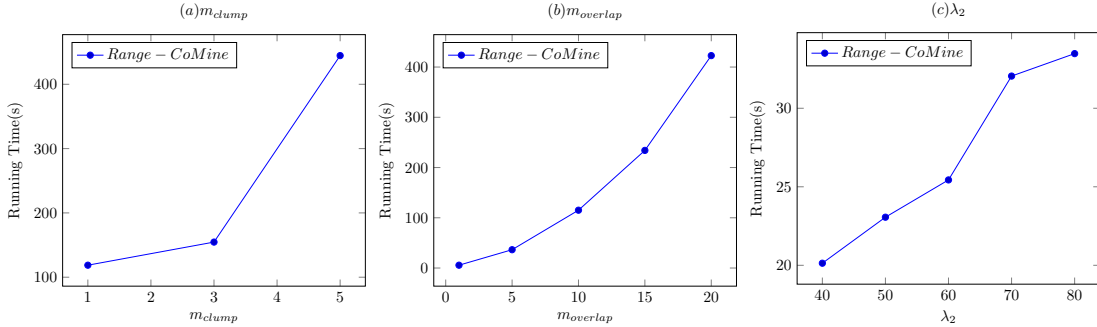


Figure 3.11: Effect of density

Effect of density: In this section, we analyze the behavior of *Range – CoMine* by varying the m_{clump} and $m_{overlap}$ values on synthetic datasets. We set min_prev as 0.0001 and the Range threshold as [1,2]. Experiments conducted on m_{clump} , $m_{overlap}$ and λ_2 are shown in Figure 3.11(a), Figure 3.11(b) and Figure 3.11(c), respectively. We observe that the running time increases with an increase in m_{clump} value because the total number of objects increases with an increase in m_{clump} value. Also, the number of candidate instances increases with the increase in m_{clump} and $m_{overlap}$, respectively. Similarly, the running time increases with an increase in $m_{overlap}$ value because the total number of features increases with an increase in $m_{overlap}$ value. Running time

also increases with the increase in λ_2 because it increases the average number of clique instances and data would involve more objects.

To summarize, Naïve Approach performs worst in all the experiments. Both the Naïve approach and *RangeInc – Mining* approaches are not scalable. Even in the case of dense data, *Real_Data_2*, and other synthetic datasets with different densities, the *Range – CoMine* approach is scalable and performs reasonably with the change in `min_prev` threshold and distance range.

3.6.5 Explainability of real data set output:

When we run *Range – CoMine* on *Real_Data_1* in the range [100m, 400m] we discover that pattern {Social Services, Religious Institution} gets introduced at 351m and pattern {Residential, Social Services} gets introduced at 396m. Similarly, When we run *Range – CoMine* on *Real_Data_2* in the range [100m, 300m] we discover that pattern {bar, restaurant} gets introduced at 71m and pattern {recreation, supermarket} gets introduced at 203m. So, using our algorithm we are able to say which pattern gets introduced at what distance. This way we not only address the challenge of choosing a distance threshold but also assist a user analyst in forming a valid hypothesis.

3.7 Conclusion

In this work, we have introduced the problem of prevalent colocation mining over a distance range query. To our knowledge, this is the first work that discusses a computation framework for distance range queries over colocation mining. First, we have discussed a Naïve approach that gives an elementary working framework of the problem. We have discussed the computational challenges in the Naïve approach and improved it to propose an incremental range colocation mining approach *RangeInc – Mining*. We have defined the critical distance of colocation patterns that helped in the computation and in justifying the correctness of these approaches. By exploring the structural properties of colocations, we have proposed an efficient technique for computing critical distances. This helped in proposing our efficient single pass range query algorithm, namely *Range – CoMine*, to solve the problem. We experimentally demonstrate the performance of these algorithms with various experiments using both real-world and synthetic data sets. We observe that *Range – CoMine* is more scalable compared to the other two approaches. As a future extension to this problem, we would like to explore the range query challenges in colocation mining with additional overheads such as range over minimum prevalence thresholds, dynamic object instances, and computation over road networks.

Colocation Subgraph Pattern Mining

In colocation pattern mining, the focus of enumeration remains on clique instances on a single neighbourhood graph. However, in real-life scenarios, one may like to relax the clique constraint to a general subgraph over a set of neighbourhood graphs along with feature instances and their relationships having different importance. This scenario becomes very complex and would need distributed algorithms to solve it. One possible approach to solve the above pattern mining problem can be adapting existing Frequent Subgraph Mining (FSM) algorithms. We extend the FSM to include the notion of the importance of feature instances and their relationships using the Map-Reduce distributed platform and design an efficient solution to pattern mining.

This work got published in the following conference paper:

A. Khare, V. Goyal, **Srikanth Baride**, S. K. Prasad, M. McDermott and D. Shah, "Distributed Algorithm for High-Utility Subgraph Pattern Mining Over Big Data Platforms," 2017 IEEE 24th International Conference on High Performance Computing (HiPC), Jaipur, 2017, pp. 263-272, doi: 10.1109/HiPC.2017.00038.

4.1 Introduction

One approach to colocation pattern mining is to view it as a subgraph pattern mining task. In this approach, the objects are represented as nodes in a graph, and the edges between the nodes represent the spatial or temporal relationships between the objects. The task then becomes one of finding subgraphs that occur frequently in the input data. When considering distance labels, the edges in the graph are assigned labels that represent the distance between the objects. For example, in a spatial context, the labels might represent the physical distance between two objects. In a temporal context, the labels might represent the time lag between two events. To account for distance labels

in subgraph pattern mining, we need specialized algorithms that take into account both the structure of the graph and the labels on the edges.

Frequent subgraph mining (FSM) discovers subgraph patterns that occur in a given graph database with a frequency more than a user-defined threshold. This problem has been well attended by researchers and practitioners due to its various applications in the area of biological networks, social networks and web data analytics. For instance, frequent subgraphs can compactly represent information in social networks. Likewise in the bioinformatics domain, common structures in protein-protein networks can be used to predict properties of a molecular compound or functionality of a protein. Other general applications include indexing of a graph database and search of dense subnetworks in a large graph.

Generally, graphs have weights associated with nodes and edges that capture different semantics related to the importance of nodes or affinities between nodes. For example, in a gene expression network, the weight on an edge may denote the connection strength of the pair of genes. Nevertheless, FSM does not consider the relative importance of participating nodes and edges in a pattern. It considers the presence or absence of nodes and edges with certain labels. Thus, FSM may report insignificant patterns that miss relevant nodes and edges. To address this issue, an approach to consider the importance of participating edges while computing the support of a pattern has been proposed. Along the same lines, we define the problem of high-utility subgraph pattern mining (WSM) where the importance of each edge/node is considered to compute the relevance (utility) of a pattern. In WSM, only the patterns having a utility value higher than the user-defined threshold are reported. The motivation behind considering the relative importance is that it would allow discovery of those patterns that include infrequent nodes or edges but that have high importance value in the network database.

Like the FSM, WSM is computationally expensive in nature. Moreover, the WSM needs to deal with the significant issue of not holding the *anti-monotone property*. This property states that if a pattern of size K is not relevant then no superset pattern of size $K + 1$ or more will be relevant which restricts the search space in FSM and other standard pattern mining methods. A naive solution to address this problem is to enumerate all possible patterns exhaustively and compute their utility. The search space for this naive solution would be very large (exponential number of subgraph patterns of a graph), rendering it computationally prohibitive on all but small sample sizes. One of the challenges in the WSM is to devise an efficient methodology to prune the search space while capturing all high-utility patterns with no misses. We introduce a function to estimate the upper-bound utility of a pattern that satisfies the anti-monotonic property and helps in restricting the search space.

The search space for FSM is generated from a graph database that is usually comprised of many moderately sized graphs and as previously mentioned for WSM this search space, naively, can quickly grow to levels that are computationally prohibitive. Hence, mining patterns from a large database is a herculean task on a single machine. Thus, use of distributed frameworks like Map-Reduce has been advocated recently for

FSM. However, any general method to solve pattern mining is iterative in nature and the big data platform Apache Spark has proved to be the best fit for iterative jobs [1]. We therefore design and study a Spark-based solution for WSM. We observe that straightforward extension of Map-Reduce based solution for FSM to Spark does not offer any advantage. Therefore we designed several optimization strategies such as i) utilizing the Bloom Filter for restricting the search space exploration, ii) avoiding sending of the graph database information in a pattern object, and iii) avoiding sending of the pattern embeddings in a pattern object. We implement these strategies in Spark and find them to be very effective.

Our main contributions are as follows:

- We define a high-utility subgraph pattern mining problem that incorporates the relative importance of edges and nodes while mining subgraph patterns.
- We develop a distributed solution for WSM and FSM on big-data platforms such as Apache Spark and Map-Reduce.
- We design effective strategies for WSM to minimize data communication, prune non-candidates to restrict the search space and avoid unnecessary computations. Our optimization strategies are also applicable for FSM.
- We conduct an experimental study to observe that our optimization strategies are efficient and reduce the computational time by a factor of at least 10.

4.2 Preliminaries, Problem Definition, and Mathematical formulation

A graph is defined as $G = (V, E)$ where V is a set of vertices and E is a set of edges, in particular, $E \subseteq V \times V$. L is a labeling function for vertices and edges and w is a weight function associating a weight with each edge $e \in E$. $L(v)$ denotes the label of the vertex v , $L(e)$ denotes the label of the edge e , and $w(e)$ is the weight of the edge e . We represent each subgraph pattern that is to be mined from the graph database as $P = (V_P, E_P)$, and term this as pattern or pattern object.

Definition 4.1 (Subgraph isomorphism). A pattern P is said to be subgraph isomorphic to a graph $G = (V, E)$, denoted as $P \subseteq G$, if there exists an injective function $\psi : V_P \rightarrow V$ such that:

- 1) $\forall v \in V_P$, $L(v) = L(\psi(v))$, and
- 2) $\forall e = (v_r, v_s) \in E_P$, $(\psi(v_r), \psi(v_s)) \in E$ and $L(v_r, v_s) = L(\psi(v_r), \psi(v_s))$.

Definition 4.2 (Subgraph matching). An isomorphic subgraph in G corresponding to the vertices in pattern graph P , denoted as $\phi(P, G) = \{\psi(v_1), \psi(v_2), \dots, \psi(v_p)\}$, is called a matching of the pattern P in the graph G .

A pattern P may have more than one matching in an input graph. We denote the set of all matchings of P in G by $\Phi(P, G) = \{\phi_1(P, G), \phi_2(P, G), \dots, \phi_t(P, G)\}$, and k^{th} matching of pattern P in G by $\phi_k(P, G)$.

Let $S = \{G_1, G_2, G_3, \dots, G_n\}$ denote the graph database of n objects. Let $G_i = (V_i, E_i)$ be the graph representing the i^{th} object.

Definition 4.3 (Support set). Given a graph database $S = \{G_1, G_2, G_3, \dots, G_n\}$ and a pattern P , the support set of the pattern P , denoted by $SupSet(P)$, is defined as the set of graph objects that have at least one matching for P . Hence, $SupSet(P) = \{G_i : P \subseteq G_i, G_i \in S\}$.

The size of $SupSet(P)$ (cardinality) defines the support of a pattern P in the set S , denoted by P_S . The task of pattern mining needs to associate a measure for relevancy to each pattern object so that patterns with a relevancy score greater than some threshold are extracted. Support of a pattern is one of the relevancy measures used in the literature [4, 13, 26, 32]. Besides these relevancy measures, other measures to define the relevancy of a pattern P are also proposed [27, 33, 55, 57]. We use the relevance measure similar to the one proposed in [33]. Utility is a measure which has been discussed recently in the pattern mining community. This is where the relevancy of a pattern considers the utility of the pattern object's constituting components [12, 69]. For example, the utility of nodes and edges, represented as edge and node weights in a graph, can be used to define the relevancy score of a pattern.

Definition 4.4 (Utility). Let E_ϕ be the set of edges in a matching m for pattern P in a graph G , i.e., $m \in \phi(P, G)$, the utility of the matching m , denoted by $u(m, P, G)$, is defined as

$$u(m, P, G) = \sum_{e=(v_r, v_s) \in P} w(\psi(v_r), \psi(v_s))$$

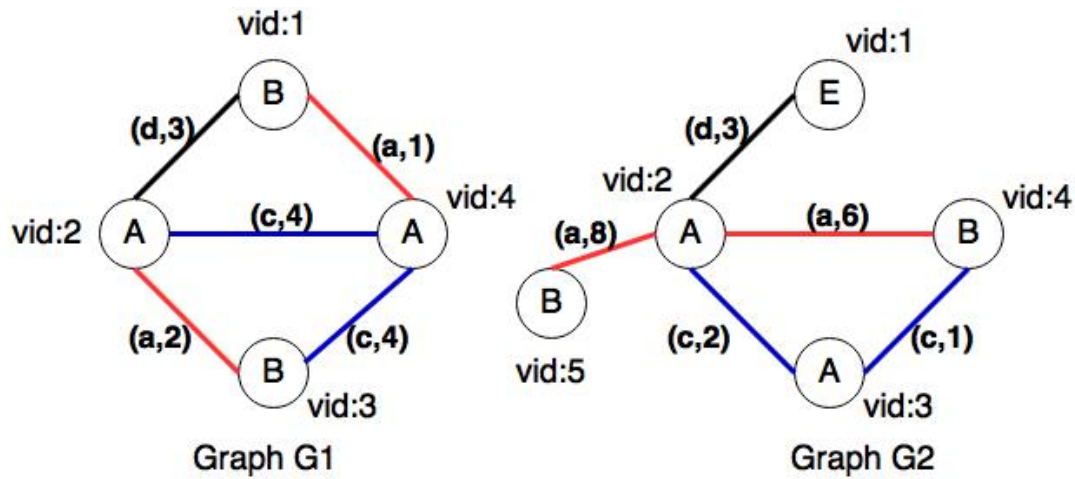
Definition 4.5 (Weight of the maximum matching). Utility of a Pattern P in a graph object G having a matching set Φ , denoted as $u(P, G)$, is defined as the weight of the maximum matching of P in G . Hence, $u(P, G) = \max_{\phi_j \in \Phi} (u(\phi_j, P, G))$, $u(P, G) = 0$ when Φ is empty.

Using this definition, we can compute the utilities of all patterns of the graph database S .

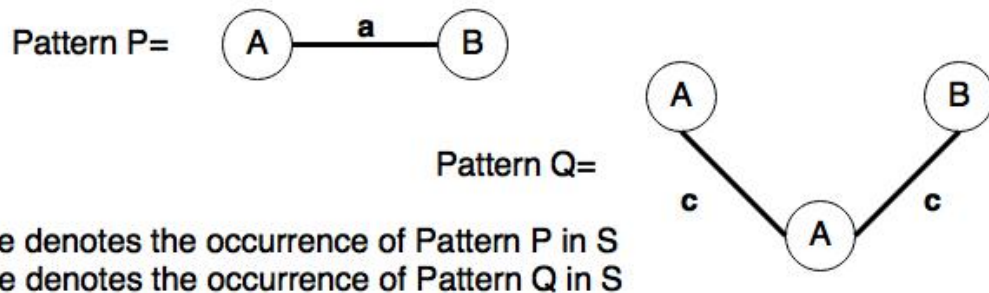
Definition 4.6 (Sum of utility of the pattern). Utility of a pattern P in a database S , denoted by $u(P, S)$, is defined as the sum of utility of the pattern P in each graph in the database. Hence,

$$u(P, S) = \sum_{G \in S} u(P, G)$$

For example, in Fig. 4.1, the graph database S contains only two graphs- G_1 and G_2 . The pattern P has two matchings in the graphs G_1 and G_2 . The utility value for the pattern P is computed as the following:



Let set of graphs $S=\{G1,G2\}$,
 The tuple above edge is (edge-label,edge-weight).



Red line denotes the occurrence of Pattern P in S
 Blue line denotes the occurrence of Pattern Q in S

Figure 4.1: An example showing various weights for pattern P

$E_{\phi_1(P,G_1)} = \{(1, 4, a, 1)\}$, $u(\phi_1, P, G_1) = 1$, and
 $E_{\phi_2(P,G_1)} = \{(2, 3, a, 2)\}$, $u(\phi_2, P, G_1) = 2$. Similarly,
 $E_{\phi_1(P,G_2)} = \{(2, 5, a, 8)\}$, $u(\phi_1, P, G_2) = 8$, and
 $E_{\phi_2(P,G_2)} = \{(2, 4, a, 6)\}$, $u(\phi_2, P, G_2) = 6$.
 $u(P, G_1) = \max(u(\phi_1, P, G_1), u(\phi_2, P, G_1)) = \max(1, 2) = 2$
 $u(P, G_2) = \max(u(\phi_1, P, G_2), u(\phi_2, P, G_2)) = \max(8, 6) = 8$ Finally, the utility of the
 Pattern P in the graph database S is $u(P, S) = u(P, G_1) + u(P, G_2) = 2 + 8 = 10$.

Problem 4.7. Weighted Subgraph pattern Mining (WSM): Given a database of graph objects $S = \{G_1, G_2, G_3, \dots, G_n\}$ and a utility threshold parameter τ , determine all patterns P such that each pattern's utility over the database is at least τ . In other words, find the set R such that $R = \{P | u(P, S) \geq \tau\}$.

Similar to frequent subgraph mining problem, WSM determines relevant patterns from a database. However, the anti-monotonicity property does not hold in WSM.

Lemma 4.8. *Anti-monotonic property in WSM does not hold.*

Proof. The proof is evident by the following example: Consider the example in Fig. 4.1. Let the utility threshold parameter τ be 10. Now consider Pattern Q in 4.1. Its utility in $S = \{G_1, G_2\}$ is 11. Also, $u(Q, S) = u(Q, G_1) + u(Q, G_2) = 8 + 3 = 11$. As $u(Q, S) \geq \tau$, Q is relevant. Suppose Q_1 is a subgraph of Q consisting of only one edge with vertex A. i.e., $Q_1 = (\{(1, A), (2, A)\}, \{(1, 2, c)\})$. Utility of pattern Q_1 in S is 6 as $u(Q_1, S) = 4 + 2 = 6$. Pattern Q_1 is not relevant however its superset pattern Q is relevant. This shows that anti-monotonic property is not satisfied in WSM. \square

4.3 Related Work

The work for frequent graph mining can be broadly categorized as following: Frequent subgraph mining over a single graph, and Frequent subgraph mining over a database of graphs.

The algorithms for mining a single graph include DISTGRAPH [55], SUBDUE [19], SiGraM [31] and GraMi [13]. SUBDUE, SiGraM and GraMi are sequential algorithms, whereas DISTGRAPH is a distributed algorithm implemented using MPI framework. As the dataset is a single graph, the challenges for the distributed algorithms are different. These include partitioning of a large graph into multiple partitions to have a balanced load at each node and computing of subgraph patterns without introducing any false negatives. Our focus is to mine a given graph database that can be partitioned easily. None of these studies consider utility or importance of edges while mining patterns and therefore can't be adopted for high-utility graph pattern mining due to not holding of downward closure property in the case of weighted pattern mining.

Frequent subgraph mining over graph databases is a celebrated field of study. The early methods for FSM include AGM [26] and FSG [32]. Similar to the Apriori algorithm [4], the candidates are generated in a level-wise manner using a breadth-first search approach. Later proposed methods like gSpan [65] and FFSM [22] use depth-first search exploration and canonical ordering of patterns. Contrary to the Apriori algorithm, no candidates are generated. However, AGM, FSG, gSpan and FFSM rely on a single node computation and are inefficient to process massive datasets. Recently some Map-Reduce based approaches are also proposed [8, 18, 39, 45]. These approaches have a low performance due to large data communication or a large number of duplicate candidates. Furthermore, the approaches proposed so far are limited in the notion of relevance and do not consider the individual importance of edges and nodes. For large scale data sets, Lin et al. [40] have devised a statistical method to predict whether a subgraph is globally frequent.

The weighted version of the graph has been approached with different relevance measures proposed. Jiang et al. introduced weighted gSpan influenced by weighted association rule mining [57] along with edge-weighted graphs [27]. Lee et al. proposed a method to mine weighted frequent subgraphs with weight and support affinities [33]. These approaches consider the node and edge importance while determining relevant patterns but fall short on scalability due to their sequential nature. Our inspiration for high-utility pattern mining originates from the approach of high-utility pattern mining for transactions which considers the relevance of each transaction item while determining relevant item-set patterns [12, 69]. However, these approaches cannot be directly applied as graph is a more complex structure as compared to an itemset.

4.4 Proposed Methods

As WSM does not follow anti-monotonic property, the search space cannot be pruned easily while searching for high utility patterns. Therefore we define an upper-bound utility function, OWU, for a pattern which returns the upper bound estimate of the maximum over the utility value of the pattern and its superset patterns. This function helps us to decide if a pattern or any of its superset patterns is relevant. The pattern search is based on classical pattern growth method that iteratively grows the pattern base, initially small in size. Patterns that have their upper bound utility value less than the user threshold are called non-candidates. The non-candidate patterns should not be grown and hence are pruned. The leftover patterns, also called *potential patterns*, are grown in a systematic way to avoid generation of redundant patterns for further utility estimation and growth. This iterative process for generation of candidates, pruning the non-candidates, and growing of potential candidates continues till there exists some potential pattern to grow after pruning non-candidates. In each iteration, the actual utility value of all candidate patterns is computed and those with high utility value are written to an output file.

Definition 4.9 (Upper-bound utility). Upper-bound utility of a pattern P over a set of graph objects S , represented as $OWU(P, S)$, is defined as the sum of utility of graph objects that have at least one matching for pattern P . Hence,

$$OWU(P, S) = \sum_{P \subseteq G_i} u(G_i, G_i)$$

Lemma 4.10. $OWU(P, S)$ satisfies anti-monotonicity.

Proof. Consider a pattern P with $OWU(P, S) \geq \tau$. Let P_{sub} be a subgraph of pattern P . It can be trivially seen that $OWU(P_{sub}) \geq \tau$. As $SupSet(P_{sub}) \geq SupSet(P)$, $OWU(P_{sub}) \geq OWU(P) \geq \tau$. Thus, P_{sub} will be a potential subgraph pattern. \square

4.4.1 Baseline Algorithm

The algorithm for WSM is based on the pattern growth method [4]. At the beginning of an iteration i , the algorithm processes all the potential patterns of size $i - 1$, denoted by F_{i-1} , as its input. The size here equals to the number of edges in a pattern. At the end of the i^{th} iteration, the algorithm generates all the relevant patterns of size i , denoted by R_i . It computes the upper bound utility value of grown patterns and prunes non-candidates. The remaining patterns are the set of potential patterns, i.e., (F_i) , which is used for $i + 1^{th}$ iteration if not empty. Algorithm 3 gives a basic outline of this method.

Procedure 3 Mining Relevant Subgraphs

```

1: procedure BASIC-WSM( $S, \tau$ )
2:    $k = 1$ , Populate  $F_k$ 
3:   while  $F_k \neq \phi$  do
4:      $C_{k+1} = \text{Candidate-generation}(F_k, S)$ 
5:     for  $c \in C_{k+1}$  do
6:       if  $\text{isomorphic-check}(c) = \text{true}$  and  $OWU(c, S) \geq \tau$  then
7:          $c.\text{upperUtility} = OWU(c, S)$ 
8:          $F_{k+1} = F_{k+1} \cup \{c\}$ 
9:       end if
10:      if  $u(c, S) \geq \tau$  then
11:         $c.\text{datasetUtility} = u(c, S)$ 
12:         $R_{k+1} = R_{k+1} \cup \{c\}$ 
13:      end if
14:    end for
15:     $k = k + 1$ 
16:  end while
17:  Return  $\cup_{i=1 \dots k-1} R_i$ 
18: end procedure

```

The main steps of the algorithm are: i) candidate generation, ii) upper bound utility estimation, and iii) actual utility computation.

Candidate Generation

Good candidate generation is a crucial step in minimizing the generation of redundant patterns for any pattern mining algorithm. The candidate generation step in the WSM-Algorithm uses an established and well-known strategy given in [8] that extends vertices on the rightmost path only. A depth-first traversal of the pattern P assigns each node a traversal-id or a time stamp when that node is visited. The rightmost path is defined as the path from the lowest traversal-id vertex to highest traversal-id vertex. The vertex with highest traversal-id is called the rightmost vertex. A pattern can have either forward

extension or backward extension. Forward extension results in addition of a new vertex as well as an edge to P , whereas backward extension adds only an edge.

- Forward extension applies to all the vertices in the rightmost path. The order of the extension is from the rightmost vertex to the vertex with the smallest traversal id in the rightmost path.
- Backward extension applies only to the rightmost vertex of Pattern P .

However, after using rightmost path extension there may still exist duplicate candidate patterns which need to be pruned. This duplicacy may exist due to the fact that a pattern may be generated from different subgraph patterns. One way to identify duplicate candidate patterns is to perform isomorphism checks on the candidates which is a computationally expensive task. Another way to identify duplicates is to associate a canonical code with each candidate pattern so that all duplicates would have the same code. We use min-dfs-code in WSM that takes the order of edges that results into the min-dfs-code. Out of all the isomorphic candidates, the candidate pattern whose insertion order of the edges is same as the min-dfs-code order is considered and selected as a validate candidate, and rest of all duplicate candidate patterns are removed.

Calculating Upper Bound Weight

To calculate the upper bound utility of a pattern we need to check the occurrence of the pattern in each graph object via subgraph isomorphism. However, subgraph isomorphism is an NP-complete problem. Inspired by [8] we instead use a data structure called occurrence list to store the locations (in terms of vertex-id) of a pattern in each graph of the graph database. We also create a Hash-Map called GraphWeights for storing graph utility of graphs with each pattern. The occurrence list and the GraphWeights together are used to compute the upper bound utility of a pattern.

Calculating Utility of a pattern

Calculation of $u(P, S)$ for a pattern P requires information about its parent graphs from the database and the information of the matchings of the pattern within the graph. To accomplish this we use a hashmap called Esets which stores a list of edge-sets corresponding to each matching for each graph-id. Each edge-set is a list of edges with each edge being represented as a 3-tuple (vertex-id1, vertex-id2, edge-weight). The utility of the pattern over the database can be computed using Esets easily as it contains information of all the matchings of a pattern in each graph.

Consider G_1, G_2 in Fig 4.1 and a pattern $P = \{(A, B, c), (A, C, a)\}$. The pattern has two matchings in G_1 and G_2 each. Therefore it has two edge-sets corresponding to the graph G_1 , i.e. $\{(1,2,21), (1,5,43)\}$ and $\{(2,3,56), (2,8,1)\}$. The edge-sets for P in G_2 can be obtained similarly.

4.4.2 Distributed WSM Algorithm

The distributed algorithm for WSM is challenging for many reasons. First, to compute upper-bound utility estimates and actual-utility of a pattern access to the whole database is needed, however, the database is partitioned and distributed to multiple nodes. In effect, utility values for any pattern would be available only with respect to the partition of database available at a node where the pattern gets generated. It therefore becomes difficult to make potentially global pruning related decisions with the utility information remaining local to a single partition. Second, performing aggregation in any distributed environment like Map-Reduce and Spark is always an expensive operation as it requires data communication between the nodes. On the other hand, since the search space is very large aggregation of the utility information over the whole database cannot be delayed for a long time. Hence, the distributed WSM algorithm runs in an iterative fashion as with basic WSM. In each iteration, the candidate patterns are generated from each of the partitions and aggregated. Each of the candidate patterns would have the upper-bound as well as actual-utility values for its generating partition associated with it. We define the utility values associated with a pattern originated from a particular partition as local upper-bound utility and local utility. Algorithm 4 represents the pseudocode for distributed-WSM algorithm to generate candidate pattern in an iteration and Algorithm 5 represents the pseudo code for the aggregation.

Algorithm Description

Conceptually, each node runs an independent WSM task over a partition of the graph dataset which is $1/k_{th}$ of the size of $|S|$. The argument F_p^k represents the set of size- k potential patterns having their upper-bound utility greater than the threshold in a specific partition S_p .

The mapper reads each pattern (say x) from the Hadoop Distributed File Systems (HDFS) as a key-value pair. The key is the min-dfs-code of the pattern ($x.min\text{-dfs-code}$) and the value is a pattern object ($x.obj$). Here the term “object” stands for its usual meaning from the object oriented programming. Then the mapper generates all possible candidates of size $k+1$ (Line 2 Algorithm 4) by extending each of the patterns in F_p^k . For each of the generated candidates (say, c), the mapper performs isomorphism checking to confirm whether c is generated from a valid generation path. In other words, it tests whether c passes the min-dfs-code based isomorphism test (Line 4 Algorithm 4). For successful candidates, the mapper calculates its upper-bound utility and actual-utility (locally) over the graphs in the partition S_p of the graph database. The $p.obj$ contains a hashmap $graphWeights_p$ that stores weights of graphs ($u(G_i, G_i)$) in the partition p and can be used to compute upper-bound utility. $p.obj$ also contains an occurrence list data structure that stores the graph-id of the graphs in partition p to which the given pattern is subgraph isomorphic. It also contains the $edgeSet_p$ that stores multiple matchings of the candidate pattern c in the graphs of partition p to which c is subgraph

Procedure 4 Mapper Of Distributed WSM

```

1: procedure MAP( $F_p^k \langle p.min\text{-dfs-code}, p.obj \rangle$ )
2:    $C_{k+1} = \text{candidate-generation}(F_p^k)$ 
3:   for  $c \in C_{k+1}$  do
4:     if  $\text{isomorphism-check}(c) = \text{true}$  then
5:        $c.upperUtility = OWU(c, p.obj)$ 
6:        $c.datasetUtility = u(c, p.obj)$ 
7:        $c.obj = \text{update-Object}(c, p.obj)$ 
8:       if  $c.upperUtility > 0$  then
9:          $\text{emit}(c.min\text{-dfs-code}, c.obj)$ 
10:      end if
11:    end if
12:  end for
13: end procedure

```

isomorphic. If the upper bound weight for candidate pattern c is greater than zero then a key-value pair of $(c.min\text{-dfs-code}, c.obj)$ is emitted.

Procedure 5 Reducer Of Distributed WSM

```

1: procedure REDUCE( $c.min\text{-dfs-code}, [c.obj]$ )
2:   for  $obj \in c.obj$  do
3:      $TotalupperUtilityobj.upperUtility$ 
4:      $TotaldatasetUtilityobj.datasetUtility$ 
5:   end for
6:   if  $TotaldatasetUtility \geq \tau$  then
7:      $\text{write}(c.min\text{-dfs-code})$  To HDFS
8:   end if
9:   if  $TotalupperUtility \geq \tau$  then
10:    for  $obj \in c.obj$  do
11:       $\text{write}(c.min\text{-dfs-code}, obj)$  to HDFS
12:    end for
13:   end if
14: end procedure

```

Algorithm 5 represents the psuedocode of the reduce phase of Map-Reduce Job. The reducer receives a set of key-value pairs, where the key is the min-dfs-code of a pattern namely $c.min\text{-dfs-code}$ and the value is a list of $c.obj$'s constructed from all partitions where the pattern c has a non-zero upper-bound utility. Reducer then iterates (Line 2 Algorithm 5) over every $c.obj$ and calculates the aggregated (total) upper-bound utility. In the same scan over the $c.obj$, it also calculates the actual-utility of the pattern (Lines 3-4 Algorithm 5). If the total actual-utility of the pattern is greater than the threshold

(τ is the threshold) then the min-dfs-code of c is written to the HDFS (Line 6 Algorithm 3). If the total upper-bound utility is greater than the threshold the reducer writes appropriate key-value pairs in the HDFS for the mappers of the next iteration (Line 9 Algorithm 5). If the number of potential $k + 1$ size patterns is zero, execution of WSM-H is stopped.

Procedure 6 Spark Implementation Algorithm

```

1: inRDD<String>= Sparkcontext.readFiles()
2: cRDD<p.min-dfs-code,p.obj>= inRDD.flatMap(o:gen-one-edge-len-candidates(o))
3: potRDD<p.min-dfs-code,[c.obj]>= cRDD.groupByKey().filter(o : filPotPatns(o))
4: rRDD<p.min-dfs-code,[c.obj]>= potRDD.filter(o: FilterRelevantPatterns(o))
5: rRDD.writeToHdfs()
6: while true do
7:   iRDD<p.obj>= potRDD.flatMap(o: gen-input(o))
8:   cRDD = iRDD.flatMap(o : gen-cand-from-prev(o))
9:   pRDD = cRDD.groupByKey().filter(o : filPotPatns(o))
10:  rRDD = pRDD.filter(o: FilterRelevantPatterns(o))
11:  rRDD.writeToHdfs()
12:  if PotentialRDD.count() = 0 then
13:    break
14:  end if
15: end while

```

Algorithm 6 represents the implementation of Algorithm 3 in the Spark framework. The Spark implementation starts with the reading of the partition files (Line 1 Algorithm 6) and creates an RDD named inRDD. It then generates size one candidate patterns from inRDD (Line 2 Algorithm 6). Next, all the pattern objects having similar min-dfs-code (isomorphic patterns) are grouped together and the patterns with their upper bound utility greater than the threshold are retained in the potRDD (Line 3 Algorithm 6). The potRDD is then processed to get the relevant patterns of size one which are saved (Lines 4-5). We extend size one patterns to size-2 patterns, and size-2 to size-3 and so on until no further extension is possible (Lines 6-13 Algorithm 6).

4.5 Optimizations

4.5.1 Use of bloom filter for Pruning of Non-candidates

A bloom filter is a space efficient probabilistic data structure widely used in set-membership test. Bloom filters can render false positives but cannot render false negatives. Bloom filters can perform addition and membership-test of an element in $O(k)$ time where k is the number of hash functions. We use the bloom filter to avoid exploration of

non-candidates. We also employ novel strategy to tighten the upper-bound utility of a potential candidate pattern by using a bloom filter.

In particular, we first add two edge’s potential patterns generated in the 2nd iteration of the WSM algorithm to the bloom filter. Then, we use them to prune non-candidate patterns of larger sizes, as illustrated in Fig. 4.2.

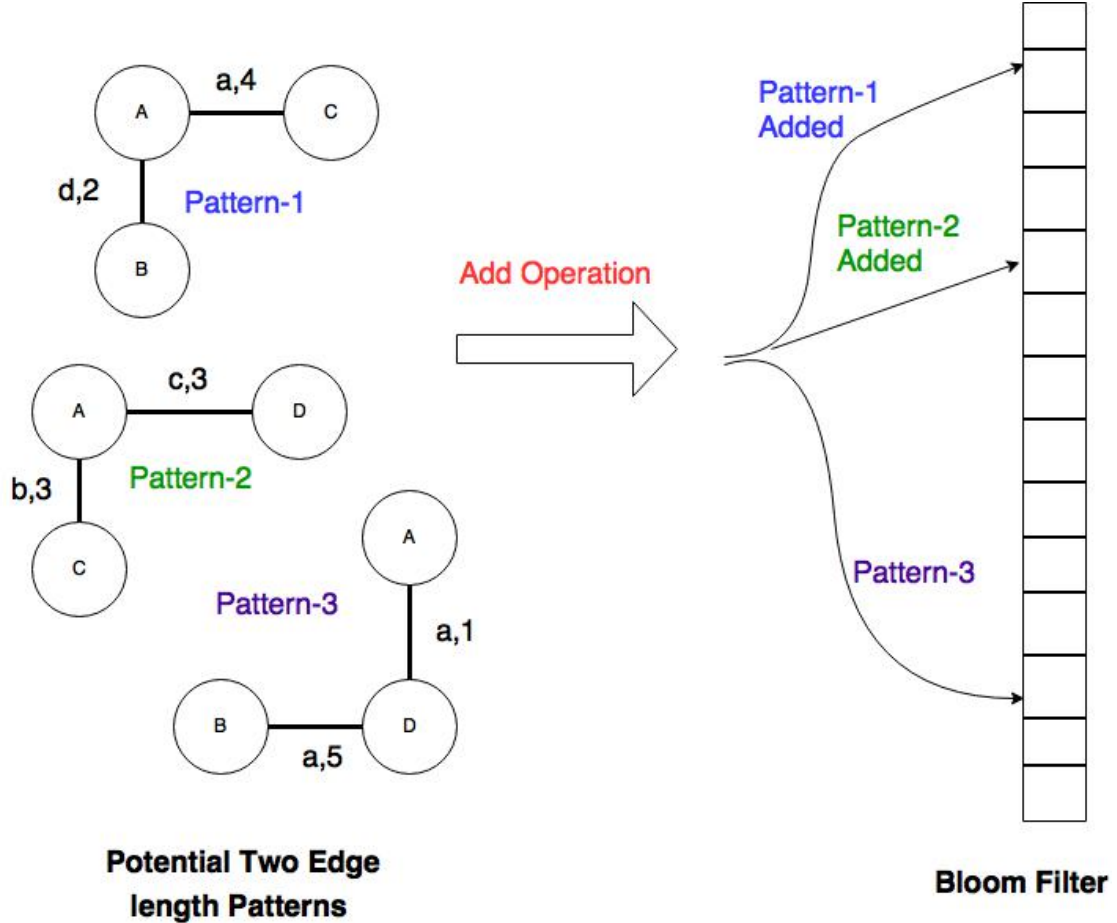


Figure 4.2: Addition of size-2 frequent patterns to the bloom filter

Whenever a potential pattern is grown by either forward extension or backward extension to generate a candidate pattern we perform a check for each pair of edges consisting of the extended edge and a neighboring edge in the bloom filter. If this test fails for a pair then the extended candidate pattern is identified as non-candidate and its utility computation is not performed.

For example in Fig. 4.3, Let P be a five-edge potential pattern. Suppose the vertex E (marked in red in Fig. 4.3) is selected for adding an edge to the pattern P . Suppose a candidate pattern $P_{candidate}$ is generated by a forward extension, and an edge with label g is added to P along with a vertex, labeled D . Dotted line shows the newly

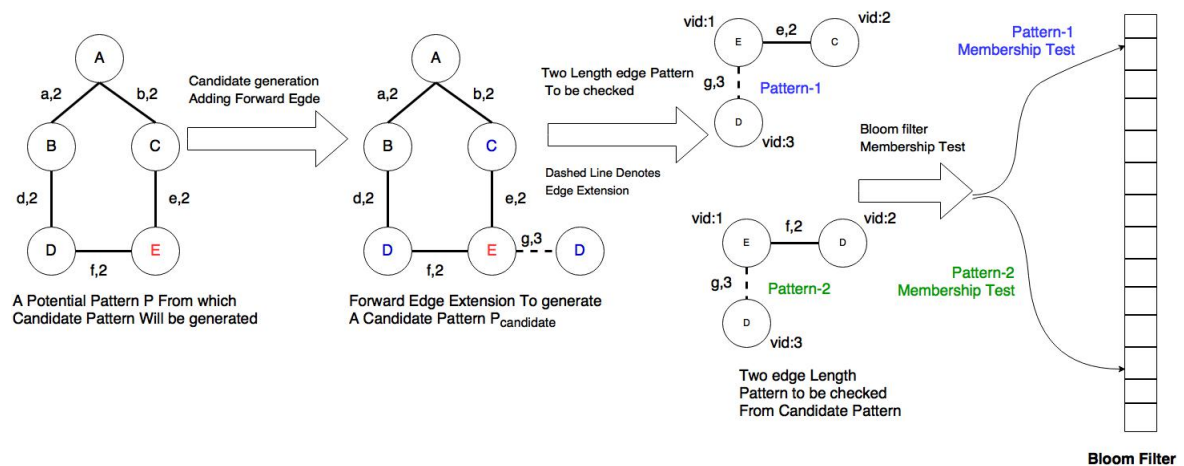


Figure 4.3: Illustration of membership test

added edge to the pattern P . Now, the neighbors of vertex E , namely C and D , are traversed and marked blue. Since vertex E has two neighbors, two two-edge patterns $P_1 = (\{(1, E), (2, C), (3, D)\}, \{(1, 2, e), (1, 3, g)\})$ and $P_2 = (\{(1, E), (2, D), (3, D)\}, \{(1, 2, f), (1, 3, g)\})$ are generated (representation according to $G(V, E)$). These two patterns P_1 and P_2 are checked to be present in the bloom filter. The utility of $P_{candidate}$ is calculated only if both of these patterns pass the membership test.

Lemma 4.11. *Bloom filter based pruning does not generate any false negative.*

Proof. Consider a potential candidate pattern P of size k which is extended to a candidate pattern Q of size $k + 1$ by adding an edge e . Let e_1, e_2, \dots, e_t be the adjacent edges. Suppose the pair (e_i, e) fails the bloom filter test, then it is an irrelevant edge. Thus, Q cannot be a potential high-utility pattern as per the anti-monotonic property of the upper-bound utility function. The pair (e_i, e) does not belong to the set of all potential high-utility pairs in the database, therefore no superset of the pair can be a potential high-utility pattern. The pattern Q contains this pair and hence it cannot be a potential high-utility pattern. \square

Tighter Upper-bound Utility

The upper-bound utility of a pattern is currently calculated as the sum of utilities of the graphs containing the pattern, which is a very loose upper-bound. A loose upper-bound of a pattern does not prune non-candidates effectively, and hence we propose a pioneering idea to have a tighter upper-bound utility using the bloom filter. Our approach is based on the observation that one can remove the utility of the irrelevant edges. These irrelevant edges could be identified while pruning using the bloom filter.

Reciting the WSM algorithm, each pattern object P generated from a partition S_p is associated with a hashmap of graph utilities of the partition S_p . Let Q be a pattern and H be a hashmap of graph utilities for graphs of Q . Suppose Q is extended to generate a candidate pattern Q_1 by adding a new edge e . If the bloom filter membership test of Q_1 fails then by Lemma 4.11, the edge e is an irrelevant edge for any extension of Q . Therefore, for every graph of Q having e , the utility of e can be subtracted. However, as the same edge can occur multiple times in a pattern and a graph, the utility value of a graph G_i containing Q is updated in the following way: Suppose the edge e occurs n times in Q , and m times in graph G_i . We update the utility of G_i by subtracting the sum of the utility value of $m - n$ edges with the least weights in G_i . This gives us a new hashmap for pattern Q that allows computation of tighter upper-bound utility value for all extensions of Q .

4.5.2 Schimmy Approach

We observe that the naive implementation of the distributed algorithm sends the complete partition information as an adjacency list data structure alongside each pattern. As a result, multiple communications of the graph database occur in each iteration. This is done for the following reasons: a) different partitions can generate the same pattern and one partition has many patterns. Due to this many-to-many relationship between patterns and partitions, each task performing map operation needs data from multiple partitions and b) if the partition information is associated within the pattern object, next candidates can be generated by accessing the pattern object data locally and the solution becomes straightforward.

Our approach is to give each partition a unique partition-id and associate it with each of its candidate patterns. This association is necessary because it gives information about the partition generating that pattern. This allows us to now know for a given pattern which partitions are needed for its extension. Whenever we need to extend the potential patterns generated from the previous iteration of the WSM-algorithm, we perform a natural join operation of these patterns with partition-ids to determine graph database partitions needed for each of the patterns. We observe that graph data pulling works better than sending the partition information along with each pattern. Due to many to many relationship between patterns and partitions, as the number of generated patterns increase it results into the graph database being sent multiple times in each iteration. This happens as the whole partition data is sent with each pattern in the case of naive implementation.

4.5.3 Lightweight Object Approach

With the Schimmy approach, pattern object does not have graph structure attached to it but it is still large as it contains information of matching and graph utility. The matching list grows larger and larger and potentially grows exponentially in size. We notice

that the min-dfs-code of a pattern gives us sufficient information to retrace the pattern generation path i.e., it takes k iterations of the DFS algorithms to generate the k -edge length pattern again. Recomputing the pattern saves us from sending the embeddings in the pattern object reducing the size of the pattern object and the communication further. Our results show that implementing this approach further reduces the running time of the algorithm.

4.6 Experimental Study

The performance of our approaches are evaluated on four standard real-life datasets. A description of these datasets is given in Table 4.1. These datasets have been used

Datasets	Number of graphs	avg # edges per graph
Yeast	79,601	22.8
P388	41,472	23.3
NCI-H23	40,353	28.6
OVCAR-8	40,516	28.1

Table 4.1: Statistics of datasets

by researchers earlier for FSM and do not have weights assigned to edges or nodes. Therefore, the weights on edges are generated synthetically with uniform probability distribution over the range from 1 to 3. To compare the performance of our approaches we port the state-of-the-art algorithm for FSM [8] to Apache Spark. We implement six different versions of WSM algorithm, namely, i) Hadoop-WSM, that is adaptation of Hadoop FSM for high-utility pattern mining, ii) Hadoop-WSM-Bloom, that extends Hadoop-WSM with bloom filter strategy, iii) Spark-WSM, that is the adaptation of Hadoop-WSM on Spark, iv) Spark-WSM-Bloom, that extends Spark-WSM with bloom filter based optimization and prunes non-candidates and revises the upper-bound utility estimate of patterns, v) Spark-WSM-Bloom-Schimmy, that extends Spark-WSM-Bloom and does not send graph database information in a pattern object, and vi) Spark-WSM-Bloom-Schimmy-Lightweight, that extends Spark-WSM-Bloom-Schimmy by not storing pattern embeddings in a pattern object but recomputes those in every iteration.

To show the effectiveness of our optimization strategies on FSM, we use the code shared by the authors of [8] and call it as Hadoop-FSM. As with WSM approaches, we adapt and implement FSM on Apache Spark to get Hadoop-FSM-Bloom, Spark-FSM, Spark-FSM-Bloom, Spark-FSM-Bloom-Schimmy, and Spark-FSM-Bloom-Schimmy-Light weight approaches.

For evaluating the comparative performance of our approaches we studied the effect of varying different parameters such as utility threshold, cluster size, and partition size

(number of graphs in a partition) on the running time and pruning of non-candidates. The parameters with their range as well as the default value are described in Table 4.2. All experiments are conducted on a heterogeneous Hadoop/Spark cluster with each node having cores ranging from 8-24 and RAM from 8-64 GB. The processor clock frequency ranges from 800-1866 MHz.

For running Hadoop-WSM algorithm, the job property in Hadoop named `mapred.task.timeout` was set as 7hrs (420 minutes). This is done to ensure that master node waits for sufficiently large amount of time for datanodes to respond. The default configuration of this setting was not enough for WSM algorithm to run on Hadoop. Another property to be set is Hadoop java heap space. We set it through the `HADOOP_HEAPSIZE` environment variable with value as 8192 MB. This is required for storing the partition’s information so that `java.Lang.outOfMemory` Exception is not thrown. In Spark configuration, the executor memory for Spark-WSM is kept as 12 GB and driver memory is kept as 20 GB.

Sr. No.	Parameter	Range	Default
1.	Utility Threshold	60%, 40%, 30%	40%
2.	Cluster Size	5,7,10,12,15	10
3.	Datasets	Yeast, P388, NCI-H23, OVCAR-8	Yeast
4.	Partition Size	100, 800, 1800 , 2800	800

Table 4.2: Parameters for experimental-study

We keep the number of mappers in Hadoop-framework equal to number of partitions and number of reducers are kept at 30% of the number of mappers. In Spark, the number of executors are kept equal to the number of partitions in the datasets. For all the experiments we use the bloom filter of size 50 bits with the false positive rate 0.02.

4.6.1 Effect of Utility Thresholds

Figure 4.4 shows the run time of six different approaches at different utility thresholds. Our approach Spark-WSM-Bloom-Schimmy-Lightweight (SWBSL) outperforms the naive approach Hadoop-WSM by at least an order on all the datasets. For some datasets, like NCI-H23 and OVCAR-8, our approach shows a run time improvement of 20 times at the threshold of 40%. Furthermore, the speedup increases with the decrease of utility thresholds. This is due to an increase in the number of patterns with a decrease in threshold which causes more data communication overhead in the case of Hadoop-WSM. The graphs also show the effectiveness of our optimization strategies in terms of reducing the run time. Each strategy contributes in reducing the run time for all the

datasets. We also study the effect of bloom filter qualitatively by measuring the statistics of non-candidate pruning both with and without bloom filter. Table 4.3 shows the statistics for different datasets. In datasets such as OVCAR-8 we observe non-candidate pruning rate as high as 82%.

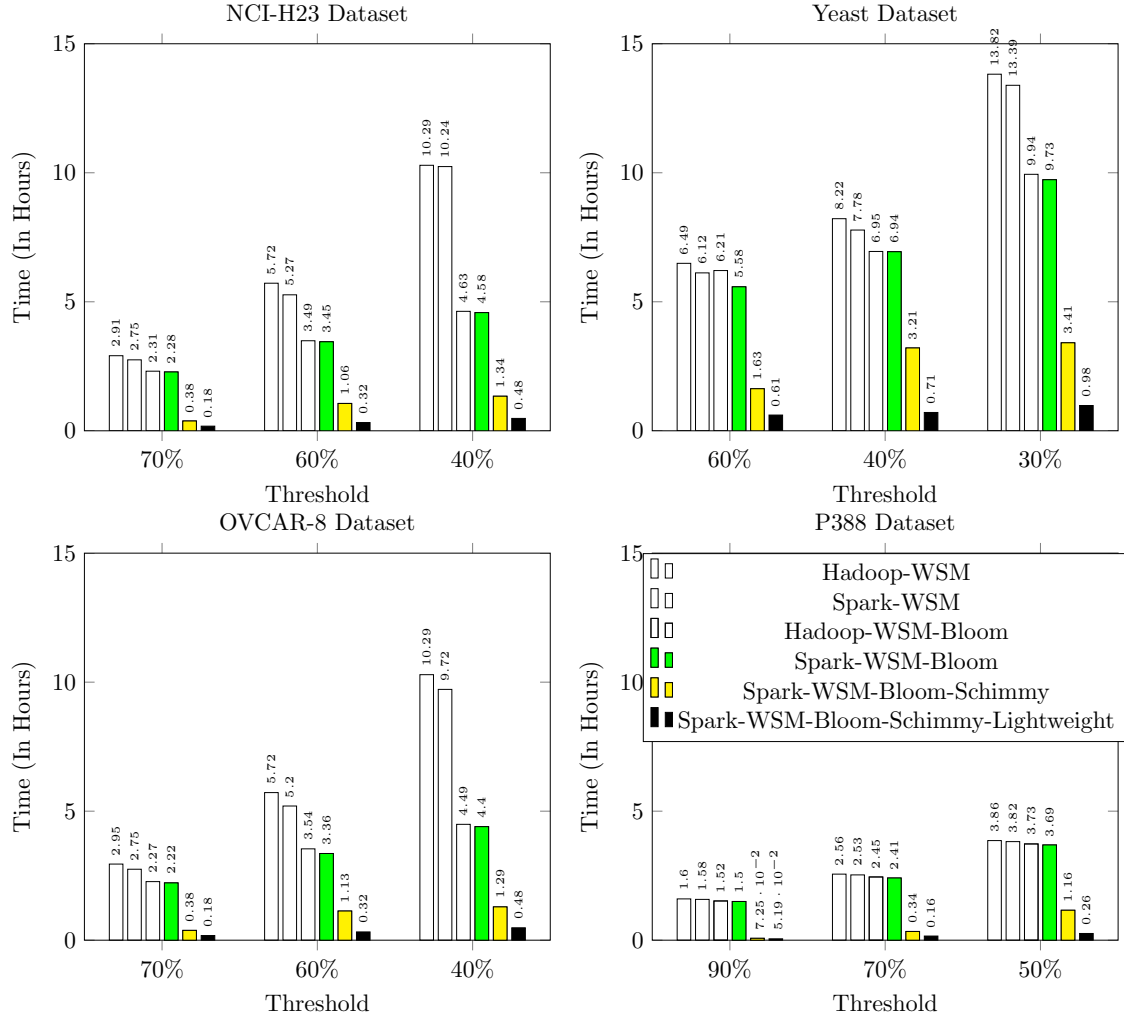


Figure 4.4: Effect of utility threshold on run time

4.6.2 Effect of Cluster Size

We study the effect of cluster size on Yeast, OVCAR-8, NCI-H23 and P388 Datasets with thresholds set at 40%, 40%, 40%, 50% respectively with the partition size set at 800 for the SWBSL approach. As can be seen in Figure 4.5, the run time decreases with the increase in the number of nodes, but this decrease in run time (or increase in speedup) is little. It can be due the fact that the increase in the number of nodes leads

Datasets	Thresholds	% Pruning
Yeast	60%	47.87
	40%	56.80
	30%	65.62
NCI-H23	70%	42.28
	60%	70.00
	40%	81.94
OVCAR-8	70%	41.84
	60%	70.38
	40%	81.94
P388	90%	6.17
	70%	26.85
	50%	47.04

Table 4.3: Effect of Bloom filter based pruning

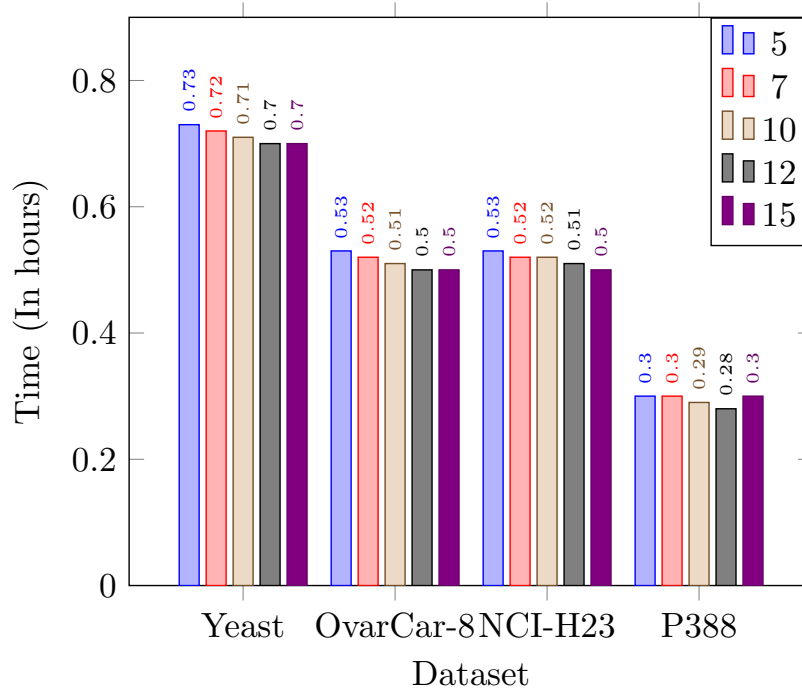


Figure 4.5: Effect of Cluster Size on Runtime

to more data communication overhead. Also, Spark-Framework runs multiple tasks on multiple threads. Therefore even in the case with less number of nodes the time taken is not linearly increasing as the nodes in the cluster are multi-core and reduce computation time due to multi-threading.

4.6.3 Effect of Partition Size

This experiment is performed against the Yeast Dataset having a threshold of 40% using the fully optimized algorithm (SWBSL) on a 10-node cluster configuration. Table 4.4 shows the result for this experiment. The partition size plays a significant role for the WSM algorithm. Having small partition sizes leads to more isomorphic patterns generated in total as the isomorphic check is performed locally to a partition by a task. This results in an increase in communication. This can be evident from Table 4.4 as the least percentage of pruning is when partition size is 100. On the other hand, if the partition size is kept high then the time taken to perform computation per partition increases. Also the data-structures for storing partition information require a large amount of memory. Therefore partition size is a crucial parameter to balance carefully. It can be seen that the time decreases with increase in partition size initially, but further increase of the partition size to 2800 increases the run time, hence corroborating the reasoning.

Partition Size	Time Taken	% Pruning
100	3269s	53.8%
800	2545s	56.8%
1800	2182s	58.9%
2800	2713s	58.9%

Table 4.4: Effect of partition size on run time and % pruning

4.6.4 Effect of Optimization Strategies in FSM

Figure 4.6: FSM: Effect of support threshold on run time

Figure 4.6 shows the effect of frequency threshold on the run time of different approaches. The graphs shows a similar trend for FSM as we see with WSM. All three optimization strategies are effective. In FSM, the bloom filter stores frequent patterns with two edges and is used for the pruning of non-candidates. The irrelevant patterns are identified as with WSM while generating candidate patterns. We observe that the Bloom filter is very effective for the case of FSM. The Schimmy and the Lightweight approaches are implemented in the same way as WSM. They help in lowering the data communication overhead and hence decrease in the run time.

4.7 Conclusion

We defined and presented novel distributed approaches for mining high-utility subgraph patterns from a graph database. To address the issue of utility measure being not anti-monotonic we define a function to get the upper-bound utility of a subgraph pattern that allows the systematic search of high-utility subgraph patterns and enables reduction of the search space. We give different optimization strategies, namely, use of bloom filters to prune the non-candidate patterns as well as to tighten the upper-bound utility estimate, a Schimmy design to reduce the communication of graph data between the nodes, and Removal of embeddings of a pattern from the pattern object to reduce the data communication. Our solution is flexible since it relies on primitive operations, such as map, filter and group-by, available on almost all distributed platforms.

We show that a straightforward porting of the state-of-the-art Map-Reduce solution to Apache Spark does not reduce the computation time for FSM. It quickly overshoots its demand of main memory and results into flushing of memory data to disk frequently. Our solution is efficient as demonstrated by the experimental study. However, the upper-bound utility estimate of a pattern is still very loose. Our future road-map includes plans to explore new mechanisms to tighten the upper-bound utility of a pattern.

Mining Co-location Patterns on Dynamic Data

This chapter is based on the following submitted journal paper:

Srikanth Baride, Anuj S. Saxena, and Vikram Goyal. "Mining Co-location Patterns on Dynamic Data." *Data Mining and Knowledge Discovery*, [Submitted].

5.1 Introduction

In applications where data changes with time, i.e., new events become available and the old events lapse, the co-location patterns also change. For example- for event organizers, in planning a new event in the vicinity of the co-located events of types musical evenings, stand-up performances, social gatherings, business meetings, food festivals, etc., updated co-location patterns from the recent data will be required to ensure better turnaround. Similarly, in planning daily police patrol routes, law enforcement agencies required updated crime locations from the latest crime data [2]. Thus, the mining task for dynamic data is exploratory and requires an analyst to explore different subsets of data corresponding to different time intervals.

The traditional co-location mining techniques are defined for static data and do not provide mechanisms to accommodate changes in the datasets. Thus, the algorithms would have to run afresh on the updated dataset. This would result in a reduction in the productivity of an analyst in addition to wastage of energy and less effective use of computing resources. Further, for applications that require online updates of co-location information, the re-computation may not be feasible due to high computational time requirements.

In this work, we are motivated to develop a computational framework that can efficiently update the co-location patterns from the previous state, i.e., the previously computed patterns, the old data state, and the changes that occurred in the data. In literature, various measures have been defined to quantify the support for co-location

patterns, such as construction-based [50], partition-based [54], enumeration-based [24], and fraction-based [9]. Fraction score addresses the issue of overcounting and undercounting of support measures in overlapped participating instances [9], and therefore, it is considered the best-suited support measure for finding co-location patterns. We intend to develop a framework for updating fraction score based co-location patterns for dynamic data. Some of the earlier studies discussed co-locations mining in evolving databases using the participation index as a support measure [3, 5, 10, 16, 21, 44, 46, 64]. To the best of our knowledge, this is the first work that discusses fraction score-based co-location pattern mining in dynamic data.

The fraction score measure gives weightage to an object based on its overlapping with different participating instances of candidate patterns. It depends on the topology of the participating objects, and therefore, the addition or deletion of objects in close proximity to the participating objects may change the score of participating objects as well as their own scores. As analyzed in subsequent sections, these changes in the fraction scores occur not only due to changes in the neighboring objects (aka., *1-nbd*) but also due to changes in the neighborhood of the neighboring objects (aka., *2-nbd*). Thus multiple score updates for affected participating objects may be required for updating the fraction score of a candidate pattern. Moreover, the same objects may be participating in different candidate patterns and may get an update call several times corresponding to different objects that are added or deleted. Also, changes in fraction score for object addition or deletion are not monotonic in nature which prevents an early termination while updating scores. These facts pose severe computational challenges in updating the fraction score from the previous state.

In this work, we address the problem of updating co-location patterns for dynamic data using a temporal-window framework, as in Figure 5.1. In the temporal-window framework, the data that changes with time is realized at a fixed time gap, say after every 15 minutes, every 1 hour, etc. Between any two consecutive time instances, say previous time (t) and current time ($t+1$), the data at time $t+1$ (denoted W^{t+1}) is updated from the previous data (denoted W^t) by adding objects O^a and deleting objects O^d that are added and deleted during the time period t to $t+1$. The co-location patterns of the current window W^{t+1} (aka., current co-locations) are desired to be updated from that of the previous window, instead of recomputing them from scratch. For efficiently mining current co-location patterns, it is required to compute the support of only those patterns that are likely to be either included in or deleted from the co-location patterns of the previous state, called candidate co-location. An efficient method for updating the support measure of candidate patterns should make only necessary updates in the fraction score whereby reusing the unchanged information. This requires identifying candidate objects for the current window W^{t+1} for which the score is likely to change, by using the past state (i.e., the previous data W^t and the previous co-location patterns) and the changes that occur in the data.

Techniques for mining co-location patterns in static data cannot be applied directly to dynamic data that requires the following updates: 1) The neighborhood relationships

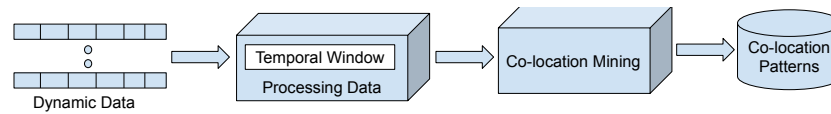


Figure 5.1: Framework for Mining Co-locations from Dynamic Data

may change as soon as some data arrives or is removed, hence there is a need to find the updated neighborhoods; 2) Using the updated neighborhood, there is need to identify the previous co-locations that disappear, new co-location patterns that appear, and those which remains unchanged. The cost of computing co-locations using classical mining algorithms becomes high as they do not reuse information from the previous state and find co-locations for the updated data from scratch. We address these computational overheads by proposing an efficient technique for updating co-locations. The main contributions of this work are:

1. A computational framework is proposed that can update the fraction score for the dynamic data in an online fashion. The proposed method is efficient as it only updates scores that are changed due to the addition or deletion of objects and reuse previous unchanged fraction score values.
2. An efficient algorithm is proposed for updating the fraction scores by avoiding multiple updates for the same score due to overlapping instances in the neighborhood sets of added and deleted objects. A lazy computation approach is proposed for the fraction score updating to avoid possibly over-computing the score due to overlapping instances.
3. An efficient algorithm named UpFS is designed for mining co-locations in the current window based on the proposed fraction score update technique. Three variants of the proposed algorithm are implemented that vary in how the neighborhood information is stored. They are (1) computing neighborhood information from scratch, (2) neighborhood information in the form of a list, (3) neighborhood information using a hash map. Experimental results show that the designed algorithm is more efficient than the baseline approach.

5.2 Related work

Various support measures have been proposed for mining co-location patterns in the last decade. The most studied measures includes, partition-based [54], construction-based [50], enumeration-based [24], and fraction-based [9]. These proposed measures have some advantages over others; however, they are not devoid of limitations. In partitioning-based co-location mining, the geographical region is divided into small grids, and the aggregate of the number of co-location patterns in various grids is used as support. Though the partitioned-based approach is easy to apply for mining frequent

co-locations, it has an obvious limitation of missing the count of those instances which overlap between different grid cells. Construction-based methods find instances of a label set using heuristics such as Voronoi partitioning, and the instance count of a feature set that is put together heuristically is used as support. A limitation of this approach is that a heuristic may not provide an optimal grouping of instances, which may result in undercounting the support of patterns. An enumeration-based approach uses the number of row instances as support where objects are said to form a row instance if they are in each other's neighborhood [79]. The enumeration-based approach can overcount co-location frequency because of overlapping instances. The recently proposed fraction-based approach [9] assigns fractions to objects based on the overlap of instances of the potential co-location patterns and takes aggregation of these fractions as support. It is considered as the best-suited support measure for finding co-location patterns as it addresses the issue of overcounting and undercounting. It is shown that the supports of the fraction-based approach are nearest to the ground truth compared to the support measures of the other mentioned approaches [9].

5.2.1 Classical Colocation Mining

Shekhar *et al.* introduced the concept of the Co-location pattern mining and proposed a Co-location Miner algorithm to mine co-locations using an event centric model [54]. Huang *et al.* extended this work with a join based algorithm to mine co-location patterns using participation index. The proposed join based algorithm was able to reduced the search space by exploiting the spatial autocorrelation properties of the spatial data [24]. Yoo *et al.* initially came up with a partial join approach to improve the efficiency of join based co-location mining algorithm [72] and then further improved it with a joinless technique using an instance-lookup scheme [70]. Mehta *et al.* gave a new perspective for mining co-locations using a distributed graph database [48]. Chan *et al.* developed an apriori like algorithm for mining co-location patterns based on fraction score and gave a filter and verification approach to decide whether an object is part of a candidate co-location or not [9].

5.2.2 Incremental Colocation Mining

There are also studies on mining co-locations in incremental databases. He *et al.* proposed the ICMP algorithm for incremental maintenance of discovered spatial co-location patterns when a new object is added [16]. Then, an algorithm named EUCOLOC for efficiently mining co-location patterns in evolving spatial databases was proposed by Yoo *et al.* [3]. EUCOLOC relies on a border concept to avoid unnecessary candidate generation and it is more efficient compared to ICMP. However, these two algorithms only consider adding new points and cannot handle point deletion. Later, Lu *et al.* designed an algorithm for incremental mining of prevalent co-locations (IMPCA) to handle both new and deleted points and also gave a pruning strategy to further increase efficiency

[44]. Lee *et al.* proposed the incremental topology miner (Inc_TMiner) algorithm to incrementally update topological patterns in spatial-temporal databases [34]. It performs database projections and searches for patterns using a depth-first search. Wang *et al.* proposed an algorithm named UUOC (Utility Update of Co-locations) to incrementally mine high utility co-locations in a database by considering both addition and deletion of data points [64]. To efficiently mine co-locations in large data Andrzejewski *et al.* proposed a parallel approach to mining co-locations using an iCPI-Tree structure [5]. An algorithm named INC-MGPUCCPM is proposed which takes the advantage of the processing power of GPUs to mine co-locations when the database is updated with new data points. Wang *et al.* proposed incremental fuzzy participation index for measuring the prevalence of the changed co-location in the updated data sets, and design the algorithm of incremental mining of prevalent co-location patterns based on FNR (the IMPCP-FNR algorithm). [61] Chang *et al.* proposed a new approach to rearrange the neighborhood relations in order to use less storage to store data information and also can avoid generating the non-incremental candidate instances [10].

5.2.3 Dynamic Co-location Mining

On the other hand to handle the dynamic nature of spatial objects, Qian *et al.* proposed an algorithm for mining co-locations with dynamic neighborhood constraints [51], which treats the co-location mining problem as an optimization problem and solves it using a greedy approach. Hu *et al.* proposed a concept of dynamic spatial co-location pattern to consider the dynamic relationships among the spatial features [21]. The idea here is to mine only maximal co-locations, which can derive all prevalent co-locations. Ma *et al.* proposed a two-step framework to discover evolving spatial co-location patterns [46]. In this framework, an extend-and-evaluate scheme is proposed to form evolving spatial co-location by selecting appropriate evolvers from top-k spatial co-location patterns at each time slot.

But none of these algorithms consider the fraction score as a support measure. This is the first attempt to analyze dynamic co-location pattern mining using fraction score as a support measure.

5.3 Preliminaries and Problem Formalization

For applications in which objects change with time, i.e., new objects become available and the old objects lapse, the co-location information also changes. A periodic update of the co-location patterns for such dynamic data is required. Most of the existing techniques for co-location pattern mining do not provide support for updating patterns for dynamic data. The re-computation of patterns may not be feasible for applications that require online updates due to high computational time requirements. In this work,

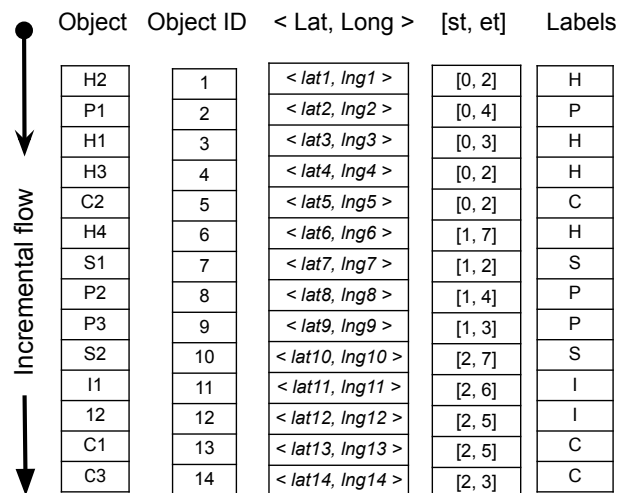
we are motivated to develop a computational framework that can efficiently update the co-location patterns for dynamic data.

We first define the terminology and notations used in our modeling. For an object, we denote its type by a fixed label depicting its feature. For example, objects M1, M2, M3, etc., of the type ‘Musical Events’ spread across a region, can be denoted by a label M. The collection of all the labels that are considered for computation is denoted by a set \mathcal{L} . The task of co-location mining for dynamic data is to find those subsets of \mathcal{L} (called labelset or pattern) whose object instances are in close proximity with sufficiently high frequency at any given time.

For mining co-location patterns, we process dynamic data at a fixed time gap using a temporal-window framework as discussed in the next Section.

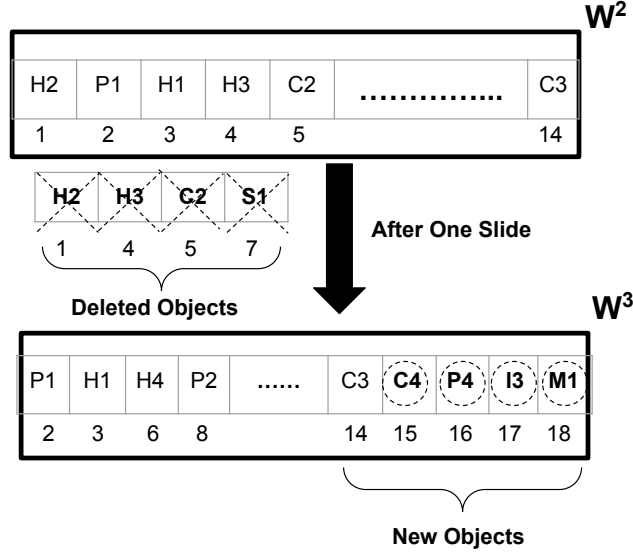
5.3.1 Dynamic data update framework

Initially, all the objects that are available are stored in (data) window W^0 at a time zero. As time progresses, we update the window W^1 at time $t = 1$ (i.e., after the first time-gap), W^2 at time $t = 2$ (i.e., after the second time-gap), and so on. Thus we consider time to be discrete, i.e., $t = 0, 1, 2, 3, \dots$, and the window W^t denotes active objects at any given time t . As time progresses from t to $t + 1$, we keep a record of the objects that are deleted (O^d) and added (O^a) during this period. This record facilitates the updating of window W^{t+1} at time $t + 1$. The window W^{t+1} now contains the active objects as follows: $(W^t \setminus O^d) \cup O^a$. The objects that are added to and deleted from the window W^t at time $t + 1$ are called the changed objects. We denote the cardinality of the changed object by λ .



Object	Object ID	< Lat, Long >	[st, et]	Labels
H2	1	< lat1, lng1 >	[0, 2]	H
P1	2	< lat2, lng2 >	[0, 4]	P
H1	3	< lat3, lng3 >	[0, 3]	H
H3	4	< lat4, lng4 >	[0, 2]	H
C2	5	< lat5, lng5 >	[0, 2]	C
H4	6	< lat6, lng6 >	[1, 7]	H
S1	7	< lat7, lng7 >	[1, 2]	S
P2	8	< lat8, lng8 >	[1, 4]	P
P3	9	< lat9, lng9 >	[1, 3]	P
S2	10	< lat10, lng10 >	[2, 7]	S
I1	11	< lat11, lng11 >	[2, 6]	I
I2	12	< lat12, lng12 >	[2, 5]	I
C1	13	< lat13, lng13 >	[2, 5]	C
C3	14	< lat14, lng14 >	[2, 3]	C

Figure 5.2: Snapshot of dynamic spatial data

Figure 5.3: Temporal moment from W^2 to W^3

In Figure 5.2, we present a snapshot of the dynamic data in which a spatial object o consists of object ID, spatial coordinate $\langle latitude, longitude \rangle$, event duration, and a label. Object IDs are assigned incrementally as distinct integers based on their arrival time. Event duration, denoted by a pair $[st, et]$, tells the start time and end time of the event. The absolute times are converted into discrete time using the time at $t = 0$ and the time-gap. Thus an event with the time interval $[2, 5]$ is first available at time $t = 2$ and remains available until $t = 5$. For our example, we have considered labels as Health Camps (H), Musical Performances (P), Cultural Events (C), Social Gatherings (S), and Industrial Exhibitions (I). Figure 5.3 illustrates the proposed temporal-window framework over the data of Figure 5.2. For demonstration purposes, we have shown a temporal-window from W^2 (at time $t = 2$) to W^3 (at time $t = 3$). In the window W^2 , there are 14 objects where objects with IDs 1-5 get added at time $t = 0$, objects with IDs 6-9 get added at time $t = 1$, and objects with IDs 10-14 get added at time $t = 2$. For ease of explanation, we have not deleted any object till time $t = 2$. The objects with IDs 1, 4, 5 and 7 are having end times $et = 2$. At the time $t = 3$, these four objects get deleted whereas the new objects $C4$, $P4$, $I3$, and $M1$ that appear in between $t = 2$ and $t = 3$ get added in W^3 at time $t = 3$. In the same way, objects get added and deleted to update the window as time progress.

5.3.2 Overview of Fraction-Score

In literature, various measures, such as fraction score [9], participation score [54], etc., are defined to ascertain a labelset to be a co-location pattern. In this work, we have

used fraction score to decide on co-location patterns, as discussed in the next Section. The reason for giving preference to the fraction score over others is that the fraction score avoids overcounting overlapped instances of a pattern by assigning fraction value to them. Thus, the mined patterns are more accurate and closer to the ground truth [9].

The central concept behind this approach is to consider each group not as a whole unit of prevalence but as a fractional unit, where the fraction value is calculated by distributing the contribution of an object across all the row instances involving that object.

In other words, instead of treating each group as a complete and independent entity, this method breaks down the prevalence of a group into smaller fractions, where each fraction represents a portion of the group’s prevalence associated with the object in question. The fraction assigned to a group is determined by how many row instances within that group involve the same object.

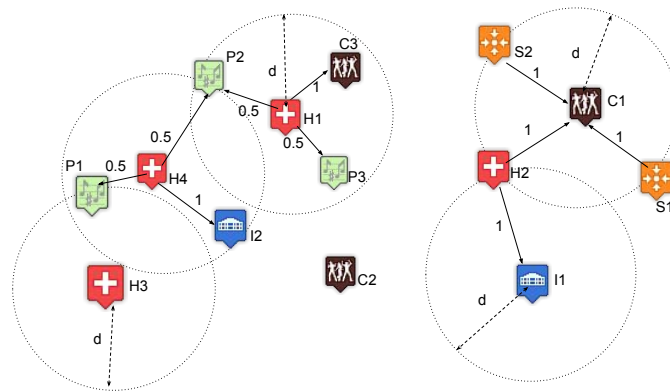
This approach is particularly useful when dealing with situations where objects are shared among multiple groups or instances, as it allows for a more fine-grained and fair representation of each group’s contribution to the overall prevalence. It ensures that the contribution of shared objects is appropriately distributed among the relevant groups, leading to a more accurate assessment of prevalence.

5.3.3 Formal Definition of Fraction-Score

In this Section, we discussed the adaptation of the fraction score computation for the temporal-window framework. For illustration, we have considered our running example of a window W^2 at time $t = 2$ as in Figure 3.2 having 14 spatial objects with five different labels $\mathcal{L} = \{H, P, C, S, I\}$, and its spatial view as in Figure 5.4. The edge is shown between the objects that are within the prespecified distance d . We have considered $d = 70m$, and used Euclidean distance for distance calculation between two spatial objects.

For a window W and an object $o \in W$, the neighbouring objects of o within a distance d are denoted by $disk(o, d, W)$. For a label $l \in \mathcal{L}$, the subset of the neighbors of o having label l is denoted $Neigh(o, l, d)$. For example, in Figure 5.4, the neighbors of some of the objects are highlighted by drawing a circle. It can be seen that neighbors of $H1$ in W^2 are $\{H1, P2, C3, P3\} (= disk(H1, 70m, W^2))$. The neighbors of all other objects are shown in Table 5.1. Also, $Neigh(H1, H, d) = \{H1\}$, $Neigh(H1, P, d) = \{P2, P3\}$, $Neigh(H1, C, d) = \{C3\}$, $Neigh(H1, S, d) = \{\}$ and $Neigh(H1, I, d) = \{\}$. The neighborhood count of all the objects for the labels $\mathcal{L} = \{H, P, C, S, I\}$ is shown in Table 5.1.

The different neighbourhood instances (hereafter called row instances) of a label-set $\mathcal{C} (\subseteq \mathcal{L})$ may not have all distinct objects. For illustration, consider a label set $\mathcal{C} = \{H, P\}$ that have row instances $H1P2, H1P3, H4P1, H4P2$ in W^2 in Figure 5.4. The object $H1$ is common in row instances $H1P2$ and $H1P3$ and thus the two objects

Figure 5.4: Fraction score Computation for W^2 (at time $t = 2$)

Object	Neighborhood	Neighborhood count					Fraction score				
		H	P	C	S	I	H	P	C	S	I
H1	H1, P2, C3, P3	1	2	1	0	0	1	1	1	0	0
H2	H2, I1, C1	1	0	1	0	1	1	0	1	0	1
H3	H3, P1	1	1	0	0	0	1	0.5	0	0	0
H4	H4, P1, P2, I2	1	2	0	0	1	1	1	0	0	1
P1	P1, H3, H4	2	1	0	0	0	1	1	0	0	0
P2	P2, H1, H4,	2	1	0	0	0	1	1	0	0	0
P3	P3, H1, C2	1	1	1	0	0	0.5	1	1	0	0
S1	S1, C1,	0	0	1	1	0	0	0	0.5	1	0
S2	S2, C1	0	0	1	1	0	0	0	0.5	1	1
I1	I1, H2	1	0	0	0	1	1	0	0	0	1
I2	I2, H4,	1	0	0	0	1	1	0	0	0	1
C1	C1, S2, H2, S1	1	0	1	2	0	1	0	1	1	0
C2	C2, P3,	0	1	1	0	0	0	1	1	0	0
C3	C3, H1	1	0	1	0	0	1	0	1	0	0

Table 5.1: Neighborhood, Neighborhood Count & fraction scores for W^2

$P2$ and $P3$ must get a $1/2$ share (i.e., *fraction value*) of $H1$ as its participation in these two instances.

The fraction value is assigned to the objects of the row instances based on the number of shared objects in different groups. Formally, for objects $o, o' \in W$, the object o receives a *fraction value* of o' as its participation in W , denoted by $\Delta_{obj}(o, o', W)$, defined as

$$\Delta_{obj}(o, o', W) = \frac{1}{\min Neigh(o', o.l, d) |} \quad (5.1)$$

Where $Neigh(o', o.l, d)$ is the sharing of o' with neighbouring objects having label $o.l$ in the window W . For our running example, for $o = P2$ and $o' = H1$, we have $Neigh(H1, P, d) = 2$ as the object $H1$ is shared with two objects in W^2 , namely $P2$

and $P3$, having the label P . Thus

$$\Delta_{obj}(P2, H1, W^2) = \frac{1}{Neigh(H1, P, d)} = \frac{1}{2} = \Delta_{obj}(P3, H1, W^2)$$

For an object $o \in W$ and a label $l \in \mathcal{L}$ such that $o.l \neq l'$, the object o may have multiple objects in its neighbourhood with label l' . The *fraction score* that an object o receives from the label l' in W , denoted by $\Delta_{label}(o, l', W)$, is the aggregation of the fraction values of those neighbouring objects that have label l' . Formally,

$$\Delta_{label}(o, l', W) = \min \left\{ \sum_{o' \in disk(o, d, W) \wedge o'.l = l'} \Delta_{obj}(o, o', W), 1 \right\} \quad (5.2)$$

The aggregate fraction score of o can not be more than 1; thus, it is bounded by 1. For example, object $P2$ in Figure 5.4 has objects $H1$ and $H4$ in its neighbour having label H . Therefore,

$$\Delta_{label}(P2, H, W) = \min\{\Delta_{obj}(P2, H1, W) + \Delta_{obj}(P2, H4, W), 1\} = 1$$

The fraction score of the objects in the window W^2 for labels $\mathcal{L} = \{H, P, C, S, I\}$ is shown in Table 5.1. In the time window framework, as time progresses, the fraction score of objects for the current labels also changes and thus requires periodic updates.

We are now ready to discuss the support computation for a label-set \mathcal{C} in a window W for deciding on co-location patterns. The defined measure quantifies the participation of a label-set by assigning fraction value to overlapped instances. Label-sets having support more than the prespecified minimum support threshold qualify as co-location patterns.

Definition 5.1 (Co-locations in a Window W). A label-set \mathcal{C} ($\subseteq \mathcal{L}$) is said to be a *co-location for a window W* if the fraction score based *support* for their object instances in W that are within a given distance d satisfies the user-specified minimum support (*min_sup*) threshold.

For computing the support of a label-set \mathcal{C} (with $|\mathcal{C}| \geq 2$), first the score of all the objects of the row instances of C in W with respect to the label-set \mathcal{C} , denoted by $\Delta_{labelset}(o, \mathcal{C}, W)$, is computed. For a fixed label $l \in \mathcal{C}$ and an object $o \in W$ with $o.l = l$, $\Delta_{labelset}(o, \mathcal{C}, W)$ is the minimum of the fraction scores object o receives with respect to other labels in \mathcal{C} . Formally,

$$\Delta_{labelset}(o, \mathcal{C}, W) = \min_{l' \in \mathcal{C} - \{o.l\}} \Delta_{label}(o, l', W) \quad (5.3)$$

The support of a label-set \mathcal{C} for a fixed label $l \in \mathcal{C}$ is the aggregation of the score of all the objects having label l , as given in equation 5.3.

$$sup(\mathcal{C} \mid l, W) = \sum_{o \in obj(l, \mathcal{C}, W)} \Delta_{labelset}(o, \mathcal{C}, W) \quad (5.4)$$

where for a label $l \in \mathcal{C}$, $obj(l, \mathcal{C}, W)$ denote the set of objects o in the row instances of \mathcal{C} in W that have label l . The support of a label-set \mathcal{C} in W is the least among the support of labels in \mathcal{C} normalized by the count of maximum objects of any label in \mathcal{C} .

$$sup(\mathcal{C}, W) = \frac{\min_{l \in \mathcal{C}} sup(\mathcal{C} | l, W)}{\max_{l \in \mathcal{C}} |\{o.l = l | o \in W\}|} \quad (5.5)$$

From the above discussion, it is evident that in a timed window framework, the neighbourhood structure of objects, as in Figure 5.4, changes. This results in a change in the neighbourhood count and the fraction score of objects as in Table 5.1 and computed using formulas in the equation 5.1-5.2. However, the computation for the support¹ of a label-set, as in equations 5.3-5.5, can be performed as proposed earlier [9].

5.3.4 Problem Formalization

Finding co-location patterns for dynamic data requires recomputing support measures periodically. A baseline approach computes the fraction score and the support from scratch for every temporal-window. However, if the number of changed objects (i.e., λ) is small compared to the window size $|W|$, the changes in the neighborhood relation are also less and ample information can be reused from the previous window. Thus, with the assumptions such as (a) $\lambda \lll W$ (b) the data updates periodically at a small-time gap, and (c) the changes in the patterns require at the run time, it is meaningful to update the fraction scores from the previous window.

The methodology to update the changes for the temporal-window between any two consecutive time instances is the same. Thus for the ease of explanation, we denote by W^c and W^o the current window (say, at time $t + 1$) and the previous window (say, at time t), respectively, such that

$$W^r = W^o \setminus O^d \quad \& \quad W^c = W^r \cup O^a$$

where O^a and O^d are respectively the added and deleted objects due to the temporal-window, and W^r are the objects retained in the current window from the previous window. Similarly, \mathcal{L}^c and \mathcal{L}^o denote distinct labels in W^c and W^o respectively such that $\mathcal{L}^c = \mathcal{L}^r \cup \mathcal{L}^{new}$, where

$$\mathcal{L}^r = \{o.l | o \in W^r\} \quad \text{and} \quad \mathcal{L}^{new} = \{o.l | o \in O^a\} \setminus \mathcal{L}^r$$

represents the unchanged labels and the newly added labels, respectively. Finding the changes in the co-location patterns due to the temporal-window requires identifying

- *Deleted Co-locations.* The co-location patterns from the previous window for which the current support is less than the minimum support threshold. Formally, finding \mathcal{C} such that

$$sup(\mathcal{C}, W^o) \geq min_support \quad \& \quad sup(\mathcal{C}, W^c) < min_support$$

¹Refer [9] by Chan *et al.* for more details about the fraction score based support computation.

- *Added Co-locations.* The label set \mathcal{C} that were not co-locations in the previous window and for which the current support value has increased above the minimum support threshold. Formally,

$$\text{sup}(\mathcal{C}, W^o) < \text{min_support} \quad \& \quad \text{sup}(\mathcal{C}, W^c) \geq \text{min_support}$$

This work proposes an efficient approach for mining *current co-locations* for the temporal-window framework by updating the fraction value (equation 5.1) and the fraction score (equation 5.2) of the current objects in the window W^c for the current labels \mathcal{L}^c .

5.4 Framework for Updating Fraction score

The change in the fraction score of an object $o \in W^c$ is due to the change in its neighborhood. An inverted list R-Tree (IR-Tree) [15, 37] is used to find the neighborhood of an object o within the user-specified distance d . For finding an updated neighborhood, the IR-Tree is updated by inserting O^a and deleting O^d . We denote by $\text{Disk}(o, d, W^o)$ and $\text{Disk}(o, d, W^c)$ the neighborhood of an object o in the old and the updated IR-Tree respectively.

Next, we define *1-nbd* and *2-nbd* of an object o in a window W to analyze the effect of the change in the neighbourhood over the fraction score.

Definition 5.2 (Neighbourhood of an object). For an object $o \in W$, an object $o' \in \text{Disk}(o, d, W)$ is said to be in *1-nbd* of o , and an object $o'' \in \text{Disk}(o', d, W^o)$ such that $o'' \notin \text{Disk}(o, d, W^o)$ is said to be in *2-nbd* of o .

We first analyze the effects of deletion and addition of a single object on the fraction score. We denote by $\Delta_{obj}(o, o', W^o)$ and $\Delta_{obj}(o, o', W^c)$ the fractional participation value that object o receives from o' (equation 5.1) in the old and the current window respectively. For notational convenience, the deletion of a single object o is represented as $O^d = \{o\}$, $O^a = \phi$, and with this change, W^o and W^c denote the old window and the current window respectively. For objects in the *1-nbd* of o , say o' , the deletion of o will decrease the fraction score of o' for the label $o.l$. Formally,

$$\forall o' \in \text{Disk}(o, d, W^o) \text{ s.t. } \Delta_{obj}(o, o', W^o) = 1 \xrightarrow{O^a, O^d} \Delta_{label}(o', o.l, W^c) = 0$$

It is to note that $\Delta_{obj}(o, o', W^o) = 1$ corresponds to the case of only one object in the *1-nbd* of o' having label $o.l$, and that is o .

The cases of deletion of an object o for which $\Delta_{obj}(o, o', W^o) < 1$ and that changes the fraction value of the neighboring objects to a non-zero value are discussed next.

Lemma 5.3. *Effect of deleting a single object o on the fraction score of objects in the 1-nbd and 2-nbd of o that have the same label as o .*

- (a) For the objects o' in the 1-nbd of o and from which o receives a fraction value that is less than 1 (i.e., $\Delta_{obj}(o, o', W^o) < 1$), the fraction value of all the objects in the 1-nbd of o' that have the same label as o will increase due to the deletion of o . Formally,

Let $o' \in Disk(o, d, W^o)$ with $\Delta_{obj}(o, o', W^o) < 1$, then for all $o'' \in Disk(o', d, W^c)$ such that $o''.l = o.l$ and $o'' \neq o$, the fraction value o'' receives from o' increases due to the deletion of o , i.e.,

$$\Delta_{obj}(o'', o', W^o) < \Delta_{obj}(o'', o', W^c)$$

- (b) Additionally, in the previous case, the fraction score of o'' for the label $o'.l$ increases or remains the same.

Proof (a): The deletion of object o reduces the neighborhood count of $o' \in Disk(o, d, W^o)$ with respect to the label $o.l$, i.e., reduces $Neigh(o', o.l, d)$ by 1. Also, as $\Delta_{obj}(o, o', W^o) < 1$, the reduced neighborhood count is still greater or equal to 1. Thus for all the objects $o'' \in Disk(o', d, W^c)$, other than o , having $o''.l = o.l$, it is found that

$$\Delta_{obj}(o'', o', W^o) = \frac{1}{Neigh(o', o.l, d)} < \frac{1}{Neigh(o', o.l, d) - 1} = \Delta_{obj}(o'', o', W^c)$$

Proof (b): The fraction score of object o'' for the label $o'.l$ is the aggregation of fraction values object o'' receives from the objects in its 1-nbd with label $o'.l$. Let us consider the case when this value is less than one in the old window. From equation 5.2 in Section 5.3.3, we have

$$\Delta_{label}(o'', o'.l, W^o) = \sum_{o_i \in Disk(o'', d, W^o) \wedge o_i.l = o'.l} \Delta_{obj}(o'', o_i, W^o)$$

Some of the terms in the above summation are the fraction values o'' received from the objects o_i that are in the 1-nbd of o . One such object is o' . Let us denote all these objects by set A , i.e., $A = \{o_i \mid o_i \in Disk(o'', d, W^o), o_i \in Disk(o, d, W^o), o_i.l = o'.l\}$, and the remaining elements by the set B , i.e., $B = \{o_i \mid o_i \in Disk(o'', d, W^o), o_i \notin Disk(o, d, W^o), o_i.l = o'.l\}$. It is to be noted that for all $a \in A$, $\Delta_{obj}(o, a, W^o) < 1$ as their neighborhood contains at least two objects with the same label as $o.l$, namely o and o'' . Now rewriting the above expression,

$$\Delta_{label}(o'', o'.l, W^o) = \sum_{a \in A} \Delta_{obj}(o'', a, W^o) + \sum_{b \in B} \Delta_{obj}(o'', b, W^o)$$

on deleting the object o , from part (a), $\Delta_{obj}(o'', a, W^o) < \Delta_{obj}(o'', a, W^c)$ for all $a \in A$, whereas $\Delta_{obj}(o'', b, W^o) = \Delta_{obj}(o'', b, W^c)$ for all $b \in B$. And therefore,

$$\Delta_{label}(o'', o'.l, W^o) \leq \Delta_{label}(o'', o'.l, W^c) \quad (5.6)$$

Thus the value of the fraction score of o'' for the label $o'.l$ increases or remains the same (when $A = \phi$) on deleting object o . Moreover, the change in the fraction score of o'' with respect to the label $o'.l$, denoted by $\Delta_{label}^{diff}(o'', o'.l)$, is

$$\Delta_{label}^{diff}(o'', o'.l) = \sum_{a \in A} \{\Delta_{obj}(o'', a, W^c) - \Delta_{obj}(o'', a, W^o)\} \quad (5.7)$$

Further, in the case when the minimum value of the fraction score is 1 in the old window (equation 5.2), despite the increase in the fraction score of o'' for the label $o'.l$ as in equation 5.6, there is no change in the actual value as it is bounded by 1. \square

Lemma 5.4. *Effect of deleting a single object o on the fraction scores of objects in its 1-nbd that have labels different than o .*

- (a) *Due to the deletion of a single object o , the fraction value that the objects in the 1-nbd of o , say o' , receives from o becomes zero, whereas the fraction value o' receives from other objects in its 1-nbd having the same label as that of o remains the same. Formally,*

For $o' \in Disk(o, d, W^o)$, we have

- i $\Delta_{obj}(o', o, W^c) = 0$*
- ii For all $o'' \in Disk(o', d, W^c)$ such that $o''.l = o.l$, we have*

$$\Delta_{obj}(o', o'', W^c) = \Delta_{obj}(o', o'', W^o)$$

- (b) *Additionally, the fraction score of o' for the label $o.l$ decreases due to deletion of o .*

Proof (a): The proof of the first part is direct as the value depends upon the fraction an object o' receives from an object o which is not present anymore in W^c . To prove the second part, it is observed that the fraction value $\Delta_{obj}(o', o'', W^c)$ depends upon the neighborhood count of o'' with respect to label $o'.l$ in W^c . The deletion of object o , which has a different label than o' will not change this count, and hence the result.

Proof (b): Let us denote $A = \{o'' \mid o'' \in Disk(o', d, W^o), o''.l = o.l\}$ and $B = \{o'' \mid o'' \in Disk(o', d, W^c), o''.l = o.l\}$ where $B = A \setminus \{o\}$. The fraction score of object o' due to label $o.l$ in the old window W^o is

$$\begin{aligned} \Delta_{label}(o', o.l, W^o) &= \sum_{o'' \in A} \Delta_{obj}(o', o'', W^o) \\ &= \Delta_{obj}(o', o, W^o) + \sum_{o'' \in B} \Delta_{obj}(o', o'', W^o) \end{aligned}$$

From part a(ii), we have $\Delta_{obj}(o', o'', W^o) = \Delta_{obj}(o', o'', W^c)$ for all $o'' \in B$, and

$$\Delta_{label}(o', o.l, W^c) = \sum_{o'' \in B} \Delta_{obj}(o', o'', W^c)$$

Therefore,

$$\Delta_{label}(o', o.l, W^o) - \Delta_{label}(o', o.l, W^c) = \Delta_{obj}(o', o, W^o) > 0$$

Hence the result. \square

The case of the addition of an object is dual to the deletion of an object, and the results are direct from the previous lemmas. For addition of an object o , we consider $O^d = \phi$ and $O^a = \{o\}$, and the windows W^o and W^c denote the old and the current windows with this change.

Lemma 5.5. *Effect of adding a single object o on the fraction scores of objects in the 1-nbd and 2-nbd of o and having the same label as that of o .*

- (a) *For the objects o' in the 1-nbd of o , the fraction value of all the objects in 1-nbd of o' and having the same label as o will decrease due to the addition of o . Formally, For $o' \in Disk(o, d, W^c)$ and for $o'' \in Disk(o', d, W^o)$ such that $o''.l = o.l$, $\Delta_{obj}(o'', o', W^c) < \Delta_{obj}(o'', o', W^o)$ on addition of o .*
- (b) *Additionally, the fraction score of o'' for the label $o'.l$ decreases.*

Proof outline: The proof of both the parts is direct from the lemma 5.3 on interchanging the role of the current and the old window. \square

Lemma 5.6. *Effect of adding a single object o on the fraction scores of objects in its 1-nbd having label different from that of o .*

- (a) *Due to the addition of a single object o , the fraction value that the objects in the 1-nbd of o , say o' , receives from other objects in the 1-nbd of o' and having a label as that of o remains the same. Formally, For $o' \in Disk(o, d, W^c)$ and for $o'' \in Disk(o', d, W^c)$ s.t. $o''.l = o.l$, we have*

$$\Delta_{obj}(o', o'', W^c) = \Delta_{obj}(o', o'', W^o)$$

- (b) *Additionally, the fraction score of o' for the label $o.l$ increases due to the addition of o .*

Proof outline: The proof is direct from lemma 5.4 on interchanging the current and the old window, and from the fact that $\Delta_{obj}(o', o, W^c) > 0$ \square

From equation 5.4 and equation 5.5 in section 5.3.3, the support $sup(\mathcal{C}, W)$ of a candidate co-location \mathcal{C} is an aggregation of fraction score of its row instances. Also from lemmas 5.3-5.6, fraction score may increase, decrease, or remain the same corresponding to the addition or deletion of objects which depends upon the neighbourhood of changed objects in the current window. As estimating the change in the support value $sup(\mathcal{C}, W^c)$ from $sup(\mathcal{C}, W^o)$ is not possible without updating the fraction score of the changed

objects, we propose an observation-based approach for efficiently updating fraction values of the relevant objects in the current window.

In the observation-based approach, the fraction values are updated only for each pair of objects that have changed, and the rest of the values are copied as it is. In the next section, we discuss this approach that utilizes the results of the addition and deletion of an object as derived in lemma 5.3-5.6. There are issues with efficiently updating fraction scores and support values after updating fraction values. This is due to multiple deletions and the addition of objects, keeping track of the changed fraction values and fraction scores. The implementation issues are discussed in Section 5.5, however, in this section, we discuss our observation-based technique for the addition and the deletion of objects.

5.4.1 Addition of an object

This Section discusses the effect of the addition of an object $o \in O^a$ on the fraction values of the objects in W^c .

- (a) For all the objects $o' \in Disk(o, d, W^c)$, $\Delta_{obj}(o', o, W^c)$ needs to be computed. It is the fraction value objects o' receives from newly added object o , and is equal to the reciprocal of the number of objects in the neighborhood of o having label $o'.l$ as in equation 5.1, Section 5.3.3. This update requires exploring of the 1-*nbd* of $o \in O^a$.
- (b) For all the objects $o' \in Disk(o, d, W^c)$, we need to compute $\Delta_{obj}(o, o', W^c)$. This requires finding objects in the neighborhood of o' having label $o.l$. This update requires exploring the 1-*nbd* and 2-*nbd* of $o \in O^a$.
- (c) For all the objects $o' \in Disk(o, d, W^c)$ and $o'' \in Disk(o', d, W^c)$ such that $o''.l = o.l$, $\Delta_{obj}(o'', o', W^c)$ (lemma 5.5) must be updated. This update also requires exploring the 1-*nbd* and 2-*nbd* of $o \in O^a$.
- (d) For all the objects $o' \in Disk(o, d, W^c)$ and $o'' \in Disk(o', d, W^c)$ such that $o''.l \neq o.l$, the $\Delta_{obj}(o'', o', W^c)$ values remain unchanged. i.e., $\Delta_{obj}(o'', o', W^c) \leftarrow \Delta_{obj}(o'', o', W^o)$
- (e) For all the objects $o' \in Disk(o, d, W^c)$ and $o'' \in Disk(o', d, W^c)$, the $\Delta_{obj}(o', o'', W^c)$ values remain unchanged (lemma 5.4). i.e., $\Delta_{obj}(o', o'', W^c) \leftarrow \Delta_{obj}(o', o'', W^o)$
- (f) For all the objects o' that are neither in 1-*nbd* nor in 2-*nbd* of $o \in O^a$, their fractions remain unchanged. That is, for all $o'' \in Disk(o', d, W^c)$, fractions $\Delta_{obj}(o', o'', W^c)$ and $\Delta_{obj}(o'', o', W^c)$ remain unchanged.

The above observations provide a guideline to compute, update and restore the fraction values due to the addition of objects. However, the spatial locations of the newly added objects make this task non-trivial. For example, the newly inserted objects having the same label and in the 1-*nbd* or 2-*nbd* of each other, the same neighborhood count and

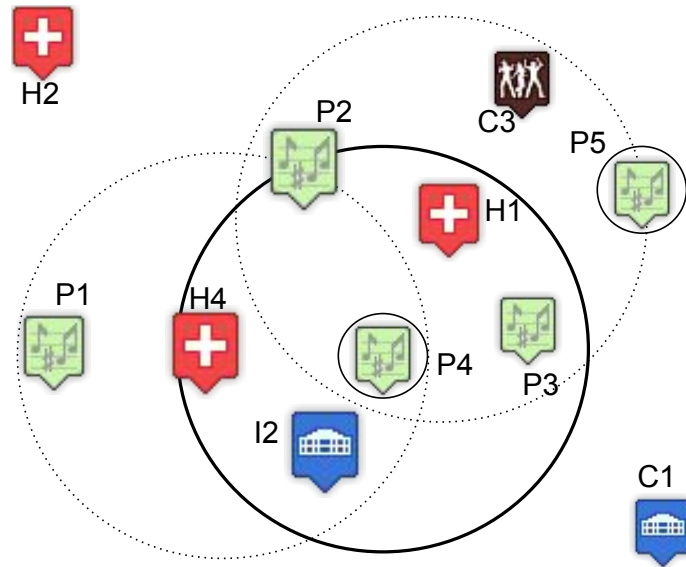


Figure 5.5: Effect of addition

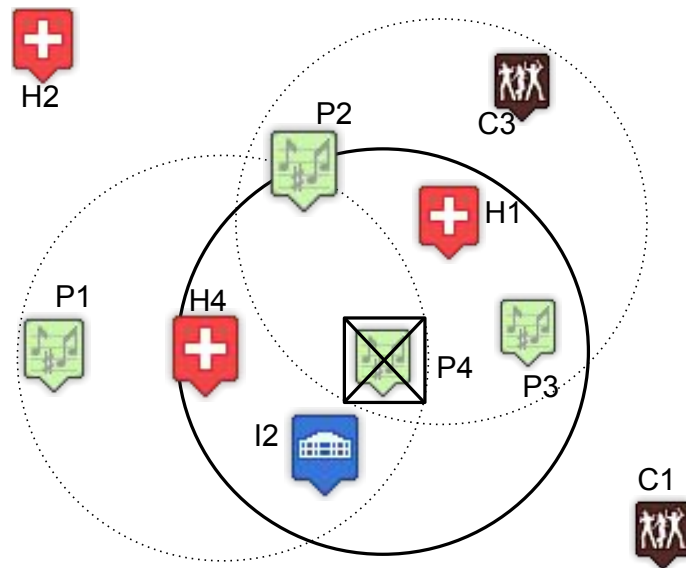


Figure 5.6: Effect of deletion

Δ values, may be updated multiple times. The above observations are illustrated next with an example.

Consider Figure 5.5, where all the objects are old except $P4$ and $P5$. The effect of adding of $P4$ having $1\text{-}nbd\ Disk(P4, d, W^c) = \{P2, P4, P3, H1, H4, I2\}$ is that the fraction values of objects in the $1\text{-}nbd$ and $2\text{-}nbd$ are updated as

1. All the objects in the 1-*nb*d of $P4$ will get a fraction of $P4$. Two objects in the 1-*nb*d of $P4$ have label H , i.e., $Neigh(P4, H, d) = \{H1, H4\}$. Thus $\Delta_{obj}(H1, P4, W^c) = \frac{1}{Neigh(P4, H, d)} = \frac{1}{2} = \Delta_{obj}(H4, P4, W^c)$. Similarly, $\Delta_{obj}(I2, P4, W^c) = 1$. These freshly computed values require exploring 1-*nb*d of $P4$ only.
2. The 1-*nb*d of the object $H4$ contains three objects with label P , including the one newly introduced $P4$. The fraction value that $P4$ will receive from $H4$ is $\Delta_{obj}(P4, H4, W^c) = \frac{1}{Neigh(H4, P, d)} = \frac{1}{3}$. This freshly computed value requires exploring the 2-*nb*d of $P4$.
3. In addition to the above, fraction values of the old object in the 1-*nb*d $H4$ with label P , i.e., $P1$ and $P2$, will also change as $\Delta_{obj}(P1, H4, W^c) = \Delta_{obj}(P2, H4, W^c) = \Delta_{obj}(P4, H4, W^c)$. That is, all the objects with label P in the 1-*nb*d of $H4$ will receive one-third fraction of it in place of the earlier score of $\frac{1}{2}$. This update on 2-*nb*d of $P4$ is performed within the computation of the previous step.
4. The fraction values $\Delta_{obj}(I2, H4, W^c)$ (in (d)) and $\Delta_{obj}(H4, P1, W^c)$, $\Delta_{obj}(H4, P2, W^c)$, $\Delta_{obj}(H4, I2, W^c)$ (in (e)) remain unaffected. It can be observed that the fraction values of all other objects that are not in 1-*nb*d or 2-*nb*d of $P4$ also remain unaffected. That is, $\Delta_{obj}(C1, P3, W^c)$, $\Delta_{obj}(P3, C1, W^c)$, $\Delta_{obj}(H2, P2, W^c)$, $\Delta_{obj}(P2, H2, W^c)$, etc., remain the same. Further, note that in the 1-*nb*d of $H1$ there are two newly inserted objects with label P , namely $P4$ and $P5$. Thus the multiple updates for $\Delta_{obj}(P4, H1, W^c)$ and $\Delta_{obj}(P5, H1, W^c)$ due to (b) and (c) should be avoided.

Next, changes in fraction values due to the deletion of objects are discussed. For explanation, the example of Figure 5.6 is used in which object $P4$ is deleted.

5.4.2 Deletion of object

This subsection presents observations about the effect of deleting an object $o \in O^d$ on the fraction values of the objects in W^c .

- (a) For all the objects $o' \in Disk(o, d, W^o)$, fraction values $\Delta_{obj}(o', o, W^o)$ and $\Delta_{obj}(o, o', W^o)$ are deleted. This update can be performed by exploring the 1-*nb*d of $o \in O^d$.

Clearly, from the example, fraction values $\Delta_{obj}(P4, H1, W^o)$, $\Delta_{obj}(P4, H4, W^o)$, $\Delta_{obj}(P4, I2, W^o)$ and $\Delta_{obj}(H1, P4, W^o)$, $\Delta_{obj}(H4, P4, W^o)$, $\Delta_{obj}(I2, P4, W^o)$ are all deleted.

- (b) The deletion of o reduces the neighborhood count of $o' \in Disk(o, d, W^o)$ with respect to the label $o.l$, i.e., $|Neigh(o', o.l, d)|$. Thus, for all the objects $o'' \in Disk(o', d, W^o)$ such that $o''.l = o.l$, it is necessary to update $\Delta_{obj}(o'', o', W^c)$ using $|Neigh(o', o.l, d)|$ (lemma 5.3). This update requires exploration of the 1-*nb*d and 2-*nb*d of $o \in O^d$.

For example, the deletion of $P4$ reduces the neighborhood counts of $| Neigh(H1, P, d) |$ and $| Neigh(H4, P, d) |$, $| Neigh(I2, P, d) |$ by 1. Thus the updated value is $\Delta_{obj}(P3, H1, W^c) = \frac{1}{2} = \Delta_{obj}(P2, H1, W^c)$. Similarly, $\Delta_{obj}(P2, H4, W^c) = \frac{1}{2} = \Delta_{obj}(P1, H4, W^c)$.

- (c) For all the objects $o' \in Disk(o, d, W^o)$ and $o'' \in Disk(o', d, W^o)$ such that $o''.l \neq o.l$, the $\Delta_{obj}(o'', o', W^c)$ values remain unchanged. (lemma 2) i.e., $\Delta_{obj}(o'', o', W^c) = \Delta_{obj}(o'', o', W^o)$

From the example, the score $\Delta_{obj}(H4, I2, W^c)$ and $\Delta_{obj}(I2, H4, W^c)$ will remain unaffected.

- (d) For all the objects $o' \in Disk(o, d, W^o)$ and $o'' \in Disk(o', d, W^o)$, such that $o''.l = o.l$, the $\Delta_{obj}(o', o'', W^c)$ values remain unchanged.

From the example, the score $\Delta_{obj}(H1, P2, W^c)$ and $\Delta_{obj}(H4, P1, W^c)$ will remain unaffected as no new object with label H is deleted in their respective neighborhoods.

- (e) For all the objects o' that are neither in the 1-*nb*d nor in 2-*nb*d of $o \in O^d$, their scores remain unchanged.

From the example, the score of $\Delta_{obj}(H2, P1, W^c)$ remains unaffected.

The implementation procedure for updating the fraction values and subsequently the fraction scores for the changed window using the observation-based approach is non-trivial. This is because the fraction value that a new object o receives from a neighboring object o' requires finding change in the neighborhood of o' . If many objects were added objects in the neighborhood of o' other than o that have label $o.l$, a recursive call to update the neighborhood count from newly added objects may update the same neighborhood count multiple times and may give an overcounting problem. The addition of an object also changes the fraction value of its neighboring objects. Thus, updating a fraction value requires identifying the effect of addition and deletion on the changed objects, their neighborhood objects, and so forth. Moreover, for efficient computation, it is required to update the fraction score (equation 5.2) together with the fraction value. The computation procedure for updating fraction values and the fraction scores is discussed in the next section. For finding co-locations, the procedure of Chan et al. [9] is adopted after computing the support measure for the updated fraction scores.

5.5 Algorithms for Mining Co-locations on Dynamic Data

This Section presents an efficient approach to updating the fraction score (Algorithm 10). It is based on finding fraction values of pair of objects in the current window W^c by restoring values from the old window W^o that remain unaffected, updating values that change, and computing values that are freshly introduced based on the added objects

O^a and deleted objects O^d . This requires updating neighborhood count and fraction score of objects based on the observations given in previous Section (5.4.1 and 5.4.2). Then, updated fraction values to efficiently compute the fraction scores of labeled objects. Since fraction score changes correspond to fraction value changes, a record of the changed values in *Update_List* is maintained.

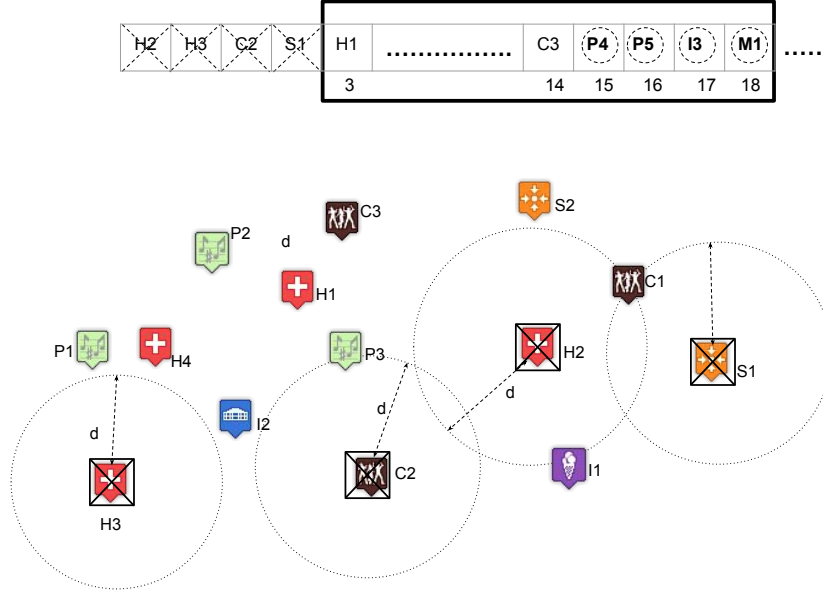


Figure 5.7: Deletion of Objects

5.5.1 Fraction_Update_List

This Section explains the structure of the *Update_List*, its creation, and maintenance. *Update_List* is a hash map containing key-value pair and is denoted as $Update_List < Key, Value >$. It stores a combination of the object (o') and label (l) as a key and a list of objects in the neighborhood of o' with label l as the value corresponding to the key. The *Fraction_Update_List* procedure is used to create and update this list (Algorithm 7). In step 1-2, it is checked if the key (o', l) is already present in the *Update_List*. If not, it is initialized to *NULL*. In step 3-5, the procedure searches for the object o'' with label l in $1-nbd$ objects of o' and append it to the *Update_List* if it is found.

As per observation (b) in Section 5.4.2, the deletion object $H3$ in the running example of Figure 5.7 updates $\Delta_{obj}(H4, P1, W^c)$. Hence, object $H4$ is added to the list as an entry with key $(P1, H)$. Similarly, as per observation (c) in 5.4.1, adding object $P4$ updates the $\Delta_{obj}(P1, H4, W^c)$. Thus, object $P1$ is added to the list as an entry with key $(H4, P)$.

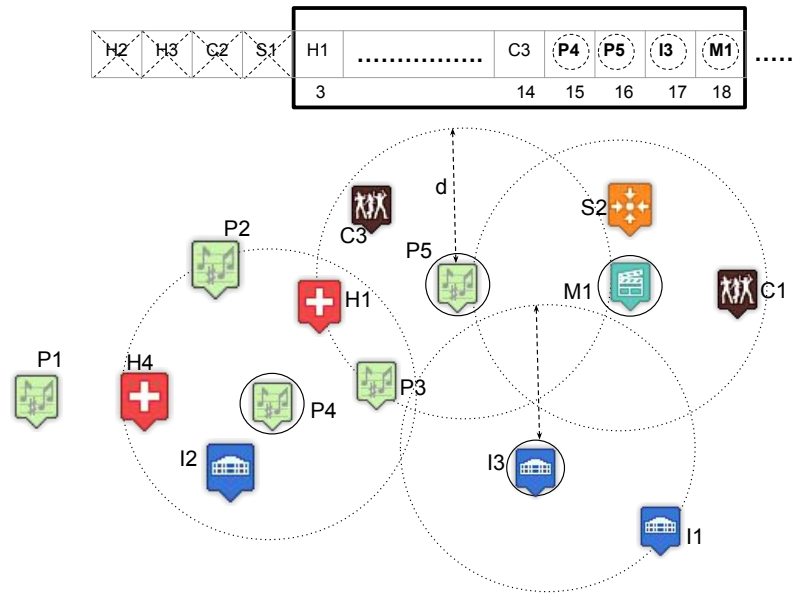


Figure 5.8: Addition of Objects

Procedure 7 $Fraction_Update_List(o', l, disk)$

- 1: **if** $(o', l) \notin Update_List.key$ **then**
 - 2: $Update_List(o', l) \leftarrow NULL$
 - 3: **for all** $o'' \in disk$ **do**
 - 4: **if** $o''.l = l$ **then**
 - 5: $Update_List(o', l).append(o'')$
 - 6: **end if**
 - 7: **end for**
 - 8: **end if**
-

5.5.2 Update_Neigh_Deletion

This subsection presents the proposed procedure for updating the neighborhood count of objects once an object is deleted. The procedure first inserts the $Update_list$ for which the fraction values need to be updated as explained in the previous subsection. After that, we reduce the neighborhood count of the objects as per observation (b) in Section 5.4.2.

In steps 1-2, a loop is done over the $1-nbd$ objects of deleted objects. In step-3, we invoke the $Fraction_Update_List$ is invoked on W^r for all the objects in $1-nbd$ with respect to the deleted object label. In step 4-5, the neighborhood count of $1-nbd$ objects is decremented by 1 as per lemma 5.3.

Procedure 8 *Update_Neigh_Deletion*

```

1: for all  $o \in O^d$  do
2:   for all  $o' \in Disk(o, d, W^o)$  do
3:     Call: Fraction_Update_List( $o', o.l, Disk(o, d, W^r)$ )
4:     if  $o' \in W^r$  then
5:        $| Neigh(o', o.l, d) | \leftarrow | Neigh(o', o.l, d) | - 1$ 
6:     end if
7:   end for
8: end for

```

5.5.3 *Update_Neigh_Addition*

This subsection presents the procedure to update the neighborhood count of objects once an object is added. Similar to the *Update_Neigh_Deletion* procedure, this procedure also does two tasks. First, it inserts in the *Update_list* the objects for which the fraction values need to be updated. Second, the neighborhood count of objects is increased as discussed in Section 5.4.1.

Procedure 9 *Update_Neigh_Addition*

```

1: for all  $o \in W^c \wedge l \in \mathcal{L}^c$  do
2:   if  $o \notin W^r \vee l \notin \mathcal{L}^r$  then
3:     initialize  $| Neigh(o, l, d) |$  and  $\Delta_{label}(o, l, W^c)$  to 0
4:   end if
5: end for
6: for all  $o \in O^a$  do
7:   for all  $o' \in Disk(o, d, W^c)$  do
8:      $| Neigh(o, o'.l, d) | \leftarrow | Neigh(o, o'.l, d) | + 1$ 
9:     Set:  $\Delta_{obj}(o, o', W^o) \leftarrow 0$ 
10:    Call: Fraction_Update_List( $o', o.l, Disk(o, d, W^c)$ )
11:    if  $o' \in W^r$  then
12:       $| Neigh(o', o.l, d) | \leftarrow | Neigh(o', o.l, d) | + 1$ 
13:    end if
14:  end for
15: end for

```

In steps 1-3, the neighborhood counts and fraction scores of newly added objects are initialized to 0. In step 4-5, the procedure iterates over the 1-*nbd* objects of newly added objects in the current window. In step 6, the procedure calculates the neighborhood count of new objects with respect to the labels of its 1-*nbd* objects. In step-7 the procedure sets the old fraction value of newly inserted objects with respect to its neighborhood objects to 0. In step-8 the *Fraction_Update_List* is invoked on W^c for all the objects in 1-*nbd* with respect to the newly added object labels. In steps 8-9, the

neighborhood count of 1-*nbd* objects is incremented by 1 as per lemma 5.5.

5.5.4 Update Fraction Score

This subsection presents the proposed algorithm to update fraction scores. This algorithm maintains global states of the variables old object set W^o , old label set \mathcal{L}^o , updated object set W^c , updated label set \mathcal{L}^c and distance threshold d . It also includes a set of computed fraction values Δ_{obj} and fraction scores Δ_{label} (given in equation 5.1 and 5.2). *Update_List* introduced in Section 5.5.1 is also a part of the global state. For the computational purpose, a retained object set W^r and retained label set \mathcal{L}^r are maintained. W^r represents the retained object set, which includes objects that were present in the previous state without any changes. These are the objects that were retained as-is from the previous state. \mathcal{L}^r , similarly, represents the retained label set. It contains labels that were present in the previous state without any changes, essentially representing labels that were retained from the previous state. This algorithm takes *new object set* O^a and *deleted object set* O^d as inputs. The objective is to update each fraction score Δ_{label} (equation 5.2) for the current window, which is the output of this algorithm.

Procedure 10 Update Fraction Scores (UpFS)

Global State: $W^o, \mathcal{L}^o, W^c, \mathcal{L}^c, d, \Delta_{obj}, \Delta_{label}, Update_List$

Input: O^a, O^d

Output: *Updated fraction scores* Δ_{label}

```

1: Update  $W^r, \mathcal{L}^r, W^c, \mathcal{L}^c$ 
2: Call: Update_Neigh_Deletion, Update_Neigh_Addition
3: for all  $o \in O^a$  do
4:   for all  $o' \in Disk(o, d, W^c)$  do
5:      $\Delta_{obj}(o', o, W^c) \leftarrow 1 / | Neigh(o, o'.l, d) |$ 
6:      $\Delta_{label}(o', o.l, W^c) \leftarrow \Delta_{label}(o', o.l, W^c) + \Delta_{obj}(o', o, W^c)$ 
7:   end for
8: end for
9: for all  $(o', l) \in Update\_List.key$  do
10:  for all  $o'' \in Update\_List(o', l)$  do
11:     $\Delta_{obj}(o'', o', W^c) \leftarrow 1 / | Neigh(o', o''.l, d) |$ 
12:     $\Delta_{label}(o'', o'.l, W^c) \leftarrow \Delta_{label}(o'', o'.l, W^c) + \Delta_{obj}(o'', o', W^c) - \Delta_{obj}(o'', o', W^o)$ 
13:  end for
14: end for

```

In step-1, W^r, \mathcal{L}^r is updated by deleting O^d and W^c, \mathcal{L}^c by adding O^a objects. Further, the set of labels \mathcal{L}^r is updated if any label completely disappears after deleting O^d , and the set of labels \mathcal{L}^c if any new label is introduced after adding O^a . Consider Figure.5, which depicts the deletion of objects $\{H2, H3, S1, C2\}$, the update of $W^r = \{H1, H4, P1, P2, P3, S2, I1, I2, C1, C3\}$ and addition of objects $\{P4, P5, M1, I3\}$. The

current window is updated as $W^C = \{H1, H4, P1, P2, P3, P4, P5, S2, I1, I2, I3, C1, C3, M1\}$. All the labels are retained in \mathcal{L}^r as there is no label that has completely disappeared due to deletion of O^d , but a label M is added to \mathcal{L}^c . Hence, $\mathcal{L}^r = \{H, P, S, I, C\}$ and $\mathcal{L}^c = \{H, P, S, I, C, M\}$. In step-2, the algorithm updates the neighborhood counts of objects that are affected using the *Update_Neigh_Deletion* and *Update_Neigh_Addition* procedures (see sections 5.5.2 and 5.5.3). In steps 3-5, the algorithm calculates the fraction values of 1-*nb*d objects with respect to the new object as per equation 5.1. Step-6 updates the fraction scores of 1-*nb*d objects as an aggregated sum of Δ values with respect to the new object label. In steps 7-9, the algorithm calculates the Δ values of 2-*nb*d objects with respect to the 1-*nb*d objects and also new objects as per equation 5.1 using the $\langle Key, Value \rangle$ pairs inserted in step-2 using *Fraction_Update_List*($o', l, disk$) procedure (explained in Section 5.5.1). In step-10 we update the fraction scores of 2-*nb*d objects as aggregated sum of fraction values with respect to 1-*nb*d object labels as per lemmas 5.4 and 5.6. The updated neighborhoods and fraction scores after deletion and addition are shown in Table 5.2.

Obj	Neighborhood	Neighborhood Count						Fraction score					
		H	P	C	S	I	M	H	P	C	S	I	M
H1	H1, P2, C3, P3	1	1	2	0	0	0	1	1	1	0	0	0
H4	H4, P1, P2, P4, I2	1	3	0	0	1	0	1	1	0	0	1	0
P1	P1, H4	1	1	0	0	0	0	0.33	1	0	0	0	0
P2	P2, P4, H1, H4,	2	2	0	0	0	0	0.33	1	0	0	0	0
P3	P3, P4, H1, C4	1	2	1	0	0	0	0.33	1	1	0	0	0
P4	P2, P3, P4, H1, H4, I2	2	3	0	0	1	0	0.66	1	0	0	1	0
S2	S2, M1, C1	0	0	1	1	0	1	0	0	1.0	1	0	1
I1	I1, I3	0	0	0	0	2	0	0	0	0	0	1	0
I2	I2, H4, P4	1	1	0	0	1	0	1	0	0	0	1	0
I3	I3, I1	0	0	0	0	1	0	1	0	0	0	1	0
C1	C1, S2, M1	0	0	1	1	0	1	0	0	1	1	0	0.5
C3	C3, C4, H1,	1	0	2	0	0	0	0.5	0	1	0	0	0
C4	C3, C4, H1, M1, P3	1	1	2	0	0	1	0.5	1	1	0	0	1
M1	M1, C1, C4, S2	0	0	1	1	0	1	0	0	1	1	0	1

Table 5.2: Snapshot After Addition

The proposed algorithm reuses the apriori-like search procedure proposed by Chan *et. al.* [9], which enumerate candidate co-location patterns iteratively and compute the support by using a filter and verification approach with combinatorial search. For each candidate co-location, the algorithm checks if it satisfies the support threshold and add it to set of co-locations if it is satisfied.

Time complexity: Given λ which is the number of changed objects, t which is the current time or the number of window changes, the algorithm updates fraction scores with a best-case complexity time $O(\lambda)$ and a worst-case time complexity $O(|W|)$.

5.5.5 Implementations

Three implementations, namely, Dynamic Mining (DM), Dynamic Mining with NBD-list (DM NBD-list), and Dynamic Mining with NBD-Hash map (DM NBD-HM) are proposed. DM updates the fraction scores of objects that are changed due to window movement, i.e., with the addition and the deletion of objects. For this, we identified the effect of the addition and deletion of objects over the neighborhood objects and proposed a computational framework for updating fraction scores. DM NBD-list and DM NBD-HM store the neighborhood of all the objects as the list of objects and as the hash map of objects over the features respectively. These two algorithms are expected to avoid the unnecessary computation of searching neighborhood objects while updating fraction scores in DM. However, if many objects are in the current window (i.e., the number of changed objects is huge), the time required to update the neighborhood and the memory requirement to store them will be high. In this case, updating fraction score becomes costlier, and it will be more beneficial to mine the patterns from scratch as compared to updating the result set from the past state.

5.6 Experiments and Results

We study the performance of the proposed algorithm in this Section.

5.6.1 Experimental Setup

In order to evaluate the performance of UpFS algorithm experiments were carried out on both real data and synthetic data. Datasets were obtained from Chan *et. al.* [9]. To make the dynamic setting we have associated a arrival timestamps for the data. We have modeled the arrival timestamps using a Poisson process.

Real dataset: This dataset describes points of interest in the UK². It contains features such as restaurants, churches, and banks It consists of 182,334 objects with 36 feature types. This dataset is referred to as Dataset-1 in the experiments.

Synthetic datasets: Several synthetic datasets were generated using the process given by Huang *et. al.* [24]. First co-locations were generated and then neighborhoods have been added followed by the addition of noise. The parameters like size of each co-location (λ_1), spatial framework size (D), size of grid cell (d'), noise label ratio (r_{noisy_label}), and noise instance ratio (r_{noisy_num}), number of co-location (N_{co_loc}) are set to 5, 10^6 , 10, 0.5, 0.5 and 20 respectively. Dataset-2 refers to a synthetic dataset of 152487 objects with 462 features, generated with the default values shown in Table 5.3.

²<https://www.pocketgpsworld.com/modules.php?name=POIs>

All algorithms were implemented in C/C++ and are memory-based. Whatever algorithms that are proposed for dynamic mining of co-locations used only the participation index as a prevalence measure. Since we did not find an algorithm for fraction score, the performance of these algorithms was compared to a baseline algorithm, which is a customization of the fraction score algorithm³ proposed by Chan *et al.* [9]. It is adapted for dynamic data by recomputing the fraction score periodically with the window movement. All experiments were conducted on the Linux operating system running on a 3.20 GHz computer with 32 GB RAM.

Parameter settings: Table 5.3 shows the experimental parameters and the values used in the experiments. The bold one indicates the default values. λ is the number of objects added or deleted from the window. S is the minimum support threshold. $|W|$ is the size of the window. t is the number of times the window is updated. d is the distance threshold of neighborhoods.

Parameter	Meaning	Values
λ	Data change rate	5 10 15 20 30
S	Minimum support thresholds	0.1 0.2 0.3 0.4 0.5 0.6
$ W $	Window size	500 1000 1500 2000 2500 3000
t	Window change rate	5 10 15 20 25 30
d	Distance	1 2 3 4 5 6
λ_2	The parameter used in Poisson distribution to construct the instances for each maximal co-location	40, 50 , 60, 70, 80
m_{clump}	Number of feature instances for each co-location in it's neighborhood	1 , 2, 3, 4, 5
$m_{overlap}$	Number of maximal co-locations generated by appending more features into co-locations	1 , 1, 5, 10 , 15, 20

Table 5.3: Experimental parameters and their values in experiments.

5.6.2 Influence of the distance threshold on performance

The influence of increasing the distance threshold on the algorithms' performance was first assessed. Results are shown in in Figure 5.9 and 5.10. For dataset-1, in Figure 5.9, for distances smaller than 3 the time for the three algorithms except DM increases exponentially. This is due to the fact that dataset-1 consists of objects with locations from the whole state that are scattered across various cities. For such locally dense but

³<https://github.com/harryckh/ICDE19-co-location>

otherwise sparse data, we observe a substantial change in the neighborhood for smaller distance values up to an extent when the neighborhood starts crossing the city limit. The same effect is visible in the memory requirement of the algorithms ‘DM NBD-list’ and ‘DM NBD-HM’. Further, even the data within the city is not very dense. As a result, the time and space requirement of ‘DM NBD-list’ and ‘DM NBD-HM’ is quite high due to the overhead in maintaining the neighborhood structure. For the baseline, the re-computation of the fraction score takes more time than any variants of the fraction score updating algorithm. DM outperforms all the three algorithms on dataset-1. For dataset-2, in Figure 2, with an increase in distance, time and memory remain constant unlike results in Figure 5.10. This is because dataset-2 consists of all objects within distance 1 and does not include a higher number of objects in the neighborhood as the distance increases. And also there are more features in dataset-2 compared to dataset-1; as a result, ‘DM NBD-list’ outperforms other algorithms in terms of time. ‘DM NBD-HM’ requires more time and memory because it indexes objects according to their feature types and stores the list of objects that belong to a feature separately. DM and the baseline take more time because they have to find the neighborhood objects again and again.

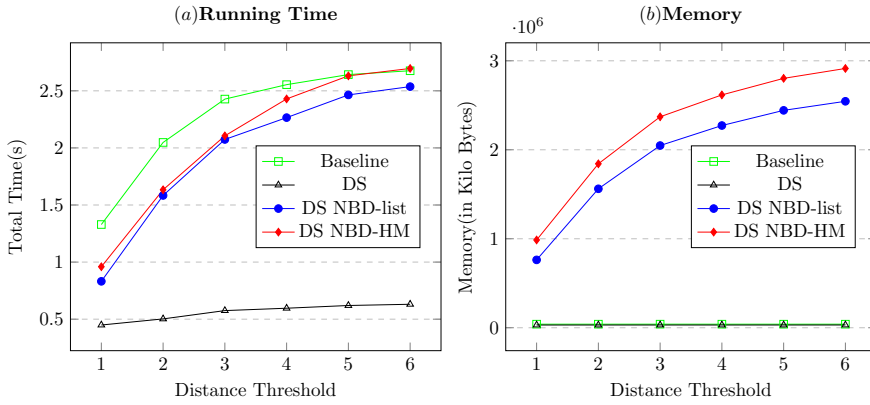


Figure 5.9: Performance with Distance Threshold for Dataset-1

5.6.3 Influence of the window size on performance

The effect of increasing the window size on the algorithms’ performance was also evaluated. Results are shown in Figure 5.11 and 5.12. For dataset-1 in Figure 5.11.a the running time increases with an increase in window size. This is clearly due to the increase in the number of objects in the neighborhood. Figure 5.11.a clarifies that for dataset-1, all the proposed algorithms take less time than the baseline. However, the memory required by DM NBD-list and DM NBD-HM increases exponentially with an increase in window size, as shown in Figure 5.11b. For dataset-2 in Figure 5.12a the running time increases with an increase in the window size. Baseline and DM take more time compared to the other two due to that fact they have to compute the number of

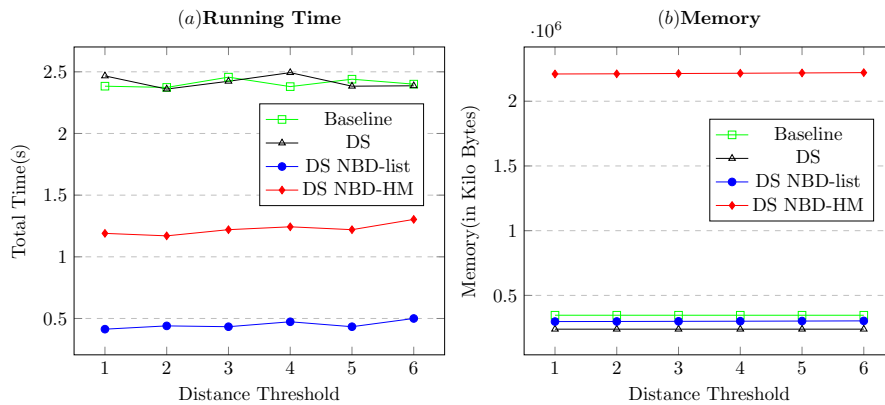


Figure 5.10: Performance with Distance Threshold for Dataset-2

neighborhoods again and again. DM NBD-HM takes much time than DM NBD-list because it has to do indexing for a large number of features in this data. Hence, from Figure 5.12a it is clear that DM NBD-list takes less time than all other algorithms. In Figure 5.12b the memory requirement of DM NBD-HM increases exponentially with an increase in window size because it maintains separate lists for each feature. However, it is constant for all other algorithms.

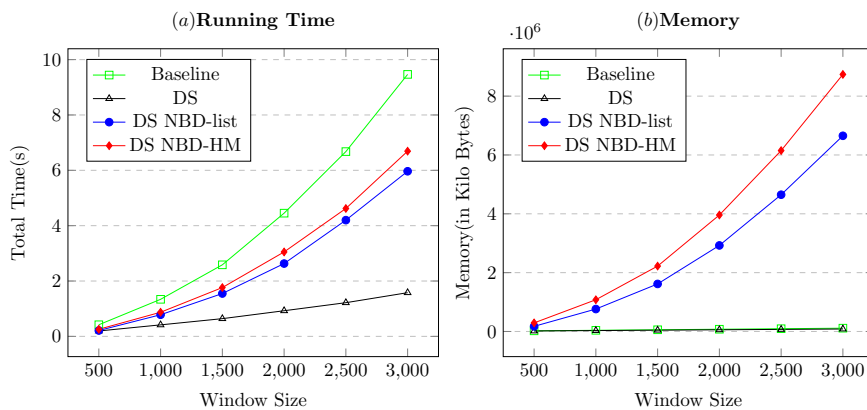


Figure 5.11: Performance with window Size for Dataset 1

5.6.4 Performance with window change rate

The effect of increasing the window change rate on the algorithms' performance was also evaluated. Results are shown in Figure 5.13 and 5.14. For dataset-1 from Figure 5.13a, it is clear that the time increases with an increase in Window change rate. DM is faster than the other algorithms as it calculates neighborhood objects only for the influenced

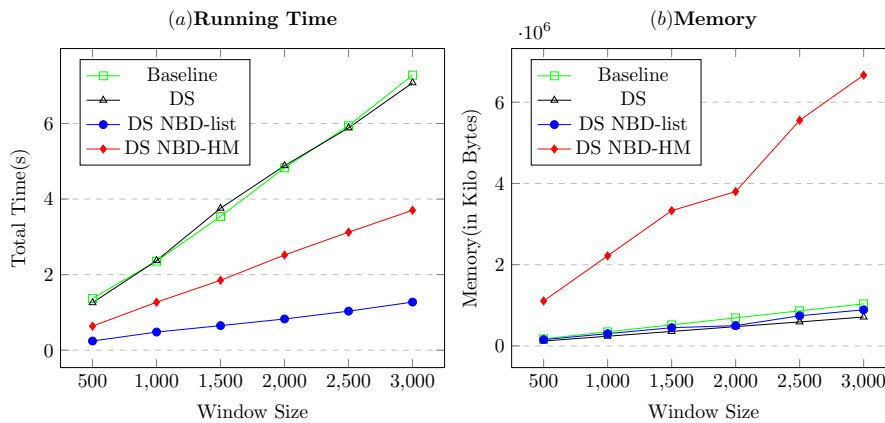


Figure 5.12: Performance with window Size for Dataset 2

objects whereas the baseline to repeated calculations with the arrival of new data. On the other hand, DM NBD-list and DM NBD-HM take more time than DM due to the maintenance cost of neighborhood structures. The memory required by DM NBD-list and DM NBD-HM rises exponentially with the increase in window movements, as shown in Figure 5.13b. This is because memory requirements increase with the increment rate. For dataset-2 in Figure 5.14a, it is observed that runtime increases with dynamic update rate. In this dataset, the computational overhead for computing neighborhood for DM is almost equivalent to that of the baseline. Whereas DM NBD-list and DM NBD-HM are taking less time because these algorithms are storing the neighborhoods. In Figure 5.14b, the memory requirement by DM NBD-HM increases exponentially as the window size is increased because it maintains separate lists for each feature. However, it is constant for all other algorithms.

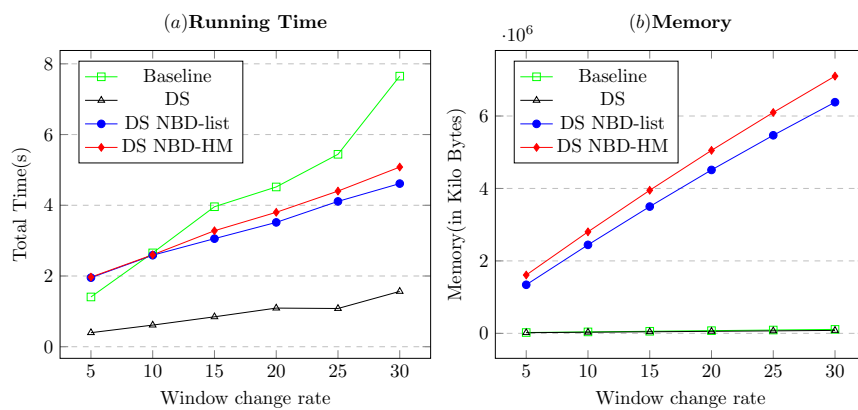


Figure 5.13: Performance with window change rate for Dataset-1

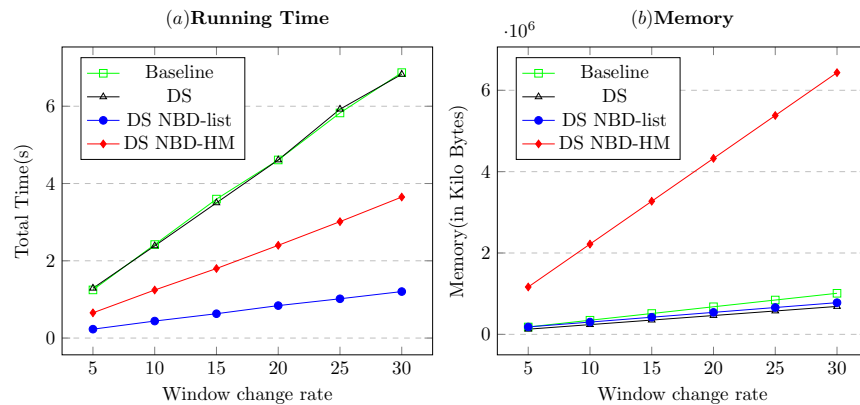
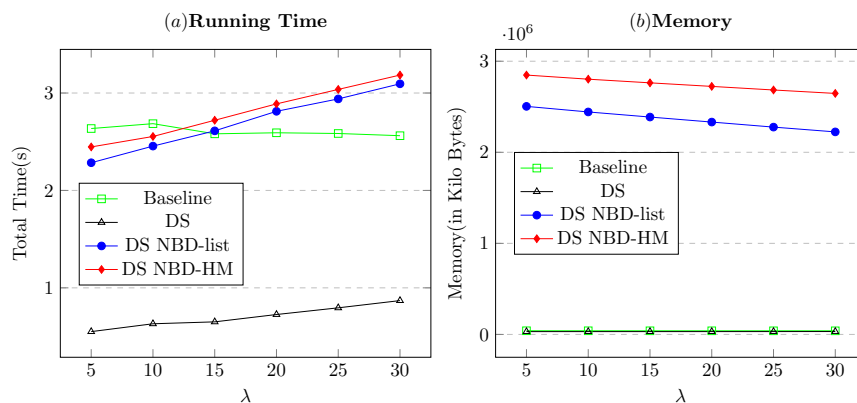


Figure 5.14: Performance with window change rate for Dataset-2

5.6.5 Influence of data change rate on performance

Another experiment examined the performance of the proposed algorithms as data change rate is increased. Results are shown in Figure 5.15 and Figure 5.16. For dataset-1 in Figure 5.15a, it is found that the DM algorithm algorithm takes less time than the baseline algorithm and the other two. The memory required by DM NBD-list and DM NBD-HM is more than the other two, as shown in Figure 5.15b. For dataset-2 in Figure 5.16a DM NBD-list takes less time than all other algorithms because it stores neighborhood objects instead of recomputing them. DM NBD-HM takes a little more time than DM NBD-list because it has to create an index on a huge number of features. DM and the baseline are taking more time as they have to recompute the neighborhood objects. In Figure 5.16b, the memory required by DM NBD-HM is much more than other algorithms because it has to maintain a separate list for each feature.

Figure 5.15: Performance with λ for Dataset-1

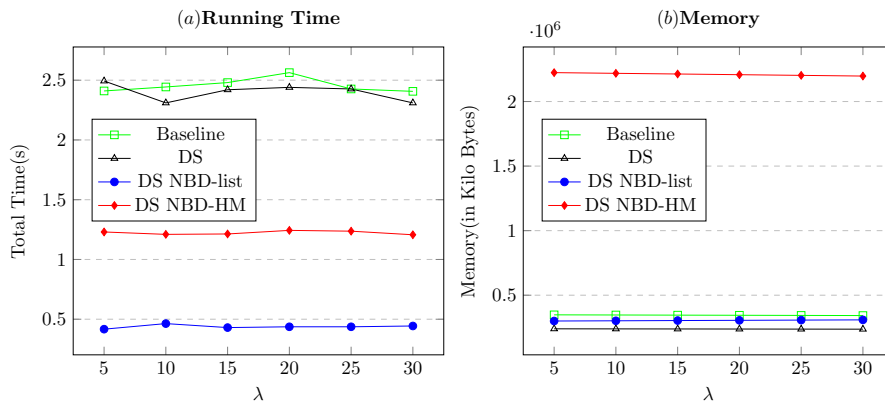


Figure 5.16: Performance with λ for Dataset-2

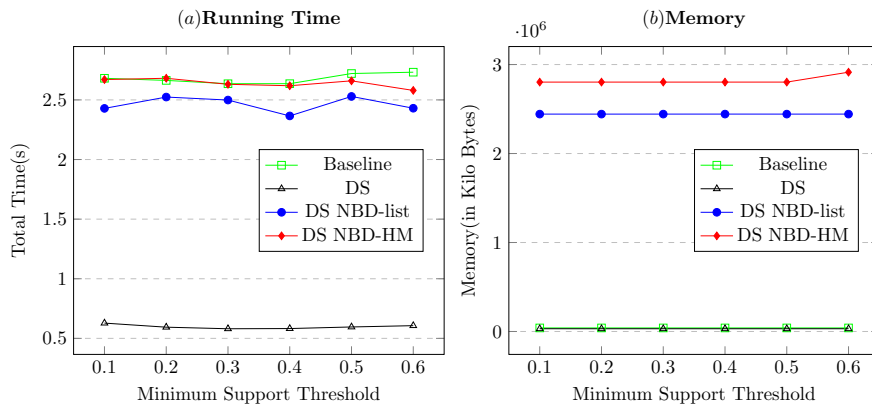


Figure 5.17: Performance with Minimum Support Threshold for Dataset-1

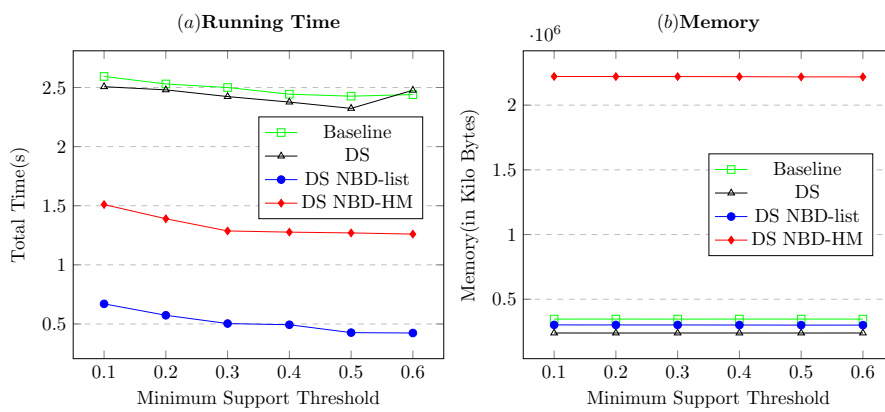


Figure 5.18: Performance with Minimum Support Threshold for Dataset-2

5.6.6 Performance with Minimum Support Threshold

The influence of the minimum support threshold on the performance of the algorithm was assessed. Results are presented in Figure 5.17 and 5.18. For dataset-1 Figure 5.17a shows that the DM algorithm takes less time than the baseline algorithm and the other two. The memory required by DM NBD-list and DM NBD-HM is more than the other two, as shown in Figure 5.17b. For dataset-2, Figure 5.18a shows that DM NBD-list takes less time than the baseline algorithm and the other two. The memory required by DM NBD-HM is much more compared to all other algorithms because it has to maintain a separate list for each feature as shown in Figure 5.18b.

5.6.7 Performance with Damped Window

Though we have previously only discussed the temporal-window model it is not a limitation of the algorithm. It can be easily extended for the damped window model by setting the number deleted objects as $O^d = 0$ and $O^a > 0$.

In this experiment we consider only the addition of objects into windows. We have examined the performance effect of the proposed algorithms as the number of objects being added is increased. Results are depicted in Figure 5.19 and Figure 5.20. For dataset-1, Figure 5.20 shows that the running time slowly increases as number of added objects is increased. DM outperforms other algorithms both in terms of time and memory. For dataset-2 in Figure 5.20a, DM NBD-list takes less time compared to all other algorithms. The memory required by DM NBD-HM is much more compared to all other algorithms because it has to maintain a separate list for each feature as shown in Figure 5.20b.

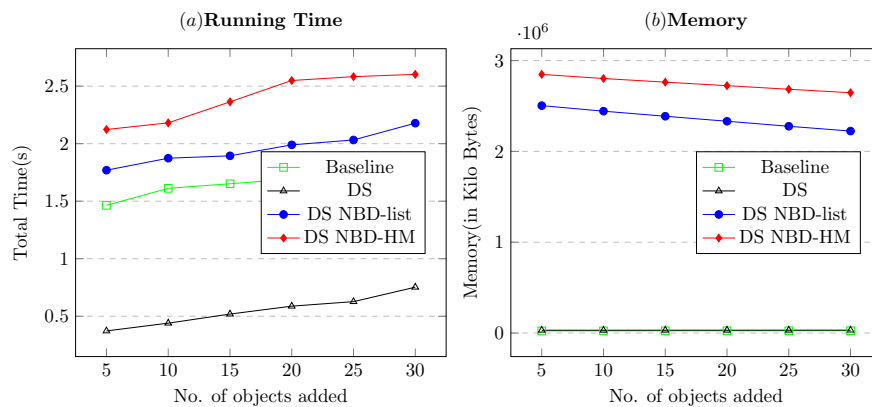


Figure 5.19: Performance with No. of objects added for Dataset -1

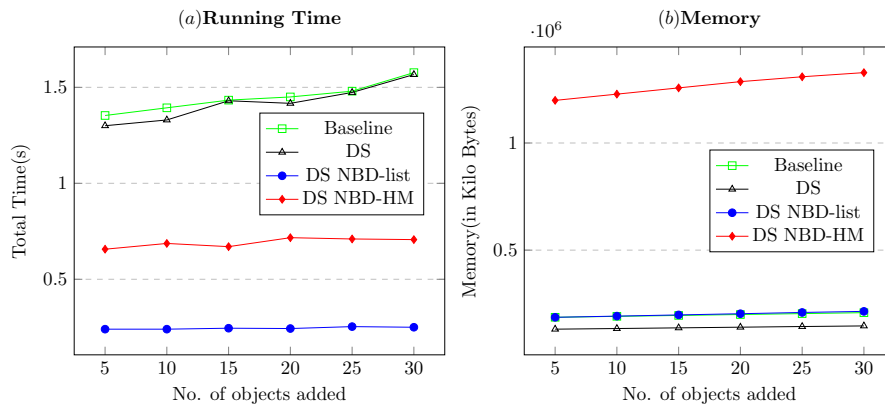
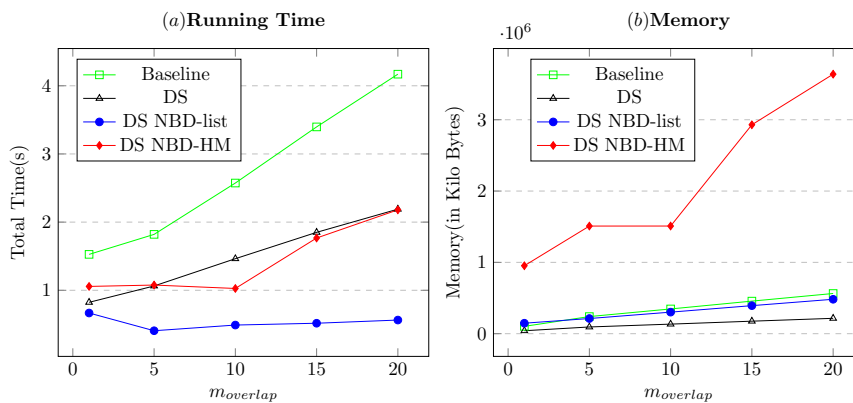


Figure 5.20: Performance with No. of objects added for Dataset-2

5.6.8 Effect of $m_{overlap}$ on synthetic datasets

The performance of the proposed algorithms was also assessed as the $m_{overlap}$ is increased. Results are shown in Figure 5.21. Running times of DM, DM NBD-list and the baseline increase with $m_{overlap}$. However, the running time of DM NBD-list remains constant. Overall, all the proposed algorithms run faster than the baseline. DM NBD-HM consumes more memory than all other algorithms.

Figure 5.21: Effect of $m_{overlap}$ on synthetic Datasets

5.6.9 Effect of m_{clump} on synthetic Datasets

The influence of m_{clump} on the proposed algorithms' performance was also evaluated. Results are shown in Figure 5.22. The running times of all algorithms increase with m_{clump} . This is because the number of objects in the neighborhood of each object

increases. Besides, DM NBD-HM needs orders of magnitude more memory compared to all other algorithms.

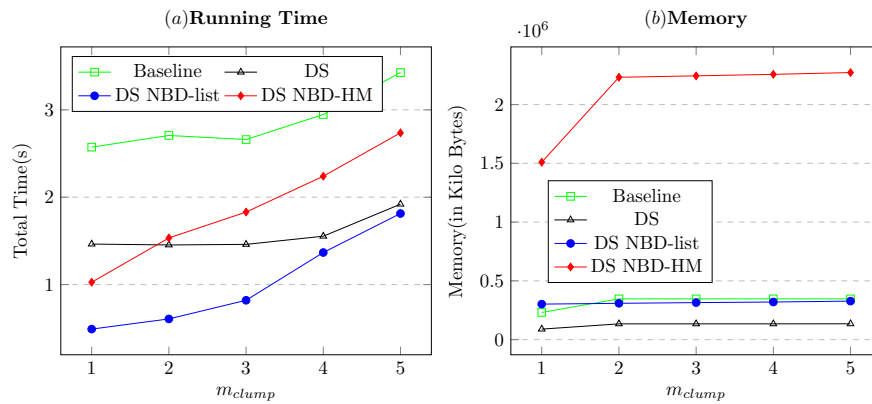


Figure 5.22: Effect of m_{clump} on synthetic Datasets

5.6.10 Effect of λ_2 on synthetic Datasets

In another experiment, the influence of increasing λ_2 on the performance of the algorithms was assessed. Results are shown in Figure 5.23. The performance of all algorithms remains constant when λ_2 is increased, in terms of running time and memory. However all our algorithms outperform the baseline. Besides DM NBD-list runs faster than the two other proposed algorithms. DM takes much less memory than all other algorithms.

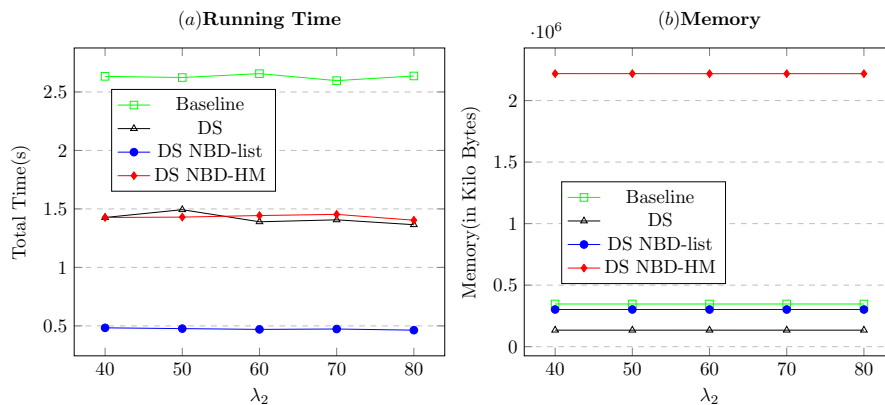


Figure 5.23: Effect of λ_2 on synthetic Datasets

5.7 Performance Evaluation and Analysis

5.7.1 Dynamic Mining (DM):

Strengths:

1. **Efficiency in Time:** DM performs well in terms of time efficiency, particularly when compared to the baseline (see Figure 5.9).
2. **Adaptability:** It efficiently calculates neighborhood objects for influenced objects, contributing to its effectiveness in dynamic scenarios.

Weaknesses:

1. **Memory Usage:** DM may face challenges with memory usage, especially when dealing with large datasets or frequent window movements (see Figure 5.9).
2. **Scalability:** The exponential increase in time with smaller distance values indicates potential scalability issues for locally dense but otherwise sparse datasets (see Figure 5.9).

5.7.2 Dynamic Mining with NBD-list (DM NBD-list):

Strengths:

1. **Time Efficiency:** DM NBD-list performs well in terms of time efficiency, especially compared to DM NBD-HM and the baseline (see Figure 5.11a and Figure 5.13a).
2. **Neighborhood Storage:** Storing neighborhood objects helps avoid unnecessary recomputation, contributing to time efficiency.

Weaknesses:

1. **Memory Usage:** Like DM, DM NBD-list may face challenges with memory usage, particularly as the window size increases (see Figure 5.11b).
2. **Scalability:** Exponential growth in memory requirements with increasing window size suggests scalability concerns (see Figure 5.11b).

5.7.3 Dynamic Mining with NBD-Hash map (DM NBD-HM):

Strengths:

1. **Feature Indexing:** DM NBD-HM uses feature indexing, which can be beneficial in scenarios with a large number of features.
2. **Neighborhood Storage:** Similar to DM NBD-list, storing neighborhood objects helps avoid unnecessary recomputation.

Weaknesses:

1. **Memory Usage:** DM NBD-HM exhibits high memory usage, especially due to maintaining separate lists for each feature (see Figure 5.12b).
2. **Computational Overhead:** The feature indexing, while useful, can lead to higher computational overhead, impacting performance.

5.7.4 Overall Considerations:

1. **Baseline (Fraction Score Algorithm):**

- **Strengths:** Provides a benchmark for comparison, especially for fraction score-related evaluations.
- **Weaknesses:** May not be optimized for dynamic settings, and re-computing fraction scores periodically can become computationally expensive.

2. **General Observations:**

- **Scalability Concerns:** Several implementations exhibit potential scalability concerns, particularly with memory requirements and computational time for larger datasets or frequent updates (see various figures).
- **Trade-offs:** There seems to be a trade-off between time efficiency and memory usage, and the choice of the algorithm may depend on the specific characteristics of the dataset and the dynamic scenario.

5.8 Conclusion

This work uses a temporal-window framework to address the problem of co-location pattern mining for dynamic data. The effect of the change of objects over the fraction score of the neighbouring objects is theoretically analyzed and used to propose an observation-based technique UpFS for updating the co-location patterns. The proposed technique updates fraction scores by reducing candidate patterns and updating the required neighborhood counts from the previous state, i.e., the previously computed patterns and the old data state, and the changes in the data. The three implementations of UpFS analyze the space and time complexity tradeoff for different datasets. The performance of the proposed algorithms is compared with a baseline approach by varying

multiple parameters and dataset characteristics such as the window size, number of updates, and distance threshold on both real and synthetic datasets. It is shown that the proposed technique for mining co-location for dynamic data performs much better than the baseline approach. As a possible extension to this work, a distributed UpFS mining algorithm can be designed to increase the dynamic data processing efficiency using distributed fault-tolerant stream processing systems like Apache Spark streaming. Indeed, considering statistical significance could provide a more comprehensive evaluation of the proposed approach.

Conclusion

In this thesis, three pertinent issues in colocation pattern mining were studied, each addressing a specific challenge in computing colocation patterns from spatial data.

The colocation mining techniques work with a single distance threshold value. The choice of this distance threshold is highly data-dependent. This makes it challenging and time-consuming for the user to find a suitable distance value. As a first work, We have addressed this issue by introducing a computational framework for prevalent colocation mining over a distance range query. As per our knowledge, this is the first work that discusses the issue of distance range query over colocation patterns. First, a Naïve approach was presented as a basic framework for the problem. This helps in identifying the computational challenges leading to the proposal of an improved method called *RangeInc – Mining*. The *RangeInc – Mining* employs incremental mining over the distance range by utilizing the subset relationship between the colocations of the consecutive distances and updating the colocations of the next stage from the previous stage. Next, the critical distance of colocation patterns was defined. By leveraging the structural properties of colocations, an efficient technique for computing critical distances was developed. This, in turn, led to the efficient single-pass range query algorithm, named *Range – CoMine*. The performance of these algorithms was experimentally evaluated using both real-world and synthetic datasets, and it was observed that *Range – CoMine* outperformed the other approaches in terms of time complexity and scalability. The possible future extension in this domain can be addressing the issues of distance range query over road networks, for dynamic data, etc. The selection of a minimum prevalence threshold is also challenging for a user. It can also be looked into that how this issue of finding a suitable prevalence threshold can be addressed. One possible approach can be exploring top-k colocation pattern mining with a distance range.

The second work focused on mining high-utility subgraph (colocation) patterns from a graph database using distributed approaches. To address the issue of the utility measure not being anti-monotonic, a function was defined to obtain the upper-bound utility of a subgraph pattern. This enabled a systematic search for high-utility subgraph patterns and reduction of the search space. Various optimization strategies were proposed, including the use of bloom filters to prune non-candidate patterns and tighten the upper-bound utility estimate. Additionally, a Schimmy design was introduced to reduce

communication of graph data between nodes, and the removal of embeddings from the pattern object was suggested to minimize data communication. The proposed solution was shown to be flexible, relying on primitive operations available on most distributed platforms. Experimental studies demonstrated the efficiency of the solution; however, it was noted that the upper-bound utility estimate of a pattern was still relatively loose. As a future direction, the authors planned to explore new mechanisms to improve the tightness of the upper-bound utility estimate.

Classical co-location mining methods are primarily tailored for static data and lack efficiency in handling dynamic updates. In the third work, we focus on extracting co-location patterns from dynamic data, where the dataset undergoes changes over time by adding and removing objects. In this work, we tackled the problem of mining co-location patterns for dynamic data using a temporal window. A theoretical framework was presented for fraction-score updation, and algorithms were developed to effectively update fraction-scores by modifying neighborhood counts due to object additions or deletions. The performance of the proposed algorithms was compared to a baseline approach using various parameters and dataset characteristics, including window size, number of updates, and distance threshold. The experimental results demonstrated the efficiency of the proposed co-location mining algorithm for dynamic data updates, outperforming the baseline approach. For possible future extensions to this problem, increasing the efficiency of dynamic data processing can be explored. Specifically, distributed UpFS mining algorithms using fault-tolerant stream processing systems like Apache Spark streaming can be designed to increase computational complexity.

The thesis made significant contributions to the field of colocation pattern mining through the introduction of novel approaches and the exploration of various challenges in prevalent colocation mining, high-utility subgraph pattern mining, and co-location pattern mining for dynamic data. The experimental evaluations provided evidence of the effectiveness and efficiency of the proposed solutions. The future extensions and plans outlined in each work offered valuable directions for further research and development in these areas.

List of Publications

1. **Srikanth Baride**, Anuj S. Saxena, and Vikram Goyal. "Efficiently Mining Colocation Patterns for Range Query." Big Data Research, (Feb 2023).
2. Alind Khare, Vikram Goyal, **Srikanth Baride**, Sushil K. Prasad, Michael McDermott and Dhara Shah. "Distributed Algorithm for High-Utility Subgraph Pattern Mining Over Big Data Platforms." HiPC, (Dec 2017).
3. **Srikanth Baride**, Anuj S. Saxena, and Vikram Goyal. "Mining Co-location Patterns on Dynamic Data." Data Mining and Knowledge Discovery, [Under Review].

Bibliography

- [1] *Putting Spark to Use: Fast In-Memory Computing for Your Big Data Applications*. <https://blog.cloudera.com>.
- [2] *San francisco crime incidents*. <https://data.sfgov.org/>.
- [3] J. S. YOO AND H. VASUDEVAN, *Effectively updating co-location patterns in evolving spatial databases*, in *Patterns 2014*, 2014, pp. 96–99.
- [4] R. AGRAWAL AND R. SRIKANT, *Fast Algorithms for Mining Association Rules in Large Databases*, in *20th International Conference on VLDB*, 1994, pp. 487–499.
- [5] W. ANDRZEJEWSKI AND P. BOINSKI, *Parallel approach to incremental co-location pattern mining*, *Information Sciences*, 496 (2019), pp. 485 – 505.
- [6] S. BARUA AND J. SANDER, *Mining statistically significant co-location and segregation patterns*, *IEEE Transactions on Knowledge and Data Engineering*, 26 (2014), pp. 1185–1199.
- [7] S. BARUA AND J. SANDER, *Mining statistically sound co-location patterns at multiple distances*, in *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, 2014.
- [8] M. A. BHUIYAN AND M. A. HASAN, *An Iterative MapReduce Based Frequent Subgraph Mining Algorithm*, *IEEE TKDE*, 27 (2015), pp. 608–620.
- [9] H. K. CHAN, C. LONG, D. YAN, AND R. C. WONG, *Fraction-score: A new support measure for co-location pattern mining*, in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 1514–1525.
- [10] Y.-I. CHANG, C.-C. WU, AND C.-Y. YEN, *Mining co-location patterns in incremental spatial databases*, in *2022 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2022, pp. 141–148.
- [11] Y. CHEN, X. CHEN, Z. LIU, AND X. LI, *Understanding the spatial organization of urban functions based on co-location patterns mining: A comparative analysis for 25 chinese cities*, *Cities*, 97 (2020), p. 102563.
- [12] S. DAWAR AND V. GOYAL, *UP-Hist tree: An efficient data structure for mining high utility patterns from transaction databases*, in *IEEE IDEAS*, 2015, pp. 56–61.

- [13] M. ELSEIDY, E. ABDELHAMID, S. SKIADOPOULOS, AND P. KALNIS, *GraMi: Frequent Subgraph and Pattern Mining in a Single Large Graph*, Proc. VLDB Endow., 7 (2014), pp. 517–528.
- [14] Y. FANG, L. WANG, AND T. HU, *Spatial co-location pattern mining based on density peaks clustering and fuzzy theory*, in Web and Big Data, Y. Cai, Y. Ishikawa, and J. Xu, eds., 2018, pp. 298–305.
- [15] A. GUTTMAN, *R-trees: A dynamic index structure for spatial searching*, SIGMOD Rec., 14 (1984), p. 47–57.
- [16] J. HE, Q. HE, F. QIAN, AND Q. CHEN, *Incremental maintenance of discovered spatial colocation patterns*, in 2008 IEEE International Conference on Data Mining Workshops, 2008, pp. 399–407.
- [17] Z. HE, M. DENG, Z. XIE, L. WU, Z. CHEN, AND T. PEI, *Discovering the joint influence of urban facilities on crime occurrence using spatial co-location pattern mining*, Cities.
- [18] S. HILL, B. SRICHANDAN, AND R. SUNDERRAMAN, *An Iterative MapReduce Approach to Frequent Subgraph Mining in Biological Datasets*, in ACM Conference on Bioinformatics, Computational Biology and Biomedicine, 2012, pp. 661–666.
- [19] L. B. HOLDER AND D. J. COOK, *Discovery of inexact concepts from structural data*, IEEE Tran. on Knowl. and Data Eng., 5 (1993), pp. 992–994.
- [20] S.-M. HU AND X.-L. QIN, *Mining co-location patterns with quantitative semantics*, Journal of Information Science, 34 (2008), pp. 287–302.
- [21] X. HU, G. WANG, AND J. DUAN, *Mining maximal dynamic spatial colocation patterns*, IEEE Transactions on Neural Networks and Learning Systems, (2020), p. 1–11.
- [22] J. HUAN, W. WANG, AND J. PRINS, *Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism*, in IEEE ICDM Conf., ICDM '03, 2003, pp. 549–.
- [23] Y. HUANG, J. PEI, AND H. XIONG, *Mining co-location patterns with rare events from spatial data sets*, GeoInformatica, 10 (2006), pp. 239–260.
- [24] Y. HUANG, S. SHEKHAR, AND H. XIONG, *Discovering colocation patterns from spatial data sets: A general approach*, IEEE Trans. on Knowl. and Data Eng., 16 (2004), p. 1472–1485.
- [25] Y. HUANG, H. XIONG, S. SHEKHAR, AND J. PEI, *Mining confident co-location rules without a support threshold*, Proceedings of the ACM Symposium on Applied Computing, (2003).

- [26] A. INOKUCHI, T. WASHIO, AND H. MOTODA, *An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data*, in 4th European Conference on Principles of Data Mining and Knowledge Discovery, 2000, pp. 13–23.
- [27] C. JIANG, F. COENEN, AND M. ZITO, *Frequent Sub-graph Mining on Edge Weighted Graphs*, in 12th International Conference on DaWaK, 2010, pp. 77–88.
- [28] L. JUNYI, W. LIZHEN, AND C. HONGMEI, *dgridtopk-fcpm: A top-k spatial co-location pattern mining algorithm based on fuzzy theory and d-grids*, Journal of Tsinghua University(Science and Technology), (2021).
- [29] K. KOPERSKI, J. ADHIKARY, AND J. HAN, *Spatial data mining: progress and challenges survey paper*, in Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Montreal, Canada, Citeseer, 1996, pp. 1–10.
- [30] P.-F. KUO AND D. LORD, *A visual approach for defining the spatial relationships among crashes, crimes, and alcohol retailers: Applying the color mixing theorem to define the colocation pattern of multiple variables*, Accident Analysis Prevention, 154 (2021), p. 106062.
- [31] H. KURAMOCHI, W. WU, AND K. KAWAMOTO, *Prediction of the behaviors of h2s and hcl during gasification of selected residual biomass fuels by equilibrium calculation*, Fuel, 84 (2005), pp. 377–387.
- [32] M. KURAMOCHI AND G. KARYPIS, *Frequent Subgraph Discovery*, in IEEE ICDM Conf., 2001, pp. 313–320.
- [33] G. LEE AND U. YUN, *Mining Weighted Frequent Sub-graphs with Weight and Support Affinities*, 6th International Workshop, MIWAI 2012, 7694 (2012), pp. 224–235.
- [34] S. LEE, C. WU, AND Y. CHEN, *Incremental maintenance of topological patterns in spatial-temporal database*, in 2013 IEEE 13th International Conference on Data Mining Workshops, dec 2011, pp. 853–860.
- [35] L. LEI, L. WANG, AND X. WANG, *Mining spatial co-location patterns by the fuzzy technology*, in 2019 IEEE International Conference on Big Knowledge (ICBK), 2019, pp. 129–136.
- [36] J. LI, A. ADILMAGAMBETOV, M. S. MOHOMED JABBAR, O. R. ZAÏANE, A. OSORNIO-VARGAS, AND O. WINE, *On discovering co-location patterns in datasets: A case study of pollutants and child cancers*, Geoinformatica, (2016).
- [37] Z. LI, K. LEE, B. ZHENG, W.-C. LEE, D. LEE, AND X. WANG, *Ir-tree: An efficient index for geographic document search*, Knowledge and Data Engineering, IEEE Transactions on, 23 (2011), pp. 585 – 599.

- [38] W. LIAN, D. CHEUNG, AND S. YIU, *An efficient algorithm for finding dense regions for mining quantitative association rules*, *Computers Mathematics With Applications - COMPUT MATH APPL*, 50 (2005), pp. 471–490.
- [39] W. LIN, X. XIAO, AND G. GHINITA, *Large-scale frequent subgraph mining in MapReduce*, in 30th ICDM Conf., 2014, pp. 844–855.
- [40] W. LIN, X. XIAO, AND G. GHINITA, *Large-scale frequent subgraph mining in MapReduce*, ICDE Conf., (2014), pp. 844–855.
- [41] Z. LIN AND S. LIM, *Optimal candidate generation in spatial co-location mining*, in Proceedings of the 2009 ACM symposium on applied computing, 2009, pp. 1441–1445.
- [42] J. B. LING LI, JIANQUAN CHENG AND X. MAI, *Geographically and temporally weighted co-location quotient: an analysis of spatiotemporal crime patterns in greater manchester*, *International Journal of Geographical Information Science*, 36 (2022), pp. 918–942.
- [43] W. LIU, Q. LIU, M. DENG, J. CAI, AND J. YANG, *Discovery of statistically significant regional co-location patterns on urban road networks*, *International Journal of Geographical Information Science*, (2021), pp. 1–24.
- [44] J. LU, L. WANG, Q. XIAO, AND YU SHANG, *Incremental mining of co-locations from spatial database*, in 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2015, pp. 612–617.
- [45] W. LU, G. CHEN, A. K. H. TUNG, AND F. ZHAO, *Efficiently extracting frequent subgraphs using MapReduce*, in IEEE International Conference on Big Data, Oct 2013, pp. 639–647.
- [46] Y. MA, J. LU, AND D. YANG, *Mining evolving spatial co-location patterns from spatio-temporal databases*, in 2022 IEEE International Conference on Big Data and Smart Computing (BigComp), 2022, pp. 129–136.
- [47] J. MATA VÁZQUEZ, J. MACÍAS, AND J. RIQUELME, *An evolutionary algorithm to discover numeric association rules.*, 01 2002, pp. 590–594.
- [48] S. V. MEHTA, S. SODHANI, AND D. PATEL, *Spatial co-location pattern mining - a new perspective using graph database*, 2018.
- [49] C. W. Y. G. MENGJIE ZHOU, TINGHUA AI AND N. WANG, *A visualization approach for discovering colocation patterns*, *International Journal of Geographical Information Science*, 33 (2019), pp. 567–592.
- [50] Y. MORIMOTO, *Mining frequent neighboring class sets in spatial databases*, in Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '01, Association for Computing Machinery, 2001, p. 353–358.

- [51] F. QIAN, Q. HE, AND J. HE, *Mining spatial co-location patterns with dynamic neighborhood constraint*, in Machine Learning and Knowledge Discovery in Databases, W. Buntine, M. Grobelnik, D. Mladenić, and J. Shawe-Taylor, eds., Berlin, Heidelberg, 2009, Springer Berlin Heidelberg, pp. 238–253.
- [52] D. SEEBACHER, *Visual Analytics of Spatial Events : Methods for the Interactive Analysis of Spatio-Temporal Data Abstractions*, PhD thesis, Universität Konstanz, Konstanz, 2021.
- [53] J. SHAO AND A. TZIATZIOS, *Mining range associations for classification and characterization*, Data Knowledge Engineering, 118 (2018), pp. 92–106.
- [54] S. SHEKHAR AND Y. HUANG, *Discovering spatial co-location patterns: A summary of results*, in Advances in Spatial and Temporal Databases, 7th International Symposium, SSTD 2001, Redondo Beach, CA, USA, July 12-15, 2001, Proceedings, C. S. Jensen, M. Schneider, B. Seeger, and V. J. Tsotras, eds., vol. 2121 of Lecture Notes in Computer Science, Springer, 2001, pp. 236–256.
- [55] N. TALUKDER AND M. J. ZAKI, *A Distributed Approach for Graph Mining in Massive Networks*, Data Min. Knowl. Discov., 30 (2016), pp. 1024–1052.
- [56] R. TANG, G. HOU, AND R. DU, *Isolated or colocated? exploring the spatio-temporal evolution pattern and influencing factors of the attractiveness of residential areas to restaurants in the central urban area*, ISPRS International Journal of Geo-Information, 12 (2023).
- [57] F. TAO, F. MURTAGH, AND M. FARID, *Weighted Association Rule Mining Using Weighted Support and Significance Framework*, in Ninth International Conf. on Know. Discovery and Data Min., 2003, pp. 661–666.
- [58] V. TRAN, L. WANG, AND H. CHEN, *A spatial co-location pattern mining algorithm without distance thresholds*, in 2019 IEEE International Conference on Big Knowledge (ICBK), 2019, pp. 242–249.
- [59] V. TRAN, L. WANG, AND L. ZHOU, *A spatial co-location pattern mining framework insensitive to prevalence thresholds based on overlapping cliques*, Distributed and Parallel Databases, (2021), pp. 1–38.
- [60] J. WANG, Y. LI, Z. ZHOU, C. WANG, Y. HOU, L. ZHANG, X. XUE, M. KAMP, X. L. ZHANG, AND S. CHEN, *When, where and how does it fail? a spatial-temporal visual analytics approach for interpretable object detection in autonomous driving*, IEEE Transactions on Visualization and Computer Graphics, 29 (2023), pp. 5033–5049.
- [61] M. WANG, L. WANG, Y. QIAN, AND D. FANG, *Incremental mining of spatial co-location patterns based on the fuzzy neighborhood relationship*, in Fuzzy Systems and Data Mining V, 2019, pp. 652–660.

- [62] M. WANG, L. WANG, AND L. ZHOU, *Spatial colocation pattern mining with the maximum membership threshold*, In Fuzzy Systems and Data Mining V, (2019), pp. 1092–1100.
- [63] X. WANG, L. LEI, L. WANG, P. YANG, AND H. CHEN, *Spatial co-location pattern discovery incorporating fuzzy theory*, IEEE Transactions on Fuzzy Systems, (2021).
- [64] XIAOXUAN WANG AND L. WANG, *Incremental mining of high utility co-locations from spatial database*, in 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), 2017, pp. 215–222.
- [65] X. YAN AND J. HAN, *gSpan: graph-based substructure pattern mining*, in IEEE ICDM Conf., 2002, pp. 721–724.
- [66] X. YAO, L. CHEN, C. WEN, L. PENG, L. YANG, T. CHI, X. WANG, AND W. YU, *A spatial co-location mining algorithm that includes adaptive proximity improvements and distant instance references*, International Journal of Geographical Information Science, 32 (2018), pp. 980–1005.
- [67] X. YAO, L. PENG, AND T. CHI, *A co-location pattern-mining algorithm with a density-weighted distance thresholding consideration*, Information Sciences, 396 (2017).
- [68] X. YAO, J. XUFENG, D. WANG, Y. LINA, L. PENG, AND T. CHI, *Efficiently mining maximal co-locations in a spatial continuous field under directed road networks*, Information Sciences, 542 (2020).
- [69] J. YIN, Z. ZHENG, AND L. CAO, *Uspan: An efficient algorithm for mining high utility sequential patterns*, in 18th International Conf. on Knowl Disc. and Data Min., KDD '12, 2012, pp. 660–668.
- [70] J. S. YOO AND S. SHEKHAR, *A joinless approach for mining spatial colocation patterns*, IEEE Trans. on Knowl. and Data Eng., 18 (2006), p. 1323–1337.
- [71] J. S. YOO, S. SHEKHAR, AND M. CELIK, *A join-less approach for co-location pattern mining: a summary of results*, in Fifth IEEE International Conference on Data Mining (ICDM'05), 2005.
- [72] J. S. YOO, S. SHEKHAR, J. SMITH, AND J. P. KUMQUAT, *A partial join approach for mining co-location patterns*, in Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems, 2004, p. 241–249.
- [73] U. YOUSAF, M. ASIF, S. AHMED, N. TAHIR, A. IRSHAD, A. A. GARDEZI, M. SHAFIQ, J.-G. CHOI, AND H. HAMAM, *Identification and visualization of spatial and temporal trends in textile industry.*, Computers, Materials & Continua, 74 (2023).

- [74] W. YU, T. AI, Y. HE, AND S. SHAO, *Spatial co-location pattern mining of facility points-of-interest improved by network neighborhood and distance decay effects*, International Journal of Geographical Information Science, (2017).
- [75] Z. YU, *A general framework for mining co-location patterns in spatial databases*, Journal of Ambient Intelligence and Humanized Computing, 5 (2014), pp. 51–64.
- [76] H. ZHANG, X. ZHOU, G. TANG, X. ZHANG, J. QIN, AND L. XIONG, *Detecting colocation flow patterns in the geographical interaction data*, Geographical Analysis, (2021).
- [77] G. ZHOU, *Data Mining for Co-location Patterns: Principles and Applications*, CRC Press, 2022.
- [78] M. ZHOU, T. AI, G. ZHOU, AND W. HU, *A visualization method for mining colocation patterns constrained by a road network*, IEEE Access, 8 (2020), pp. 51933–51944.
- [79] S. ZHOU, L. WANG, AND P. WU, *The coupling co-location pattern: A new spatial pattern for spatial data sets*, Frontiers in Artificial Intelligence and Applications, 331 (2020), pp. 290 – 303.