EAI Endorsed Transactions
on Scalable Information Systems Research Article EAI.EU

Finding Multidimensional Constraint Reachable Paths for Attributed Graphs

Bhargavi B.[1], K. Swarupa Rani[2,*], and Arunjyoti Neog[3]

[1]School of Computer and Information Sciences, University of Hyderabad, Hyderabad, Telangana, India
[2]School of Computer and Information Sciences, University of Hyderabad, Hyderabad, Telangana, India
[3]Cognizant Technology Solutions, DLF, Hyderabad

Abstract

A graph acts as a powerful modelling tool to represent complex relationships between objects in the big data era. Given two vertices, vertex and edge constraints, the multidimensional constraint reachable (MCR) paths problem finds the path between the given vertices that match the user-specified constraints. A significant challenge is to store the graph topology and attribute information while constructing a reachability index. We propose an optimized hashing-based heuristic search technique to address this challenge while solving the multidimensional constraint reachability queries. In the proposed technique, we optimize hashing and recommend an efficient clustering technique based on matrix factorization. We further extend the heuristic search technique to improve the accuracy. We experimentally prove that our proposed techniques are scalable and accurate on real and synthetic datasets. Our proposed extended heuristic search technique is able to achieve an average execution time of 0.17 seconds and 2.55 seconds on MCR true queries with vertex and edge constraints for Robots and Twitter datasets respectively.

Received on 07 November 2021; accepted on 04 August 2022; published on 22 August 2022

Keywords: hashing, attributed graph, matrix factorization, constraint reachability.

doi:10.4108/eetsis.v9i4.2581

1. Introduction

Graph mining is the process of extracting useful knowledge from graphs. Here, graphs are used to model the data. Some of the important operations of graph mining include extracting subgraphs [45], finding the reachability satisfying the given constraints and detecting the communities in a graph. Graph reachability is one of the basic operations that finds the existence of paths between the vertices of the given graph. However, in real-time, some queries require certain constraints to be satisfied while finding the reachability of the graph. The constraints are usually the conditions on the values of vertex attributes or edge attributes or both. An attributed graph is a graph that stores attribute information of vertices and edges. This attributed graph acts as an efficient modeling tool to represent information networks [9, 30]. Figure 1 illustrates an attributed graph with the vertex labels, edge labels, vertex attributes and edge attributes. The description of Fig 1 is as follows: (i) vertex label (dark shaded circles) with values such as the name of the author, paper details and affiliation details like university or organization; (ii) vertex attributes (rectangular box) having values such as state and location (categorical values), keyword (categorical values), age (numerical values) and (iii) edge attribute (shaded rectangular box) having values such as volume and issue (numerical values), order (numerical values), *studentOf* (boolean value) (iv) edge labels having values such as *knows*, *published*, *authorOf*, *affiliatedTo* and *citedBy*. For instance, consider the vertex label 'Paper1'; its vertex attributes include 'keyword' whose value is 'graph'. In general, in the real scenario of social networks, the vertex label denotes the name of a person or organization of the user. The edge labels include relationships like *friendOf*, *colleagueOf* or *supervisorOf*.

A multidimensional constraint reachability query finds the existence of a path from the source vertex

*Corresponding author. Email: swarupacs@uohyd.ac.in

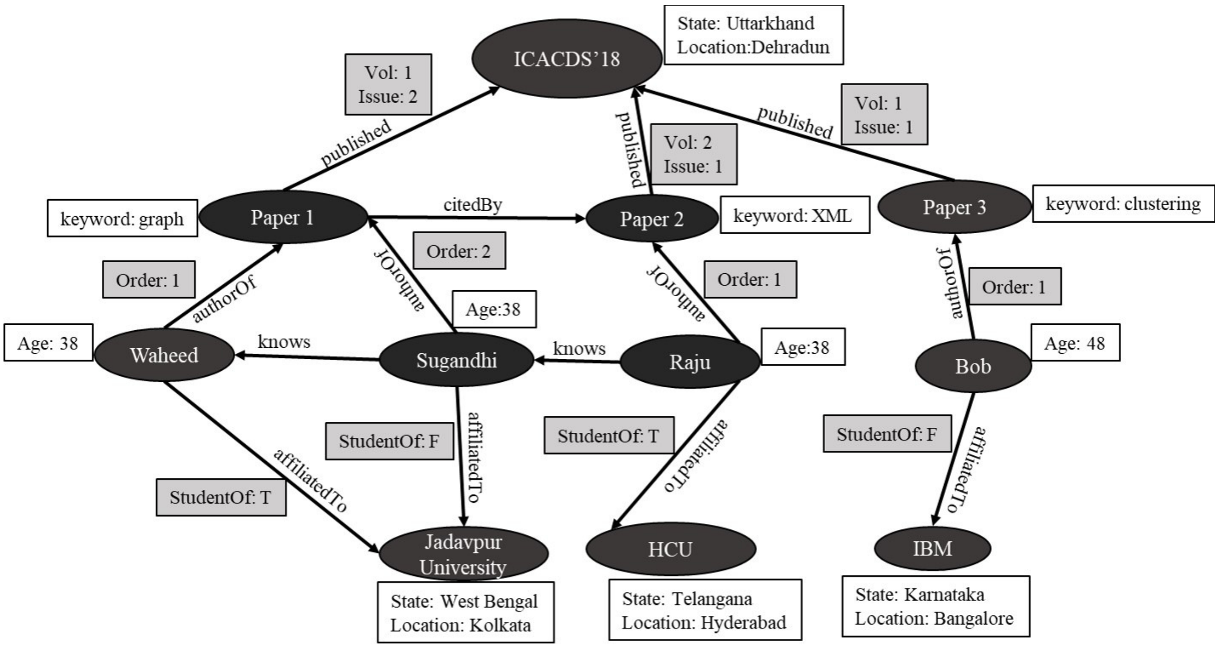


Figure (1). Example of an attributed graph

to the destination vertex satisfying the given attribute constraints. In this paper, we use MCR query, to denote the term **M**ultidimens-ional **C**onstraint **R**eachability query.

For instance, for the given attributed graph G of Figure 1, the query is to find if there is a path from Raju to Waheed whose *Age* is 38 through edge labels *knows*. The vertex constraint is *Age*=38 and the edge constraint is *knows*. From the figure 1, we find that the path exists between Raju and Waheed satisfying the given constraints. Hence, it returns *true* for the given MCR query.

One of the challenges of constraint reachability is that we need to store both graph topology and attribute information while indexing the reachability. Another challenge is that there is no prior information of constraints before query processing. This problem is applicable for many real-time information networks like social networks, transportation networks and metabolic networks. These observations motivate us to find a faster and efficient solution to solve multidimensional constraint reachability queries.

Duncan Yung et al. [9] developed a constraint verification approach to solve MCR queries to find only the path's existence but not the resultant path. They also implemented a heuristic search technique that offered direct passage across graph regions which are likely to satisfy attribute constraints from source to destination. The heuristic search involved the construction of a super graph. They used clustering based on BFS by choosing a random subset of vertices and their traversal forming clusters.

We observed from the current state-of-the-art literature ([2, 9, 15, 16]) on constraint reachability and attribute clustering techniques that we can further optimize the hashing through the Murmur hash function, which has least collision. We also observed the need to identify an efficient clustering technique that considers both graph topology and attributes information while clustering. Furthermore, we observed that there is a scope to extend the problem of MCR queries to derive the resultant paths.

We enhanced the heuristic search [9] by using optimized hashing to handle multidimensional attributes and recommended to apply an efficient clustering technique based on matrix factorization. We further improved the clustering technique by applying gap-statistic [26] to detect the number of clusters for efficient supergraph construction. In addition, we extended the problem of MCR queries by finding not only the existence of the path but also the resultant paths.

Our proposed solution is based on an improved heuristic search that considers both graph topology and attribute information while creating super graph for the given attributed graph. Thus, we can solve the constraint reachability queries faster for even large attributed graphs.

1.1. Assumptions

The assumptions in this paper include

1. We assume that the values of vertex attributes and edge attributes are single and discrete.

2. We assume that reachability can be found along the super path of the super graph.

The assumptions are formed in order to simplify the constrained reachability problem and efficiently find the constrained reachable paths.

1.2. Contributions

The main contributions in this paper include

- We performed a comprehensive literature survey on recent structural and attributed graph clustering techniques [2, 4, 6, 8, 16, 32–34] and constrained reachability techniques described in Table 2.
- We identified and improved the structural and attributed graph clustering technique [2] based on matrix factorization and applied it during super graph construction to solve multidimensional constraint reachability queries.
- We solved the multidimensional constrained reachability queries by computing the path information instead of finding the existence of paths [9].
- We proposed an extended heuristic search technique to reduce the false negative outcomes. We compared our proposed techniques to the existing techniques [9] to solve multidimensional constraint reachability queries on real and synthetic datasets.

Section 2 describes the terminologies and problem statement. In section 3, we describe the literature on attributed graphs, the constrained reachability techniques and attributed graph clustering techniques. Section 4 clearly emphasizes our proposed approach with examples. In section 5, we describe our proposed extended heuristic search technique. Section 6 describes the experiments and evaluation of our proposed techniques. In section 7, we provide conclusion and scope for further research.

2. Preliminaries

Definition 1. (Attributed Graph) " *An attributed graph, G, is a graph denoted as G=(V, E, V_a, E_a), where V is a set of vertices, E ⊆ V × V is a set of edges, V_a is a set of vertex-specific attributes and E_a is a set of edge-specific attributes*" [30].

Let V_a = (V_{a1}, V_{a2}, ..., V_{ax}) is a set of x vertex-specific attributes. For each vertex $p \in$ V, there exists a multidimensional tuple V_a(p) denoted as V_a(p) = (V_{a1}(p), V_{a2}(p), ..., V_{ax}(p)). Let E_a=(E_{a1}, E_{a2}, ..., E_{ar}) is a set of r edge-specific attributes. There is a multidimensional tuple E_a(q) denoted as E_a(q)= (E_{a1}(q), E_{a2}(q), ..., E_{ar}(q)) for every edge q ∈ E.

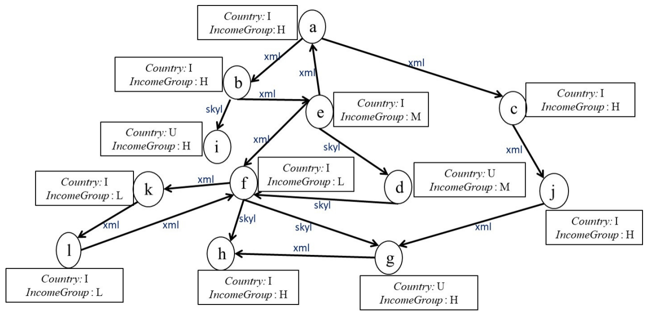


Figure (2). A toy dataset of an email network

For instance, let us consider the attributed graph for an email network as shown in Figure 2. Let the vertex attributes be *Country* and *IncomeGroup*. The domain of attribute *Country* is $V_{Country}$={ India (I), United Kingdom (U)} and the attribute *IncomeGroup* is $V_{IncomeGroup}$={ High (H), Medium (M), Low (L)}. The domain of edge attribute communication content is { XML (xml), Skyline (skyl)}. Thus, for vertex *'a'*, $V_{Country}(a)=I$ and $V_{IncomeGroup}(a)=H$. Similarly, the value of the edge attribute between vertices *'a'* and *'c'* is *xml*.

Table 1 shows the different notations used in this paper with their description.

2.1. Problem Statement

Definition 2. (Multidimensional Constraint Reachability) "Given an attributed graph G, a source vertex s, a destination vertex t, vertex constraint CV_a, and edge constraint CE_a, the multidimensional constraint reachability query on attributed graph verifies whether s can reach t under vertex and edge constraint CV_a, CE_a" [9].

We define Multidimensional Constraint Reachable path (or MCR path) as the resultant path from the given source vertex to the destination vertex while satisfying the vertex or edge attribute constraints.

Let us consider the MCR query *q1('a', 'j', 'I:H', 'xml')*, for the attributed graph of Fig. 2 as an example. The given MCR query *q1* returns true as the source vertex *'a'* can reach the destination vertex *'j'* via vertex *'c'* while satisfying the given vertex constraints 'I:H' and edge constraint 'xml'. Thus, the MCR path is {'a', 'c', 'j' }. Consider another instance of MCR query *q2('b', 'c', 'I:M', 'xml')*. The MCR query *q2* returns no path as the source vertex *'b'* cannot reach *'c'* as well as the given constraints are not satisfied.

The objective of our research is to find the resultant paths for MCR queries faster and propose a scalable solution based on clustering of large attributed graphs. We observe that we can optimize hashing for faster hash generation and constraint verification. We identified the need to find an efficient graph clustering algorithm that considers both graph topology and attributes

Table (1). Notations

Notation	Description
V_a	Set of vertex attributes
V_a(p)	Set of vertex attributes values for vertex *p*
E_a	Set of edge attributes
E_a(q)	Set of edge attributes values for edge *q*
CV_a	Constraints on Vertex attribute values
CE_a	Constraints on Edge attribute values
G(V,E,V_a,E_a)	Attributed graph
G_s	Super graph
SV	Super Vertex
SE	Super Edge
SP	Super Path

while clustering. Thus, we solve MCR queries based on efficient clustering technique in our proposed approaches described in section 4 and section 5.

3. Related Work

In this section, we describe the survey related to constraint reachability techniques [9, 11, 20, 23, 42, 43] and attributed graph clustering techniques [2, 4, 6, 8, 14, 16, 22, 32–34, 39–41]. We also discuss our important observations derived to solve MCR queries efficiently and effectively.

3.1. Constraint Reachability Techniques

Many graph reachability techniques exist, which include 2-hop [19], 3-hop [21], Dual labeling [28] and Path-Tree cover [35] indices in the literature. But, these indices do not consider attribute information. Hence, we cannot apply the reachability techniques directly to solve the constraint reachability queries.

Ruoming Jin et al. [20] introduced formally the problem of Label Constraint Reachability (LCR) query, which is a special case of attribute constraint reachability queries. They developed a spanning tree based solution to solve LCR queries. With this cited state-of-the-art [20], we performed an extensive survey about different types of constraint reachability queries and techniques [13]. Besides, we developed landmark index based path indexing and query processing techniques [11], [13] to find bounded paths for LCR queries in case of edge labeled weighted directed graphs.

An attributed graph acts as a modeling tool to represent information networks [9, 30]. Sakr et al. [31] developed G-SPARQL, a query execution engine with the defined algebraic operators on the graph by using join operations to find the reachability for large attributed graphs. They designed a model that stored the topology of the graph in main memory and accessed the attributes of the graph from the secondary memory. The attributes from the secondary memory are stored in fully decomposed model which includes storing the unique vertex attributes and edge attributes in separate tables. That is, every vertex attribute, its values in the graph and vertex id are stored in the relational form in the disk. G-SPARQL system mainly solves graph pattern matching queries [46, 47].

We observed that Yung et al. [9] developed hashing-based index instead of fully decomposed model to store vertex attributes or edge attributes for attributed graphs. The hash index involved assigning unique hash values for a group of vertex attributes or edge attributes. The attributes and the corresponding hash values are stored in secondary storage. They also designed BFS based heuristic search using random clustering to solve the multidimensional constraint reachability queries.

Zhao et al. [12] studied the problem of finding temporal paths in dynamic attributed graphs for multiple constraints. They used the constraints on total time and cost as inputs. They developed forward and backward pass approximation algorithms to find the temporal paths between the vertices. Wang et al. [16] performed an extensive survey on different types of queries in attributed graphs. They developed a taxonomy for the variety of queries and also compared the state-of-the-art literature to solve them. But, in this literature, only label constrained reachability queries and techniques are discussed. Peng et al. [23] developed 2-hop label indexing and pruning methods to answer label constraint reachability queries.

Guo et al. [15] studied the problem of graph pattern matching with multiple vertex constraints. They developed sampling-based estimation algorithms to find the matching of the spatial path patterns. Namaki et al. [14] developed Q-Chase based algorithms to handle unexpected entities and missing entities during pattern matching. The Q-Chase algorithms perform query writing and query optimization using atomic operators and pruning.

Table 2 describes the observations of recent studies on constraint reachability query processing and attributed graph clustering techniques. We observed that the current state-of-the-art techniques are not directly related to MCR queries. Hence, they cannot be applied to find MCR paths in attributed graphs. We also established the fact that there is no progress of research work done or studied to solve MCR queries in attributed graph after Yung et al. contributions [9]. Therefore, based on the observations, we extend to improve and enhance the works of Yung et al. [9, 10].

We observed that Yung et al. [9, 10] used a probability cost metric by sampling attributes for each super vertex. They mentioned the probability cost metric computation vaguely. So, we improved and enhanced their technique to solve MCR queries efficiently which is described in Section 4.

3.2. Attributed Graph Clustering Techniques

Zhou et al. [4] designed **S**tructure and **A**ttribute clustering (SA clustering) which is one of the prominent attributed graph clustering techniques based on random walks over augmented attributed graph. SA clustering is limited to small networks with few attribute values. Xu et al. [33] developed a Bayesian model-based approach to cluster attributed graphs. But, we observed that this approach is slow and not scalable.

Z. Wu et al. [6] developed **S**tructure and **A**ttributes using **G**lobal structure and **L**ocal neighborhood features (SAGL) clustering algorithm. SAGL clustering considers both the global importance of the vertex and local neighbours structure while assigning weights to different topological links. SAGL clustering technique is faster than SA clustering as the former technique doesn't increase the size of attributed graph yet uses both global importance of the vertex and attribute information to find clusters. We observed that SAGL clustering used page rank [38, 44] centrality measure in computing clusters. We also observed that though SAGL clustering [6] is faster than SA clustering to find the clusters in an attributed graph, it relied on SA clustering to find the attribute similarity between the vertices.

Issam Falih et al. [2] developed **A**ttributed **N**etwork **C**lustering **A**lgorithm (ANCA) based on matrix factorization of both graph topology and vertex attributes. ANCA clustering algorithm is developed by considering the shortest path metric for topological measure and Euclidean distance for attribute similarity. Then, matrix factorization is applied on both topological and attribute similarity measures. Finally, they used k-means clustering on the resultant matrix to form k clusters.

Guo Qi et al. [34] used a matrix factorization-based technique on edge content to detect communities. Yang et al. [32] developed non-negative matrix factorization based model to identify disjoint or overlapping communities at large scale. Amin et al. [8] developed a technique based on matrix factorization and gradient descent to identify polarization and clusters in social networks, specifically Twitter.

Wang et al. [7] developed a graph-based system for multi-view clustering. The system can generate clusters for text data, audio and video data based on their features. The clustering and optimizing algorithms in the system used a similarity-induced graph matrix based on different views of data. This system only considered the features of the data while computing clusters. We cannot apply this clustering technique to our problem as they did not consider the graph topology information during clustering.

From the literature, we observed that matrix factorization is a standard technique that has scope to find similarity by considering graph topology and vertex /edge attributes. Hence, we apply matrix factorization in supergraph construction without the probability cost metric [9] and develop a heuristic-based BFS search to solve the MCR queries using hashing.

4. Proposed Approach: Heuristic search using Hashing and Matrix Factorization

This section describes our proposed techniques to solve the MCR queries for finding the resultant paths.We adopt the hashing and heuristic search developed by Yung et al. [9] by improving and enhancing the technique to solve MCR queries efficiently and effectively. The improvements and extensions to Yung et al. [9] are briefly described as follows:

1. We perform optimized hashing of collated values of vertex attributes or edge attributes.

2. We identify an efficient attributed graph clustering algorithm [2] based on matrix factorization for supergraph construction.

We observed that Yung et al. [10] used a probability cost metric by sampling attributes for each super vertex that is vaguely mentioned. In our proposed approach, we use the supergraph without considering any probability cost metric. Thus, with the above two main aspects, we propose techniques to efficiently solve MCR queries.

4.1. Techniques

This section describes the optimized hashing and supergraph construction approaches used in our proposed techniques to solve MCR queries efficiently.

Hashing based index. Initially, we construct an attribute hash index by collating attribute values of every vertex into a single string s_a. Every unique s_a is compressed to a hash value and stored in primary storage for answering queries. This hash value is mapped to its vertex. For

Table (2). Survey of Queries and Graph Clustering Techniques on Attributed Graphs [2015-2021]

S. No.	Technique (Authors, Year)	Observations
1	Taxonomy of queries in attributed graphs (Wang Y. et al., 2021 [16])	Performed extensive study and derived taxonomy of types of queries in attributed graphs.
2	Two Pass-Approximation Algorithms (Zhao et al., 2020 [12])	Derived temporal path patterns with multiple constraints on total time and cost.
3	Label- constrained 2-hop indexing techniques (Peng et al., 2020 [23])	Solved label-constrained reachability queries by using 2-hop indexing and pruning strategies.
4	Sampling based estimation algorithms (Guo et al., 2020 [15])	Found matching of spatial path patterns with multiple vertex constraints in attributed graphs.
5	Multi-view clustering technique (Wang H. et al. , 2020 [7])	Implemented using similarity-induced graph matrix based on different views of data.
6	ANCA Clustering (Falih et al., 2018 [2])	Computed clusters in attributed graphs considering both graph topology and attribute information through matrix factorization.
7	LandmarkIndex and Query algorithms (Valstar et al., 2017 [3])	Found reachability satisfying given label constraint with significant speedups in query processing for Label Constrained Reachability queries.
8	Ensemble gradient descent algorithm based on matrix factorization (Amin et al., 2017 [8])	Identified polarization and clusters in social networks specifically Twitter through matrix factorization.
9	Heuristic Search Technique through GuidedBFS and hashing (Duncan Yung et al., 2016 [9])	Developed heuristic search technique and computed reachability with multi-dimensional constraints in attributed graphs faster.
10	SAGL clustering (Z. Wu et al., 2016 [6])	Developed clustering technique based on page rank and weighted attribute similarity in attributed graphs.

instance, consider vertex attribute values of vertex *'b'* in Figure 2, i.e., V_a(b)={I, H}. The hash value computed for collated attribute values 'I:H' is 2555692664 as shown in Table 3. Similarly, for every vertex and edge, the corresponding hash values for the attribute values are computed and stored in primary memory.

The hash value for the vertex/edge attribute constraints of the given query is computed. This hash value is verified against stored hash values in primary memory without approaching the secondary storage. Hence, it leads to faster query processing.

Algorithm 1 (HashIndexConstruct) describes the Hash Index Construction of vertex attributes for an attributed graph. The functions and variables used in the algorithm are described as follows:

- *GetHashAttr(u,G)* returns the hash value of the given vertex *u*.
- *AttrIO(u,G)* retrieves the attribute values for the vertex *u* from secondary memory.
- *GenerateMHash()* generates the hash value for the collated attribute value using Murmur hash function [37].
- *hInd* stores the hash value, its corresponding attribute values and *count*.

Algorithm 1: HashIndexConstruct

```
Input  : Attributed graph G
Output: Hash Index, hInd
procedure HashIn(G, hInd)
for all u ∈ G do
    h ← GetHashAttr(u, G)
    aio ← AttrIO(u, G)
    if h==NULL then
        h ← GenerateMHash(aio);
        Set count=1
        Append (aio, h, count) to hInd
    else Append aio to hInd[h] and update count of the
      hash value
```

- *aio* is the variable that stores the collated attribute value for the vertex.
- *h* is the variable that consists of the hash value of the current vertex *u*.
- *count* stores the frequency of assigning the hash value for the unique collated attribute value.

Algorithm 1 considers the attributed graph as input and constructs hash index *hInd*. Initially, it gets the attribute set of vertex *u* from secondary memory. This attribute set is concatenated and stored in variable *aio*. Then, it checks whether its hash value *h* is present in *hInd*. If it is NULL, then the new hash value of *u* is generated using *GenerateMurmurHash()* for the attribute values *aio*. The resultant hash value (*h*), corresponding set of attributes and *count* are stored in *hInd*.

A non-cryptographic hash function like Murmur hash function [37] generates hash values for the collated attribute values. Murmur hash function has no hash collision. During heuristic search, the constraints of vertex are verified with the constraints given by the user by retrieving the hash of constraints and comparing with the hash of collated vertex attribute values (derived from hash index *hInd*) stored in primary memory. This reduces the need to access secondary memory for multidimensional attributes verification.

For instance, let us consider the attributed graph of Fig. 2. Table 3 shows the computed hash values for the collated vertex attribute values of Fig. 2. Let us consider the computation of hash value of collated attribute values for vertex *'a'*. Its collated vertex attribute value is 'I:H' and the corresponding hash value is 2555692664 as shown in Table 3.

Table (3). Hash Index with vertex attributes and hash values

vattrHash	attr	*count*
1071913501	U:M	1
1139838478	I:L	1
2555692664	I:H	1
2608081465	U:H	1
29059796901	I:M	1

Super graph Construction. We divide the attributed graph based on clustering and construct a structure called super graph. Yung et al. [10] built the supergraph for undirected graphs. We define super graph for directed graphs as follows:

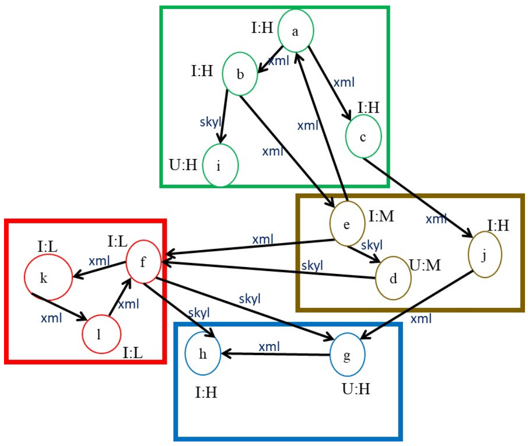
Definition 3. (Super Graph): A super graph G_s is a directed graph constructed by computing super vertices and super edges for the given attributed graph G.

Definition 4. (Super Vertex): A super vertex, SV_i, is a vertex in G_s such that every vertex p of G belongs to only one super vertex SV_i. Thus, $\forall\, p \in V$ in G, $p \in SV_i$, $p \notin SV_j$, if $i \neq j$ where $SV_i, SV_j \in G_s$.

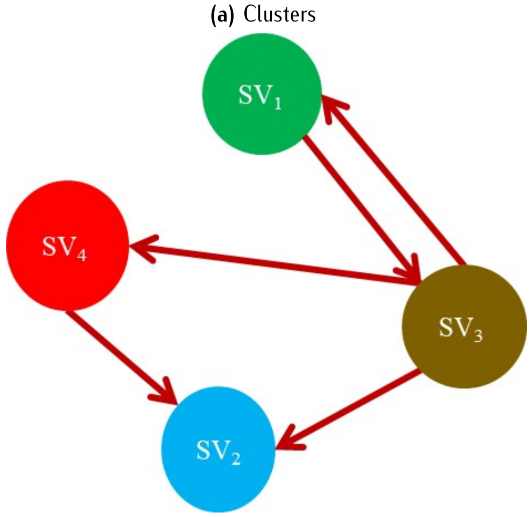
Definition 5. (Super Edge): A super edge SE_{ij}, is a directed edge in G_s formed between the super vertices SV_i and SV_j. This edge is formed only when, for any pair of vertices (p, q) ∈ G such that $p \in SV_i$ and $q \in SV_j$, there exists an edge between p and q. Thus, if there exists an edge $e(p, q) \in E$ in G, $p \in SV_i$, $q \in SV_j$ and $i \neq j$ then $\exists\, SE_{ij}\,(SV_i, SV_j) \in G_s$.

Definition 6. (Super Path): A super path, SP_i, is a simple path in G_s formed by sequence of super vertices $(SV_1, SV_2, ..., SV_d)$ such that $(SV_{i-1}, SV_i) \in G_s$.

For instance, Fig. 3b shows the super graph for the attributed graph of Fig. 2. Thus, the super vertices include SV_1, SV_2, SV_3 and SV_4. For instance, in Fig. 2, there exists edge between vertices *'j'* and *'g'*. The super vertex of *'j'* is SV_3, while the super vertex of *'g'* is SV_2. Hence, we add the super edge (SV_3, SV_2). Thus, the super edges are { (SV_1, SV_3), (SV_3, SV_2), (SV_3, SV_1), (SV_4, SV_2), (SV_3, SV_4) }. A super path from SV_1 to SV_2 is (SV_1, SV_3, SV_2).



(a) Clusters



(b) Super graph

Figure (3). Clusters and the Resultant Super graph of Fig. 2

During super graph construction, we choose K vertices as super vertices from K clusters using an efficient clustering algorithm. Algorithm 2 (SuperGraphMF) describes the Super Graph construction using clustering based on Matrix Factorization adopted from [2]. We further improved the ANCA clustering [2] by applying gap-statistic [26] to find the optimal K value. The significant functions and variables used in the algorithm are described below:

- *SPath(u, v)* returns the shortest distance between the vertices u and v.
- *findSuperVertex(v)* returns the super vertex of v from the super graph G_s.
- M_{topo} stores the topological matrix of G.
- M_{attr} stores the attribute matrix of G.
- U_{topo} stores the topological matrix after singular value decomposition.
- U_{attr} stores the attribute matrix of after singular value decomposition.
- K denotes the number of clusters.
- svu and svv store the super vertices of u and v respectively.

- In the algorithm 2, initially, the subset of vertices are identified as seeds. The seeds are selected by considering top 15% of vertices by using centrality measures such as highest degree, closeness centrality, page rank and eigen vector centrality [2]. The seeds also include the outlier vertices for coverage by considering 5% of vertices with the least centrality [2].
- Next, we compute the topological matrix for the vertices and seeds based on shortest path distance between them. This matrix is normalized and singular value decomposition is applied.
- We then find the attribute similarity between the vertices by computing euclidean distance [29] between them. The euclidean distance between the two vertices $u, v \in V$ is given by equation 1.

$$d(u,v) = \sqrt{\sum_{j=1}^{t}(|A_j(u) - A_j(v)|)^2}, \forall u, v \in V \tag{1}$$

- We use matrix factorization on attribute similarity between the vertices.
- We join the topological similarity and attribute similarity factorized vectors to get the decomposed matrix U and normalize it.
- Then, we apply k-means clustering on the decomposed matrix U to get resultant K clusters.
- If there exist $p \in SV_i$ and $q \in SV_j$ such that there exists edge from p to q in G, then we add the super edge SE_{ij} to the super graph G_s as described in steps 11 to 16 of the algorithm.

Algorithm 2: SuperGraphMF

Input : Attributed graph G(V, E, V_a), Number of clusters K
Output: Super Graph G_s with K Clusters

1 Select subset of vertices as seeds S.
2 Compute topological closeness between the vertices and seeds using shortest path metric
3 Form topological matrix, M_{topo}[v,s]=SPath(v, s), $\forall$v $\in$ V and $\forall$s $\in$ S.
4 Apply singular value decomposition on M_{topo} to get U_{topo}.
5 Compute attribute similarity matrix M_{attr} between the vertices using Euclidean distance based on equation 1.
6 Apply matrix factorization on M_{attr} to get U_{attr}.
7 U$\leftarrow$ U_{topo} +U_{attr}
8 Normalize each of U's rows defined by $U_{ij}=U_{ij}/\sqrt{\sum_j U_{ij}^2}$
9 Apply k-means clustering on U to get K clusters.
10 Construct Super graph G_s (V_s,E_s) with K vertices V_s={sv_1, sv_2,..., sv_K}with each cluster considered as super vertex sv_i
11 **for** *each edge e(u,v)* $\in$ *E* **do**
12   svu=findSuperVertex(u)
13   svv=findSuperVertex(v)
14   **if** *(svu≠svv)* **then**
15   Add edge se_i=(svu, svv) to E_s
16   i$\leftarrow$i+1

Illustration of super graph construction Let us consider the attributed graph of Fig. 2. The resultant set of seeds based on the centrality measures is { *'a'*, *'e'*, *'f'*, *'h'*, *'i'*}. We use singular value decomposition as described in the SuperGraphMF algorithm by considering both topological distance and attribute similarity. Let us assume the number of clusters K=4. We can also compute an optimal K value by applying gap statistic [26]. The resultant clusters with K=4 after applying k-means algorithm are the subsets {'a', 'b', 'c', 'i'}, { 'g', 'h'} , {'d', 'e', 'j'} and {'f', 'k', 'l'}. These clusters are denoted by vertices as SV_1, SV_2, SV_3 and SV_4 respectively which form the super vertices. We add the super edge based on existence of edge between vertices of the clusters. For instance, in Fig. 2, there exists edge between vertices *'e'* and *'f'*. The super vertex of *'e'* is SV_3, while the super vertex of *'f'* is SV_4. Hence, we add the super edge (SV_3, SV_4). Thus, the resultant super graph constructed is shown in Fig. 3b.

4.2. Proposed constrained hash search technique

We propose the constrained hash search technique through optimized hashing to solve MCR queries efficiently. Algorithm 3 (ConstrainedHash) describes our proposed constrained hash search technique. The algorithm is based on BFS and constraint verification using optimized hashing.

Algorithm 3: ConstrainedHash

Input : Attributed graph G, source vertex s, destination vertex t, Vertex Constraint C_v, Hash index *hInd*.
Output: Resultant path *rp*/*"No path"*

```
Let q be queue
Enqueue (s)
while isEmpty(q) do
    Dequeue v
    if (visited[v] = true) then
        continue
    for v′ ∈ v.adjList do
        if (visited[v′]=true) then
            continue
        if (CheckConstraint(v′, hInd, C_v,G)=true) then
            if (v′=t) then
                return rp
            Enqueue v′
    visited[v] ← true
return "No path"
procedure CheckConstraint(v,HashIndex,C_v)
hc←GenerateMHash(C_v)
hv←GetHashAttr(v,G)
if (hc≠hv) then
    return false
if (getCount(HashIndex,hc)=1) then
    return true
else attr← Get attributes from secondary storage
if (CheckAttrConstraint(attr,C_v)=true) then
    return true
```

The significant functions and variables used in the algorithm are described below:

- *CheckConstraint(v,hInd,* C_v*, G)* verifies if the attribute values of the vertex v match with the given constraints C_v.
- *getCount(hInd, h)* returns the count of the hash value h.
- *CheckAttrConstraint()* compares the constraints with attributes retrieved from secondary memory.
- *hc* stores the hash value of given constraints.
- *hv* is hash value of collated attribute values for vertex v.

Optimized Hashing In Algorithm 3, we optimize the hash retrieval through *CheckConstraint()* procedure. In this procedure, we retrieve the hash of given constraints and compare it with the hash of vertex. If both the hash values are same, we check the *count* by retrieving it from the hash index. If the *count* is one, then, we need not check the secondary storage; we declare the two hash values are equal and return true.

Illustration Let us consider the MCR query *q1('a', 'j', 'I:H')* for the attributed graph of Figure 2. The vertex constraint is collated and its hash value is computed. Thus, the hash value of constraint 'I:H' is 2555692664. Using Algorithm 3, during traversal, we compare the hash value of the given vertex constraint to the existing hash value of every vertex from the hash index table (Table 3). As the match exists, the algorithm traverses to the adjacent vertices of the current vertex and verifies the constraints. This is repeated until the destination vertex (j) is reached. Thus, our proposed constrained hash technique technique returns the path {'a', 'c', 'j'} for the MCR query $q1$.

4.3. Proposed heuristic search technique

We also propose a heuristic search technique that uses both optimized hashing and efficient attributed graph clustering to solve MCR queries efficiently. Algorithm 4 (HeuristicSearchMF) describes the Heuristic Search based on Matrix Factorization to find the MCR path between the given vertices. The significant functions and variables used in the algorithm are described below:

- *FindPathBFS(*SV_i*,* SV_j*,* G_s*)* returns the path between the super vertices SV_i and SV_j from the super graph based on BFS.
- *sPath* stores the super path between the super vertices.
- *superSrc* stores the super vertex of s.
- *superDst* stores the super vertex of t.
- *superiv* stores the super vertex of an intermediate vertex iv.

In this algorithm, for the given source vertex, destination vertex and constraints, initially, the super vertex of source vertex is identified from the super graph. Then, the super vertex of destination vertex is identified. The path between these super vertices is detected using FindPathBFS() and stored in *sPath*. We find the MCR path between the vertices using BFS and *sPath* information by verifying the user given constraints through optimized hashing (from Algorithm 1).

Algorithm 4: HeuristicSearchMF

```
Input  : Attributed graph G, source vertex s,
         destination vertex t, Vertex Constraint C_v,
         Super Graph G_s, Hash index hInd.
Output: Resultant path rp/"No path"
1  Let q be queue
2  Enqueue (s)
3  superSrc←findSuperVertex(s)
4  superDst←findSuperVertex(t)
5  sPath ← FindPathBFS(superSrc, superDst, G_s)
6  while isEmpty(q) do
7      Dequeue v
8      if (visited[v] = true) then
9          continue
10     for v′ ∈ v.adjList do
11         if (visited[v′]=true) then
12             continue
13         superiv← findSuperVertex(v′)
14         if (superiv ∈ sPath) then
15             visited[v′]=true
16             if (CheckConstraint(v′, hInd, C_v,G)=true)
               then
17                 if (v′=t) then
18                     return rp
19                 Enqueue v′
20     visited[v] ← true
21 return "No path"
```

Heuristic In the proposed approach, we assume that if reachability exists, it is found along the super path (*sPath*). By including this heuristic, we can find the reachability between the vertices faster as we traverse only the vertices that belong to the super vertices of *sPath*, thus minimizing the search space. In algorithm 4, we include the heuristic through finding super path (*sPath)* in step 5 and verifying if each super vertex of adjacent vertex (step 13) belongs to *sPath* in step 14.

Illustration For instance, consider the MCR query *q1*(*'a'*, *'j'*, *'I:H'*). The super vertex of *'a'* is SV_1 and super vertex of *'j'* is SV_3. There exists a super path in the super graph from SV_1 to SV_3, i.e., (SV_1, SV_3). Thus, our proposed heuristic search technique traverses only the vertices within super vertices SV_1 and SV_3. The vertex constraint is combined and its hash value is computed. While traversing, the hash value of the given vertex constraint is compared to the existing hash value in the hash index table (Table 3). If the match exists, it traverses to the next adjacent vertices until the destination vertex (*'j'*) is reached. Thus, our proposed heuristic search technique returns the path {'a', 'c', 'j'} for the MCR query *q1*.

Let us consider another MCR query *q2*(*'c'*, *'h'*, *'I:H'*). The super vertex of *'c'* is SV_1 and super vertex of *'h'* is SV_2. There exists the super path from SV_1 to SV_2, i.e., (SV_1, SV_3, SV_2). Thus, our proposed heuristic search technique traverses only the vertices within super vertices SV_1, SV_3 and SV_2. As the vertex constraints do not match at vertex *'g'*, the result of query *q2* is *'No path'*.

5. Proposed Extended Heuristic Search

In our proposed HeuristicSearchMF algorithm, we observed that there might exist a path that is not included in the super path of the constructed supergraph. We modified our proposed approach to overcome this problem by extending the heuristic to include those vertices whose super vertices have destination super vertex as the adjacent vertex in the supergraph.

Heuristic We assume that the vertex attribute values and edge attribute values are single and discrete. We assume that if reachability exists, it is found along the super path of the super graph. The heuristic also includes the super path having adjacent super vertices to the destination super vertex.

We extended the heuristic by including those vertices whose super vertices have destination super vertex as the adjacent vertex. Algorithm 5 (Extended-HeuristicSearchMF) describes the Extended Heuristic Search technique based on Matrix Factorization. The algorithm applies both optimized hashing and efficient graph clustering based on matrix factorization. In the algorithm, the extensions are described from step 13 to step 19. For each adjacent vertex traversed, we find its super vertex and check if it is neighbor to the super vertex of the destination vertex using the edgeExists() procedure. If such an adjacent vertex exists, then its attribute values are verified with the given constraints. This process is repeated till the destination vertex is reached and the resultant path *rp* is retrieved. Thus, our proposed extended heuristic search technique improves the accuracy by finding most of the MCR paths.

Let us consider the example of Fig. 4. To find the path from source vertex *'s'* to destination vertex *'t'*, we first compute the super path between the super vertices of *'s'* and *'t'*. Each dotted rectangular box is considered as super vertex. The super vertices thus formed include SV_1, SV_2, SV_3, SV_4, SV_5, SV_6 and SV_7. The resultant super path is (SV_1, SV_2, SV_4, SV_3). When we execute the HeuristicSearchMF algorithm, we cannot reach the destination vertex through the super path. But, in the ExtendedHeuristicSearchMF algorithm, we can reach the destination vertex via intermediate vertex *'v9'* whose super vertex SV_7 is adjacent to the destination super vertex SV_3. This is because of including the extended heuristic of considering adjacent vertices of the destination super vertex while finding the path.

Thus, we can reduce the number of missed reachable paths using the extended heuristic efficiently.

Algorithm 5: ExtendedHeuristicSearchMF

Input : Attributed graph G, source vertex s, destination vertex t, Vertex Constraint C_v, Super Graph G_s.
Output: *rp/"No path"*

Let q be queue
Enqueue (s)
superSrc←findSuperVertex(s)
superDst←findSuperVertex(t)
sPath ← FindPathBFS(superSrc, superDst, G_s)
while *isEmpty(q)* **do**
  Dequeue v
  reached← **false**
  if *(visited[v] = **true**)* **then**
    continue
  for $v' \in v.adjList$ **do**
    if *(visited[v′]=true)* **then**
      continue
    superiv← findSuperVertex(v′)
    for $v'' \in v'.adjList$ **do**
      superiv2← findSuperVertex(v″)
      if *(edgeExists(superiv2,superDst,G_s)* ***OR*** *superiv2=superDst)* **then**
        reached←**true**
    if *((superiv ∈ sPath)**OR** reached=**true**)* **then**
      if *(CheckConstraint(v′, G_h, C_v)=**true**)* **then**
        if *(v′=t)* **then**
          return *rp*
        Enqueue v′
        if *(reached=**true**)* **then**
          break
  visited[v] ← **true**
return "No path"

6. Experiments and Results

This section describes the datasets, the parameters, the experiments' domain and the result analysis of our proposed techniques.

6.1. Experiment Setup

All experiments are conducted in laptop with Core i3 2GHz CPU (2-core), 8GB RAM in Ubuntu Linux OS. We implemented our proposed and conventional techniques in C++. For secondary storage, we used the MySQL database. For hashing, we used the Murmur hash function [37]. We constructed the supergraph by using the existing R code [2] and implemented the naive clustering [9] in R programming. We set the number of super vertices to 15 (default) and

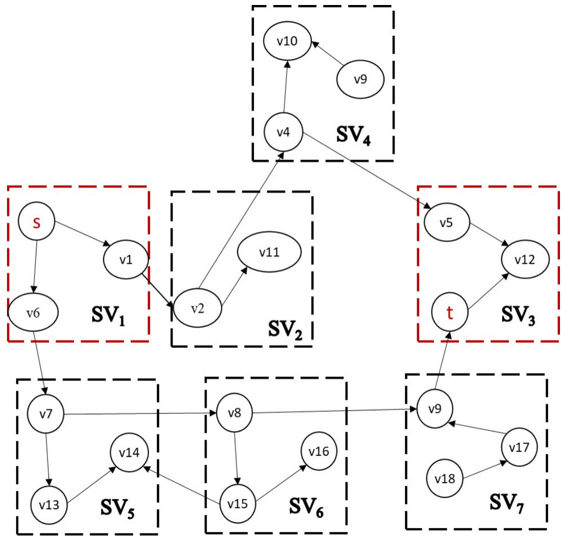


Figure (4). Example for proposed extended heuristic search technique

constructed the supergraph for all the datasets. Besides, we also computed optimal K value by applying gap statistic [26].

Table 4 describes the real and synthetic datasets used for experiments. Table 6 describes the different parameter settings used in the experiments adopted from [9]. We used vertex attributes and vertex constraints throughout our experiments. Besides, we used edge attributes and edge constraints along with vertex constraints for the real datasets to demonstrate that we can extend our proposed techniques to edge constraints. This paper focused on vertex constraints as our proposed techniques mainly use vertex attribute values during clustering. We generated 25 to 100 MCR queries (whose path length is greater than 1) for the real and synthetic datasets by randomly selecting attribute values and verifying the constraints through constrained breadth-first search and traversal. We performed experiments on synthetic graphs by varying graph size to test the scalability of our proposed techniques.

6.2. Baselines

We evaluated the efficiency of our proposed approaches (HeuristicSearchMF and ExtendedHeuristicSearchMF algorithms) by comparing them with two existing techniques described below:

(1) Breadth First Search or BFS [27] is the baseline idea. Here, the constraints are also checked while performing breadth first search from source vertex till the destination vertex is reached [3].

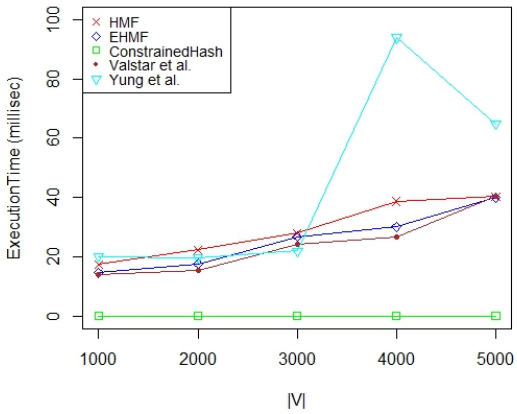
(2) Yung et al. developed BFS based heuristic search technique using naive clustering [9].

To solve MCR queries, we have the following three approaches with respect to proposed techniques.

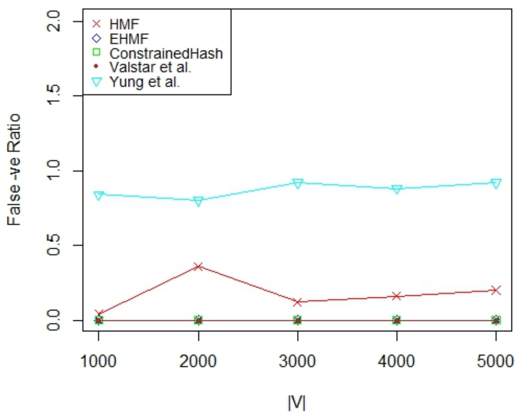
(i) We can solve only by using hashing mechanism described in section 4.2 (with optimized hashing only). We denote this approach by *Constrained-Hash*.

(ii) We can solve using both hashing and clustering mechanism to obtain the resultant path efficiently. We consider *HMF* as the implementation of our proposed HeuristicSearchMF algorithm described in section 4.3.

(iii) We consider *EHMF* technique as the implementation of our proposed ExtendedHeuristicSearchMF algorithm described in section 5.



(a) Average Execution Time



(b) False -ve Ratio

Figure (5). Varying Graph Size for ForestFire graphs

6.3. Datasets Description

Table 4 summarizes the real and synthetic datasets used for experiments. We generated synthetic graphs from SNAP [18] in C++. We randomly assigned vertex attribute values for the vertices and edge attribute values for edges. Table 5 states the synthetic vertex attributes that are assigned randomly to the datasets.

Table (4). Datasets Overview

Real Graph	**|V|**	**|E|**
Robots [36]	1724	3596
Twitter[1]	2511	37154
Synthetic Graph	**|V|**	**|E|**
Erdos-Renyi [18]	1000	2000
	1000	3000
	2000	6000
	3000	9000
	4000	12000
	5000	15000
ForestFire [18]	5000	12620
	4000	10252
	3000	7751
	2000	4865
	1000	2833

Robots. Robots is a real trust network [36] with edge labels that denote the level of trust interaction between the users. We pre-process the dataset by assigning unique identifiers to the vertices, resulting in 1724 vertices and 3596 edges. Each vertex has synthetic attributes whose values are randomly assigned (Table 5). We considered each edge had *Trustlevel* as the real attribute and derived its value from the data set. The trust level can be Master (M), Apprentice (A), Journeyer (J) or Observer (O). Besides, we randomly assigned values for the synthetic attributes for every edge of the Robots dataset.

Twitter. Twitter is a real pre-processed dataset [1] with 2511 vertices and 37154 edges. We processed the vertex attributes further into two real attributes that denote the visibility and the tag of the vertices. In addition, we randomly assigned two synthetic edge attributes described in Table 5 to evaluate MCR queries with both vertex and edge constraints.

Erdos-Renyi Graph. Erdos-Renyi (E-R) graphs are the synthetic graphs that follow power-law distribution [25]. These graphs have vertices with nearly uniform degree distribution. We generate E-R graph using SNAP [18] with the number of vertices set to 1000 and maximum degree for each vertex set to 2. Besides, we assign two attributes (as described in Table 5) for each vertex. The attribute values are randomly assigned within the domain. We also generated E-R graphs to test for scalability by varying vertices from 1000 to 5000 with maximum degree set to 3 as shown in Table 4.

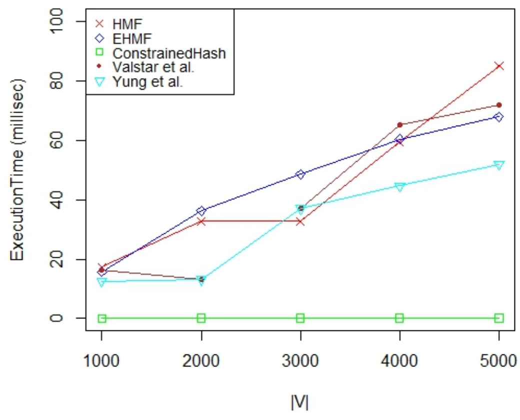
ForestFire Graph. ForestFire graphs are the synthetic random graphs [24]. The ForestFire model graphs exhibit the properties of time-evolving real-world graphs [24] that include densification of graphs and decreasing effective diameter. We generate ForestFire graphs using SNAP [18] for testing scalability with the number of vertices varying from 1000 to 5000 and the maximum degree for each vertex set to 4. The forward probability is set to 0.4, the backward probability is set to 0.2 [18] and the maximum degree of each vertex is set to 2. Besides, we assigned two attributes for each vertex as described in Table 5.

Table (5). Vertex Attributes and Edge Attributes

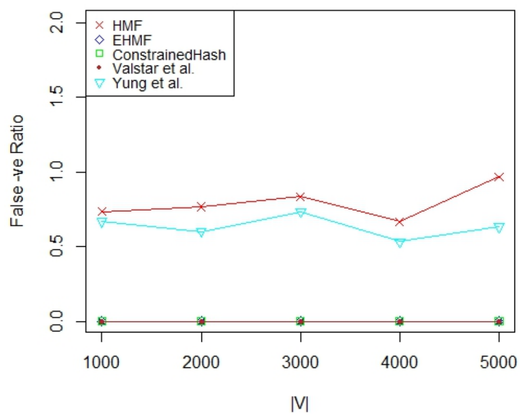
Vertex Attribute	Domain Size, Distribution
Country	5, *uniform*
Region	3, *uniform*
Gender	2, *uniform*
Edge Attribute	**Domain Size, Distribution**
Trustlevel	4, *real*
isFamily	2, *uniform*
isFriend	2, *uniform*

Table (6). Parameter Values

Parameter	Value
Number of Vertex Attributes	2, 3
Number of Edge Attributes	3
Number of Super-vertex (K)	**15**, 50
Number of Vertex Constraints	2
Number of Edge Constraints	1



(a) Average Execution Time



(b) False -ve Ratio

Figure (6). Varying Graph Size for E-R graphs

6.4. Results and Analysis

We evaluated the efficiency of our proposed techniques based on average execution time and false negative ratio. The MCR true queries are the constrained reachable queries with at least one path between the given vertices. The accuracy of our proposed techniques is based on false negative ratio for true queries. The false negative ratio (τ) is defined as *"The fraction of queries which fail to return any path that satisfies the given constraint, although at least one such path exists"* [5].

Table 7 shows the average execution time and false negative ratio of MCR queries for Robots dataset using our proposed techniques compared to existing techniques. We computed optimal K value for Robots dataset by applying gap statistic [26]. The resultant computed K value is 15. We generated 100 MCR queries for the evaluation of Robots dataset.

From Table 7, using our proposed *HMF* approach, the false negative ratio (τ) is 0.32. Based on our proposed extended heuristic technique, i.e., *EHMF*, the false negative ratio reduced to 0. Our proposed *HMF* and *EHMF* techniques executed faster than the existing technique [9]. Furthermore, our proposed *ConstrainedHash* technique executed faster than all the techniques.

Table 8 shows the average execution time and false negative ratio of 25 MCR queries with vertex constraints and edge constraints for Robots dataset. We choose the *TrustLevel* as edge constraint and generated MCR true queries based on constrained BFS. For clustering, we assumed K to be 50. We observed that our proposed techniques have a lesser false negative ratio than that of the existing technique [9]. Besides, we observed that *ConstrainedHash* technique executed faster for MCR queries than the other techniques.

Table 9 shows the average execution time and false negative ratio of 30 MCR queries with vertex

Table (7). Average Execution Time (AET) of queries for Robots dataset with only vertex constraints

S.No.	Technique	AET (s) for true queries	τ	AET (s) for false queries
	Proposed			
1	*HMF*	0.21464	0.32000	0.03071
2	*EHMF*	0.22904	0.00000	0.02829
3	*ConstrainedHash*	0.00011	0.00000	0.00048
	Conventional			
4	*Valstar et al. [3]*	0.30114	0.00000	0.53873
5	*Yung et al. [9]*	0.09214	0.60000	0.18706

Table (8). Average Execution Time (AET) of queries for Robots dataset with vertex constraints and edge constraints

S.No.	Technique	AET (s) for true queries	τ	AET (s) for false queries
	Proposed			
1	*HMF*	0.46840	0.80000	0.00680
2	*EHMF*	0.17826	0.00000	0.00674
3	*ConstrainedHash*	0.01519	0.00000	0.00182
	Conventional			
4	*Valstar et al. [3]*	0.30061	0.00000	0.20006
5	*Yung et al. [9]*	0.03488	0.96000	0.00605

Table (9). Average Execution Time (AET) of queries with only vertex constraints for Twitter dataset

S.No.	Technique	AET (s) for true queries	τ	AET (s) for false queries
	Proposed			
1	*HMF*	5.16573	0.96700	0.25429
2	*EHMF*	4.12166	0.30000	1.58136
3	*ConstrainedHash*	0.01461	0.00000	0.00558
	Conventional			
4	*Valstar et al. [3]*	4.72921	0.00000	2.26847
5	*Yung et al. [9]*	0.73027	0.73000	0.21685

Table (10). Average Execution Time (AET) of queries with vertex constraints and edge constraints for Twitter dataset

S.No.	Technique	AET (s) for true queries	τ	AET (s) for false queries
	Proposed			
1	*HMF*	0.95781	0.80000	0.16164
2	*EHMF*	2.57739	0.03000	1.9176
3	*ConstrainedHash*	2.55130	0.00000	2.0261
	Conventional			
4	*Valstar et al. [3]*	9.34007	0.00000	3.74253
5	*Yung et al. [9]*	0.89415	0.46667	0.31173

constraints for Twitter dataset. For clustering, we assumed K to be 50. In Table 10, we describe the execution of MCR queries with both vertex and edge constraints for Twitter dataset. We observed that our proposed *EHMF* technique has a lesser false negative ratio than that of the existing technique [9]. Further, we observed that *ConstrainedHash* technique executed faster than the other techniques. From Table 10, we also observed that with the increase in the constraints, the proposed HMF technique has least execution time for false queries.

Table 11 shows the average execution time and false negative ratio of MCR queries using our proposed techniques compared with existing techniques for E-R graphs. From the table 11, we find that there is considerable decrease in the false negative ratio for our proposed extended heuristic technique i.e. *EHMF* than that of *HMF*. We also observed that *ConstrainedHash* technique executed faster for MCR queries than the other techniques.

Figure 5a shows the average execution time for Forest Fire graphs with varying graph size from 1000 vertices to 5000 vertices. Figure 6a shows the average execution time for E-R graphs with varying graph size from 1000 vertices to 5000 vertices whose maximum degree is set to 3. From Fig. 5a and Fig. 6a, we observe that

Table (11). Average Execution Time (AET) of queries for Erdos-Renyi graph with only vertex constraints

S.No.	Technique	AET (s) for true queries	τ	AET (s) for false queries
	Proposed			
1	*HMF*	0.01280	0.65000	0.00294
2	*EHMF*	0.01290	0.05000	0.00296
3	*ConstrainedHash*	0.00009	0.00000	0.00002
	Conventional			
4	*Valstar et al. [3]*	0.01162	0.00000	0.00651
5	*Yung et al. [9]*	0.01304	0.55000	0.00342

ConstrainedHash technique executed faster than the other proposed techniques and existing techniques. We also observed that , in case of ForestFire graphs, our proposed *HMF* and *EHMF* techniques executed faster than the existing technique [9] with the increase in the number of vertices. Figure 5b shows that the false negative ratio varied from 0.04 to 0.36 for the MCR true queries on ForestFire graphs using our proposed *HMF* approach. But, in the case of E-R graphs, we observed higher false negative ratio using our proposed *HMF* approach than the conventional approach with increase in the number of vertices as shown in Figure 6b. By using our proposed *EHMF* and *ConstrainedHash* techniques, we find that the false negative ratio is 0 for both the synthetic datasets.

7. Conclusion and Future Scope

In this paper, we solved MCR queries on attributed graphs by finding the resultant paths. We proposed an efficient heuristic search technique that includes hashing and clustering. The hash value is computed for multidimensional attribute values using optimized hashing. We used matrix factorization based graph clustering on the attributed graph to construct supergraph. We used the shortest path from the super graph and hashing to match the constraints in our proposed approach and efficiently found MCR paths. Further, we proposed an extended heuristic search technique that increased the accuracy. We find that our proposed techniques are scalable and solved MCR queries efficiently evaluated on real and synthetic datasets.

We plan to extend the research by developing an efficient index for membership-based constraint reachability queries. Besides, we can use optimization techniques [10] of computing extra hash values to find reachability between two vertices with constraints specified on only some of the vertex/edge attributes. We can also use our proposed technique to solve constrained reachability queries for single source vertex and multiple destination vertices. We can also extend our proposed approaches for streaming graph data by using incremental clustering techniques [22].

Acknowledgement

This work is part of the first author's Ph.D. thesis titled "A Study of Constrained Reachability Query Processing in Directed Graphs" submitted to the University of Hyderabad, Hyderabad, India during December 2020 [17].

References

[1] T. He and K. C. C. Chan (2018) *Discovering Fuzzy Structural Patterns for Graph Analytics*, IEEE Transactions on Fuzzy Systems, Vol. 26(5), pp. 2785–2796.

[2] Falih, Issam and Grozavu, Nistor and Kanawati, Rushed and Bennani, Younès (2018) *ANCA : Attributed Network Clustering Algorithm*, Complex Networks & Their Applications VI, Springer International Publishing, pp. 241–252.

[3] L.D.J. (Lucien) Valstar (2016) *Landmark Indexing for Scalable Evaluation of Label-Constrained Reachability Queries*, Masters Thesis, Department of Mathematics and Computer Science Web Engineering Research Group, Eindhoven University of Technology, Netherlands.

[4] Zhou, Yang and Cheng, Hong and Yu, Jeffrey Xu (2009) *Graph Clustering Based on Structural/Attribute Similarities*, Proc. VLDB Endow., Vol. 2(1):718–729.

[5] Likhyani, Ankita and Bedathur, Srikanta(2013) *Label Constrained Shortest Path Estimation*, Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, pp. 1177–1180.

[6] Wu, Zhonggang and Lu, Zhao and Ho, Shan-Yuan (2016) *Community Detection with Topological Structure and Attributes in Information Networks* ACM Trans. Intell. Syst. Technol., Vol. 8(2), pp. 33:1–33:17.

[7] Wang, Hao and Yang, Yan and Liu, Bing and Fujita, Hamido(2019) *A study of graph-based system for multi-view clustering* Knowledge-Based Systems , ELsevier, Vol. 163, pp. 1009–1019.

[8] M. T. Al Amin and C. Aggarwal and S. Yao and T. Abdelzaher and L. Kaplan (2017) *Unveiling polarization in social networks: A matrix factorization approach*, Proceedings of IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, pp. 1–9.

[9] Duncan Yung and Shi-Kuo Chang (2016) *Fast Reachability Query Computation on Big Attributed Graphs* 2016 IEEE International Conference on Big Data, pp.3370–3380.

[10] Ka Wai (Duncan) Yung (2017) *Query Processing on Attributed Graphs* PhD Thesis, University of PittsBurgh.

[11] B. Bhargavi and Rani K. Swarupa, (2018), *Bounded Paths for LCR Queries in Labeled Weighted Directed Graphs*, Advances in Computing and Data Sciences, pp. 124–133.

[12] Zhao, Anqi and Liu, Guanfeng and Zheng, Bolong and Zhao, Yan and Zheng, Kai, (2020), *Temporal paths discovery with multiple constraints in attributed dynamic graphs*, World Wide Web, Springer, Vol. 23(1), pp. 313–336.

[13] Bhargavi B. and Swarupa Rani K. (2019) *Implicit landmark path indexing for bounded label constrained reachable paths*, International Journal of Recent Technology and Engineering (IJRTE), Vol. 8(4):p.10.

[14] Namaki, Mohammad Hossein and Song, Qi and Wu, Yinghui and Yang, Shengqi (2019) *Answering why-questions by exemplars in attributed graphs*, Proceedings of the 2019 International Conference on Management of Data , pp. 1481–1498.

[15] Guo, Ying and Zheng, Lianzhen and Zhang, Yuhan and Liu, Guanfeng, (2020) *MCOPS-SPM: Multi-Constrained Optimized Path Selection Based Spatial Pattern Matching in Social Networks*, Cloud Computing, Smart Grid and Innovative Frontiers in Telecommunications, Springer International Publishing, pp. 3–19.

[16] Wang, Yanhao and Li, Yuchen and Fan, Ju and Ye, Chang and Chai, Mingke (2021) *A survey of typical attributed graph queries*, World Wide Web, Springer, Vol. 24(1), pp. 297–346.

[17] Bhargavi B. (2020) *A Study of Constrained Reachability Query Processing in Directed Graphs* Preprint, PhD Thesis, University of Hyderabad.

[18] Jure Lescovec, (2016), *SNAP: A general purpose network analysis and graph mining library in C++*, <http://snap.stanford.edu/snap>.

[19] Cohen, Edith and Halperin, Eran and Kaplan, Haim and Zwick, Uri (2002) *Reachability and Distance Queries via 2-hop Labels* Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '02, pp.937–946.

[20] Jin, Ruoming and Hong, Hui and Wang, Haixun and Ruan, Ning and Xiang, Yang (2010) *Computing Label-constraint Reachability in Graph Databases* Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 123-134.

[21] Jin, Ruoming and Xiang, Yang and Ruan, Ning and Fuhry, David (2009) *3-HOP: A High-compression Indexing Scheme for Reachability Query*, Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, pp. 813–826.

[22] Zarayeneh, Neda and Kalyanaraman, Ananth (2019) *A Fast and Efficient Incremental Approach toward Dynamic Community Detection*,Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 9–16.

[23] Peng, You and Zhang, Ying and Lin, Xuemin and Qin, Lu and Zhang, Wenjie (2020) *Answering Billion-Scale Label-Constrained Reachability Queries within Microsecond*, VLDB Endowment, Vol. 13(6), pp. 812–825.

[24] Leskovec, Jure and Klecberg, Jon and Faloutsos, Christos (2007) *Graph Evolution: Densification and Shrinking Diameters*, ACM Trans. Knowl. Discov. Data, Vol. 1(1), pp. 2-44.

[25] Réka Albert and Albert-lászló Barabási (2002) *Statistical mechanics of complex networks*, Rev. Mod. Phys, pp. 47–94.

[26] Robert Tibshirani and Guenther Walther and Trevor Hastie (2000) *Estimating the number of clusters in a dataset via the Gap statistic*, Journal of Royal Statistical Methodology, Vol. 63, pp. 411–423.

[27] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford (2009) *Introduction to Algorithms, Third Edition*.

[28] Wang, Haixun and He2, Hao and Yang, Jun and Yu, Philip S. and Yu, Jeffrey Xu and Yu, Jeffrey Xu (2006) *Dual Labeling: Answering Graph Reachability Queries in Constant Time*, Proceedings of the 22nd International Conference on Data Engineering, IEEE Computer Society, pp. 1–12.

[29] Duda, Richard O. and Hart, Peter E. and Stork, David G. (2000) *Pattern Classification (2nd Edition)*, Wiley-Interscience.

[30] Zhengkui Wang and Qi Fan and Huiju Wang and Tan, Kian Lee and Divyakant Agrawal and El Abbadi, Amr (2014) *Pagrol: Parallel Graph OLAP over large-scale attributed graphs*, Proceedings - ICDE, IEEE Computer Society, pp. 496–507.

[31] Sakr, Sherif and Elnikety, Sameh and He, Yuxiong (2011) *G-SPARQL: A Hybrid Engine for Querying Large Attributed Graphs*, Microsoft Technical Report.

[32] Yang, Jaewon and Leskovec, Jure (2013) *Overlapping Community Detection at Scale: A Nonnegative Matrix Factorization Approach*, Proceedings of the Sixth ACM International Conf. on Web Search and Data Mining, Vol. 13(6), ACM, pp. 587–596.

[33] Xu, Zhiqiang and Ke, Yiping and Wang, Yi and Cheng, Hong and Cheng, James (2012) *A Model-Based Approach to Attributed Graph Clustering*, Proceedings of the 2012 ACM SIGMOD International Conf. on Management of Data, pp. 505–516.

[34] G. Qi and C. C. Aggarwal and T. Huang (2012) *Community Detection with Edge Content in Social Media Networks*, 2012 IEEE 28th ICDE, pp. 534–545.

[35] Aggarwal, Charu C. and Wang, Haixun (2012) *Managing and Mining Graph Data*, Springer Publishing Company, Incorporated.

[36] (2017) *Robots Dataset*, <http://tinyurl.com/gnexfoy/>.

[37] (2016) *MurmurHash*, <https://github.com/aappleby/smhasher>.

[38] Page, L. and Brin, S. and Motwani, R. and Winograd, T. (1998) *The PageRank citation ranking: Bringing order to the Web*, Proceedings of the 7th International World Wide Web Conference, (pp. 161–172).

[39] Z. Lin, Z. Kang, L. Zhang and L. Tian, (2021) *Multi-view Attributed Graph Clustering* in IEEE Transactions on Knowledge and Data Engineering, doi: 10.1109/TKDE.2021.3101227.

[40] Atzmueller, M., Günnemann, S. and Zimmermann, A. (2021) *Mining communities and their descriptions on attributed graphs: a survey.* Data Min Knowl Disc, Vol. 35, pp. 661–687. https://doi.org/10.1007/s10618-021-00741-z.

[41] Isma Hamid and Yu Wu and Qamar Nawaz and Runqian Zhao (2017) *An improved attributed graph clustering method for discovering expert role in real-world communities*,

In proceedings of 10th EAI International Conference on Mobile Multimedia Communications, EAI MOBIMEDIA, https://doi.org/10.4108/eai.13-7-2017.2270341.

[42] Xin Chen, You Peng, Sibo Wang, and Jeffrey Xu Yu (2022) *DLCR: efficient indexing for label-constrained reachability queries on large dynamic graphs*. Proc. VLDB Endow., Vol. 15(8), pp. 1645–1657, https://doi.org/10.14778/3529337.3529348.

[43] Yangjun Chen and Gagandeep Singh (2021) *Graph Indexing for Efficient Evaluation of Label-constrained Reachability Queries* ACM Trans. Database Syst. 46, 2, Article 8 (June 2021), 50 pages. https://doi.org/10.1145/3451159.

[44] Arasu, Arvind and Cho, Junghoo and Garcia-Molina, Hector and Paepcke, Andreas and Raghavan, Sriram. (2001) *Searching the Web* ACM Trans. Internet Technol., Vol. 1(1), pp. 2–43.

[45] Bhargavi B. and K. Swarupa Rani (2020) *Finding Frequent Subgraphs and Subpaths through Static and Dynamic Window Filtering Techniques*, EAI Endorsed Transactions on Scalable Information Systems, Vol. 7(27):p.13.

[46] B. Bhargavi, K. P. Supreethi (2012) *Graph Pattern Mining: A Survey of Issues and Approaches*, International Journal of Information Technology and Knowledge Management, Vol. 5(2), pp. 401-407.

[47] Bhargavi Balla, Supreethi Kalyandurgam Pujari (2012) *An Efficient Computation of Reachability Labeling for Graph Pattern Matching*, Second International Conference on Social Eco-Informatics, IARIA, pp. 58-62.