Information Sciences 607 (2022) 961–1000

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Personalized query techniques in graphs: A survey

Peiying Lin [a], Yangfan Li [a,*], Wensheng Luo [a], Xu Zhou [a], Yuanyuan Zeng [a], Kenli Li [a], Keqin Li [b]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan 410082, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 13 July 2021
Received in revised form 17 May 2022
Accepted 5 June 2022
Available online 8 June 2022

Keywords:
Community search
Graph search
Keyword query
Reachability query
Similarity query
Shortest path query

ABSTRACT

Graph is a famous data structure that has prevalent applications in the real world, including social networks, biological networks, and computer networks. In these applications, graph management operators are powerful tools for mining important information hidden in large-scale graphs. As important graph data management operators, personalized graph queries are playing an increasingly significant role in providing users with effective decision support. In particular, the purpose of personalized graph queries is to compute personalized results which can meet the preferences of different users from the three aspects of specified query vertices, structures, and attributes. In this paper, we conduct a survey to offer a comprehensive view of the current personalized graph queries which need users to specify query vertices over simple and attributed graphs, respectively. These queries fall into three categories, including point-related, path-related, and subgraph-related graph queries, whose query results have distinct structures. We analyze existing approaches to personalized graph queries and highlight current challenges. In addition, we also offer guidelines for future graph queries.

1. Introduction

The graph is a prevalent data structure used to model complex networks in many practical applications, including computer networks, social networks, and biological networks. In a graph, vertices and edges are utilized to represent entities and relationships between entities, respectively. For example, vertices in a social network represent different people. If two people have a social relationship, they are connected by an edge. In a biological network of protein–protein-interaction, a vertex represents a protein and an edge represents a biochemical interaction between two proteins.

Graph queries are important graph management operators, which are powerful tools for managing and analyzing graph data. These queries have become core techniques in many recommendation systems for the sake of providing decision support and information services. As introduced in [1], recommendation results without personalization may fail to meet the various needs of users because of ignoring their preferences and diversity. In practice, users usually are interested in recommendations related to themselves. Now, numerous personalized recommendations have come into our lives to help users explore the most-watched movies (e.g., Netflix), favorite products (e.g., Amazon), potential friends (e.g., Facebook), headline

[*] Corresponding author.
E-mail addresses: peiying_lin@hnu.edu.cn (P. Lin), yangfanli@hnu.edu.cn (Y. Li), luowensheng@hnu.edu.cn (W. Luo), zhxu@hnu.edu.cn (X. Zhou), zyy95@hnu.edu.cn (Y. Zeng), lkl@hnu.edu.cn (K. Li), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.ins.2022.06.023
0020-0255/© 2022 Elsevier Inc. All rights reserved.

news (e.g., Google News), to name just a few [2]. The recommendations are all gained by taking into account the different preferences of users.

For a given graph *G*, the goal of these personalized graph queries is to compute personalized results, including point, path, and subgraphs, which can satisfy the various preferences of users. At present, abundant personalized graph queries have been proposed and received growing attention for their significant roles in personalized recommendations. As shown in Fig. 1, personalized graph queries are widely used in path planning, biomedical research, social activity organization, and other real-world applications. They can efficiently manage and analyze large-scale graphs in social networks, spatial networks, and biological networks.

Personalized graph queries can help users search for important knowledge hidden in large-scale graphs and provide strong support for them to make ideal decisions. As an example, article recommendations are a powerful tool for users to find the most related articles to the current article in Google Scholar, Baidu Scholar, and other academic search sites. These personalized article recommendations can be gained by similarity queries [1]. Additionally, in road networks, the shortest path query is effective in finding an optimal path with the minimum time overhead for travelers [3]. Shortest path queries play vital roles in online mapping and navigation services and bring great convenience to our daily travel [4,5]. Furthermore, in social networks, community searches are useful for organizers to identify a group of users, each of whom is well-acquainted [6].

There have been many studies of personalized graph queries that achieve different goals and are related to different preferences of users. However, to the best of our knowledge, it lacks a comprehensive survey of these personalized graph queries. Therefore, it is significant to organize and analyze these studies.

In this paper, we focus on personalized graph queries, including similarity queries, reachability queries, shortest path queries, and community searches, over different types of graphs. These personalized graph queries aforementioned can cover three aspects of user preferences, which are query vertices, structures, and attributes. First, they all need to specify query vertices as important inputs. Second, they output different structures, including points, paths, and subgraphs, respectively. Finally, for the user preferences of attributes, personalized graph queries over different types of graphs are considered.

As shown in Table 1, with respect to the user preferences of different structures, in this paper, we first divide personalized graph queries into the following three categories.

- **Point-related graph queries.** Graph queries compute one or more individual points. Similarity queries are famous point-related graph queries which retrieve vertices similar to given query vertices. They are widely utilized in article recommendations [1].
- **Path-related graph queries.** Graph queries are closely related to paths between vertices. In this paper, shortest path queries and reachability queries whose goals are to compute the shortest path and check the reachability between two given query vertices are classified as path-related graph queries. These queries play an important role in online mapping and navigation services.
- **Subgraph-related graph queries.** Graph queries return subgraphs meeting specified user preferences. For instance, community searches, which aim to search for special dense subgraphs, belong to subgraph-related graph queries. These queries are significant in advertising and viral marketing, content recommendation, team building, and other real-life applications.

Besides, personalized graph queries in the same category are further divided into different subcategories in terms of different types of graphs. As well as simple graphs without special attributes, other graphs have different types of attributes, including time, location, keywords, weights, probabilities, etc. As introduced in [6], simple graphs only consider links between vertices, while attributed graphs consider both links and attributes. It is worth noticing that these different types of graphs are closely related to diverse user preferences. For instance, in temporal graphs, users can query the similarity of certain vertices in users' caring for time intervals [44]. Users can find related communities by specifying the common likelihood in keyword graphs [109–112]. Based on location graphs, users can find friends within a certain distance [38,113–115].

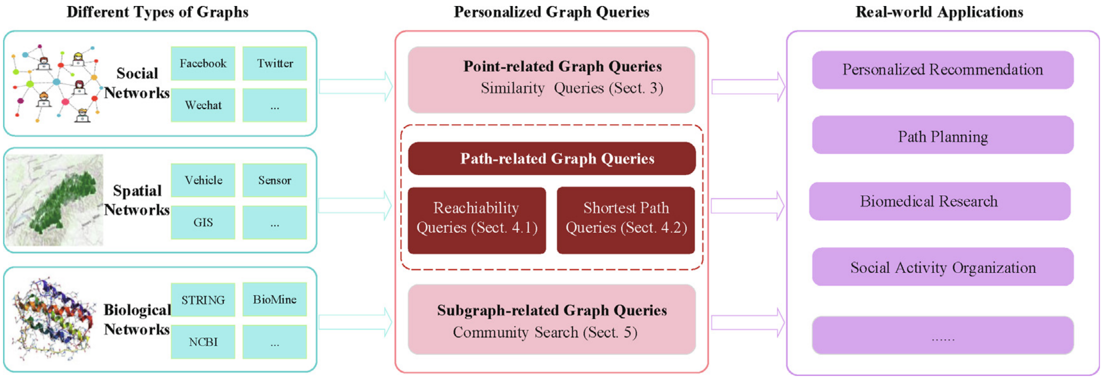

Different Types of Graphs
Personalized Graph Queries
Real-world Applications
Social Networks
Facebook
Twitter
Wechat
...
Spatial Networks
Vehicle
Sensor
GIS
...
Biological Networks
STRING
BioMine
NCBI
...
Point-related Graph Queries
Similarity Queries (Sect. 3)
Path-related Graph Queries
Reachability Queries (Sect. 4.1)
Shortest Path Queries (Sect. 4.2)
Subgraph-related Graph Queries
Community Search (Sect. 5)
Personalized Recommendation
Path Planning
Biomedical Research
Social Activity Organization
......

Fig. 1. Applications of Personalized Graph Queries.

Table 1
Classification of personalized graph queries.

Categroies	Problems	Simple Graphs	Attributed Graphs					
			Keyword	Location	Temporal	HINs	Weighted	Probability
Point-related	SimRank Similarity Query	[7–47]			[44]	[48,49]	[50,23]	[51,52]
Path-related	Reachability Query	[53–58,57,59–65]			[66–68]	[69–72]		[73–77]
	Shortest Path Query	[78–86]		[87,4,88–93,3,94–96]	[97–100]	[101]		[102,103]
Subgraph-related	k-core-based Community Search	[104–108]	[109–112]	[113,114,38,115]	[116]		[117–121]	
	k-truss-based Community Search	[122–124]	[125]				[126]	

Moreover, for personalized graph queries belonging to the same category, we further divided these queries due to the composition of query vertices. Finally, we classify algorithms for the same personalized graph query into two categories, online algorithms, and index-based algorithms, according to whether indexes are utilized to organize the given graphs for boosting query performance.

There are several overviews for special graph queries. In pointed-related queries, Zhang et al. [127] summarized the relationship between different computation methods of SimRank before 2015. In path-related queries, Sommer et al. [128] surveyed the shortest-path queries just in static graphs before 2014. In subgraph-related queries, Malliaros et al. [129] reviewed the core decomposition problems from three aspects: basic concepts, algorithms, and applications. In [6], Fang et al. summarized the existing work on community search problems. Moreover, they also analyzed the advantages and disadvantages of different community models and compared the effectiveness and efficiency of the corresponding approaches. Apart from the surveys on community searches, [130] focused on the distinctive features and challenges of dynamic community discovery. Bian et al. [131] reviewed the existing works on top k nodes identification in social networks, top k influential nodes and top k significant nodes, from theory and application. In [132], they briefly introduced the existing community query-related research topics and future directions.

Unlike the surveys above, this paper provides a comprehensive review of the research on graph queries based on personalized graph queries in the pointed, path, and subgraph-related categories, while Wang et al. [133] introduced attributed graph queries which are based on whether the query is structured. We also give the definition of different attribute graphs, where the attribute is related to the personalized query needing. And we introduce state-of-the-art approaches to handle the corresponding queries. It is beneficial for readers to review the current research thoroughly and gain directions for future research. All in all, the contributions of this paper are mainly as follows.

- We carry out a comprehensive literature review of personalized graph queries and classify these queries into different categories based on different structures returned.
- We analyze the representative and state-of-the-art approaches to personalized graph queries by considering the types of graphs.
- We exploit the challenges faced by personalized graph queries and offer directions for further study.

The rest of this paper is organized as follows. In Section 2, we review graphs closely related to personalized graph queries. In Section 3, we focus on similarity queries that are point-related. In Section 4, we discuss shortest path queries and reachability queries, both of which are path-related. In Section 5, we pay attention to community searches with the goal of computing dense subgraphs. In Section 6, we exploit the challenges faced by personalized graph queries and offer research directions. Finally, in Section 7 we conclude this paper.

2. Types of graphs

In this paper, the three aspects of user preferences, including specified query vertices, structures, and attributes, are taken into account. We mainly focus on the personalized graph queries which need users to specify query vertices. Based on the structure returned, all these queries are divided into three categories: point-related, path-related, and subgraph-related graph queries. Moreover, personalized graph queries of the same category are further divided into different subcategories in terms of attributes.

In a graph, a vertex denotes an entity and an edge denotes a connection relationship between two entities. User preferences for attributes correspond to different types of graphs, including simple and attributed graphs. A simple graph is a graph that only considers the topology between the vertices [133]. Attributed graphs consider both the topology between vertices and the feature of vertices and edges. Attributed graphs considered in this paper mainly include temporal graphs, location-based graphs, keyword-based graphs, weight-based graphs, heterogeneous graphs, and probability graphs. These graphs are closely related to different real-world scenarios, and play a crucial role in personalized graph queries.

In the following, we introduce various types of attributed graphs that are prevalent in personalized graph queries.

2.1. Temporal graphs

A temporal graph is a graph with a time attribute [130], and it can be formally defined as follows.

Definition 1 (*Temporal Graph*). A temporal graph $G = (V, E)$ consists of a vertex set V and an edge set E. Each vertex $v \in V$ is a triplet (v, t_s, t_e), where $[t_s, t_e]$ is the time interval of v, i.e., t_s and t_e denote the start and end times of the vertex v. Each edge $e \in E$ is a quadruplet (u, v, t_s, t_e) where $u, v \in V$, $[t_s, t_e]$ is the time interval of the edge (u, v), i.e., t_s and t_e are the start and end times of the edge (u, v).

Fig. 2(a) shows a temporal graph where edges exist at different time instances. For instance, the edge (v_1, v_3) exists at time 3, and the edge (v_2, v_3) is active at time 5. In the time interval $[1, 6]$, we can get a projected graph shown in Fig. 2 (b) by combining all the edges existing in this interval.

Temporal graphs can be naturally adapted to many practical applications where relationships between entries only persist for a time interval. Take a social network as an example. For a person, his/her comments on the message of his/her friends have a time attribute because these comments are submitted at different times.

In this paper, we summarize personalized queries, including reachability queries (Section 4.1.2), shortest path queries (Section 4.2.3), and community searches (Section 5.1.2) over temporal graphs, respectively.

2.2. Location-based graphs

A graph with location information is called a location-based graph, which can be defined as follows.

Definition 2 (*Location-based Graph*). A location-based graph $G(V, E)$ is a graph in which each vertex $v \in V$ is associated with a location $v.l = (v.x, v.y)$. Here $v.x \in \mathbb{R}$ and $v.y \in \mathbb{R}$ are the coordinates of latitude and longitude.

Fig. 3 illustrates a location-based graph with 13 vertices. As illustrated, each vertex representing a person is associated with a geographical location.

Personalized graph queries over location-based graphs have also attracted much attention for their wide use in location-based service (LBS) applications such as travel recommendations and social activity organizations. In these LBS applications, users are allowed to share their location information with their friends. As a result, we can find people who are geographically close and have close social connections.

In this paper, we review studies on personalized queries, including shortest path queries (Section 4.2.2) and community searches (Section 5.3.2) over location-based graphs, respectively.

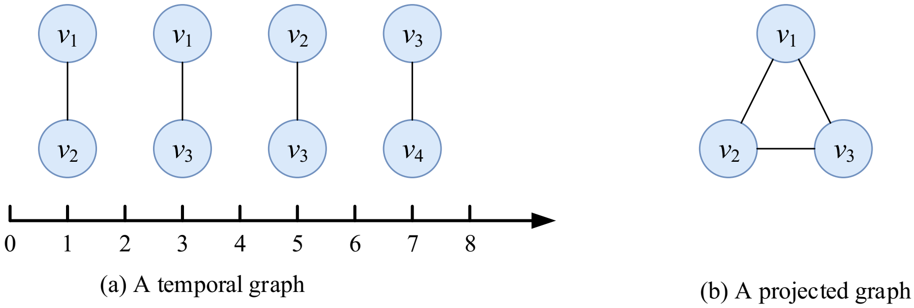
2.3. Keyword-based graphs

A keyword-based graph is a graph where each vertex is associated with keywords. Formally, the keyword-based graph can be introduced as follows.

Definition 3 (*Keyword-based Graph*). A keyword-based graph $G = (V, E)$ consists of a vertex set V and an edge set E, where each vertex $v \in V$ is associated with a keyword set $v.\wp = \{s_1, s_2, \ldots, s_i\}$ (s_i is a string attribute description). Each keyword, within $v.\wp$, represents different attributes of the vertex v.

Fig. 4 shows a social network graph with 14 vertices, where each vertex is associated with a keyword set to denote its traits.

The keyword-based graphs are widespread in many real-life applications including social networks, bibliographical networks, knowledge graphs, and so on [110]. As introduced in [134], the keywords of each user (vertex) can represent location,



(a) A temporal graph

(b) A projected graph

Fig. 2. A Temporal Graph and its Projected Graph at the Time Interval $[1, 6]$.

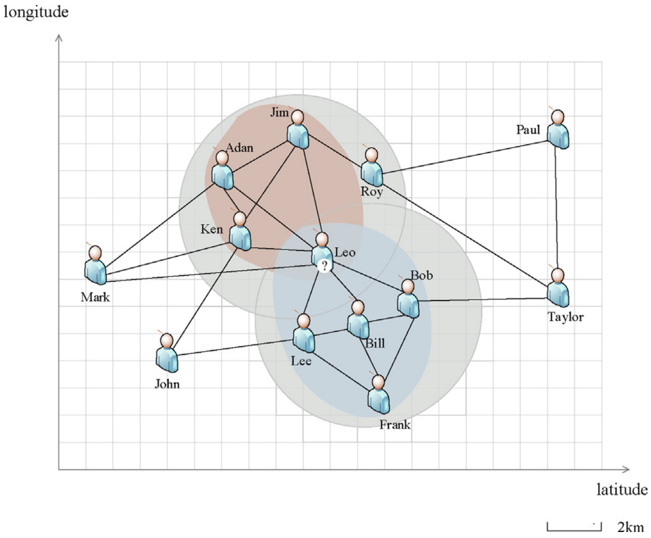


Fig. 3. An Example of a Location-based Graph [38].

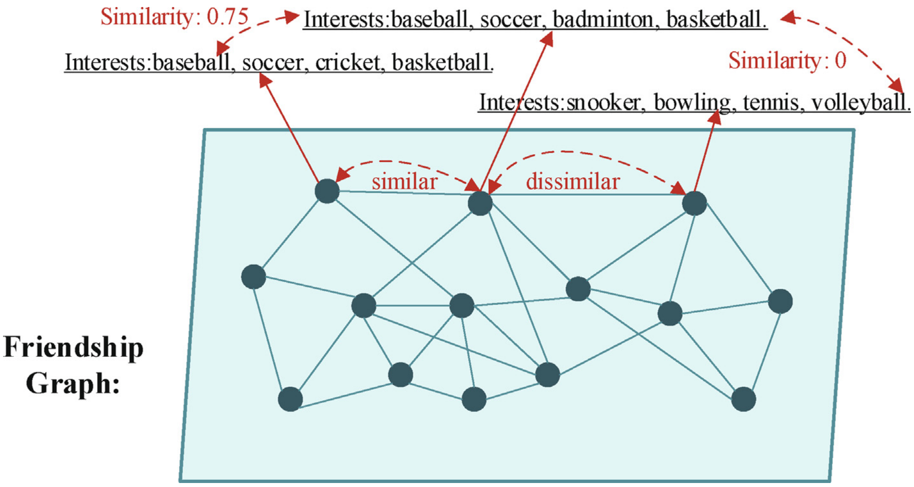

Similarity: 0.75
Interests:baseball, soccer, badminton, basketball.
Interests:baseball, soccer, cricket, basketball.
Similarity: 0
Interests:snooker, bowling, tennis, volleyball.
similar
dissimilar
Friendship Graph:

Fig. 4. An Example of a Keyword-based Graph [135].

interests, and other contents generated by the user. Over the keyword-based graph, we can obtain user groups composed of users having similar interests, which make these groups stable and active.

In this paper, we review related work on community searches (Section 5.3.2) over keyword-based graphs.

2.4. Weight-based graphs

A graph with numerical attributes is called a weight-based graph, which could be formulated as follows.

Definition 4 (*Weight-based Graph*). A weight-based graph $G = (V, E)$ consists of a vertex set V and an edge set E, where each vertex $v \in V$ or each edge $e \in E$ is associated with a weight (i.e., influence) $v.w \in \mathbb{R}$ or $e.w \in \mathbb{R}$. Here the weight w of a vertex or an edge represents its importance.

There is an example of a weight-based graph in Fig. 5 where each edge $e \in E$ has a weight $e.w$ denoting the similarity between two vertices.

Weight-based graphs are widely utilized in social networks, road networks, and other fields. Take the road network as an example. An edge in a road network has a weight to represent the travel cost or distance between two vertices. Weight information is a significant factor affecting shortest path queries on road networks.

In this paper, we summarize the related work of similarity queries and community searches over weight-based graphs in Sections 3.3.1 and 5.3.4, respectively.

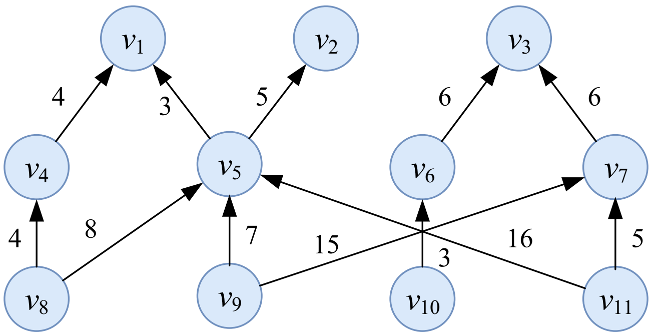


Fig. 5. An Example of a Weight-based Graph.

2.5. Heterogeneous graphs

In bibliographic networks and knowledge graphs, heterogeneous information networks are prevalent. These networks are composed of various types of vertices and various connections between vertices. In this paper, heterogeneous information networks are also called heterogeneous graphs.

Definition 5 (*Heterogeneous Graph [136]*). A heterogeneous graph $G = (V, E)$ is a directed graph in which each vertex $v \in V$ has a vertex type mapping function $\psi : V \to \mathscr{V}$ and each edge has a vertex type mapping function $\phi : E \to \mathscr{E}$. Each vertex has a vertex type $\psi(v) \in \mathscr{V}$ and an edge $e \in E$ has an edge type $\phi(e) \in \mathscr{E}$.

As shown in Fig. 6, there is a heterogeneous graph of a database system and logic programming (DBLP) network which contains four different vertex types, i.e., the authors $a_1, \ldots, a_4$, the papers $p_1, \ldots, p_4$, the topic t_1, and the venue v_1. At the same time, there are three kinds of directed edge types denoting different relationships, that are, "author$\overset{\text{write}}{\to}$paper", "paper$\overset{\text{mention}}{\to}$topic", and "paper$\overset{\text{publn}}{\to}$venue".

In this paper, we review the research on reachability queries (Section 4.1.2), shortest path queries (Section 4.2.2), and community searches (Section 5.3.5) over heterogeneous graphs, respectively.

2.6. Probability graphs

Due to privacy preservation, noise measurements, and inconsistent information sources, probability graphs exist widely in social networks, biological networks, and mobile networks. In this paper, we focus on the personalized queries over probability graphs in which edges are uncertain.

Definition 6 (*Probability Graph*). A probability graph $\mathscr{G} = (V, E, p)$, also known as an uncertain graph, is composed of a vertex set V and an edge set E where each edge $e \in E$ has an existence probability $e.p$.

The probability graph usually carries possible world semantics [137,138], and the existence probability of a possible world g is calculated as

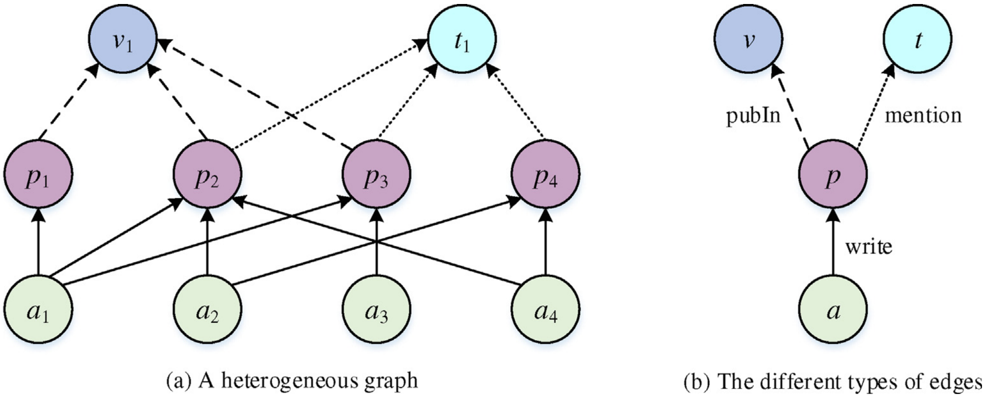


Fig. 6. An Example of a Heterogeneous Graph.

$$\Pr(g) = \prod_{e \in E'} e.p \times \prod_{e \in E \setminus E'} (1 - e.p), \tag{1}$$

where E' includes all the edges within $\mathscr{G}$ and $e.p$ represents the existential probability of an edge e.

Fig. 7(a) shows an uncertain graph where each connection/edge has an existence probability. All possible worlds and their existence probabilities are depicted in Fig. 7(b).

In numerous applications, uncertainty has been an inherent property of graph data. Personalized graph queries over probability graphs have attracted extensive attention. In this paper, we survey similarity queries (Section 3.3.2), reachability queries (Section 4.1.2), and shortest path queries (Section 4.2.2) over probability graphs, respectively. Fig. 8. Table 1.

Table 2 shows the frequently-used symbols in this paper. Fig. 9.

3. Point-related personalized queries

In point-related personalized graph queries, similarity query is focused a lot. It is a fundamental problem to measure similarities among different vertices in the applications of graph analysis and mining, such as recommendation systems [139],

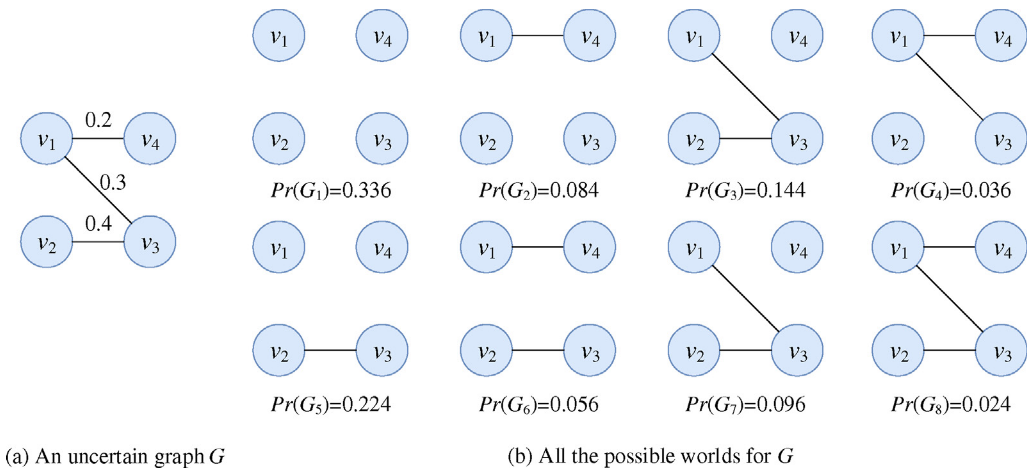


Fig. 7. An Example of Possible Worlds [51].

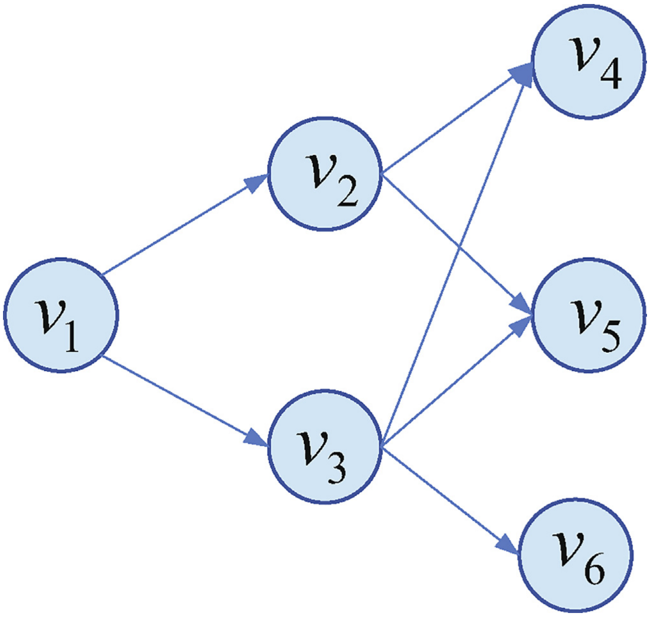
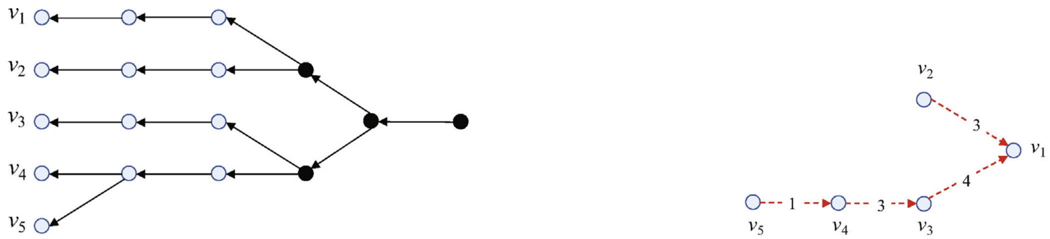


Fig. 8. An Example of the Iterative and Random Walk Methods.

Table 2
The Summary of Frequently-used Symbols.

Symbol	Definition
G	A simple graph
u, v	A vertex
e	An edge
V, E	The vertex set and edge set of G
n, m	The number of vertices and edges
q, Q	A query vertex and a query vertex set
$deg(v)$	The degree of a vertex v
t_s, t_e	The start and end times of a vertex v
$\wp$	A keyword set
$e.p$	The existence probability of an edge e
$Pr(G)$	The existence probability of a possible world instance G
$Sim(u, v)$	The SimRank similarity score of two vertices u and v
$v.w$, $e.w$	The weight of a vertex v and an edge e
$v.l = (v.x, v.y)$	The location of a vertex v
$\psi : V \rightarrow \mathscr{V}$	The mapping function of vertex type
$\phi : E \rightarrow \mathscr{E}$	The mapping function of edge type



(a) First meeting times of coalescing reversed walks of v_1, v_2, v_3, v_4, and v_5.

(b) The fingerprint tree for v_1, v_2, v_3, v_4, and v_5.

Fig. 9. An Example of a Fingerprint Tree [8].

link prediction [140], spam detection [141], and graph mining [142]. In general, a similarity measure can be utilized to cluster different objects, such as for collaborative filtering in a recommender system, where similar users and items are grouped according to user preferences [7]. Table 3.

In Section 3.1, we give an overview of the famous similarity models. Consider the SimRank similarity model is acknowledged as one of the most popular and promising models to measure the similarity between two vertices [41]. In the rest part of this section, Simrank computing models and SimRank similarity queries, which belong to classic point-based personalized graph queries, over different types of graphs are reviewed in Section 3.2 and Section 3.3.

3.1. Preliminary

In this section, we introduce the similarity model utilized to measure the similarity between vertices.

3.1.1. Similarity models

At present, there are many different connection-based, also called link-based, similarity models, including Jaccard [143], Dice [144], Cosine [145], Bibliographic Coupling [146], Co-citation [147], and SimRank [7]. In Table 3, we illustrate different similarity models for two vertices u and v, where $N(u)$ $(N(v))$ denotes the neighbor set vertices of vertex u (v), $c(c \in [0, 1])$ is a damping factor, $I(u)$ $(I(v))$ and $O(u)$ $(O(v))$ are separately the in-neighborhood and out-neighborhood vertices of vertex u (v), $|I(u)|$ $(|I(v)|)$ denotes the in-neighborhood number of vertex u (v), and $I_i(u)$ $(I_j(v))$ is the i^{th} (j^{th}) in-neighborhood of vertex u (v).

The similarity models, Jaccard, Dice and Cosine, are widely utilized in information retrieval. The co-citation similarity model was first introduced by Small et al. [147] in the fields of citation analysis and bibliometrics as a fundamental metric to characterize the similarity between scientific papers. Besides, Bibliographic Coupling was proposed by Kessler et al. [146] to measure paper similarities. It is designed based on the observation that authors of papers on the same topic tend to cite the same papers. Unlike the above models, SimRank is proposed on the basis of the intuition that two objects are similar if they are referenced by similar objects [7].

Table 3
Similarity Models.

Similarity models	Equations
Jaccard	$\lvert N(u) \cap N(v)\rvert / \lvert N(u)) \cup N(v)\rvert$
Dice	$2\lvert N(u) \cap N(v)\rvert / \lvert N(u)\rvert + \lvert N(v)\rvert$
Cosine	$\lvert N(u) \cap N(v)\rvert / \sqrt{\lvert N(u)\rvert^2 + \lvert N(v)\rvert^2}$
Co-citation	$\lvert I(u) \cap I(v)\rvert$
Bibliographic Coupling	$\lvert O(u) \cap O(v)\rvert$
SimRank	$\frac{c}{\lvert I(u)\rvert \lvert I(v)\rvert} \sum_{i=1}^{\lvert I(u)\rvert} \sum_{j=1}^{\lvert I(v)\rvert} s(I_i(u), I_j(v))$

Consider the popular similarity models shown in Table 3. The Jaccard, Dice, Cosine, Co-citation, and Bibliographic Coupling similarity models only consider the neighborhood vertices of the query vertices and can only mine the local similarity of vertices. Compared with these similarity models, SimRank iterative takes into account the neighborhood of the neighborhood as shown in row 6 of Table 3. Then, the SimRank model has two advantages: the first one is capturing the entire topology of a given graph; the another one is effectively evaluating the similarity between each pair of vertices. Accordingly, SimRank is recognized as the most popular model in the existing similarity models. Therefore, in this paper, we mainly focus on the SimRank model and SimRank-based similarity queries.

3.1.2. SimRank computing methods

Since Jeh et al. [7] proposed the SimRank model, different computing methods of SimRank were designed. These methods mainly contain three categories: iterative method [7], random walk method [7,8], and non-iterative method [127]. The calculation formulas of the three methods are shown in Table 4. Here, **S** is the SimRank score matrix that contains the SimRank scores of any two vertices in a given graph, **W** denotes the column normalized matrix of an adjacency matrix, **I** represents an identity matrix, $\vee$ is the operator that sets the diagonal elements of the left-hand side to the corresponding elements of the right-hand side, t is the number of iterations, and **D** is a diagonal matrix. In the random walk method, for two vertices u and v, $p_{ft}(u, v, w)$ represents the probability of a pair of random walks, i.e., it starts from u and v, respectively, walks on the length of t and first meets at vertex w.

Consider the iterative and random walk methods [7]. The iterative method obtains the similarity of any two vertices through an iterative formula, while the random walk method computes the similarity of any two vertices according to the first-meeting probability of the reverse random walk. Fig. 8 shows an example of SimRank's iterative and random walk methods. Let the damping factor c be 0.6. Based on the iterative method, the similarity of the vertices v_4 and v_5 is $s(v_4, v_5) = [0.6/(2 \times 2)] \times [s(v_2, v_2) + s(v_2, v_3) + s(v_3, v_2) + s(v_3, v_3)] = 0.48$, where the similarity between each vertex and itself is 1, and $s(v_2, v_3) = [0.6/(1 \times 1)] \times s(v_1, v_1) = 0.6$. Besides, according to the random walk method, vertices v_4 and v_5 meet in two cases:

1) by two paths $v_4 \leftarrow v_2 \rightarrow v_5$ and $v_4 \leftarrow v_3 \rightarrow v_5$ with $t = 1$;
2) by two paths $v_4 \leftarrow v_2 \leftarrow v_1 \rightarrow v_3 \rightarrow v_5$ and $v_4 \leftarrow v_3 \leftarrow v_1 \rightarrow v_2 \rightarrow v_5$ with $t = 2$.

As a result, we have $s(v_4, v_5) = \frac{1}{2} \times \frac{1}{2} \times 0.6^2 + \frac{1}{2} \times \frac{1}{2} \times 0.6^2 + \frac{1}{2} \times \frac{1}{2} \times 0.6^1 + \frac{1}{2} \times \frac{1}{2} \times 0.6^1 = 0.48$.

As pointed out in [11], the iterative method is infeasible and inefficient, especially when the graph changes dynamically and frequently. To bridge this gap, Li et al. [11] developed a non-iterative method that can be utilized to calculate the similarity scores for arbitrary subsets of vertices. In addition, these similarity scores can be updated incrementally. To enable non-iterative calculations, the SimRank equation is rewritten as a non-iterative form by adopting the Kronecker product and vectorization operator. In [20], Maehara et al. introduced a SimRank linearization technique which can efficiently transform the SimRank computing to a linear equation problem. Besides, they use a more precise diagonal matrix **D** to replace $(1 - c)\mathbf{I}$ in the formula of the non-iteration method in [20]. Later, in [127], Zhang et al. analyzed the relationship between the non-iterative method and the random walk method. They inferred that the non-iterative method can be transformed to the random walk method without guaranteeing first-meeting [127].

Among the three SimRank computing methods, iterative methods have the highest accuracy. However, they are not appropriate for large and dynamic graphs. For static graphs, current iterative solutions are not efficient in time and space.

Table 4
SimRank Computing Methods.

Computing Method	Formula
Iterative [7]	$\mathbf{S}_t = (c\mathbf{W}'\mathbf{S}_{t-1}\mathbf{W}) \vee \mathbf{I}$
Random Walk [7]	$Sim(u, v) = \sum_{t=0}^{\infty} c^t \sum_{w \in V} p_{ft}(u, v, w)$
No-iterative [11,20]	$\mathbf{S} = c\mathbf{W}'\mathbf{S}\mathbf{W} + (1 - c)\mathbf{I}$ $\mathbf{S} = c\mathbf{W}'\mathbf{S}\mathbf{W} + \mathbf{D}$

Non-iterative methods have lower time and space complexities than iteration methods. Since the computing procedure has uncertainty, the accuracy of random walk methods cannot be ensured.

3.2. SimRank similarity queries over simple graphs

SimRank similarity queries in simple graphs can be divided into four categories: single-pair, single-source, partial-pairs, and all-pairs SimRank similarity queries. Particularly, a single-pair SimRank similarity query returns the SimRank score for two given vertices, a single-source SimRank similarity query returns the SimRank scores for a vertex and all vertices in a given vertex set, a partial-pairs SimRank similarity query retrieves the SimRank scores for all vertex pairs between two given vertex sets, and an all-pairs SimRank similarity query retrieves the SimRank scores for all vertex pairs in a given graph. In this section, in addition to simple SimRank similarity queries, we also introduce their important variants, top k and threshold-based SimRank similarity queries, which aim to compute vertex pairs with size constraints.

3.2.1. Single-pair SimRank similarity queries

In [40], Wang et al. proposed the pairwise SimRank estimation (PSE) problem, which belongs to the single-pair SimRank similarity query.

Problem 1 (*Pairwise SimRank Estimation [40]*). Given a graph $G = (V, E)$, a vertex pair (u, v) for $u, v \in V$, and an error-bound ϵ, return the SimRank score $\hat{s}(u, v)$ such that $|\hat{s}(u, v) - s(u, v)| < \epsilon$.

For single-pair SimRank similarity queries, there are online algorithms and index-based algorithms designed based on the no-iterative and random walk methods, respectively.

In the online algorithms, Li et al. [148] designed a random walk method, which repeatedly aggregates the meeting probability of k-hop from the query vertices u and v. The time complexity is $O\left(Kd^2 \times \min\left\{d^K, n^2\right\}\right)$, where $K = \lceil \log_c \epsilon \rceil - 1$. In [20], a no-iterative method with a diagonal correction matrix **D** was first designed, which can convert the calculation of SimRank to a linear equation problem. The query time needs $O(tm)$. For accelerating the method of [20], Li et al. [24] devised a parallel algorithm, namely CloudWalker-MCSP. It computes a length-n indexing vector on the Apache Spark platform. Although the approaches in [20,24] can effectively handle single-pair SimRank similarity queries, neither can gain accurate query results. In [40], the authors presented an online algorithm, namely backward local push and MC sampling (BLPMC), based on the random walk method. BLPMC applies the technique of backward local push to reduce the size of random walk sampling.

There have also been plenty of index-based algorithms using the random walk method [8,18,30,40]. In [8], the single-pair SimRank similarity query was first resolved based on the random walk method. A fingerprints tree index was constructed to efficiently organize the random walk information. Fig. 9 illustrates an example of a fingerprint tree. Since the first meeting times for vertex u_1 and u_2 is 3, so the value of directed edge (u_2, u_1) in fingerprint tree is 3.

Furthermore, He et al. [18] introduced a position matrix that can efficiently save the first-meeting probability of any two random walks. By iteratively computing the position matrix, final query results can be gained directly. In [30], an index-based algorithm, SimRank via local updates and sampling (SLING), was designed based on a reverse random walk method by adopting local update and sampling techniques. SLING algorithm needs $O(1/\epsilon)$ query time, $O(n/\epsilon)$ index space, and $O(m/\epsilon + n/\log n/(\delta\epsilon^2))$ index constructing time, where δ is the failure probability and ϵ is the maximum additive error allowed in any SimRank score. The state-of-the-art work of single-pair SimRank similarity queries over simple graphs was developed by Wang et al. [40]. They designed two index-based algorithms, namely forward local push and MC sampling (FLPMC) and forward and backward local push and MC sampling (FBLPMC), to process static and dynamic single-pair SimRank similarity queries, respectively. These algorithms adopt backward and forward local push strategies for reducing the size of random walk samples.

3.2.2. Single-source SimRank similarity queries

A single-source SimRank similarity query can be formally defined as follows.

Problem 2 (*Simple Single-source SimRank Similarity Query [39]*). Given a graph $G = (V, E)$ and a vertex $u(u \in V)$, compute the SimRank scores $s(u, v)$ of vertex pairs (u, v) for all the vertices $v \in V$.

Top k and threshold-based single-source SimRank similarity queries are two important variants of the simple single-source SimRank similarity query, which return k vertices with the highest SimRank scores and vertices with SimRank scores larger than a specified threshold, respectively.

- **Simple Single-source SimRank Similarity Query.** Yu et al. [26] proposed the SimRank$^\sharp$ ($SR^\sharp$) algorithm on the basis of the non-iterative method. They designed an efficient method that can accurately calculate the diagonal correction matrix **D**. This method can solve the connectivity trait problem of SimRank through a kernel-based calculation. Here the connectivity trait problem is a phenomenon in which the SimRank score decreases with the increase of paths between vertices. Later, Liu et al. [34] presented an online algorithm based on the random walk method with a provable approximation guarantee. Besides, three pruning rules were designed for computing the first-meeting probability of a given vertex u with respect to partial walks. In [42], the authors noted that the SimRank similarity method has the disadvantage of zero similarity, i.e., "a path between two vertices does not contribute to the SimRank score if its length is odd", and then a SimRank* model was developed. Based on SimRank*, the authors devised ss-gSR* for geometric SimRank* computing and ss-eSR* for exponential SimRank* computing, respectively, where ss-gSR* and ss-eSR* are both memory-efficient algorithms. In [44], Li et al. proposed CrashSim which is more efficient to READS [32], SLING [30], or Probesim [34]. They also extended CrashSim to CrashSim-T for the SimRank similarity query in temporal graphs. In [47], a novel matrix random sampling approach was proposed to accelerate the computation with less memory cost. Wang et al. [43] pointed out that the exact SimRank algorithm is time-consuming for a large graph with more than 10^6 vertices. To address this concern, they investigated the approach to compute the exact single-source and top k SimRank results on large graphs.

Apart from the above online algorithms, there are also index-based algorithms for simple single-source SimRank similarity queries. In [30,39], a single-source SimRank similarity query was computed based on the property that the SimRank score $s(u, v)$ of two given vertices u and v is equivalent to the probability that two $\sqrt{c}$-walks meet. The $\sqrt{c}$-walks meet means that a vertex u in the graph stops at the current vertex with a probability of $1-\sqrt{c}$, or walks to the in-degree neighbor of the current vertex with a probability of $\sqrt{c}$. In [30], an index was constructed to store all the random walk information, but the solution in [39] only considered hub vertices which can reduce space cost without sacrificing query efficiency. Due to the limitations of memory and computational power, a single machine fails to process a large graph. Therefore, Wang et al. [149] researched single source SimRank queries in a distributed environment. They proposed a distributed framework, namely DISK, based on the linearized formulation of SimRank which includes offline and online phases.

- **Top k Single-source SimRank Similarity Query.** Lee et al. [14] introduced two random walk methods, master and slave random walks, and then proposed a local top k search method, namely TopSim. The master random walk starts from any vertex and ends at the query vertex. The start vertex of the slave random walk is consistent with the master random walk. It passes through the same long path as the master random walk and stops at certain points in the master random walk path. The final results are the k vertices with the highest scores on the product graph. Kusumoto et al. [21] adopted a non-iterative method to calculate the SimRank scores of vertex pairs. Furthermore, two upper limits are utilized to prune unqualified vertex pairs through MC simulation. Finally, the final results are obtained by adaptive tactics.

  In [25], Shao et al. designed an index-based algorithm on the basis of sampling random walks. First, a random walk index is constructed by sampling a set of one-way graphs. A one-way graph is a graph in which each vertex in the reversed graph contains only one random walk. Then, the results are computed by utilizing the connectivity of the one-way graph, which can be updated efficiently.
- **Threshold-based Single-source SimRank Similarity Query.** The threshold-based single-source SimRank similarity query was first researched in [16]. Focusing on the threshold-based single-source SimRank similarity query, Fujiwara et al. [16] designed the SimMat algorithm without an index. SimMat can also be utilized to process the top k single-source SimRank similarity query. It first calculates the approximate SimRank similarity of partial vertex pairs based on the Sylvester equation and then prunes unqualified vertex pairs according to the Cauchy-Schwartz inequality. Later, in [45], Liu et al. improved the query efficiency by integrating several random walk sampling strategies and solved both the threshold and top k single-source SimRank similarity queries based on the problem of multi-armed bandits.

3.2.3. Partial-pairs SimRank similarity queries

The simple partial-pairs SimRank similarity query was first investigated in [27]. It is also called partial-pairs SimRank assessment and can be defined as follows.

Problem 3 (*Simple Partial-pairs SimRank Similarity Query*). Given a graph $G = (V, E)$, and two collections of vertices A and B, compute the SimRank score $s(u, v)$ of each vertex pair (u, v) for $u \in A$ and $v \in B$.

Different from the simple partial-pairs SimRank similarity query, the top k and threshold-based partial-pairs SimRank similarity queries return k vertex pairs with the highest SimRank scores and vertex pairs with SimRank scores larger than a specified threshold, respectively.

- **Simple Partial-pairs SimRank Similarity Query.** For a simple partial-pairs SimRank similarity query, there is only an online approach. In [27], Yu et al. proposed an iterative method based on a seed germination technique. Moreover, a hashing structure was introduced to decrease unnecessary computations without loss of accuracy.
- **Top k Partial-pairs SimRank Similarity Query.** Sun et al. [12] first investigated the top k partial-pairs SimRank similarity query, which is also called the link-based similarity join. Given two sets of vertices in a graph, it returns k vertex pairs with the highest similarity according to an e-function of a common measurement similar to SimRank. They presented an iterative deepening join algorithm, which is an online algorithm adopting the iterative computation method. In each iteration, unqualified vertex pairs could be identified according to their approximate SimRank similarity.

  Wang et al. [41] studied an approximate algorithm for the top k SimRank query where partial-pair is a pair set Q instead of two collections of vertices A and B. Although such query can be processed by the approaches to multiple single-pair similarity queries or typical partial-pair similarity queries, it may lead to inefficiency due to the existence of many redundant calculations. Inspired by this, the Carmo algorithm was developed. It first uses a heap to improve the state-of-art pairwise estimation method BLPMC [40], and then introduces the upper and lower bound estimation method to prune the search space for better query performance.

- **Threshold-based Partial-pairs SimRank Similarity Query.** In [35], Zheng et al. presented a SimRank-based join (SRJ) query that aims to compute vertex pairs from two vertex sets U and V that satisfy a threshold constraint. A filter-and-refine framework was proposed for the SRJ query. In the filter phase, it computes a shortest path distance-based upper bound of SimRank to avoid redundant SimRank computations. Besides, they also proposed a simple method to identify whether a vertex u is in the h-hop neighbors of another vertex v; this method can be utilized to avoid computation on the shortest path distance. In the verification phase, the SimRank scores of the left vertex pairs are computed on the basis of an $h - go\ cover^{+}$ vertex set. Here, the $h - go\ cover^{+}$ of a vertex-pair graph G is a set of vertices whose removal leaves a graph G' without a circle and each simple path in G' is of length no longer than h for $h \geqslant 1$.

3.2.4. All-pairs SimRank similarity queries

There are three types of all-pair SimRank similarity queries, including simple, top k and threshold-based all-pair SimRank similarity queries. In particular, a simple all-pairs SimRank similarity query is formulated as follows.

Problem 4 (*Simple All-pairs SimRank Similarity Query*). Given a graph $G = (V, E)$, return SimRank scores of all vertex pairs (u, v) for $u, v \in V$.

Again, the goal of top k and threshold-based all-pairs SimRank similarity queries is to return k vertex pairs with the highest SimRank scores and vertex pairs whose SimRank scores exceed a specified threshold, respectively. It is worth noticing that all algorithms for all-pairs SimRank similarity queries are online.

- **Simple All-pairs SimRank Similarity Query.** In [17,19], the partial sum was introduced to the iterative calculation of SimRank. In particular, the partial sum for two given vertices u and v is

$$s(u, v) = \frac{c}{|I(u)||I(v)|} \sum_{x \in I(u)} \underbrace{\sum_{y \in I(v)} s(I(x), I(y))_{Partial_{I(v)}(x)}},$$

where $Partial_{I(v)}(x)$ denotes the partial sum of vertex v. As shown in Fig. 10, vertices u and v have a common in-neighbor vertex a. To get $s(u,w)$ and $s(v,w)$ for a given vertex $w, Partial_{I(v)}(a) = s(a,d)$ needs to be computed twice because $s(u,w) = \frac{0.8}{4\times1} \times [s(a,d) + s(b,d) + s(c,d) + s(d,d)]$, and $s(v,w) = \frac{0.8}{4\times1} \times [s(a,d) + s(b,d) + s(c,d) + s(d,d)]$.In [19], the computational order of partial sums is adjusted based on the observation that the partial sum of a vertex is closely related to that of the vertex having the most common in-neighbors.

Many parallel approaches to SimRank have been developed to gain better performance. He et al. [10] introduced graphics processing units (GPUs) to efficiently implement the no-iterative method proposed in [11]. Cao et al. [13] designed an incremental calculation equation of SimRank, called Delta-SimRank, and devised a feasible parallel algorithm based on MapReduce. Li et al. [24] used Spark to compute the diagonal correction matrix **D** in parallel and then adopted the MC method to speed up the online query. In [37], Huang et al. designed a method that divides an objective graph due to the idea of modularity maximization and constructs a collapsed graph based on blocks. Moreover, the similarity between vertices inside a block and between different blocks is computed. Finally, the approximate SimRank similarity of all the vertex pairs is obtained. In [46], Wang et al. devised a novel local push based algorithm for processing all-pairs SimRank. Moreover, they proposed an iterative parallel two-step framework for local push to exert the advantage of modern hardwares with multicore CPUs.

There are also numerous studies on the optimization of SimRank similarity models. In [9,23,42], the zero similarity problem of SimRank was considered, i.e., $s(u, v) = 0$ if there is no node having equal distance to both vertex $u$and v. In Fig. 11, when $c = 0.6$ and iterative number $T = 20$, the SimRank values of partial pairs are shown in Table 5. As shown, there are

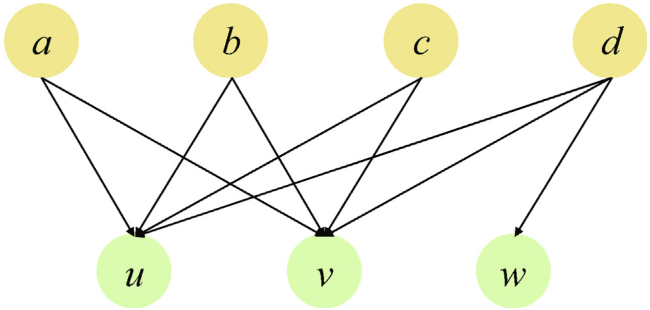


Fig. 10. An Example of Partial Sum.

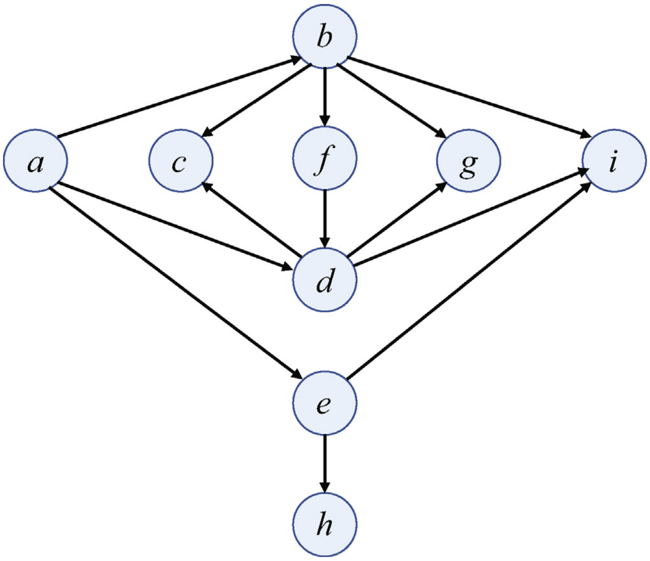


Fig. 11. A Zero-similarity Example.

Table 5
The SimRank Scores of $s(*,f)$.

Vertex Pairs	(a,f)	(b,f)	(c,f)	(d,f)	(e,f)	(f,f)	(g,f)	(h,f)	(i,f)
SimRank score	0	0	0.39	0	0	1	0.39	0.36	0.38

many zero-similarity scores. On one hand, the SimRank values of $s(a,*)$ is 0 (e.g. $s(a,f) = 0$) for $I(a) = 0$. On the other hand, because the lengths of the in-linked paths between two vertices are not equal, there are many zero-similarity scores.

Zhao et al. [9] first noted the zero-similarity problem and designed a P-Rank model in consideration of both internal and external link relationships. Although P-Rank can reduce the partial zero similarity, SimRank is still equal to zero when the in-link and out-link paths of the two given vertices have unequal lengths. Additionally, Chen et al. [23] proposed Asymmetric Network Structure Context Similarity (ASCOS) based on the observation that "if the length of the path between two given vertices is an odd number, this path does not contribute to SimRank". To some extent, ASCOS resolves the zero-similarity problem. However, when the length is even, SimRank may also be zero. In [42], SimRank* was presented to record the information of all the reachable paths for two given vertices, which essentially resolves the zero-SimRank problem. Table 6.

- **Top k All-pairs SimRank Similarity Query.** In [22], Tao et al. proposed the top k all-pairs SimRank similarity query. Moreover, two online algorithms were developed by converting the SimRank calculation between vertices into dot product calculations. It encodes each vertex as a vector, selects $2k$ candidate vertices, and chooses the last k solutions by a tree-based WAND algorithm. This method requires too many redundant calculations to re-scan the $2k$ candidate vertices. Li et al. [33] designed an incremental algorithm for computing SimRank. They also introduced an iterative batch pruning framework, where unqualified vertex pairs can be iteratively pruned.
- **Threshold-based All-pairs SimRank Similarity Query.** In [28], Maehara et al. formulated a threshold-based all-pairs SimRank similarity query and proposed an approximate algorithm with an arbitrary accuracy without an index. This algorithm consists of two phases, filter and verification, based on a no-iterative method. In the filter phase, the Gauss-Southwell algorithm and the stochastic threshold technique are used to calculate candidate vertex pairs, which is efficient to reduce the redundant calculations of SimRank. In the verification phase, the MC algorithm is invoked to prune unqualified vertex pairs that do not meet a given threshold constraint.

3.3. SimRank similarity queries over attributed graphs

3.3.1. Weight-based graphs

In [50], Antonellis et al. focused on query rewriting for sponsored search over an undirected weight-based graph. They pointed out that if overlooking the weighted information in the graph, it may get inaccurate similarity values for given vertex pairs. For example, in Fig. 12, the weights of edges $e(a,b)$ and $e(b,c)$ are 1 and 10, respectively. Apparently, compared to a,b is

Table 6
Classification of SimRank-based Similarity Queries.

Query/ Graph	Algorithms			
	Method	**Type**	**Accuracy**	**Algorithm**
Single-pair/Simple	Random Walk	Index-based	yes	*Fingerprints*[8], FLPMC/BFLPMC [40], ISP [18], SLING [30]
		Online	yes	BLPMC [40], SinglePair [148]
	No-iterative	Online	no	*CloudWalker-MCSP* [24], LIN [20]
Single-source/Simple	Random Walk	Index-based	yes	PRsim [39], READS [32], SLING [30], TSF(tk) [25]
		Online	yes	CrashSim(td) [44], Exactsim(tk) [43], Probesim(tk) [34], SimTab (tk,td) [45], TopSim(tk) [14]
	No-iterative	Online	yes	MSR(tk) [21], SR$^\sharp$ [26]
			no	*CloudWalker-MCSS* [24], LIN(tk,td) [20], RSSD [47], SimMat(tk,td) [16]
	Iterative	Online	no	Ss-gSR*/ss-eSR* [42]
Partial-pairs/Simple	Random Walk	Online	no	Carmo(tk) [41]
	No-iterative	Online	yes	PrunPar/Par-SR [27]
	Iterative	Index-based	yes	h-go over$^+$(td) [35]
		Online	no	IDJ-LB1/LB2(tk) [12]
All-pairs/Simple	Random Walk	Online	yes	SRK-Join(tk) [22]
			no	ASCOS [23], KSimJoin(tk) [33]
	No-iterative	Online	yes	*FLP-SR [153], Inc-SR [36], LIN(tk,td) [20], MonteCarlo(td) [28], SR$^\sharp$ [26]*
			no	*CloudWalker-MCAP* [24], *GPUSRB* [10], LIN(tk,td) [20], NI_Sim [11], PBiGG [154]
	Iterative	Online	yes	C-Rank [31], Naive [7], OIP-DMST [17], OIP-SR [19], Psum-SR [155]
			no	*Delta* [13], LP [46], P-Rank [9], *PartitionSimRank* [37], SOR-SR [15], SimRank* [42], WebSim [29]
All-pairs/Weight	Random Walk	Online	no	ASCOS++ [23]
	Iterative	Online	no	SimRank++ [50]
Single-source/Temporal	Random Walk	Online	no	CrashSim-T(td,na) [44]
Single-source/Probabilistic	Random Walk	Online	no	IDP [51], SS-OP [52]
Single-pair/Probabilistic	Random Walk	Online	no	SR-TS/SR-SP [52]
Single-pair/Heterogeneous	Random Walk	Online	no	Hetesim [49]
Single-source/Heterogeneous	Iterative	Online	no	Pathsim(tk) [48]

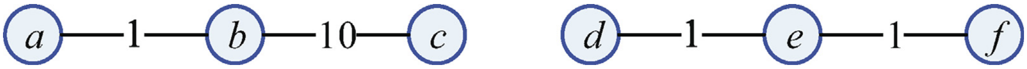


Fig. 12. An Example of Weight-based Graphs.

more similar to c, however, their SimRank values are the same without considering the weight information. To address this concern, SimRank over a weight-based graph was investigated in [50] and computed by the random walk method. In addition, two algorithms were designed on the basis of the weights of edges and evidence in a click graph. The weight-based SimRank was demonstrated to be the best method for creating rewrites on the basis of the click graph.

Later, Chen et al. [23] pointed out that SimRank faces the problem that paths with odd numbers are ignored in the computation of SimRank. To address this issue, two new similarity measures, ASCOS and ASCOS++, were developed. Compared to SimRank, ASCOS can gain a more complete similarity. Furthermore, ASCOS++ improves ASCOS by taking into account the weights of edges in all the paths between two given vertices. Both ASCOS++ and ASCOS are appropriate for distributed environments.

3.3.2. Temporal graphs

To the best of our knowledge, there is only one work that focuses the SimRank similarity query in temporal graphs, i.e., the CrashSim-T algorithm [44].

Problem 5 (*Single-source SimRank Similarity Queries over Temporal Graphs*). Given a temporal graph $G = (V, E)$, a query vertex $u \in V$, and a time interval $[T_1, T_t]$, the problem aims to find a vertex set ω such that each vertex v in $\omega, s(u, v)$ can meet the certain query requirement during the given time interval $[T_1, T_t]$.

Two certain query requirements are proposed by Li et al. [40] that is trend and threshold. The trend and threshold mean that in the time interval $[T_1, T_t]$, the all $s(u, v)$ should continuously increase with time and no less than a threshold all the time, separately. Li et al. improved their CrashSim algorithm, which is based on the random walk model and solved the simple graph single-source SimRank similarity query problem, to CrashSim-T. CrashSim-T first implements the CrashSim algorithm on the temporal graph and then uses a delta pruning and a difference pruning strategy to reduce the time-consuming.

3.3.3. Probability graphs

In real-life applications, including social networks and biological networks, there are many uncertain graphs where edges are associated with probability values. Du et al. [51] first investigated similarity queries over uncertain graphs.

Problem 6 (*All-pairs Similarity Queries over Probability Graphs*). Given an uncertain graph $G = (V, E, P)$, where V is a vertex set, E is an edge set, and P denotes the probabilities of the edges within E, compute the probabilistic SimRank scores of all vertex pairs. The probabilistic SimRank score $S(u, v)$ of two vertices u and v can be computed as $S(u, v)^k = cW^TS(u, v)^{k-1}W + (1-c)I$,where $c \in [0, 1]$ is the decay factor, I is an identity matrix with $S^0 = I$, and W is a probabilistic transition matrix.

In [51], the probabilistic SimRank was used to evaluate the similarity between two vertices in the probability graphs. In an uncertain graph G and two given vertices u and v, when u and v reach the same vertex, the SimRank score of the vertex pair (u, v) is equal to the inverse of the stationary probability. In addition, a probabilistic transition matrix is utilized to describe the transition probability of two vertices. The transition probabilities can be gained by computing those on multiple subgraphs. Based on this observation, the SubG algorithm was designed to calculate the probabilistic transition matrix. After that, a dynamic programming algorithm was developed to boost query performance.

In [51], the k-step transition probability matrix $\mathscr{W}^{(k)}$ was supposed to have an equivalent relationship with the k^{th} power of the one-step transition probability matrix $\mathscr{W}^{(1)}$. As introduced in [150,52], this assumption is unreasonable because it does not correspond to the commonly used possible world model. To solve this problem, Zhu et al. [150,52] exploited a new SimRank measurement for uncertain graphs. For two given vertices u and v, the SimRank similarity is computed on the basis of the probabilities of two random walks that start from u and v, and meet at the same vertex after k transitions for all k. To compute SimRank effectively, efficient algorithms for single-pair and single-source top k SimRank similarity were designed. Finally, a backward tracking technique and an upper bound of the SimRank scores were introduced to further improve the processing efficiency of top k single-source SimRank similarity queries.

3.3.4. Heterogeneous graphs

The similarity query over heterogeneous graphs not only faces the data sparseness problem but also meets the difficulty of mining latent relationships between heterogeneous data objects. To address these issues, Xi et al. [151] studied the information integration problem about how to combine the broad variety of heterogeneous data and relationships effectively and efficiently for the purpose of improving the performance of information applications. A Unified Relationship Matrix (URM) was proposed to represent a set of heterogeneous data objects and their interrelationships. For a given heterogeneous graph with N type data, URM is

$$\mathbf{L}_{URM} = \begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_{12} & \cdots & \mathbf{L}_{1N} \\ \mathbf{L}_{21} & \mathbf{L}_{22} & \cdots & \mathbf{L}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}_{N1} & \mathbf{L}_{N2} & \cdots & \mathbf{L}_{NN} \end{bmatrix},$$

where $\mathbf{L}_i$ (for $1 \leqslant i \leqslant N$) denotes data objects within the same data space, which is a $m \cdot m$ matrix (m is the total number of objects of the data type i); $\mathbf{L}_{ij}(i \neq j)$ represents data objects within different data space, which is a $m \cdot n$ matrix (m is the total number of objects of the data type i, n is the total number of objects of the data type j). After that, they also presented a unified similarity-calculating algorithm, SimFusion, where URM is combined with the iterative computing model. SimFusion measures the similarity of heterogeneous data objects by taking into account relationships from multiple sources.

In [48], Sun et al. proposed a variant of SimRank, Pathsim, for heterogeneous graphs. It is a new variant of SimRank based on Meta-Path to calculate the similarity between vertices of the same type in heterogeneous graphs. They verified the effectiveness of PathSim by calculating the top k vertices with the highest similarity for a given type of vertex. Later, [49] extended the similarity search in heterogeneous graphs and proposed the Hetesim model where as well as the similarity between two objects with the same type, the similarity between two objects with different types needs to be taken into account. Hetesim utilized the relevance path instead of the meta-path to modify the SimRank model. As a result, it is successfully used to predict and verify disease genes [152].

3.4. Discussion

In Table 6, we conduct a comprehensive classification of SimRank similarity queries, where the “td” and “tk” in parentheses after the algorithm name represent the corresponding threshold variant problem and top k variant problem, respectively and no brackets indicate the non-variant problem and the algorithm names in italics indicate parallel algorithms. From Table 6, we can conclude that: 1) there is considerable research on SimRank similarity queries over simple graphs, while the research of SimRank similarity queries over attributed graphs is still in its early days. 2) Among the three computational methods of SimRank, random walk methods are mostly adopted in single-pair and single-source similarity queries, while iterative methods are frequently utilized in all-pairs and partial-pairs similarity queries. 3) For SimRank similarity queries, most of the proposed algorithms are online, and only a few algorithms are devised based on indexes. 4) All the accuracy of SimRank similarity queries on attributed graphs is not given. 5) Only a few parallelism works for similarity query work.

It is a chance to investigate more insight into regular patterns for attribute graph similarity queries and improve the accuracy of the query. Also, more parallelism work can be designed to accelerate the queries.

4. Path-related personalized queries

In this section, we survey the studies about path-related graph queries, including reachability queries and shortest path queries, which are widely used in online mapping and navigation services. These queries are two personalized graph queries that are closely related to paths. We review the latest studies of these queries on simple graphs and attributed graphs. We also classify the approaches to reachability queries and shortest path queries into two categories: online algorithms and index-based algorithms.

4.1. Reachability queries

The reachability query is an important graph management operator which checks if there is at least one path between two given vertices. It is widely used in online social networks, biological networks, and other real-life applications. In this subsection, we summarize some latest and representative work of reachability queries over simple graphs, temporal graphs, heterogeneous graphs, and probability graphs.

4.1.1. Simple graphs

The reachability query over simple graphs can be formulated as follows.

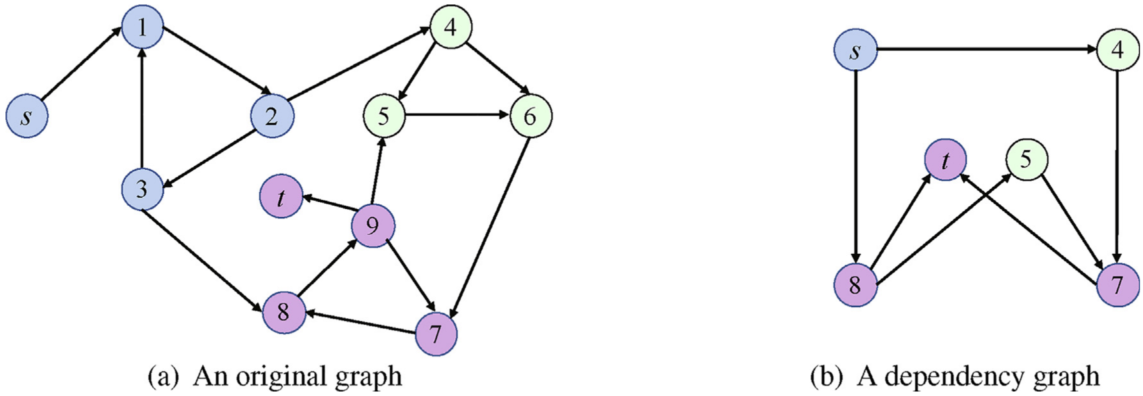
Problem 7 (*Reachability Query over Simple Graphs*). For a given directed graph $G = (V, E)$, a source vertex v_s and a target vertex v_t, decide whether there is a directed path from v_s to v_t in the graph G.

For reachability queries over simple graphs, there are both online algorithms and index-based algorithms.

Online algorithms. The breadth-first search (BFS) algorithm is a classical algorithm for reachability queries. Given a graph G, a source vertex v_s, and a target vertex v_t, BFS traverses all the reachable vertices and edges of the given graph G from v_s until v_t is found or all the other vertices in G are visited. The BFS algorithm is time-consuming, especially for large graphs. To resolve this issue, there are many follow-up studies in [156,54,157]. Then et al. [54] devised the multi-source BFS (MS-BFS) algorithm which improves query performance by running multiple concurrent BFSs for the same graph. MS-BFS can achieve efficient graph traversal by exerting its advantages of sharing common computations across concurrent BFSs, greatly reducing the number of random memory accesses, and avoiding synchronization costs.

Moreover, a distributed algorithm for reachability queries was developed by Fan et al. [53]. This method collects corresponding paths from each subgraph and constructs a dependency graph to reduce redundant computations. As shown in Fig. 13, a given graph is divided into three subgraphs, P_1, P_2, and P_3, which are induced by blue, green and purple vertices, respectively. For a query $q(s, t)$, it returns two paths $p(s, 4)$ and $p(s, 8)$ from P_1. Similarly, the paths $p(4, 7)$ and $p(5, 7)$ are returned from P_2 and the paths $p(7, t), p(8, t), p(7, 5)$ and $p(8, 5)$ are returned from P_3. Based on these graphs, a dependency graph depicted in Fig. 13(b) is constructed. After that, we can get the final result directly based on the dependency graph.

Although the dependency graph proposed in [53] helps reduce the search space and improve the query performance, it needs to construct dependency graphs for different queries, which is time-consuming. In addition, the dependency graph is only stored in a single machine, and it is apparent that the query efficiency will be seriously affected when the scale of the dependency graph is very large.



(a) An original graph

(b) A dependency graph

Fig. 13. An Example of an Original Graph and a Dependency Graph [53].

Apart from the above accurate algorithms, Sengupta et al. [61] proposed ARROW which adopts a random walk to obtain approximate results. The basic idea of ARROW is to constrain the range and length of random walk. Notice that the efficiency and accuracy of ARROW are affected by two parameters r and l, which are the number of random walks and the length of each walk, respectively. The approximate idea is also used to deal with the problem of reachability queries under regular simple paths [70].

Index-based algorithms. In [65], Gurajada et al. presented a graph-based index for distributed set reachability (DSR) queries. Instead of constructing the global dependency graph proposed in [53], DSR builds a boundary graph for the given graph and stores it in each computing node to improve the query efficiency by processing those subgraphs that contain source vertices in parallel. Besides, there exists at most one round of message exchanges performed during the query. However, DSR suffers a limitation of huge space overhead because the boundary graph needs to be stored in all the computing nodes. In addition, an extra centralized index is deployed on each computing node to get better query performance, resulting in redundant space costs.

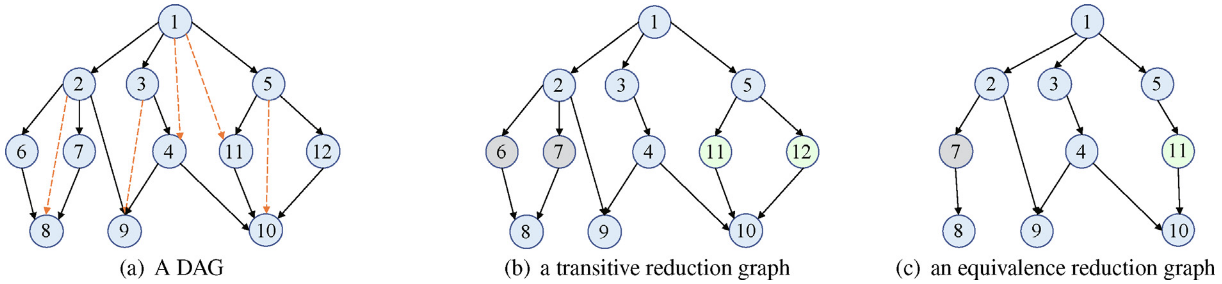
In [64], Zhou et al. constructed a transitive reduction (TR) graph for retaining long paths as much as possible, and then built an equivalence reduction (ER) graph by merging vertices with the same in-neighbors and out-neighbors. This method can efficiently reduce the scales of graphs and prune the redundant computations of paths. Three types of graphs are shown in Fig. 14. Specifically, due to the two paths from vertex 1 to vertex 4 in Fig. 14(a), the path $p(1,4)$ can be deleted since the path $p(1,3,4)$ is longer than it. In addition, vertices 6 and 7 will be merged because of $in^*_G(6) = in^*_G(7) \wedge out^*_G(6) = out^*_G(7)$. Although the ER graph is useful in reducing the search space, it causes excessive redundant calculations for irrelevant vertices, especially when the source vertex is close to the destination vertex in the original graph.

Different to the indexes mentioned above, there are also many indexes constructed on the basis of hop labels. In [55], Cohen et al. proposed the 2-hop index that builds two reachable vertices as 2-hop neighbors. Due to its huge space complexity $O(n \cdot m^{1/2})$, this index is not appropriate for large graphs. After that, the SCARAB method [59] was designed based on a backbone structure. SCARAB can achieve a reasonable trade-off between the index space overhead and query performance. Specifically, a given query $q(s,t)$ can be transformed to a new query $q(B_s, B_t)$. Here B_s and B_t are the sets of backbone vertices which can be reached from the vertices s and reach the vertex t, respectively.

Hop labels based on topological folding (TF) were proposed in [56] to accelerate the reachability query. For a given directed acyclic graph (DAG) G, the transformed topological folding graph G^* is established to construct hop labels of all vertices. It is effective to decrease the size of hop labels. Based on the transformed topological folding graph, the query procedure contains three steps. First, $G = (V_G, E_G)$ is transformed to many subgraphs $G_1, \ldots, G_{tf(G)}$ where each subgraph G_n is equipped with a transformed topological folding $G^*_n = \{V^*_n, E^*_n\}$. The subgraph G_{i+1} and the corresponding topological level G^*_{i+1} are produced based on G_i and G^*_i. Second, for each $v \in V^*, u \in V^*$ is recorded in $v's$ in-label set $L_{in}(v)$ when u can reach v. Likewise, $w \in V^*$ is recorded in $v's$ out-label set $L_{out}(v)$ when v can reach w. Third, given two vertices s and t, it returns "true" if $L_{out}(s) \cap L_{in}(t) \neq \emptyset$. This means that vertices s and t are reachable. The structure of graphs based on TF are shown in Fig. 15 where G_2 is constructed based on G^*_1, and G_3 is constructed based on G^*_2.

In [58], Zhang et al. constructed a scalable hop-based index in the distributed environment. Specifically, it constructs a fixed number of labels for each vertex and adopts effective pruning strategies to reduce the search space. However, the query efficiency is affected by the number of labels. In the worst case, its query efficiency is not better than BFS as it needs to traverse all reachable paths.

Most existing algorithms use greedy methods to construct hop labels of the vertices, which causes a high cost of materializing the transitive closure. To alleviate this problem, a hierarchical-labeling scheme [57] was developed to effectively reduce the construction cost while maintaining a good query performance. Moreover, a distributed-labeling scheme was proposed to optimize the recursive hierarchical decomposition operations in the hierarchical-labeling scheme and further narrow the size of hop labels. In [63], the independent permutation (IP) labeling scheme was presented to process reachability queries by employing randomness. After that, due to the high cost of checking vertices, a k-minwise independent permutation was employed. Moreover, to further improve the performance of IP [63], the bloom-filter labeling scheme (BFL) [62] was presented. BFL can achieve better pruning ability and improve query performance accordingly. However, both of these two approaches can only gain the approximate results of reachability queries.

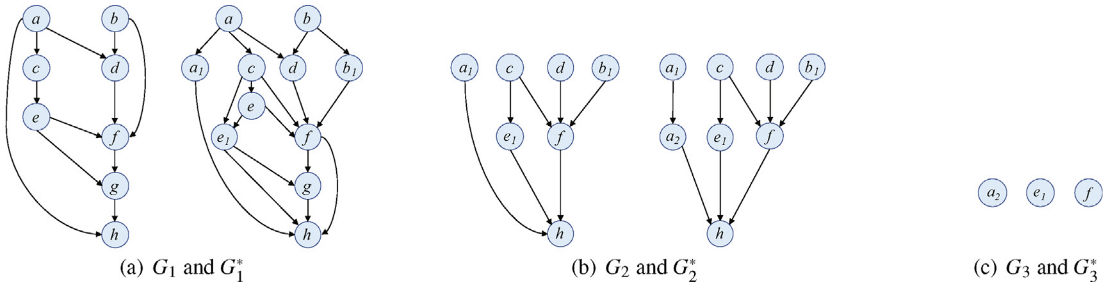


(a) A DAG

(b) a transitive reduction graph

(c) an equivalence reduction graph

Fig. 14. An Example of an Graph Construction [64].



(a) G_1 and G_1^*

(b) G_2 and G_2^*

(c) G_3 and G_3^*

Fig. 15. An Example of TF Structure [56].

4.1.2. Attributed graphs

Attributed graphs involving path-related personalized queries mainly contain temporal graphs, heterogeneous graphs, and probability graphs.

- **Temporal Graphs.** The reachability query over temporal graphs can be defined as follows.

Problem 8 (*Reachability Query over Temporal Graphs*). Given a temporal graph $G = (V, E)$ consisting of a vertex set V and an edge set E, where each edge $e = (u, v)$ is associated with a time interval to represent its active time, a source vertex v_s, a target vertex v_t, and a time interval $[t_s, t_e]$, determine whether v_s can reach v_t within the given time interval.

The latest algorithms for reachability queries over temporal graphs are all index-based. In [66], Wu et al. proposed TopChain, which is a labeling scheme based on hop labels. Specifically, a temporal graph G is first transformed into a DAG, and a set of chains is then constructed to boost query performance. An original temporal graph and its transformed graph are shown in Fig. 16. Assume that the travel time for each edge is only 1 unit, and the weight of each edge represents its start time. From the edge $e(a, b, 1)$ in the original graph, we obtain two vertices $(a, 1)$ and $(b, 2)$. This means that a can reach b within the minimal time interval [1,2].

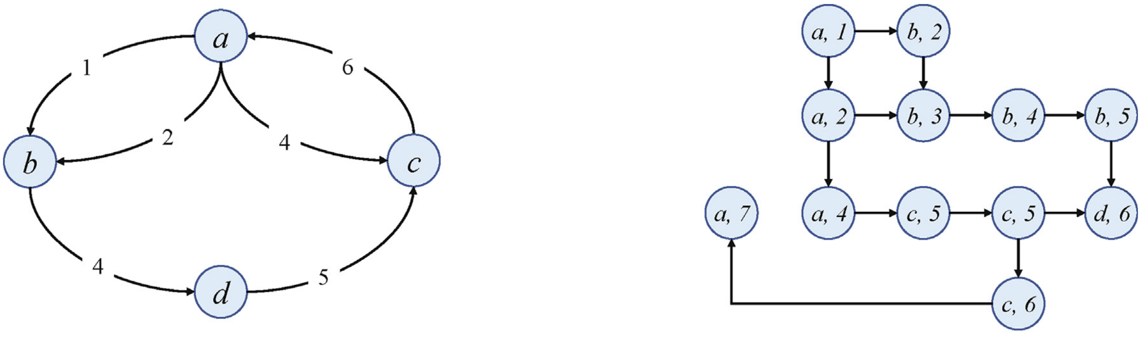
To avoid the extra overhead of graph transformation in [66], Zhang et al. [67] developed a scalable hop index TVL which is a distributed labeling scheme for temporal graphs. Furthermore, TVL adopts effective pruning strategies to accelerate the query process. Moreover, TVL is constructed based on the message propagation technique, which is efficient in reducing the time cost of index construction. However, the performance of TVL depends heavily on the number of labels. In [68], Wen et al. applied the 2-hop index to span-reachability queries on temporary graphs which aims to explore a temporal path between each two vertices in a given time interval.

- **Heterogeneous Graphs.** In [71,70], the authors investigated the reachability query over heterogeneous graphs.

Problem 9 (*Reachability Query over Heterogeneous Graphs*). Given a multi-relation direction graph $G(V, E, L)$, where V is a vertex set, E is an edge set, and L is an edge label set, a source vertex v_s, and a target vertex v_t, determine whether there is at least one path from v_s to v_t consisting of edges with labels in L.

Reachability query algorithms for heterogeneous graphs can also be divided into two types: online and index-based.

-Online algorithm. Recently, Wadhwa et al. [70] designed an online algorithm, namely ARRIVAL, to address reachability



(a) An original graph

(b) A transformed graph

Fig. 16. An Example of TopChain Structure [66].

queries over regular simple paths. A sampling method and a random walk method are adopted to improve the query performance of the BFS-based method. Meanwhile, the length of the walk is adjusted to achieve a trade-off between time overhead and the accuracy of results.

-Index-based algorithm. In [71], an index-based algorithm was developed to solve label-constrained reachability (LCR) queries over large graphs. This algorithm was presented based on a new landmark-based index that selects a small number of landmark vertices to store their reachable vertices. Specifically, for a given query $q(v_s, v_t)$, if v_s is a landmark vertex, the query can be directly answered by hop labels of v_s. Otherwise, this method explores the graph until finding another landmark vertex or the vertex v_t. In the above approach, it is difficult to specify the number of landmark vertices, which seriously affects query performance and space cost. To further improve the query efficiency, in [72], Peng et al. applied the 2-hop index to solve the LCR query. This method can efficiently reduce the redundant computation by adjusting the order of traversal paths. This also contributes to reducing the time cost of index construction.

- **Probability Graphs.** Reachability queries and their variants over probability graphs were widely studied in [76,74,77,75].

Problem 10 (*Reachability Query over Probability Graphs*). Given a probability graph $G = (V, E, P)$ consisting of a vertex set V and a directed edge set E, where P is a probability function assigning a probability of existence to each edge in E, a source vertex v_s, and a target vertex v_t, return the probability that v_s can reach v_t.

Existing approaches to the reachability query on probability graphs can also fall into two categories: online algorithms and index-based algorithms.

-Online algorithms. In [74], Li et al. proposed a recursive stratified sampling (RSS) method, to measure reliability in probability graphs. Compared with the Monte Carlo (MC) sampling method [73], it provides better efficiency in simplification of graphs and variance reduction. In [75], the lazy propagation sampling method was developed to avoid unnecessary probing of edges, which contributes to improving query performance.

-Index-based algorithms. Apart from the above online algorithm, there were two index-based algorithms exploited in [76,77]. Li et al. [76] adopted a bit-vector-based compact structure to minimize space overhead and invoked BFS to deal with reliability queries based on a compacted structure. In [77], Maniu et al. devised the ProbTree index which can dramatically improve the efficiency of reliability queries. Specifically, the ProbTree index is first built for a given original uncertain graph G. Then, the MC sampling method is adapted to process an equivalent graph G_q, which is constructed from the ProbTree index.

4.1.3. Discussion

Table 7 shows some representative approaches of reachability queries. In Table 7, **Single** represents reachability queries that aim to identify the reachability between the source vertex and the target vertex, while **Batch** represents reachability queries for multiple source and target vertices which can also be called batch reachability queries. As illustrated, there is limited research on batch reachability queries.

Both index-based and online algorithms are developed to resolve reachability queries. The online algorithms can be utilized to process graphs that are frequently updated, but it is inefficient to deal with reachability queries in large graphs. Over large graphs, there are two methods to improve the query performance of reachability queries. The first method is to design a parallel algorithm with better performance than traditional serial algorithms. Another method is to organize large graphs as indexes and design effective algorithms to narrow the search space.

4.2. Shortest path queries

The shortest path query is a fundamental problem in graph data management. It has numerous applications such as GPS navigation and route planning. Similar to the reachability query, we carry out a classified summary for the related work of

Table 7
Classification of Representative Solutions to Reachability Queries.

Graph	**Query**	**Algorithm**	
		Type	**Technique**
Simple	Single	Online	Dependency graph [53], Mutil-source BFS [54]
		Index-based	Graph reconstruction [64], Hierarchy-based [57,59,60], Hop-based [55–58], Random walk [61], Sampling and hop-based [62,63]
	Batch	Index-based	Graph reconstruction [65]
Temporal	Single	Index-based	Graph reconstruction [66], Hop-based [67,68]
Heterogeneous	Single	Online	Iteratively update path [69], Random walk [70]
		Index-based	Landmark and hop-based [71,72]
Probabilistic	Single	Online	Sampling [73–75]
		Index-based	Compact structure [76], The tree index [77]

shortest path queries over simple graphs, location-based attributed graphs, temporal graphs, heterogeneous graphs, and probability graphs.

4.2.1. Simple graphs

The shortest path query over simple graphs can be formulated as follows.

Problem 11 (*Shortest Path Query over Simple Graphs*). Given a graph $G = (V, E)$ with a vertex set V and an edge set E, where each edge $e = (u, v) \in E$ is associated with a weight (distance) $e.w$, a source vertex v_s, and a target vertex v_t, return the path between v_s and v_t with the smallest total weights (distances).

For the shortest path queries over simple graphs, both online and index-based algorithms were developed. Some representative work is introduced in the following.

Online algorithms. Dijkstra's algorithm [78] is a famous method to find the shortest path by network traversal. Based on Dijkstra's algorithm, the A$*$ algorithm [79] can obtain better query performance. It benefits from a heuristic function to compute the priority of each vertex and reduces redundant paths. Likewise, in [80], Fredman et al. further improved Dijkstra's algorithm by using a Fibonacci heap (F-heap). However, it is not suitable for these algorithms to deal with large-scale graphs since they produce too many redundant calculations on irrelevant vertices.

Index-based algorithms. The hop-based label has been used in many existing studies about the shortest path query [81,82,158]. The main idea is to precompute a set of hop labels for each vertex [158]. In [81], Akiba et al. proposed an exact method PLL for shortest path queries on large-scale networks. It executes a breadth-first search to recompute distance labels for vertices, and adopts effective pruning strategies to improve the building efficiency. In addition, this method can also provide an extremely high query efficiency. In [84], Li et al. presented PSL which supports building the PLL index in parallel. In addition, two effective pruning strategies are deployed to restrain the search space. Notice that both these two methods are more suitable for small-world graphs. In [86], Li et al. further devised CT-Index which is more scalable than PSL. Specifically, CT-Index adopts a core-decomposition to partition the data graph as the core part and the forest part. Then, it builds PSL and the tree-index for the parts of core and forest, respectively.

In [83], Takuya et al. first proposed two kinds of dynamic indexing schemes for contemporary queries and historical queries, respectively. Compared with other existing methods, these two methods can get exact query results and efficiently handle dynamic graph updates. In [85], Takanori et al. combined an online bidirectional breadth-first search and offline shortest-path trees (SPTs) to avoid full computation of prohibitively large data structures. In addition, the proposed SPTs structure can also be updated accurately in dynamic graphs.

4.2.2. Attributed graphs

The attributed graphs involving path-related personalized queries mainly contain location-based graphs, temporal graphs, heterogeneous graphs, and probability graphs.

- **Location-based Graphs.** In [94,3], the shortest path query over location-based graphs is researched, which can be formulated as follows.

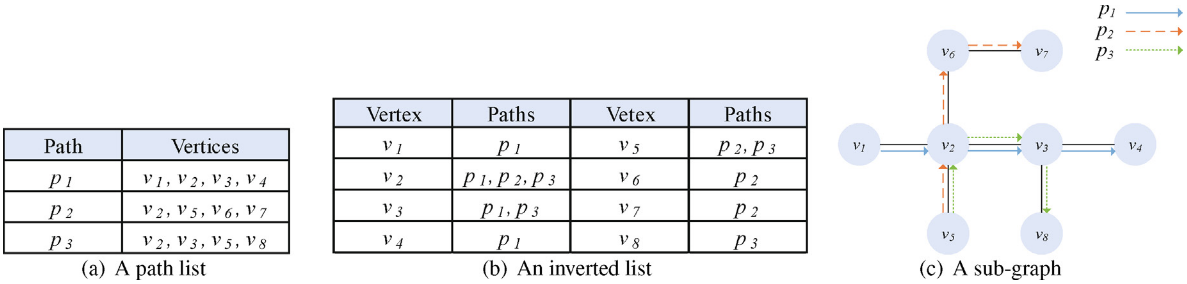
Problem 12 (*Shortest Path Query over Location-based Graphs*). Given a location-based graph $G = (V, E)$ with a vertex set V and an edge set E, where each vertex represents a road junction and each edge represents a road segment associated with a weight (length) $e.w$, a source vertex v_s, and a target vertex v_t, return the path from v_s to v_t that has the minimum sum of edge weights (lengths).

The shortest path query over a location-based attributed graph is similar to that over simple graphs. The following online and index-based algorithms were designed to process the shortest path query.

Online algorithms. The main idea of the online algorithms for the shortest path query is to utilize a cache to boost query performance. In [94], Thomsen et al. proposed a global cache algorithm for batch shortest path queries over location-based graphs. Here, the cache structure stores the path information with the most benefit. As introduced in [95], the global cache algorithm in [94] has a low cache hit ratio. To address this issue, a cache refreshing algorithm was developed to increase the cache hit ratio to a certain extent. However, due to the lack of a suitable scheduling scheme, it is still a major challenge to achieve a high cache hit ratio. Recently, a query decomposition algorithm was designed by Li et al. [3] for the purpose of further improving the cache hit ratio. In [3], a cache-based technique was adopted to avoid redundant computations. In particular, a given query set (S, T), where S is a source vertex set and T is a target vertex set, is divided into several query subsets according to an angle threshold. Then, for each query $q_i = (s_i, t_i)$, if it cannot gain final results by only visiting the paths stored in the cache, the A^* algorithm is invoked to traverse all the vertices in a given graph.

Fig. 17 shows an example of the cache structure presented in [3]. For a path $p_1 = (v_1, v_2, v_3, v_4)$, it is stored in inverted lists of related vertices v_1, v_2, v_3, and v_4 as shown in Fig. 17(b). According to the inverted list in Fig. 17(b), the sub-graph depicted in Fig. 17(c) is constructed again by combing all the paths in the path list shown in Fig. 17(a).

Index-based algorithms. In the index-based algorithms for the shortest path query, two famous indexes, G-tree and hierarchy-based indexes, have been widely used.



Path	Vertices
p_1	v_1, v_2, v_3, v_4
p_2	v_2, v_5, v_6, v_7
p_3	v_2, v_3, v_5, v_8

(a) A path list

Vertex	Paths	Vetex	Paths
v_1	p_1	v_5	p_2, p_3
v_2	p_1, p_2, p_3	v_6	p_2
v_3	p_1, p_3	v_7	p_2
v_4	p_1	v_8	p_3

(b) An inverted list

(c) A sub-graph

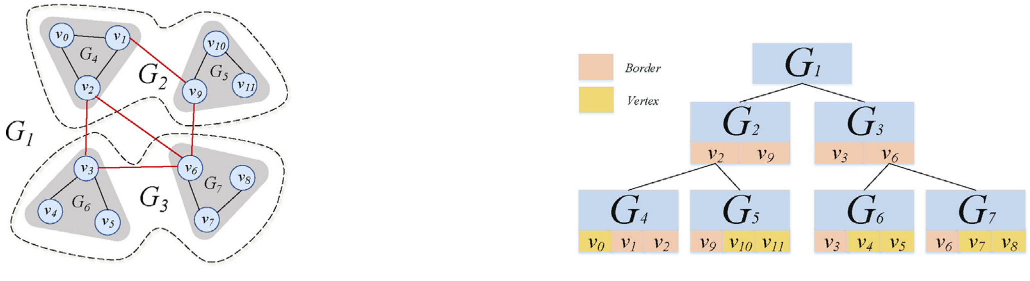
Fig. 17. An Example of the Cache Structure [3].

In [93], Zhong et al. designed a balanced search tree, namely G-tree, to organize a road network which is recursively divided into sub-networks. Here, each node of the G-tree represents a sub-network. Fig. 18 shows an example of the G-Tree. As shown, there are border vertices v_1 and v_2 in G_4. Likewise, G_2 contains two border vertices v_2 and v_9.

The G-tree is an efficient index for boosting the query performance of the shortest path query in most cases. However, it has no advantage for the shortest path query where source and target vertices are close in a road network but distant in the G-Tree. To resolve this issue, Li et al. [88] presented a novel index, namely G∗-tree, which can avoid redundant traversal costs by adding shortcuts between leaf nodes. In the G∗-tree, shortcuts are inserted between some leaf nodes. As shown in Fig. 19, the basic structure of the G∗-tree is similar to the G-tree [93]. From Fig. 19(b), there are two shortcuts in the G∗-tree, one is between G_4 and G_6, and the another is between G_5 and G_7. For a given query $q(v_2, v_3)$, it can be accelerated by the shortcut between G_4 and G_6.

Geisberger et al. [87] first proposed a hierarchy-based index, called contraction hierarchy (CH), to accelerate the shortest path query. In CH, each vertex v is first assigned a specific order $r(v)$ as the vertex hierarchy by some predefined criteria. Then, for each vertex v, it visits all the neighbors $N(v)$ of v and adds edges between neighbor pairs. In [4], Zhu et al. developed an arterial hierarchy (AH) index that integrates spatial information into CH. Notice that CH and AH cannot be utilized to effectively process the shortest path query when a source vertex s is far from a target vertex t.

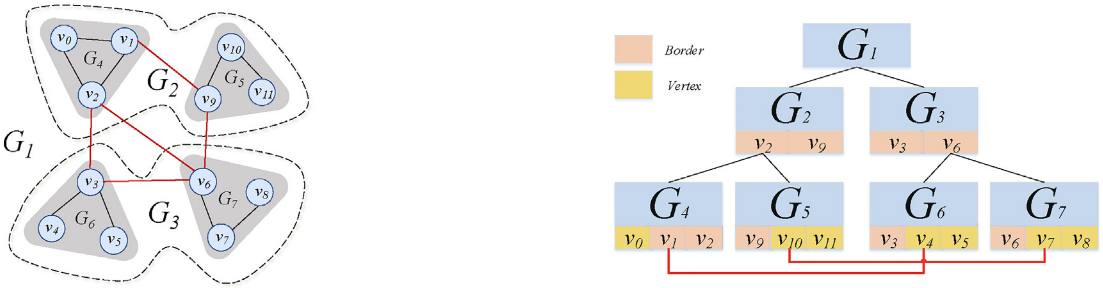
In [90], Ouyang et al. proposed a shortcut-centric paradigm focusing on exploring a small number of *shortcuts* to maintain the index of a dynamic road network. Given a road network $G = (V, E, \phi)$, before constructing the shortcut index G', the vertices are ranked by their importance, and we obtain a total vertex order γ. For each neighbor pair (u, w) of a given vertex v, a new edge with a weight of $\phi(u, v) + \phi(v, w)$ is added if it satisfies (1) there is not an edge between u and v and (2) $\gamma(u) > \gamma(v)$



(a) A partitioning graph

(b) A G-Tree

Fig. 18. An Example of the G-Tree [93].



(a) A partitioning graph

(b) A G^*-Tree

Fig. 19. An Example of G^*-Tree [88].

and $\gamma(w) > \gamma(v)$. If there is an edge between vertices u and v, its weight is set to $min\{\phi(u,w), \phi(u,v) + \phi(v,w)\}$. By this way, G' is transformed to a shortcut supporting graph (SS-Graph) G^* which is a directed graph that contains two types of vertices, including shortcut type vertices and supporting relation type vertices. Fig. 20 depicts an example of SS-Graph. Here, a shortcut between v_1 and v_2 needs to be constructed since the topological order of v_0 is the smallest among the three vertices. Although avoiding extra computations when updating the weight values of edges in dynamic graphs on SS-Graphs, the topological order of vertices will seriously affect the number of shortcuts and the size of SS-Graph, which may cause a huge space overhead. And it will also affect the performance of the algorithm.

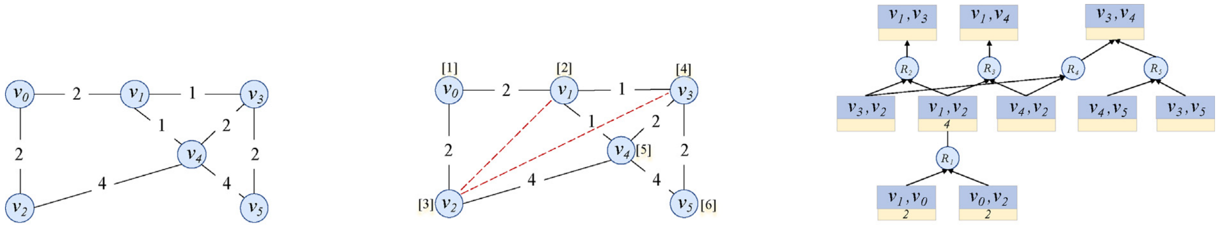
Besides the above hierarchy-based indexes, Dian et al. [89] designed a hierarchical 2-hop index (H2H) that aims to achieve a trade-off between space overhead and query performance. In H2H, each vertex in a given graph is assigned a specific order according to a decomposition tree constructed by an improved tree decomposition algorithm [159]. For each vertex in the graph, it stores the shortest paths for all the ancestor nodes as well as the positions of the vertices in the same tree node. Although H2H is conducive to reducing the size of hop labels, the construction of a decomposition tree requires extra time and space overhead. In addition, H2H cannot be used to deal with dynamic graphs since it is difficult to update the decomposition tree.

Fig. 21(c) shows an example of the H2H index based on the decomposition tree in Fig. 21(b). For the vertex v_7, the corresponding node in the H2H index stores the shortest distances between itself and all its ancestor vertices v_1, v_2, v_3, v_4, v_5, and v_7. In Fig. 21(c), a node of the decomposition tree stores a vertex and its neighbors. Because vertex v_7 and its neighbors v_3 and v_1 are at the 6th, 4th and 5th levels of the decomposition tree, the node of v_7 also contains this position information.

In [92], Wei et al. proposed the UE index for shortest path queries on dynamic road networks. In UE, the shortcut between each two vertices in a valley path is constructed based on a specific ranking strategy. Here, for any two vertices u and v, the path $p(u,v)$ is a valley path if $r(w) < min\{r(u), r(v)\}$ where w is a middle vertex. In addition, an improved bi-directional query method is devised to enhance the query efficiency. Given a query vertex pair (u,v), it expands upward edges $e(u,w)$ to $r(w) > r(u)$ from the source vertex u. Similarly, downward edges $e(w,v)$ where $r(w) > r(v)$ are expanded from the target vertex v. The query process can be early terminated when u and v both reach the same middle vertex.

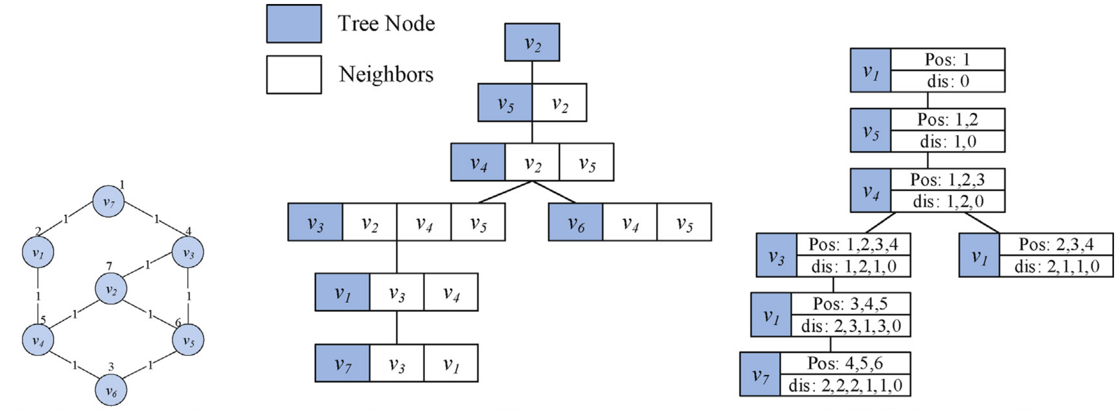
- **Temporal Graphs.** In [98], the time-dependent shortest path query was researched.

Problem 13 (*Time-dependent Shortest Path Query*). Given a time-dependent graph (road network) $G = (V, E, W)$, where V is a vertex set and E is a directed edge set, each edge $e \in E$ is associated with a time-dependent weight function w_e, a source vertex v_s with a departure time t, and a target vertex v_t, retrieve the exact fastest travel time from v_s to v_t and its associated path.



(a) An original graph (b) A shortcut graph (c) A shortcuts supporting graph

Fig. 20. An Example of a Graph Transformation Process [90].



(a) An original graph (b) A decomposition tree (c) A H2H index construction

Fig. 21. An Example of the H2H Index [89].

Recently, two index-based algorithms have been developed for the time-dependent shortest path query. Wang et al. [98] proposed a new height-balanced tree-structured index, namely TD-G-tree, for fast processing shortest path queries over temporal graphs. A hierarchical graph partitioning technique is utilized to divide a given temporal graph (road network) into hierarchical partitions. In this way, a balanced tree is built, where the tree nodes correspond to different partitions and each parent–child pair represents a partition and its sub-partitions. Then, time-dependent shortest paths are computed for border vertices linking different partitions. Based on TD-G-tree, an effective algorithm is developed to compute the optimal solution through dynamic programming and chronological divide-and-conquer techniques. Li et al. [97] proposed the time-dependent hop labeling (THop) with partitioning graphs into sub-graphs. An online approximate and bottom-up compression techniques were introduced to further decrease the space overhead of this index, save construction time, and gain better query efficiency. It combines a linear piecewise function and a 2-hop-based labeling scheme to solve the shortest path query on the time-dependent graph.

In [100], Wu et al. investigated the problem of classic shortest path queries in temporal graphs. Then, the authors proposed the one-pass algorithm and a graph transformation technique, respectively. In [99], Konstantinos et al. constructed an appropriate reachability index TimeReach. For each node u, it maintains a list of elements in time-dependent strongly connected components (SCCs). It is conducive to efficiently get the query results when the two nodes are in the same SCC. For the query that the two nodes belong to different SCCs, it is split into a series of sub-queries. The results of corresponding sub-queries are combined to produce the answer for the query through an AND (OR) operator for conjunctive (disjunctive) queries.

- **Heterogeneous Graphs.** In [101], Zhang et al. focused on the shortest path query over heterogeneous graphs.

Problem 14 (*Correlation Constrained Shortest Path Query*). Given a multi-relation graph $G(V, E, \Re, F)$, where V is a vertex set, E is an edge set, $\Re$ is a set of allowable relations in G, and $F : E \rightarrow R$ for $R \subseteq \Re$ and $R \neq \varnothing$ is a mapping function assigning each edge $e \in E$ a subset of $\Re$, two vertices $v_s, v_t \in V$, and a correlation constraint C, compute the shortest path p_{st} from v_s to v_t such that each vertex in p_{st} satisfies the constraint C.

Zhang et al. [101] investigated a general approach to the shortest path query over heterogeneous graphs. The correlation constraints including necessity and denial were taken into account. Additionally, the authors presented a vertex encoding scheme, called hybrid relation encoding, which is a lightweight index that can efficiently reduce the search space. Based on the hybrid relation encoding, the BFS algorithm and an efficient heuristic algorithm are extended to effectively process the shortest path query under relevant constraints.

- **Probability Graphs.** In [102], Chen et al. studied the threshold-based shortest path query over probability graphs.

Problem 15 (*Threshold-based Shortest Path Query over Probability Graphs*). For a given probability graph $G = (V, E)$ consisting of a vertex set V and an edge set E, where each edge is associated with a probability of existence, two vertices $v_s, v_t \in V$, and a probabilistic threshold τ, compute paths between two vertices v_s and v_t with shortest path probabilities larger than the threshold τ. The shortest path probability of a path P is calculated as

$$SP(P) = \sum Pr(G_i),$$

where G_i represents a possible world instance of the given graph G and P is the shortest path in G_i.

Table 8
Classification of Representative Solutions to Shortest Path Queries.

Graph	Query	Algorithm	
		Type	Technique
Simple	Single	Online	Greedy [78–80]
		Index-based	Hop-based and Tree-based [86], Pruned hop-based [81–84], Shortest-path trees [85]
Location	Single	Index-based	G-Tree [93], Hierarchical [87,4], G∗-Tree [88], Hierarchical and hop [89], Shortcuts and hierarchical [90], Shortcuts [91,92]
	Batch	Index-based	Cache refreshing [95], Global cache [94], Graph reconstruction [96], Task decomposition and local cache [3]
Temporal	Single	Index-based	Condense graph[99], G-Tree Hop [97,98], Graph transform [100]
Heterogeneous	Single	Index-based	Hierarchy [101]
Probabilistic	Single	Online	Filter framework [103], Sampling [102]

In most uncertain graph models, edges are assumed to be independent. However, this assumption is invalid in numerous applications. Inspired by this, Cheng et al. [102] proposed a new uncertain graph model that can effectively show some hidden relationships among edges sharing the same vertices. They devised several sampling approaches to compute approximate results of shortest path probabilities on the correlated uncertain graph.

4.2.3. Discussion

As shown in Table 8, we make a simple classification of some representative research on shortest path queries. In Table 8, **Single** represents the shortest path query from a source vertex to a target vertex, and **Batch** represents the shortest path query with many source vertices and target vertices. Similar to reachability queries, there are limited studies on batch shortest path queries.

For shortest path queries, there are online algorithms and index-based algorithms. Online algorithms can process the shortest path query on frequently updated graphs. However, they are not appropriate to process large graphs. To effectively process the shortest path query over large-scale graphs, the index-based algorithms are better choices than online algorithms.

5. Subgraph-related graph queries

In this section, we survey the studies about community searches (CS) [132], which belong to subgraph-related graph queries. Specifically, for a graph *G*, CS retrieves connected cohesive subgraphs that contain given query vertices. CS is significant in real-life applications including advertising and viral marketing, content recommendation, and team building.

There are numerous common cohesive subgraph models, such as *k*-core [160], *k*-ECC [161], *k*-truss[162,163], and *k*-clique [164]. In these cohesive subgraph models, the more cohesive the model is, the less inefficient the query is [6]. Considering the cohesiveness, query efficiency, and popularity, we focus on community searches based on *k*-core and *k*-truss in this paper. And more cohesive subgraph definitions can be found in [6]. In the following, we divide CS problems into two categories: CS over graphs without attributes which are called simple graphs; CS over attributed graphs, which include temporal graphs, location-based graphs, keyword-based graphs, weight-based graphs, and heterogeneous graphs.

5.1. Preliminary

In this subsection, we introduce the basic concepts of *k*-core and *k*-truss.

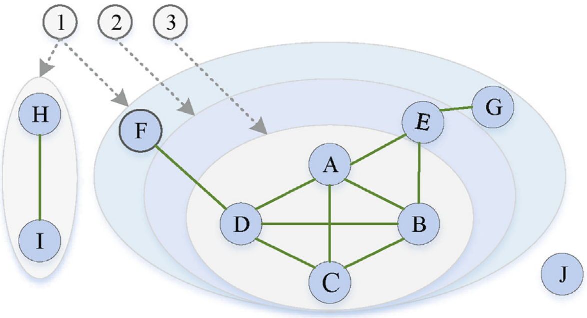
Definition 7 (*k-Core*). Given a graph G and an integer $k(k \geqslant 0)$, the k-core of G is defined as the largest subgraph of G, where the degree of each vertex is not less than k.

Definition 8 (*Core Number*). Given a vertex $v \in V$, the core number of v is the largest value of k such that the corresponding k-core includes v.

Fig. 22 shows *k*-cores and the core number of each vertex in the graph. As illustrated, the 3-core of the given graph consists of four vertices, including A, B, C, and D, and the 2-core contains five vertices that are A, B, C, D, and E. It is obvious that $3\text{-}core \subseteq 2\text{-}core \subseteq 1\text{-}core$.

Different from *k*-core, *k*-truss is a cohesive subgraph based on triangles.

Definition 9 (*Support*). Given a graph $G(V,E)$, the support of an edge $e \in E$, denoted by $sup(e,G)$, is the number of triangles containing e.



Core number	Vertices
0	J
1	F, G, H, I
2	E
3	A, B, C, D

Fig. 22. An Example of a *k*-core.

Definition 10 (*Subgraph Trussness*). Given a graph $G(V, E)$, the subgraph trussness of $G' \in G$, denoted by $\tau(G')$, is the largest $k(k \leq 2)$ for all edges in G' that satisfies $sup(e, G) \leqslant k - 2$.

Definition 11 (*Edge Trussness*). Given a graph $G(V, E)$, the edge trussness of $e \in E$, denoted by $\tau(e)$, is the maximum subgraph trussness of G', which must contain edge e.

Definition 12 (*k-truss*). The k-truss is the maximal subgraph of G such that it contains at least three vertices and every edge is incident to at least k triangles, i.e., the support of each edge in k-truss is no less than $k - 2$.

Fig. 23 shows a 4-truss $\{A, B, C, D\}$ and a 3-truss $\{A, B, C, D, E\}$ of the graph.

5.2. *k*-core-based community search

5.2.1. Simple graphs

Simple graphs include undirected graphs and directed graphs. We survey the research of k-core-based community searches over undirected and directed graphs.

- **Undirected Graphs.** A graph $G(V, E)$ is an undirected graph if all edges within G are undirected. On undirected graphs, we focus on three CS-related problems that are size-unbounded CS [104], size-bounded CS [104], and the best k-core [165]. In [104], Sozio et al. investigated the size-unbounded CS problem over undirected graphs for the first time.

Problem 16 (*Size-unbounded CS [104]*). Given an undirected simple graph $G(V, E)$, a set of query vertices $Q \subset V$, and a cohesive constraint, find a connected subgraph $H(V_H, E_H)$ of G, such that $Q \subseteq V_H$, and there is no other subgraph H' of G meeting the cohesive constraint and containing H.

To solve the size-unbounded CS problem, there are both online and index-based algorithms.

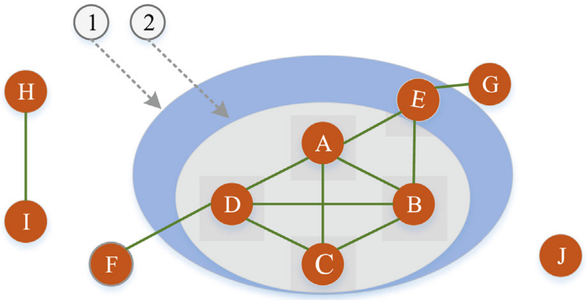
There are two online algorithms, Global [104] and Local [105], devised for the size-unbounded CS. Sozio et al. [104] presented a greedy algorithm, called Global, to search size-unbounded communities. The algorithm follows the framework in [166] to iteratively compute cohesive subgraphs [167] by removing vertices that do not satisfy a cohesive constraint. In [105], Cui et al. designed the Local algorithm which extends its neighbors incrementally to a community from the query vertex until the current community is the best. Although Global and Local have the same time complexity, Local always achieves higher efficiency in practice. This is because it dispenses with accessing the entire graph.

In [106], Barbieri et al. devised the ShellStruct index that organizes all k-shells of a given graph G into a tree. Specifically, the i^{th} layer of the tree contains the i^{th} shell for $1 \leqslant i \leqslant k$. The space cost of ShellStruct is $O(n)$. The connected component tree can also be stored in the suitable lowest-common-ancestor (LCA) data structure, which was introduced in [168].

As pointed out in [104], the CS problem of size-unbounded faces a limitation that the returned communities may be very large. Consider that in many applications (including organizing a meeting), the number of participants cannot exceed the predetermined size. As a result, communities with a bounded size are desirable, and the size-bounded CS problem was presented to compute communities with at most k vertices [104]. Some heuristic algorithms were proposed in [104] to improve its query efficiency.

In [169], Li et al. researched the minimum k-core search problem whose goal is to compute the minimum k-core including given query vertices.**Definition 13** (*Minimum k-core*). Given a graph $G = (V, E)$ and a query set Q, the minimum k-core is the k-core of G such that it contains all the vertices within Q and it has the minimum size.

The minimum k-core search problem was proven to be NP-hard. The heuristic algorithms for this problem are efficient but they face two limitations, i.e., designing on the basis of scoring functions and having no guarantee of the query result.



Truss number	Edges
4	(A, B), (A, C), (A, D) (B, C), (B, D), (C, D)
3	F, G, H, I
2	E

Fig. 23. An Example of k-truss.

Inspired by this, Li et al. [169] provided new lower and upper bounds of the optimal result and devised a progressive search algorithm (PSA) that can achieve a good trade-off between the quality of the final result and query performance. Three lower bounds were presented to effectively obtain the lower bound for the optimal result. In addition, an onion layer based upper bound algorithm, namely L-Greedy, was developed based on the idea of expanding the partial subgraphs by adding the promising vertices incrementally.

The common k-core-based community search, such as the minimum k-core search problem [169], requires to input k manually. Obviously, without a guideline, it is difficult for users to specify an appropriate value of k. Inspired by this, Chu et al. [165] introduced the concept of the best k-core.**Definition 14** (*Best k-core [165]*). Given a graph $G = (V, E)$, the best k-core of G is defined as the k-core with the highest scores where $0 \leqslant k \leqslant k_{max}$ and k_{max} is the maximum core number of vertexes in the graph G.

Chu et al. [165] studied the problem of computing the highest score of k-core with two community scoring metrics, including modularity and average degree, respectively. A direct method is to calculate the scores of all the possible k-cores and pick out the k-core with the highest score. However, this method is inefficient for large-scale graphs. To alleviate this issue, a light-weight vertex ranking method and a core forest were designed to incrementally calculate the scores. Specifically, the authors proposed a vertex ranking strategy where the neighbor sets of vertices are arranged in ascending order. Then, a core forest is adopted to organize all the k-cores for $0 \leqslant k \leqslant k_{max}$ as trees; each tree represents a connected component in the graph. Based on it, the proposed algorithm computes the scores of k-cores incrementally with k reducing from k_{max} to 0.

- **Directed Graphs.** For a direct graph $G(V, E)$ with a vertex set V and an edge set E, let $d_{in}(v)$ and $d_{out}(v)$ are the in-degree and out-degree of a vertex $v \in V$. Moreover, for a subgraph $G' \subseteq G$, its in-degree $d_{in}(G')$ and out-degree $d_{out}(G')$ are denoted as $\min d_{in}(v)$ and $\min d_{out}(v)$ for $v \in G'$, respectively.

Problem 17 (*Community Search on Directed Graph (CSD)*). Given a directed graph $G(V, E)$, a query vertex q, and two nonnegative integers k and l, obtain a connected subgraph $G_q \subseteq G$ satisfying G_q contains the vertex $q, \forall v \in G_q, d_{in}(G_q) \geqslant k$ and $d_{out}(G_q) \geqslant l$, i.e., G_q is a D-core.

A straightforward method for CSD is to iteratively delete unqualified vertices that do not satisfy the D-core constraint until all remaining vertices form a D-core. Similar to the Global algorithm in [104], the method requires $O(m + n)$ time, and it is inefficient when processing large graphs. To further improve query performance, Fang et al. [107] developed an index where a two-dimensional table is utilized to organize all the D-cores. Furthermore, the space cost of the index is reduced dramatically based on the nested property of D-cores.

5.3. Attributed graphs

5.3.1. Temporal graphs

In [116], Li et al. first investigated the persistent community search problem on temporal graphs, which can be formulated based on the following concepts.

Definition 15 (*Maximal* (θ, k)*-Persistent-Core Interval*). For a given temporal graph $G = (V, E)$ and two nonnegative parameters θ and k, a time interval $[t_s, t_e]$ with $t_e - t_s \geqslant \theta$ is a maximal (θ, k)-persistent-core interval of G if it holds that for each $t \in [t_s, t_e - \theta]$, the projected graph of G in $[t, t + \theta]$ is also a connected k-core subgraph and there is no other time interval containing $[t, t + \theta]$ and satisfying the above constraints.

Definition 16 (*Core Persistence*). Let $T = \{[t_{s_1}, t_{e_1}], \ldots, [t_{s_r}, t_{e_r}]\}$ be the interval set corresponding to all the intervals of the maximal (θ, k)-persistent-core of G. $F(\theta, k, G)$ denotes the core persistence of G with parameters θ and k. Specifically, core persistence is the total length of all the maximal (θ, k)-persistent core intervals of G minus $(r - 1)\theta$ if and only if T is not empty.

Problem 18 (*Persistent Community Search (PCS) Problem*). For a given temporal graph G, three nonnegative parameters θ, τ, and k, return an induced subgraph G' of G where the core persistence of G' is not less than τ, and G' is maximal.

In [116], the PCS problem was proven to be NP-hard. A temporal graph reduction approach was proposed to refine the input graph. This approach adopts the meta-interval decomposition technique to decompose the entire time interval into a set of meta-intervals, and then uses the meta-intervals to calculate and hold the degree of persistence of the vertices. Based on the refined temporal graph, a branch and bound algorithm was developed by integrating some pruning rules.

5.3.2. Location-based graphs

There are many CS problems over geo-social networks, such as geo-social group queries with a minimum acquaintance constraint [115], spatial-aware community searches [113], radius-bounded k-core searches [38], and skyline cohesive group queries [170]. All these CS problems satisfy the k-core constraint. However, they meet different spatial constraints. In the following, we introduce four CS queries and their related work.

- **Spatial-aware Community Search.** Two famous SACs are the minimum covering circle (MCC) search and the spatial-aware community (SAC) search.

Problem 19 (*Minimum Covering Circle (MCC) Search*). Given a vertex set S where each vertex is associated with a location, calculate the spatial circle that contains all the vertices in the given set S and is with the smallest radius.

Problem 20 (*Spatial-aware Community (SAC) Search*). For a given geo-social network $G(V,E)$, a query vertex q, and an integer k, return a connected induced subgraph G_q of G such that it contains the query vertex q, the degree of each vertex within G_q is not less than k, and MCC of the vertex set in G_q has the smallest radius.

In [171], the authors pointed out that three points on the boundary can determine a spatial circle. Accordingly, a basic exact algorithm of the SAC search problem first computes a community and then finds a subset of the community that meets the cohesive and spatial constraints. This algorithm requires $O(m \cdot n^3)$ time. Obviously, for large-scale graphs, it is computationally expensive. To address this problem, Fang et al. [113] proposed two approximation algorithms, namely AppInc and AppAcc. AppInc takes the smallest circle centered on q as a feasible solution. Its approximation ratio is 2. Another approximate algorithm, AppAcc, can achieve a ratio of $1 + \epsilon_A$ where $0 < \epsilon_A < 1$. Moreover, this algorithm can identify and output final results progressively, as introduced in [114].

- **Radius-bounded k-core Search.** In [38], the radius-bounded k-core search (RB-k-core search) was identified for the first time.

Problem 21 (*Radius-Bounded (RB) k-Core Search [38]*). For a given geo-social network $G(V,E)$, a positive integer k, a radius r, and a query vertex $q \in V$, find all the connected subgraphs G_q of G containing q such that $\forall v \in G_q, deg(v) \geqslant k$, the radius of MCC of G_q is less than r and there is no other subgraph G'_q containing G_q.

For the RB k-core search problem, Wang et al. [38] designed three algorithms: TriV, BinV, and RotC. TriV contains three steps: generating all the candidate circles containing q; checking the given radius bound; and computing the maximum k-core in each candidate circle. The time complexity of TriV is $O(mn^3)$. Then, the authors developed a binary-vertex-based algorithm called BinV to reduce candidate circles. For each pair of vertices in G, BinV only generates a circle passing through a pair of vertices with a radius of r. The RotC algorithm shares the computation of adjacent circles to improve query efficiency.

- **Geo-social group queries.** In [115], Zhu et al. formulated the geo-social group query with a minimum acquaintance constraint.

Problem 22 (*Geo-social Group Queries (GSGQ) with a Minimum Acquaintance Constraint*). For a given geo-social network $G(V,E)$ with a positive integer k, a query vertex $q \in V$, and a spatial constraint Λ, return a connected subgraph $G_q \subseteq G$ containing q such that $\forall v \in G_q, deg(v) \geqslant k, G_q$ meets the constraint Λ, and there is no other subgraph G'_q satisfying the same constraints with G_q and $G_q \subset G'_q$.

Zhu et al. [115] introduced three different spatial constraints: (1) a spatial rectangle containing G_q; (2) a circle with a center vertex q and its radius is less than the k^{th} nearest distance between q and vertices in G_q; and (3) a circle satisfying (2) and G_q contains $k+1$ vertices.

To gain good query performance, the Social Aware R-tree (SAR tree) index was proposed in [115]. In particular, the index integrates the spatial locations and social relationships between different vertices. To further reduce I/O cost, a SaR∗-tree was developed. Based on these two indexes, effective algorithms [115] were presented to solve GSGQs under different spatial constraints.

- **Skyline Cohesive Group Queries.** Existing work on cohesive group queries over geo-social networks faces two limitations, i.e., the lack of flexibility and the impracticality of spatial distance. Recently, Li et al. [170] investigated the skyline cohesive group query in large road-social networks.

Definition 17 (*Road-social Group*). Given a road-social network $G = (G_r, G_s)$ consisting of a road network G_r and a social network G_s, a query vertex q, a meeting point set P, a distance threshold r, two non-negative parameters k and l, a subgraph C of G_s is a road-social group if it satisfies the following two properties: (1) C is a (k,l)-core (k-core of size l) containing q; and (2) there is at least a meeting point $p \in P$ such that the upper bound of the distances between p and each vertex $v \in C$ is not longer than r.

Problem 23 (*Skyline Cohesive Group Query*). Given a road-social network $G = (G_r, G_s)$, return road-social skyline groups that are not dominated by any other road-social group. Here, for two given road-social groups C_1 and C_2, C_1 dominates C_2 if C_1 is better than C_2 in terms of the social and spatial cohesiveness, or C_1 and C_2 have the same core number, however, C_1 is better than C_2 in terms of spatial cohesiveness.

To address the skyline cohesive group query, Li et al. [170] first devised efficient pruning strategies and proposed an exact algorithm. Furthermore, a novel *cd-tree* was designed to organize the social and the distance information. Based on the *cd-tree*, greedy algorithms were developed to gain better query performance.

5.3.3. Keyword-based graphs

The community search on keyword-based graphs has been extensively studied [109–112]. In the following, we introduce attributed community queries on keyword-based graphs.

Problem 24 (*Attributed Community Query (ACQ)*). Given a keyword-based graph $G(V,E)$, a positive integer k, a query vertex $q \in V$, and a keyword set $S \subseteq W(q)$, compute an attributed community (AC) set of q. AC holds the following properties: (1) it is a maximal connected induced subgraph of G containing the query vertex q; (2) the degree of each vertex is not less than k; and (3) all the vertices in AC share the largest set of keywords in S.

A simple approach to ACQ was proposed by Fang et al. [111]. It first enumerates all nonempty subsets of S and checks whether each subset is connected and contains q. The subgraphs with the most shared keywords are returned as the final results. The method is time-consuming and impractical for large-scale graphs. In addition, a hierarchical structure index, namely CL-tree, was proposed to organize vertices associated with different keyword sets. It builds an inverted list for each vertex p. For each keyword e of the vertex p, the ID list of vertices that contains e is computed and maintained. To construct the CL-tree, it requires $O(m\alpha(n))$ time and $O(ln)$ space overhead, where l is the average size of $W(v)$ over the vertex set V. Fang et al. [110] also devised a maintenance algorithm for the CL-tree. By using the CL-tree to organize the graph, final results can be obtained by identifying the k-core containing q with the largest keyword set.

Fig. 24 shows an example of the CL-tree. In node r_2, the inverted list of keywords z contains E. For the node r_3, the inverted list of keywords y includes A, C, and D.

5.3.4. Weighted-based graphs

At present, there are two types of community searches on weighted graphs, namely single-dimensional influential CS and multi-dimensional influential CS. In [6], the authors reviewed the single-dimensional influential CS. In this paper, we focus on the multi-dimensional influential CS.

If each vertex corresponds to a set of multi-dimensional numerical vectors, the undirected graph is a multi-valued graph. Li et al. [121] first defined the skyline community in multi-valued graphs.

Definition 18 (*Domination*). Let G_1 and G_2 be two induced subgraphs of G with influence values $f(G_1)$ and $f(G_2)$, respectively. If $f(G_1) \geqslant f(G_2)$ for all dimensions and $f_i(G_1) > f_i(G_2)$ for a dimension i, then G_1 dominates G_2, denoted as $G_1 \prec G_2$.

Problem 25 (*Skyline Community Search*). Given a multi-valued graph G and an integer k, find all the skyline communities G', which are induced subgraphs, such that they are connected k-cores and are not dominated by any other connected k-core, or connected k-cores containing G' whose influences are equal to $f(G')$.

Li et al. [121] focused community searches over the graphs where each vertex is associated with two-dimensional attributes. They proposed the SkylineComm2D algorithm to process the skyline community search problem. It calculates the maximum value of the f_1 dimension to obtain the skyline community with influence (f_1, f_2). Then, it refines constraints to compute the maximal f_2. After obtaining f_2, it executes the above process until there is no k-core meeting the given constraints. The time complexity of SkylineComm2D is $O(s(m+n))$ and the space complexity of SkylineComm2D is $O(m+n+s)$ where m and n are the number of edges and vertices, respectively, and s is the number of 2D skyline communities.

In view of the higher-dimensional situation, a space-partition algorithm was proposed, which divides the irregular two-dimensional space into several overlapping regular two-dimensional subspaces. The approach was then extended to a higher-dimensional case. The time and space complexities of the 3D skyline community search algorithm are $O(s^2(m+n))$ and $O(m+n+s)$, respectively.

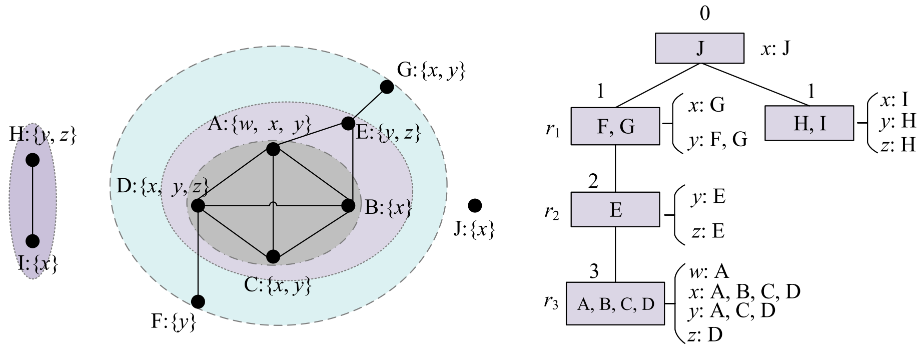


Fig. 24. An Example of the CL-tree [111].

5.3.5. Heterogeneous graphs

In [136], Fang et al. investigated the problem of community search on large heterogeneous graphs. For a given heterogeneous graph and a query vertex q, it returns a community containing q such that all the vertices in this community are of the same type as q as well as close relationships. The community search problem on heterogeneous graphs was presented based on the following concepts.

Definition 19 (*HIN (heterogeneous graph) Schema*). Given an HIN $G = (V, E)$, where V is a vertex set and E is an edge set, mappings $\psi : V \rightarrow \mathscr{A}$ and $\phi : E \rightarrow \mathscr{R}$ are from V to a vertex type set A, and from E to an edge type set R, respectively. The schema of G denoted by T_G is a directed graph over $\mathscr{A}$ and $\mathscr{R}$, i.e., $T_G = (\mathscr{A}, \mathscr{R})$.

Definition 20 (*Meta Path*). For an HIN schema $T_G = (\mathscr{A}, \mathscr{R})$, a meta-path P of length l is defined as $A_1 \xrightarrow{\mathscr{R}_1} \mathscr{A}_2 \xrightarrow{\mathscr{R}_2} \ldots \xrightarrow{\mathscr{R}_l} \mathscr{A}_{l+1}$, where $\mathscr{A}_i \in \mathscr{A}$ and $\mathscr{R}_i \in \mathscr{R}$ for $1 \leqslant i \leqslant l$.

Let P be a meta-path linking two vertices of the target type and the P-neighbors for a given vertex v are all vertices which can form the P meta-path with v. Given a vertex v and a vertex set S where each vertex is of the target type, Fang et al. [136] defined basic-degree (b-degree) $\alpha(v, S)$ as the number of P neighbors of v within the set S. Based on b-degree, they formulated three (k, P)-core models, that is, basic (k, P)-core, edge-disjoint (k, P)-core, and vertex-disjoint (k, P)-core.

The community search over HINs can be formulated as follows.

Problem 26 (*Community Search over HINs*). Given an HIN G, a query vertex q, a symmetric meta-path P, an integer $k (k > 0)$, and a specific (k, P)-core model, return the (k, P)-core containing q.

Fang et al. [136] designed a basic algorithm to calculate basic (k, P)-cores. This algorithm constructs an induced homogeneous graph G_P, and then returns the connected k-core containing q from G_P. To speed up the search procedure, the authors further developed a batch search strategy and two labeling strategies. For edge-disjoint (k, P)-cores, two efficient query algorithms were devised by peeling vertices progressively and pruning vertices in a batch manner. These approaches can also be adjusted to compute vertex-disjoint (k, P)-cores. Finally, to further improve query efficiency, meaningful meta-path cores are precomputed and organized as a compact index, namely CoreIndex.

5.4. k-truss-based community search

In this section, we introduce the related work about k-truss-based community search over simple and attributed graphs, respectively.

5.4.1. Simple graphs

In this subsection, triangle-connected truss community and closest truss community searches over simple graphs are reviewed, respectively.

Triangle-connected Truss Community. For an undirected graph $G(V, E)$, Huang et al. [122] proposed a problem of triangle-connected k-truss community search with the goal of computing the communities containing a specified query vertex. The authors also proposed the notion of triangle connectivity in [122]. Two triangles are adjacent if and only if they have a common edge. For two given edges $e_1, e_2 \in E$, they are triangle-connected if they belong to the same triangle, or can be reached by a series of adjacent triangles.

Based on the triangle connectivity, the triangle-connected truss community (TTC) search is defined as follows.

Problem 27 (*TTC Search*). Given an undirected graph $G(V, E)$, a query vertex $q \in V$, and an integer $k \geqslant 2$, retrieve subgraphs $H \subseteq G$ which satisfy the following properties:

1. H contains the query vertex q such that $\forall e \in E(H), sup(e,H) \geqslant (k-2)$;
2. $\forall e_1, e_2 \in E(H), e_1$ and e_2 are triangle-connected;
3. There is no other subgraph H' of G that contains H and satisfies the above properties.

For the graph in Fig. 25(a) with a query vertex q and $k = 5$, there are 2 triangle-connected 5-truss communities containing the vertex q as shown in Fig. 25(b).

An online approach [122] and two index-based approaches, namely TCP-index [122] and EquiTruss [123], were proposed to solve the TTC search problem. Huang et al. [122] proposed an online approach to search TTC for G. It firstly computes the trussness of all edges in G by truss decomposition [172]. Then it searches the incident edges of the query vertex q and checks the edges with trussness not less than k to obtain triangle-connected truss communities. The online algorithm computes TTC in a BFS manner and iterates until all incident edges of q have been processed. Finally, k-truss communities which include the query vertex q are obtained. However, this algorithm is inefficient because it will produce a lot of redundant edge accesses when detecting unqualified edges. Meanwhile, Huang et al. [122] also presented a triangle connectivity-preserving index, namely TCP-index, to improve the query efficiency. For each vertex in G, the truss number and triangle adjacency relationship are stored in a compact tree-shape index. For each edge, it needs to access the index twice [122]. The query time is linear to the size of the community, which is optimal. The time complexity of TCP-index construction is $O\left(\sum_{(u,v)\in E} \min deg_G(u), deg_G(v)\right))$ and space complexity is $O(m)$.

Fig. 26 illustrates the TCP-index of the graph G in Fig. 25(a). When querying 5-truss communities, it retrieves two vertex sets of 5-truss edges, that are, $\{x_1, x_2, x_3, x_4\}$ and $\{s_1, s_2, s_3, s_4\}$.

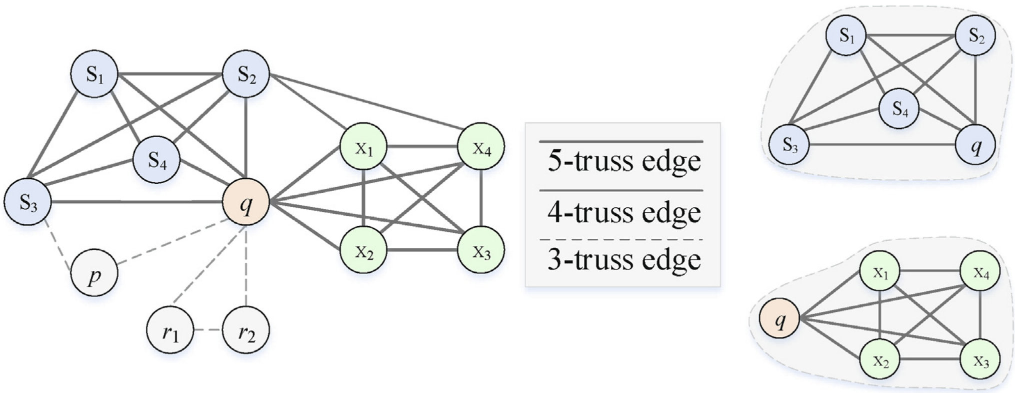
In [123], Akbas and Zhao proposed k-truss equivalence to further improve the query efficiency. The k-truss equivalence can be used to represent both the triangle connectivity and k-truss cohesiveness in the triangle-connected truss community. Specifically, given two edges $e_1, e_2 \in E, e_1$ and e_2 are k-truss equivalence, if and only if (1) $\tau(e_1) = \tau(e_2) = k$, and (2) e_1 and e_2 are triangle-connected in a k-truss. Accordingly, edges in a given graph G can be divided into mutually exclusive equivalence classes, each of which represents a TTC. As a result, it can directly find the triangle-connected communities containing the vertex q on EquiTruss without accessing the graph G. Fig. 27 shows the EquiTruss index of the graph in Fig. 25(a). Different from TCP-index, the truss community query based on EquiTruss-index is more efficient because it only needs to visit each edge once [123].

Closest Truss Community. For a query vertex q, the TTC model can find all overlapping communities. However, because of the strict constraint of triangle connectivity, TTC may fail to search communities for multiple query vertices. To address this issue, Huang et al. [124] presented the problem of the closest truss community (CTC) search.

Problem 28 (*CTC Search*). Given an undirected simple graph $G(V,E)$ and a query vertex set $Q \in V$, compute a subgraph $H \subseteq G$ which satisfies the following properties:

1. H is a connected k-truss and contains query vertices within Q, i.e., $Q \subseteq H, \forall e \in E(H), sup(e,H) \geqslant (k-2)$;
2. H is a subgraph with the smallest diameter among all subgraphs satisfying Property 1.

As shown in Fig. 28(a), the subgraph in the region shaded gray is a 4-truss containing the query vertex set $Q = \{q_1, q_2, q_3\}$ with diameter 4. Fig. 28(b) shows a 4-truss containing Q with a diameter 3.



(a) Graph G

(b) TTCs

Fig. 25. The TTC Search ($k = 5$).

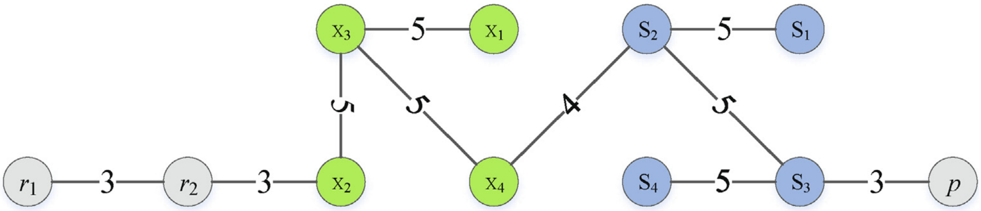


Fig. 26. TCP-index T_q for Vertex q of G in Fig. 25(a).

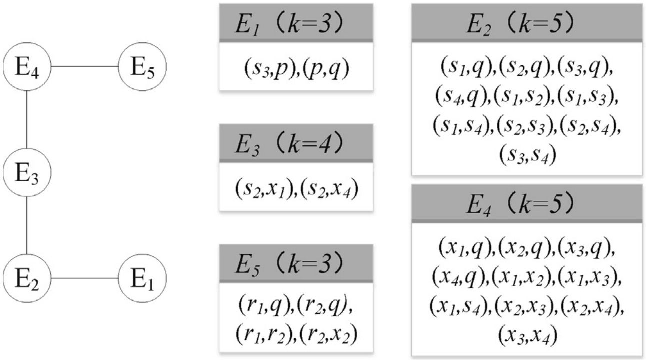

E4
E5
E3
E2
E1
E1 (k=3)
(s3,p),(p,q)
E3 (k=4)
(s2,x1),(s2,x4)
E5 (k=3)
(r1,q),(r2,q),(r1,r2),(r2,x2)
E2 (k=5)
(s1,q),(s2,q),(s3,q),(s4,q),(s1,s2),(s1,s3),(s1,s4),(s2,s3),(s2,s4),(s3,s4)
E4 (k=5)
(x1,q),(x2,q),(x3,q),(x4,q),(x1,x2),(x1,x3),(x1,s4),(x2,x3),(x2,x4),(x3,x4)

Fig. 27. EquiTruss Index of G in Fig. 25(a).

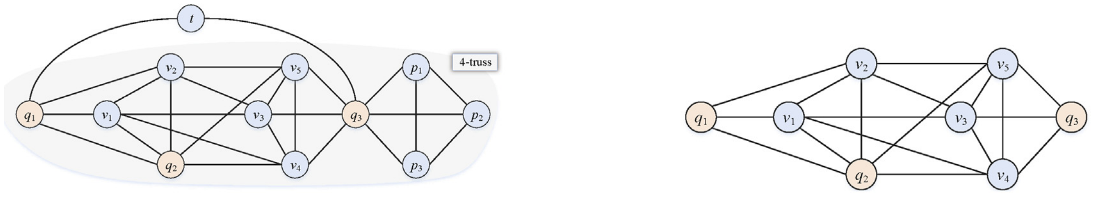
Huang et al. [124] proved that the problem of finding a k-truss with the minimum diameter is NP-hard. It is difficult to obtain a solution with an approximation ratio of less than 2 since it is closely related to the minimum diameter. In [124], the authors also proposed the method which first finds a maximal connected k-truss containing given query vertices with the largest trussness. Then it iteratively removes the vertex farthest from the query vertices and maintains the trussness. To further improve the efficiency of the CTC search, Huang et al. proposed two optimization strategies, namely bulk deletion and local exploration, respectively. The bulk deletion is a greedy strategy to achieve quick termination at the expense of the accuracy of results. It accelerates the pruning process by removing at least k vertices in batches. Besides, local exploration is a heuristic strategy to obtain the closest truss community from the local neighborhoods of query vertices.

5.4.2. Keyword-based attributed graphs

To find the communities which consist of vertices with similar attributes and contain given query vertices, Huang and Lakshmanan [125] investigated an attribute-driven truss community (ATC) search based on (k,d)-truss and an attribute score function. A (k,d)-truss is a subgraph H of G where each edge is contained by at least $k-2$ triangles. Besides, the communication cost between the vertices of H and query vertices is no greater than d.

Problem 29 (*ATC Search*). Given a graph G, a query $Q = (V_q, W_q)$, and two parameters k and d, retrieve an attributed truss community (ATC) H that satisfies the following properties:

1. H is a (k,d)-truss containing query vertices V_q;



(a) Graph G

(b) Closest Truss Community for $Q = \{q_1, q_2, q_3\}$

Fig. 28. The CTC Search.

2. H owns the maximum attribute score $f(H, W_q)$ among the subgraphs satisfying Property 1.

Here, the attribute score of the community H is denoted by W_q. The score function is $f(H, Wq) = \sum_{w \in W_q} \frac{score(H,w)^2}{|V(H)|}$, where $score(H, w) = |V_w \cap V(H)|$ is the number of vertices covering the query attribute w.

Fig. 29(a) shows a keyword-based attributed graph G. Given a query vertex set $V_q = \{q_1, q_2\}$ and a query attribute set W_q = {'DB', 'DM'}, H is a (k, d)-truss for V_q in G with $k = 4$ and $d = 2$ as depicted in Fig. 29(b).

The ATC search is proven to be NP-hard. To help efficiently processing of the ATC search, Huang et al. [125] presented a greedy framework and proposed an algorithm for ATC in a top-down manner. The algorithm first finds the maximal (k, d)-truss of G, then it prunes vertices with the smallest "attribute marginal gain" from the (k, d)-truss and ensures the remaining nodes meet the (k, d)-truss constraints. At last, it returns the (k, d)-truss with the maximum attribute score among all maximal (k, d)-trusses as the final result. To further boost the query efficiency, Huang et al. [125] presented a novel index, namely attributed truss index (ATindex). This index maintains structural trussness and attribute trussness at the same time. In addition, a local exploration strategy is proposed for efficiently computing a small neighborhood subgraph around the query vertices.

5.4.3. Weighted graphs

Zheng et al. [126] proposed a weighted truss community (WTC) model which aims to search k-truss communities over edge-weighted graphs.

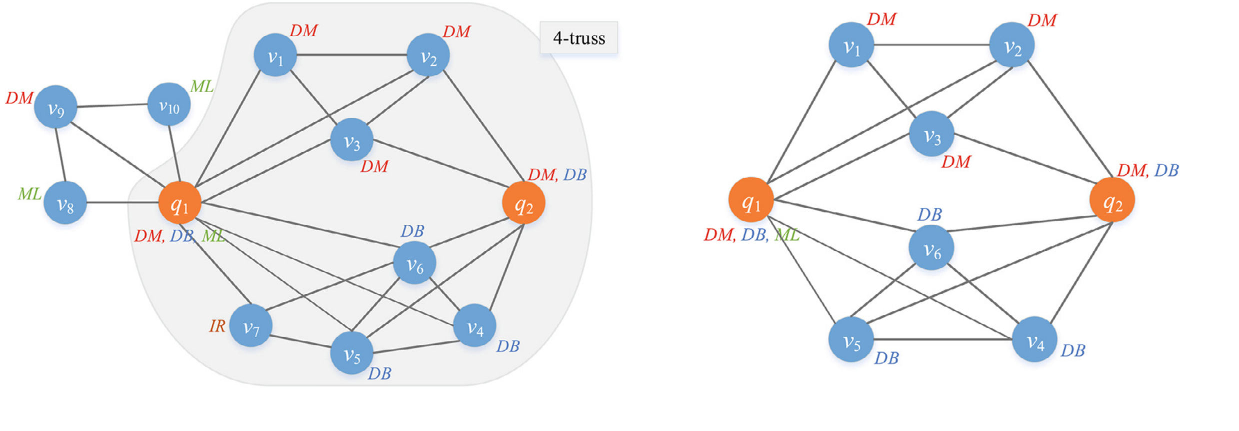
Definition 21 (*Weighted Truss Community*). Given an undirected weighted graph $G = (V, E, W)$ and an integer k, a weighted k-truss community is an induced subgraph $H \subseteq G$ such that the following properties hold:

1. $\forall e_1, e_2 \in E(H), e_1$ and e_2 are triangle connected in H;
2. $\forall e \in E(H), sup_H(e) \geqslant k - 2$;
3. There is no other subgraph H' of G that $H \subseteq H'$ and H' satisfies the above properties.

Problem 30 (*WTC Search*). Given an undirected weighted graph $G(V, E, W)$, parameters k and r, return r weighted k-truss communities H with the largest weights $w(H)$. Here, the community weight of H is $w(H) = min_{e \in E(H)} w(e)$.

Fig. 30 shows a weighted graph G with $k = 5$ and $r = 1$. The 5-truss community C_1 has the largest weight $\omega(C_1) = 0.8$. Therefore, C_1 is returned as the result of WTC.

A straightforward approach to find r communities with the largest community weights is to enumerate all weighted k-truss communities. However, it is time-consuming, especially in large graphs. To improve the query efficiency, Zheng et al. [126] proposed an index structure, namely KEP-Index, which organizes all the communities in a given graph G as a tree-shaped structure. All the weighted k-truss communities form a partial order relationship. As a result, the query time of WTC based on the KEP-index is linear to the size of the results.



(a) An attribute graph G

(b) H, the (4,2)-truss community for V_q={q_1, q_2} and W_q={$'DB', ' DM'$}

Fig. 29. An Example of the ATC Search.

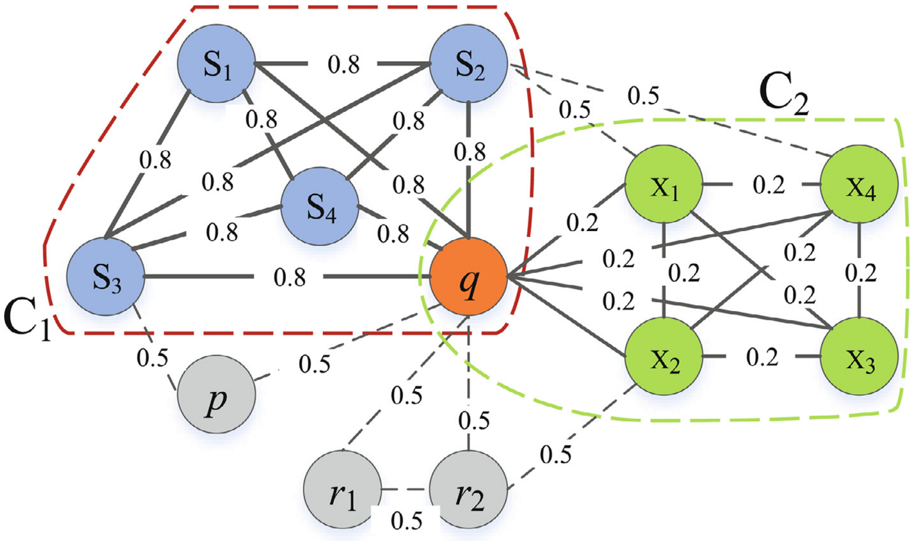


Fig. 30. An Example of the WTC Search.

5.5. Discussion

Tables 9 and 10 show the related work on k-core-based and k-truss-based community searches over different types of graphs, respectively. Different from point-related and path-related personalized graph queries, there are abundant studies about the CS problem, one of the most famous subgraph-related personalized graph queries, over various attributed graphs. Most of these CS problems were proposed based on the k-core model, while the research on k-truss-based CS is relatively less.

Table 9
Classification of k-core-based Community Search.

Graph	Problem	Algorithm	
		Type	Technique
Simple	Size-unbounded CS	Index-based	ShellStruct index [106]
		Online	Greedy algorithm [104], Local [105]
	Size-unbounded CS	Online	Heuristic algorithms [104]
	Minimum k-core	Online	Progressive search algorithm and L-Greedy [169]
	The best k-core	Online	Vertex rank and core forest [165]
	D-core CS	Index-based	NestIdx index [107], PathIdx index [107], and UnionIdx index [107]
	Persistent CS	Online	Prune-and-search approach [116]
Location-based	Spatial-aware CS	Online	AppInc [113] and AppAcc [113]
	RB-k-core CS	Online	TriV [38], and RotC [38]
	Geo-social CS	Index-based	Social-aware Rtree [115]
	Skyline Cohesive GQ	Index-based	Cd-tree [170]
Keyword-based	Attributed CS	Index-based	CL-tree [111]
Weighted-based	Skyline CS	Online	SkylineComm2D [121]]
HINs	(k, P)-core CS	Index-based	CoreIndex [136]

Table 10
Classification of k-truss-based Community Search.

Graph	Problem	Algorithm	
		Type	Technique
Simple	Triangle-connected Truss CS	Index-based	EquiTruss index [123], TCP-index [122]
		Online	Online search [122]
	Closest Truss CS	Online	Bulk deletion and local exploration, Greedy algorithm [124]
Keyword-based	Attribute-driven truss CS	Online	Greedy framework [125]
Weighted-based	Weighted truss CS	Index-based	KEP-Index [126]

As shown in Tables 9 and 10, the algorithms for solving the CS problems are divided into two categories: index-based algorithms and online algorithms. According to the nested property of k-core and k-truss, many index-based algorithms were proposed where all the communities are organized as a tree structure. Unlike index-based algorithms, most online algorithms first calculate a maximal cohesive subgraph and then prune unqualified vertices which do not satisfy the constraints. In addition, there are also online algorithms that filter the vertices unsatisfying the attribute constraints and then verify whether the remaining vertices constitute a cohesive subgraph. Compared to online algorithms, index-based algorithms require less query time. In contrast, online algorithms have an obvious advantage in terms of space overhead.

6. Future work

In this section, we discuss the challenges faced by personalized graph queries and propose a list of interesting future directions.

6.1. Personalized graph queries for new application scenarios

In many real-life applications, new types of graphs, including heterogeneous graphs, multilevel graphs, and signed graphs, attract increasing attention. Although there are colorful works about personalized graph queries, there is very little research on the corresponding queries over the aforementioned graphs. Take the similarity query as an example. At present, it also lacks an efficient similarity model for the similarity query over heterogeneous graphs, multilevel graphs, or signed graphs. Additionally, there are also many applications whose requirements cannot be satisfied by one type of personalized graph query. When organizing a dinner, it may need to find a group of users that have a close social relationship and at least one favorite POI. Consequently, it is imperative to research personalized graph queries over these new types of graphs. Related research outcomes are significant for meeting the requirements of users in many new real-life applications.

There are also some personalized graph hybrid graph queries, such as skyline cohesive group query [38] and (k, r)-core-based community search [135], that are proposed to satisfy complex application requirements of users. The skyline cohesive group query proposed in [38] has a close relationship with both the path-related and subgraph-related graph query. Consider the (k, r)-core-based community search in [135]. It combines the point-related and subgraph-related graph query. With the rapid development of society, the preferences of users will be more complex and diversified. Accordingly, new hybrid personalized graph queries need to be investigated to meet these complex requirements.

6.2. User-friendly personalized graph queries

In most personalized graph queries, the query results depend on the important input parameters specified by users. In general, different parameters bring different query results. For instance, in the k-truss-based community search, there is no solution that could help users specify the parameter k instead of manual selection. It is a major challenge for users to provide appropriate parameters without guidelines.

In [6], Fang et al. suggested that the values of parameters can be computed automatically on the basis of historical query logs. Moreover, the crowdsourcing platform can also be utilized to facilitate query suggestions about parameters. In addition to the aforementioned directions, another direction is identifying new personalized graph queries that are parameter-free and studying interactive personalized graph queries. For example, Lin et al. [165] presented the best k-core with the goal of computing the k-core with the highest scores.

6.3. Why-not personalized graph queries

In practice, users always gain unexpected results without appropriate parameters. Inspired by this, why-not problems have received considerable attention because they help users explore the reason for obtaining unexpected results and provide suggestions for users to obtain their expected results. According to a designed loss function and user requirements, the main idea is to automatically adjust the query point or specify parameters. For example, in [173], the why-not subgraph matching problem adjusts the query subgraph until the expected results are obtained. The why-not problem is useful to improve the quality and usability of database systems. There have been some related studies in data management [174–176]. However, the research of why-not personalized graph queries is still in the early stage.

6.4. Personalized graph queries with size constraints

Some personalized graph queries face the challenge of a prohibitively larger number of query results. Accordingly, it is difficult for users to make a final decision from so many results manually.

To address this issue, it is an effective direction to research personalized graph queries with size constraints. In addition, it is better to offer users various choices to meet their distinct requirements. For example, we can investigate the top k similarity query that aims to return the best k results with the largest difference.

Most of the existing algorithms for personalized graph queries cannot obtain any result before completing the calculation of the entire result set. In addition to new personalized graph queries that can return a controllable size of results, another direction is to devise progressive algorithms for corresponding queries. As a result, users can gain query results progressively, and they can stop the query in a timely manner when enough results are returned.

6.5. Distributed approaches to personalized graph queries

The existing approaches to personalized graph queries are mainly in-memory algorithms. They are based on the assumption that the memory of a single machine is large enough to store a given graph. With the explosive growth in the size of graphs, this assumption is invalid in most cases. For example, a well-known social network data set Friendster consists of 65,608,366 vertices and 1,806,067,135 edges. A single machine cannot meet the memory requirement of personalized graph queries over such a big data set.

To address this concern, it is urgent to research these queries over distributed environments and propose effective distributed algorithms. To reduce communication costs, effective graph partition and pruning strategies are necessary. In addition, index techniques can also be introduced to boost query performance.

6.6. High-performance approaches to personalized graph queries

Numerous personalized graph queries have been proved to be NP-hard. In most cases, we cannot obtain exact results in a reasonable time. As a result, many approximate approaches are presented to trade off some accuracy for efficiency. However, over large-scale graphs, these approximate approaches may also have performance bottlenecks.

Recently, the technology of general-purpose graphics processing units (GPGPU) has made considerable progress and development. GPU has attracted many researchers for its features of massive parallelism and high memory access bandwidth [177]. It has played an important role in accelerating various graph queries. To address the performance problem of personalized graph queries, especially over large graphs, one direction is to investigate these queries in a heterogeneous environment and exert the advantages of GPUs which is a high degree of parallelism.

6.7. Graph neural networks for personalized queries

Graph neural networks (GNN), based on deep learning technologies for graphing structured data, have achieved superior performance across various graph-based tasks, such as computer vision, natural language processing and recommendation systems [178]. Recently, researchers have investigated solving various graph query tasks with GNNs [179–183]. Graph embedding is able to embed vertices into low-dimensional space while reserving the graph topology information. In this latent space, it is empirical to provide fast graph similarity computations [181] and conduct similarity search based on proximity embedding [179,180]. ICS-GNN [183] proposed a community search method based on GNN to locate the target community over a subgraph collected on the fly from an online network. It is more flexible and lightweight than the traditional rule-based methods. [184] utilizes GNN to detect communities with overlapping.

Although these works have demonstrated the effectiveness of GNNs for graph query tasks, it still remains a big challenge to learn the rich attribute information such as time, location and so on through GNNs for personalized query tasks in practise.

6.8. Graph query languages and personalized queries

Graph query languages use special declarative languages to support high-level interfaces to query the graphs [133]. Recently there are some basic graph query languages that are proposed [185–188] and are developing. These graph query languages can be categorized into two kinds, i.e., RDF graph model-based (SPARQL [188]) and property graph model based (PGQL [185], Cyper [186], and G-core [187]). The property graph model-based graph query languages can support the definition of attributed graphs in the personalized graph queries, while the RDF is not straight weight suitable [189]. However, there is no standard graph query language for property graph query right now, unlike the SPARQL for the RDF-based graph model.

Therefore, there are opportunities for graph query languages due to the complexity of personalized graph queries. For example, the supporting queries can support a lot of path-related queries, but there are no languages that can tackle the similar queries and the community queries mentioned in this paper. To deal with these graph queries, more utility syntax and operators should be designed while considering the scalability for more complex queries. The research of personalized graph queries not only gives inspiration to graph query languages to design suitable operators and syntax to support these queries, but also comprised the foundation solving methods for graph query languages. Almost all of the existing graph query languages use path-match as the essential technique for many graph queries, then the methods proposed for path-related graph queries in this survey can be taken into account as the basic method in the graph query language engines.

7. Conclusion

In this paper, we conduct a comprehensive survey on personalized graph queries, whose goal is to compute personalized query results for users on the basis of their personalized preferences in terms of specified query vertices, structures, and attributes. We focus on the latest research on personalized graph queries, as well as some classic studies. These researches are classified into three categories: point-related, path-related, and subgraph-related graph queries for the first time. After analyzing the state-of-the-art approaches to these personalized graph queries, we discuss the challenges they face and also point out a list of future directions as guidelines for researchers.

CRediT authorship contribution statement

Peiying Lin: Conceptualization, Methodology, Formal analysis, Investigation, Writing – original draft, Writing – review & editing. **Yangfan Li:** Investigation, Writing – original draft. **Wensheng Luo:** Investigation, Writing – original draft. **Xu Zhou:** Investigation, Writing – original draft. **Kenli Li:** Writing – review & editing, Resources. **Keqin Li:** Writing – review & editing, Resources.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research was supported by China Scholarship Council, the NSFC (Grant Nos. 61802032, 61772182, 61772088), the Hunan Province Key Laboratory of Industrial Internet Technology and Security (2019TP1011), the Key Area Research Program of Hunan (2019GK2091), and the Emergency Special Project of NSFC (Grant No. 61751204).

References

[1] Z. Du, J. Tang, Y. Ding, Polar++: Active one-shot personalized article recommendation, IEEE Trans. Knowl. Data Eng. (2019) 1.
[2] M. Sarwat, R. Moraffah, M.F. Mokbel, J.L. Avery, Database system support for personalized recommendation applications, ICDE (2017) 1320–1331, https://doi.org/10.1109/ICDE.2017.174.
[3] L. Li, M. Zhang, W. Hua, X. Zhou, Fast query decomposition for batch shortest path processing in road networks, in: 36th IEEE International Conference on Data Engineering (ICDE'20), IEEE, 2020, pp. 1189–1200.
[4] A.D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, S. Zhou, Shortest path and distance queries on road networks: towards bridging theory and practice, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, ACM, 2013, pp. 857–868.
[5] V.J. Wei, R.C.-W. Wong, C. Long, Architecture-intact oracle for fastest path and time queries on dynamic spatial networks, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 1841–1856.
[6] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, X. Lin, A survey of community search over big graphs, VLDB J. 29 (1) (2020) 353–392.
[7] G. Jeh, J. Widom, Simrank: a measure of structural-context similarity, in: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002, pp. 538–543.
[8] D. Fogaras, B. Rácz, Scaling link-based similarity search, in: Proceedings of the 14th international conference on World Wide Web (WWW'05), 2005, pp. 641–650.
[9] P. Zhao, J. Han, Y. Sun, P-rank: a comprehensive structural similarity measure over information networks, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM'09), ACM, 2009, pp. 553–562.
[10] G. He, H. Feng, C. Li, H. Chen, Parallel simrank computation on large graphs with iterative aggregation, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2010, pp. 543–552.
[11] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, T. Wu, Fast computation of simrank for static and dynamic information networks, in: 13th International Conference on Extending Database Technology (EDBT'10), Vol. 426, ACM, 2010, pp. 465–476..
[12] L. Sun, C. Cheng, X. Li, D. Cheung, J. Han, On link-based similarity join, PVLDB 4 (11) (2011) 714–725.
[13] L. Cao, B. Cho, H.D. Kim, Z. Li, M.-H. Tsai, I. Gupta, Delta-simrank computing on mapreduce, in: Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: Algorithms, systems, programming models and applications, ACM, 2012, pp. 28–35..
[14] P. Lee, L.V. Lakshmanan, J.X. Yu, On top-k structural similarity search, in: IEEE 28th International Conference on Data Engineering (ICDE'12), IEEE, 2012, pp. 774–785.
[15] W. Yu, W. Zhang, X. Lin, Q. Zhang, J. Le, A space and time efficient algorithm for simrank computation, World Wide Web 15 (3) (2012) 327–353.
[16] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, M. Onizuka, Efficient search algorithm for simrank, in: 29th IEEE International Conference on Data Engineering (ICDE'13), IEEE, 2013, pp. 589–600.
[17] W. Yu, X. Lin, W. Zhang, Towards efficient simrank computation on large networks, in: 29th IEEE International Conference on Data Engineering (ICDE'13), 2013, pp. 601–612.
[18] J. He, H. Liu, J.X. Yu, P. Li, W. He, X. Du, Assessing single-pair similarity over graphs by aggregating first-meeting probabilities, Inf. Syst. 42 (2014) 107–122.
[19] W. Yu, X. Lin, W. Zhang, J.A. McCann, Fast all-pairs simrank assessment on large graphs and bipartite domains, IEEE Trans. Knowl. Data Eng. 27 (7) (2014) 1810–1823.
[20] T. Maehara, M. Kusumoto, K.-I. Kawarabayashi, Efficient simrank computation via linearization, arXiv preprint arXiv:1411.7228 (2014)..
[21] M. Kusumoto, T. Maehara, K.-I. Kawarabayashi, Scalable similarity search for simrank, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, ACM, 2014, pp. 325–336.
[22] W. Tao, G. Li, Efficient top-k simrank-based similarity join, in: International Conference on Management of Data, SIGMOD 2014, ACM, 2014, pp. 1603–1604..
[23] H.-H. Chen, C.L. Giles, Ascos++ an asymmetric similarity measure for weighted networks to address the problem of simrank, ACM Trans. Knowl. Discovery Data 10 (2) (2015) 1–26.

[24] Z. Li, Y. Fang, Q. Liu, J. Cheng, R. Cheng, J. Lui, Walking in the cloud: Parallel simrank at scale, PVLDB 9 (1) (2015) 24–35.
[25] Y. Shao, B. Cui, L. Chen, M. Liu, X. Xie, An efficient similarity search framework for simrank over large dynamic graphs, PVLDB 8 (8) (2015) 838–849.
[26] W. Yu, J.A. McCann, High quality graph-based similarity search, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2015, pp. 83–92.
[27] W. Yu, J.A. McCann, Efficient partial-pairs simrank search on large networks, PVLDB 8 (5) (2015) 569–580.
[28] T. Maehara, M. Kusumoto, K.-I. Kawarabayashi, Scalable simrank join algorithm, in: 31st IEEE International Conference on Data Engineering (ICDE'15), IEEE, 2015, pp. 603–614.
[29] M. Zhang, H. Hu, Z. He, L. Gao, L. Sun, Efficient link-based similarity search in web networks, Expert Syst. Appl. 42 (22) (2015) 8868–8880.
[30] B. Tian, X. Xiao, Sling: A near-optimal index structure for simrank, in: Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, ACM, 2016, pp. 1859–1874.
[31] S.-H. Yoon, S.-W. Kim, S. Park, C-rank: A link-based similarity measure for scientific literature databases, Inf. Sci. 326 (2016) 25–40.
[32] M. Jiang, A.W.-C. Fu, R.C.-W. Wong, Reads: a random walk approach for efficient and accurate dynamic simrank, PVLDB 10 (9) (2017) 937–948.
[33] R. Li, X. Zhao, H. Shang, Y. Chen, W. Xiao, Fast top-k similarity join for simrank, Inf. Sci. 381 (2017) 1–19.
[34] Y. Liu, B. Zheng, X. He, Z. Wei, X. Xiao, K. Zheng, J. Lu, Probesim: scalable single-source and top-k simrank computations on dynamic graphs, PVLDB 11 (1) (2017) 14–26.
[35] W. Zheng, L. Zou, L. Chen, D. Zhao, Efficient simrank-based similarity join, ACM Transactions on Database Systems (TODS) 42 (3) (2017) 1–37.
[36] W. Yu, X. Lin, W. Zhang, J.A. McCann, Dynamical simrank search on time-varying networks, VLDB J. 27 (1) (2018) 79–104.
[37] X. Huang, X. Gao, J. Tang, G. Wu, A parallel method for all-pair simrank similarity computation, Algorithms and Architectures for Parallel Processing – 18th International Conference (ICA3PP'18), vol. 11334, Springer, 2018, pp. 593–607.
[38] K. Wang, X. Cao, X. Lin, W. Zhang, L. Qin, Efficient computing of radius-bounded k-cores, in: 34th IEEE International Conference on Data Engineering (ICDE'18), IEEE, 2018, pp. 233–244.
[39] Z. Wei, X. He, X. Xiao, S. Wang, Y. Liu, X. Du, J.-R. Wen, Prsim: Sublinear time simrank computation on large power-law graphs, in: Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, ACM, 2019, pp. 1042–1059.
[40] Y. Wang, L. Chen, Y. Che, Q. Luo, Accelerating pairwise simrank estimation over static and dynamic graphs, VLDB J. 28 (1) (2019) 99–122.
[41] Y. Wang, Z. Feng, L. Chen, Z. Li, X. Jian, Q. Luo, Efficient similarity search for sets over graphs, IEEE Trans. Knowl. Data Eng. (2019).
[42] W. Yu, X. Lin, W. Zhang, J. Pei, J.A. McCann, Simrank*: effective and scalable pairwise similarity search based on graph topology, VLDB J. 28 (3) (2019) 401–426.
[43] H. Wang, Z. Wei, Y. Yuan, X. Du, J.-R. Wen, Exact single-source simrank computation on large graphs, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 653–663.
[44] M. Li, F.M. Choudhury, R. Borovica-Gajic, Z. Wang, J. Xin, J. Li, Crashsim: An efficient algorithm for computing simrank over static and temporal graphs, in: 2020 IEEE 36th International Conference on Data Engineering (ICDE), IEEE, 2020, pp. 1141–1152.
[45] Y. Liu, L. Zou, Q. Ge, Z. Wei, Simtab: accuracy-guaranteed simrank queries through tighter confidence bounds and multi-armed bandits, Proceedings of the VLDB Endowment 13 (12) (2020) 2202–2214.
[46] Y. Wang, Y. Che, X. Lian, L. Chen, Q. Luo, Fast and accurate simrank computation via forward local push and its parallelization, IEEE Trans. Knowl. Data Eng. (2020).
[47] J. Lu, Z. Gong, Y. Yang, A matrix sampling approach for efficient simrank computation, Inf. Sci. 556 (2021) 1–26.
[48] Y. Sun, J. Han, X. Yan, P.S. Yu, T. Wu, Pathsim: Meta path-based top-k similarity search in heterogeneous information networks, Proceedings of the VLDB Endowment 4 (11) (2011) 992–1003.
[49] C. Shi, X. Kong, Y. Huang, S.Y. Philip, B. Wu, Hetesim: A general framework for relevance measure in heterogeneous networks, IEEE Trans. Knowl. Data Eng. 26 (10) (2014) 2479–2492.
[50] I. Antonellis, H. Garcia-Molina, C. Chang, Simrank++ query rewriting through link analysis of the clickgraph (poster), in: Proceedings of the 17th international conference on World Wide Web (WWW'08), ACM, 2008, pp. 1177–1178.
[51] L. Du, C. Li, H. Chen, L. Tan, Y. Zhang, Probabilistic simrank computation over uncertain graphs, Inf. Sci. 295 (2015) 521–535.
[52] R. Zhu, Z. Zou, J. Li, Simrank on uncertain graphs, IEEE Trans. Knowl. Data Eng. 29 (11) (2017) 2522–2536.
[53] W. Fan, X. Wang, Y. Wu, Performance guarantees for distributed reachability queries, PVLDB 5 (11) (2012) 1304–1316.
[54] M. Then, M. Kaufmann, F. Chirigati, T.-A. Hoang-Vu, K. Pham, A. Kemper, T. Neumann, H.T. Vo, The more the merrier: Efficient multi-source graph traversal, PVLDB 8 (4) (2014) 449–460.
[55] E. Cohen, E. Halperin, H. Kaplan, U. Zwick, Reachability and distance queries via 2-hop labels, SIAM J. Comput. 32 (5) (2003) 1338–1355.
[56] J. Cheng, S. Huang, H. Wu, A.W.-C. Fu, Tf-label: a topological-folding labeling scheme for reachability querying in a large graph, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013, pp. 193–204.
[57] R. Jin, G. Wang, Simple, fast, and scalable reachability oracle, PVLDB 6 (14) (2013).
[58] T. Zhang, Y. Gao, C. Li, C. Ge, W. Guo, Q. Zhou, Distributed reachability queries on massive graphs, in: Database Systems for Advanced Applications - 24th International Conference (DASFAA'19), vol. 11448, Springer, 2019, pp. 406–410..
[59] R. Jin, N. Ruan, S. Dey, J.Y. Xu, Scarab: scaling reachability computation on large graphs, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, ACM, 2012, pp. 169–180.
[60] S. Seufert, A. Anand, S. Bedathur, G. Weikum, Ferrari: Flexible and efficient reachability range assignment for graph indexing, in: 29th IEEE International Conference on Data Engineering (ICDE'13), 2013, pp. 1009–1020.
[61] N. Sengupta, A. Bagchi, M. Ramanath, S. Bedathur, Arrow: Approximating reachability using random walks over web-scale graphs, in: 35th IEEE International Conference on Data Engineering (ICDE'19), IEEE, 2019, pp. 470–481.
[62] J. Su, Q. Zhu, H. Wei, J.X. Yu, Reachability querying: Can it be even faster?, IEEE Trans Knowl. Data Eng. 29 (3) (2016) 683–697.
[63] H. Wei, J.X. Yu, C. Lu, R. Jin, Reachability querying: An independent permutation labeling approach, PVLDB 7 (12) (2014) 1191–1202.
[64] J. Zhou, S. Zhou, J.X. Yu, H. Wei, Z. Chen, X. Tang, Dag reduction: Fast answering reachability queries, in: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, ACM, 2017, pp. 375–390.
[65] S. Gurajada, M. Theobald, Distributed set reachability, in: Proceedings of the 2016 International Conference on Management of Data, ACM, 2016, pp. 1247–1261.
[66] H. Wu, Y. Huang, J. Cheng, J. Li, Y. Ke, Reachability and time-based path queries in temporal graphs, in: 32nd IEEE International Conference on Data Engineering (ICDE'16), IEEE, 2016, pp. 145–156.
[67] T. Zhang, Y. Gao, L. Chen, W. Guo, S. Pu, B. Zheng, C.S. Jensen, Efficient distributed reachability querying of massive temporal graphs, VLDB J. 28 (6) (2019) 871–896.
[68] D. Wen, Y. Huang, Y. Zhang, L. Qin, W. Zhang, X. Lin, Efficiently answering span-reachability queries in large temporal graphs, in: 36th IEEE International Conference on Data Engineering (ICDE'20), IEEE, 2020, pp. 1153–1164.
[69] S. Gao, K. Anyanwu, Prefixsolve: efficiently solving multi-source multi-destination path queries on rdf graphs by sharing suffix computations, in: Proceedings of the 22nd international conference on World Wide Web (WWW'13), 2013, pp. 423–434.
[70] S. Wadhwa, A. Prasad, S. Ranu, A. Bagchi, S. Bedathur, Efficiently answering regular simple path queries on large labeled networks, in: Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, ACM, 2019, pp. 1463–1480.
[71] L.D. Valstar, G.H. Fletcher, Y. Yoshida, Landmark indexing for evaluation of label-constrained reachability queries, in: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, ACM, 2017, pp. 345–358.
[72] Y. Peng, Y. Zhang, X. Lin, L. Qin, W. Zhang, Answering billion-scale label-constrained reachability queries within microsecond, PVLDB 13 (6) (2020) 812–825.

[73] G.S. Fishman, A comparison of four monte carlo methods for estimating the probability of s-t connectedness, IEEE Trans. Reliab. 35 (2) (1986) 145–155.
[74] R.-H. Li, J.X. Yu, R. Mao, T. Jin, Recursive stratified sampling: A new framework for query evaluation on uncertain graphs, IEEE Trans. Knowl. Data Eng. 28 (2) (2015) 468–482.
[75] Y. Li, J. Fan, D. Zhang, K.-L. Tan, Discovering your selling points: Personalized social influential tags exploration, in: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, ACM, 2017, pp. 619–634.
[76] R. Zhu, Z. Zou, J. Li, Top-k reliability search on uncertain graphs, in: 2015 IEEE International Conference on Data Mining (ICDM'15), IEEE, 2015, pp. 659–668.
[77] S. Maniu, R. Cheng, P. Senellart, An indexing framework for queries on probabilistic graphs, ACM Transactions on Database Systems (TODS) 42 (2) (2017) 1–34.
[78] E.W. Dijkstra et al, A note on two problems in connexion with graphs, Numerische mathematik 1 (1) (1959) 269–271.
[79] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[80] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (3) (1987) 596–615.
[81] T. Akiba, Y. Iwata, Y. Yoshida, Fast exact shortest-path distance queries on large networks by pruned landmark labeling, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, 2013, pp. 349–360.
[82] T. Akiba, Y. Iwata, K.-I. Kawarabayashi, Y. Kawata, Fast shortest-path distance queries on road networks by pruned highway labeling, in: 2014 Proceedings of the sixteenth workshop on algorithm engineering and experiments (ALENEX), SIAM, 2014, pp. 147–154.
[83] T. Akiba, Y. Iwata, Y. Yoshida, Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling, in: Proceedings of the 23rd international conference on World wide web (WWW'14), ACM, 2014, pp. 237–248.
[84] W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, X. Lin, Scaling distance labeling on small-world networks, in: Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, ACM, 2019, pp. 1060–1077.
[85] T. Hayashi, T. Akiba, K. Kawarabayashi, Fully dynamic shortest-path distance query acceleration on massive networks, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM'16), ACM, 2016, pp. 1533–1542.
[86] W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, X. Lin, Scaling up distance labeling on graphs with core-periphery properties, in: Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, ACM, 2020, pp. 1367–1381.
[87] R. Geisberger, P. Sanders, D. Schultes, D. Delling, Contraction hierarchies: Faster and simpler hierarchical routing in road networks, in: International Workshop on Experimental and Efficient Algorithms, Springer, 2008, pp. 319–333.
[88] Z. Li, L. Chen, Y. Wang, G*-tree: An efficient spatial index on road networks, in: 35th IEEE International Conference on Data Engineering (ICDE'19), IEEE, 2019, pp. 268–279.
[89] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, Q. Zhu, When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks, in: Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, ACM, 2018, pp. 709–724.
[90] D. Ouyang, L. Yuan, L. Qin, L. Chang, Y. Zhang, X. Lin, Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees, PVLDB 13 (5) (2020) 602–615.
[91] D. Zhang, D. Yang, Y. Wang, K.-L. Tan, J. Cao, H.T. Shen, Distributed shortest path query processing on dynamic road networks, VLDB J. 26 (3) (2017) 399–419.
[92] V.J. Wei, R.C. Wong, C. Long, Architecture-intact oracle for fastest path and time queries on dynamic spatial networks, in: Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, ACM, 2020, pp. 1841–1856.
[93] R. Zhong, G. Li, K.-L. Tan, L. Zhou, Z. Gong, G-tree: An efficient and scalable index for spatial search on road networks, IEEE Trans. Knowl. Data Eng. 27 (8) (2015) 2175–2189.
[94] J.R. Thomsen, M.L. Yiu, C.S. Jensen, Effective caching of shortest paths for location-based services, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, ACM, 2012, pp. 313–324.
[95] J.R. Thomsen, M.L. Yiu, C.S. Jensen, Concise caching of driving instructions, in: Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2014, pp. 23–32.
[96] S. Wang, X. Xiao, Y. Yang, W. Lin, Effective indexing for approximate constrained shortest path queries on large road networks, PVLDB 10 (2) (2016) 61–72.
[97] L. Li, S. Wang, X. Zhou, Time-dependent hop labeling on road network, in: 35th IEEE International Conference on Data Engineering (ICDE'19), IEEE, 2019, pp. 902–913.
[98] Y. Wang, G. Li, N. Tang, Querying shortest paths on time dependent road networks, PVLDB 12 (11) (2019) 1249–1261.
[99] K. Semertzidis, E. Pitoura, K. Lillis, Timereach: Historical reachability queries on evolving graphs, in: Proceedings of the 18th International Conference on Extending Database Technology (EDBT'15), Vol. 15, OpenProceedings.org, 2015, pp. 121–132..
[100] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, Y. Xu, Path problems in temporal graphs, PVLDB 7 (9) (2014) 721–732.
[101] X. Zhang, M.T. Özsu, Correlation constraint shortest path over large multi-relation graphs, PVLDB 12 (5) (2019) 488–501.
[102] Y. Cheng, Y. Yuan, G. Wang, B. Qiao, Z. Wang, Efficient sampling methods for shortest path query over uncertain graphs, International Conference on Database Systems for Advanced Applications (DASFAA'14), Vol. 8422, Springer, 2014, pp. 124–140.
[103] L. Zou, P. Peng, D. Zhao, Top-k possible shortest path query over a large uncertain graph, in: Web Information System Engineering (WISE'11), vol. 6997, Springer, 2011, pp. 72–86..
[104] M. Sozio, A. Gionis, The community-search problem and how to plan a successful cocktail party, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2010, pp. 939–948.
[105] W. Cui, Y. Xiao, H. Wang, W. Wang, Local search of communities in large graphs, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, ACM, 2014, pp. 991–1002.
[106] N. Barbieri, F. Bonchi, E. Galimberti, F. Gullo, Efficient and effective community search, Data mining and knowledge discovery 29 (5) (2015) 1406–1433.
[107] Y. Fang, Z. Wang, R. Cheng, H. Wang, J. Hu, Effective and efficient community search over large directed graphs, IEEE Trans. Knowl. Data Eng. 31 (11) (2018) 2093–2107.
[108] C. Giatsidis, D.M. Thilikos, M. Vazirgiannis, D-cores: measuring collaboration of directed graphs based on degeneracy, Knowl. Inf. Syst. 35 (2) (2013) 311–343.
[109] Y. Fang, R. Cheng, On attributed community search, in: MATES Workshop in PVLDB, Springer, 2017, pp. 1–21..
[110] Y. Fang, R. Cheng, Y. Chen, S. Luo, J. Hu, Effective and efficient attributed community search, VLDB J. 26 (6) (2017) 803–828.
[111] Y. Fang, R. Cheng, S. Luo, J. Hu, Effective community search for large attributed graphs, PVLDB 9 (12) (2016) 1233–1244.
[112] J. Shang, C. Wang, C. Wang, G. Guo, J. Qian, An attribute-based community search method with graph refining, J. Supercomput. (2017) 1–28.
[113] Y. Fang, R. Cheng, X. Li, S. Luo, J. Hu, Effective community search over large spatial graphs, PVLDB 10 (6) (2017) 709–720.
[114] Y. Fang, Z. Wang, R. Cheng, X. Li, S. Luo, J. Hu, X. Chen, On spatial-aware community search, IEEE Trans. Knowl. Data Eng. 31 (4) (2018) 783–798.
[115] Q. Zhu, H. Hu, C. Xu, J. Xu, W.-C. Lee, Geo-social group queries with minimum acquaintance constraints, VLDB J. 26 (5) (2017) 709–727.
[116] R.-H. Li, J. Su, L. Qin, J.X. Yu, Q. Dai, Persistent community search in temporal networks, in: 34th IEEE International Conference on Data Engineering (ICDE'18), IEEE, 2018, pp. 797–808.
[117] R.-H. Li, L. Qin, J.X. Yu, R. Mao, Influential community search in large networks, PVLDB 8 (5) (2015) 509–520.
[118] S. Chen, R. Wei, D. Popova, A. Thomo, Efficient computation of importance based communities in web-scale networks using a single machine, CIKM (2016) 1553–1562.

[119] F. Bi, L. Chang, X. Lin, W. Zhang, An optimal and progressive approach to online search of top-k influential communities, PVLDB 11 (9) (2018) 1056–1068.
[120] R.-H. Li, L. Qin, J.X. Yu, R. Mao, Finding influential communities in massive networks, VLDB J. 26 (6) (2017) 751–776.
[121] R.-H. Li, L. Qin, F. Ye, J.X. Yu, X. Xiao, N. Xiao, Z. Zheng, Skyline community search in multi-valued networks, in: Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, ACM, 2018, pp. 457–472.
[122] X. Huang, H. Cheng, L. Qin, W. Tian, J.X. Yu, Querying k-truss community in large and dynamic graphs, SIGMOD (2014) 1311–1322.
[123] E. Akbas, P. Zhao, Truss-based community search: a truss-equivalence based indexing approach, PVLDB 10 (11) (2017) 1298–1309.
[124] X. Huang, L.V. Lakshmanan, J.X. Yu, H. Cheng, Approximate closest community search in networks, arXiv preprint arXiv:1505.05956 (2015)..
[125] X. Huang, L.V. Lakshmanan, Attribute-driven community search, PVLDB 10 (9) (2017) 949–960.
[126] Z. Zheng, F. Ye, R.-H. Li, G. Ling, T. Jin, Finding weighted k-truss communities in large networks, Inf. Sci. 417 (2017) 344–360.
[127] Z. Zhang, Y. Shao, B. Cui, C. Zhang, An experimental evaluation of simrank-based similarity search algorithms, PVLDB 10 (5) (2017) 601–612.
[128] C. Sommer, Shortest-path queries in static networks, ACM Computing Surveys (CSUR) 46 (4) (2014) 1–31.
[129] Y. Fang, X. Huang, L. Qin, W. Zhang, C. Cheng, X. Lin, D.M. Fragkiskos, G. Christos, N.P. Apostolos, V. Michalis, The core decomposition of networks: theory, algorithms and applications, VLDB J. 29 (2020) 61–92.
[130] G. Rossetti, R. Cazabet, Community discovery in dynamic networks: A survey, ACM Computing Surveys (CSUR) 51 (2) (2018) 1–37.
[131] R. Bian, Y.S. Koh, G. Dobbie, A. Divoli, Identifying top-k nodes in social networks: A survey, ACM Comput. Surv. 52 (1) (2019).
[132] X. Huang, L.V. Lakshmanan, J. Xu, Community search over big graphs: Models, algorithms, and opportunities, in: 2017 IEEE 33rd international conference on data engineering (ICDE), IEEE, 2017, pp. 1451–1454..
[133] Y. Wang, Y. Li, J. Fan, C. Ye, M. Chai, A survey of typical attributed graph queries, World Wide Web 24 (1) (2021) 297–346.
[134] C. Pizzuti, A. Socievole, A genetic algorithm for community detection in attributed graphs, in: International Conference on the Applications of Evolutionary Computation, Vol. 10784, Springer, 2018, pp. 159–170..
[135] F. Zhang, Y. Zhang, L. Qin, W. Zhang, X. Lin, When engagement meets similarity: efficient (k, 2016, r)-core computation on social networks, arXiv preprint arXiv:1611.03254.
[136] Y. Fang, Y. Yang, W. Zhang, X. Lin, X. Cao, Effective and efficient community search over large heterogeneous information networks, PVLDB 13 (6) (2020) 854–867.
[137] N. Narayanaswamy, R. Vijayaragunathan, Parameterized optimization in uncertain graphs - A survey and some results, Algorithms 13 (1) (2020) 3.
[138] L. Li, H. Wang, J. Li, H. Gao, A survey of uncertain data management, Front. Comput. Sci. (2020) 1–29.
[139] Z. Abbassi, V.S. Mirrokni, A recommender system based on local random walks and spectral methods, in: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis, Springer, 2007, pp. 102–108.
[140] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, J. Am. Soc. Inf. Sci. Technol. 58 (7) (2007) 1019–1031.
[141] N. Spirin, J. Han, Survey on web spam detection: principles and algorithms, ACM SIGKDD explorations newsletter 13 (2) (2012) 50–64.
[142] Y. Zhou, H. Cheng, J.X. Yu, Graph clustering based on structural/attribute similarities, PVLDB 2 (1) (2009) 718–729.
[143] P. Jaccard, Étude comparative de la distribution florale dans une portion des alpes et des jura, Bull Soc Vaudoise Sci Nat 37 (1901) 547–579.
[144] L.R. Dice, Measures of the amount of ecologic association between species, Ecology 26 (3) (1945) 297–302.
[145] R. Baeza-Yates, B. Ribeiro-Neto, et al, Modern information retrieval, vol. 463, ACM Press, New York, 1999.
[146] M.M. Kessler, Bibliographic coupling between scientific papers, American documentation 14 (1) (1963) 10–25.
[147] H. Small, Co-citation in the scientific literature: A new measure of the relationship between two documents, J. Am. Soc. Inf. Sci. 24 (4) (1973) 265–269.
[148] P. Li, H. Liu, J.X. Yu, J. He, X. Du, Fast single-pair simrank computation, in: ICDM, SIAM, 2010, pp. 571–582..
[149] Y. Wang, R. Xu, Z. Feng, Y. Che, L. Chen, Q. Luo, R. Mao, Disk: a distributed framework for single-source simrank with accuracy guarantee, Proceedings of the VLDB Endowment 14 (3) (2020) 351–363.
[150] R. Zhu, Z. Zou, J. Li, Simrank computation on uncertain graphs, in: ICDE, IEEE, 2016, pp. 565–576..
[151] W. Xi, E.A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, D. Zhuang, Simfusion: measuring similarity using unified relationship matrix, SIGIR (2005) 130–137.
[152] X. Zeng, Y. Liao, Y. Liu, Q. Zou, Prediction and validation of disease genes using hetesim scores, IEEE/ACM Trans. Comput. Biol. Bioinf. 14 (3) (2016) 687–695.
[153] Y. Wang, X. Lian, L. Chen, Efficient simrank tracking in dynamic graphs, ICDE, IEEE (2018) 545–556.
[154] J. Lu, Z. Gong, X. Lin, A novel and fast simrank algorithm, IEEE TKDE 29 (3) (2016) 572–585.
[155] D. Lizorkin, P. Velikhov, M. Grinev, D. Turdakov, Accuracy estimate and optimization techniques for simrank computation, PVLDB 1 (1) (2008) 422–433.
[156] S. Hong, T. Oguntebi, K. Olukotun, Efficient parallel graph exploration on multi-core cpu and gpu, in: 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT'11), IEEE, 2011, pp. 78–88.
[157] J. Chhugani, N. Satish, C. Kim, J. Sewall, P. Dubey, Fast and efficient graph traversal algorithm for cpus: Maximizing single-node efficiency, in: 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS'12), IEEE, 2012, pp. 378–389.
[158] Y. Li, L.H. U, M.L. Yiu, N.M. Kou, An experimental study on hub labeling based shortest path algorithms, PVLDB 11 (4) (2017) 445–457..
[159] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in ak-tree, SIAM J. Algebraic Discrete Methods 8 (2) (1987) 277–284.
[160] S.B. Seidman, Network structure and minimum degree, Social networks 5 (3) (1983) 269–287.
[161] A. Gibbons, Algorithmic graph theory, Cambridge University Press, 1985.
[162] Q. Liu, Y. Zhu, M. Zhao, X. Huang, J. Xu, Y. Gao, Vac: Vertex-centric attributed community search, in: 36th IEEE International Conference on Data Engineering (ICDE'20), IEEE, 2020, pp. 937–948.
[163] Q. Liu, M. Zhao, X. Huang, J. Xu, Y. Gao, Truss-based community search over large directed graphs, in: Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, ACM, 2020, pp. 2183–2197.
[164] M. Danisch, O. Balalau, M. Sozio, Listing k-cliques in sparse real-world graphs, in: Proceedings of the 2018 World Wide Web Conference on World Wide Web (WWW'18), 2018, pp. 589–598.
[165] D. Chu, F. Zhang, X. LIN, W. Zhang, Y. Zhang, C.Z.Y. Xia, Finding the best k in core decomposition: A time and space optimal solution, in: 36th IEEE International Conference on Data Engineering (ICDE'20), IEEE, 2020, pp. 685–696..
[166] M. Charikar, Greedy approximation algorithms for finding dense components in a graph, in: Approximation Algorithms for Combinatorial Optimization, Third International Workshop (APPROX'00), Springer, 2000, pp. 84–95..
[167] A.V. Goldberg, Finding a maximum density subgraph, University of California Berkeley, 1984.
[168] H.N. Gabow, R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. Syst. Sci. 30 (2) (1985) 209–221.
[169] C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, X. Lin, Efficient progressive minimum k-core search, PVLDB 13 (2019) 362–375.
[170] Q. Li, Y. Zhu, J.X. Yu, Skyline cohesive group queries in large road-social networks, in: 36th IEEE International Conference on Data Engineering (ICDE'20), IEEE, 2020, pp. 397–408.
[171] J. Elzinga, D.W. Hearn, Geometrical solutions for some minimax location problems, Transp. Sci. 6 (4) (1972) 379–394.
[172] J. Wang, J. Cheng, Truss decomposition in massive networks, arXiv preprint arXiv:1205.6693 (2012)..
[173] M.S. Islam, C. Liu, J. Li, Efficient answering of why-not questions in similar graph matching, IEEE Trans. Knowl. Data Eng. 27 (10) (2015) 2672–2686.
[174] M.S. Islam, R. Zhou, C. Liu, On answering why-not questions in reverse skyline queries, in: 29th IEEE International Conference on Data Engineering (ICDE'13), IEEE, 2013, pp. 973–984.
[175] L. Chen, Y. Li, J. Xu, C.S. Jensen, Towards why-not spatial keyword top-k) queries: A direction-aware approach, IEEE Trans. Knowl. Data Eng. 30 (4) (2018) 796–809.

[176] L. Chen, Y. Li, J. Xu, C.S. Jensen, Direction-aware why-not spatial keyword top-k queries, in: 33rd IEEE International Conference on Data Engineering (ICDE'17), IEEE, 2017, pp. 107–110.
[177] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, Q.-S. Hua, Graph processing on gpus: A survey, ACM Computing Surveys (CSUR) 50 (6) (2018).
[178] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Networks Learn. Syst. 32 (1) (2020) 4–24.
[179] Z. Liu, V.W. Zheng, Z. Zhao, Z. Li, H. Yang, M. Wu, J. Ying, Interactive paths embedding for semantic proximity search on heterogeneous graphs, in: Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining, 2018, pp. 1860–1869.
[180] Z. Liu, V.W. Zheng, Z. Zhao, H. Yang, K.C.-C. Chang, M. Wu, J. Ying, Subgraph-augmented path embedding for semantic user search on heterogeneous social network, in: Proceedings of the 2018 World Wide Web Conference, 2018, pp. 1613–1622.
[181] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, W. Wang, Simgnn: A neural network approach to fast graph similarity computation, in: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, 2019, pp. 384–392.
[182] Y. Hao, X. Cao, Y. Sheng, Y. Fang, W. Wang, Ks-gnn: Keywords search over incomplete graphs via graphs neural network, Advances in Neural Information Processing Systems 34 (2021).
[183] J. Gao, J. Chen, Z. Li, J. Zhang, Ics-gnn: lightweight interactive community search via graph neural network, Proceedings of the VLDB Endowment 14 (6) (2021) 1006–1018.
[184] O. Shchur, S. Günnemann, Overlapping community detection with graph neural networks, arXiv preprint arXiv:1909.12201 (2019)..
[185] O. van Rest, S. Hong, J. Kim, X. Meng, H. Chafi, Pgql: a property graph query language, in: Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, 2016, pp. 1–6.
[186] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, Cypher: An evolving query language for property graphs, in: Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 1433–1445.
[187] R. Angles, M. Arenas, P. Barceló, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, et al, G-core: A core for future graph query languages, in: Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 1421–1432.
[188] W. Ali, M. Saleem, B. Yao, A. Hogan, A.-C.N. Ngomo, A survey of rdf stores & sparql engines for querying knowledge graphs, VLDB J. (2021) 1–26.
[189] A. Debrouvier, E. Parodi, M. Perazzo, V. Soliani, A. Vaisman, A model and query language for temporal graph databases, VLDB J. 30 (5) (2021) 825–858.