

# Eficient Betweenness Centrality Computation over Large Heterogeneous Information Networks

Zhejiang Normal University ShanDong University ShanDong Universityy honggao@zjnu.edu.cn xzcheng@sdu.edu.cn dxyu@sdu.edu.cn

Xinrui Wang  $\frac{1}{2}$  Yiran Wang  $\frac{1}{2}$  Xuemin Lin  $\frac{1}{2}$  Jeffrey Xu Yu xrwang@sdu.edu.cn yr\_wang@mail.sdu.edu.cn xuemin.lin@gmail.com yu@se.cuhk.edu.hk

Hong Gao Xiuzhen Cheng Dongxiao Yu<sup>⊠</sup>

ShanDong University ShanDong University Shanghai Jiaotong University The Chinese University of Hong Kong

# ABSTRACT

Betweenness centrality (BC), a classic measure which quantifes the importance of a vertex to act as a communication "bridge" between other vertices in the network, is widely used in many practical applications. With the advent of large heterogeneous information networks (HINs) which contain multiple types of vertices and edges like movie or bibliographic networks, it is essential to study BC computation on HINs. However, existing works about BC mainly focus on homogeneous networks. In this paper, we are the frst to study a specifc type of vertices' BC on HINs, e.g., fnd which vertices with type  $A$  are important bridges to the communication between other vertices also with type  $A$ ? We advocate a meta pathbased BC framework on HINs and formalize both coarse-grained and fne-grained BC (cBC and fBC) measures under the framework. We propose a generalized basic algorithm which can apply to computing not only cBC and fBC but also their variants in more complex cases. We develop several optimization strategies to speed up cBC or fBC computation by network compression and breadth-frst search directed acyclic graph (BFS DAG) sharing. Experiments on several real-world HINs show the signifcance of cBC and fBC, and the efectiveness of our proposed optimization strategies.

# PVLDB Reference Format:

Xinrui Wang, Yiran Wang, Xuemin Lin, Jefrey Xu Yu, Hong Gao, Xiuzhen Cheng, and Dongxiao Yu. Efficient Betweenness Centrality Computation over Large Heterogeneous Information Networks. PVLDB, 17(11): 3360 - 3372, 2024. [doi:10.14778/3681954.3682006](https://doi.org/10.14778/3681954.3682006)

#### PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at [https://github.com/1ran/BccH.](https://github.com/1ran/BccH)

# 1 INTRODUCTION

Betweenness centrality (BC), a fundamental metric in network analytics, measures the importance of each vertex to act as a communication "bridge" between other vertices in the network. Specifically, in a conventional homogeneous network which contains

<span id="page-0-0"></span>

only one type of vertices and one type of edges, the BC of a vertex  $v$  is defined as the fraction of the shortest paths between all pairs of vertices that pass through  $v$  [\[21\]](#page-12-0). That is, a vertex with high BC would pass though more shortest paths. In reality, each shortest path represents one communication way between a pair of vertices. Thus, vertices' BC refect their abilities to control the communication between other vertices in the network[\[21,](#page-12-0) [35\]](#page-12-1). Vertices with high BC can facilitate, impede or bias the transmission of messages in the network[\[21\]](#page-12-0), which are greatly meaningful in practice [\[17,](#page-12-2) [24,](#page-12-3) [28,](#page-12-4) [31,](#page-12-5) [33\]](#page-12-6). For example, in social networks, users with high BC can promote communication between other unfamiliar users[\[31\]](#page-12-5). In power transmission networks, the failures of power grid components with high BC would lead to widespread power outage incidents[\[26\]](#page-12-7). Furthermore, vertices' BC has widely been used for improving the quality of other graph analysis tasks, such as link prediction[\[25\]](#page-12-8) and graph visualisation[\[54\]](#page-12-9).

In reality, many information networks are inherently heterogeneous, containing multiple types of vertices and multiple types of edges which represent diferent semantic relations. Typical heterogeneous information network (HIN) examples are bibliographic networks (e.g., DBLP), movie networks (e.g., IMDb), and biomolecular reaction networks (e.g., KEGG). Fig. [1\(](#page-0-0)a) shows an example of a movie HIN which describes the relationships among diferent types of vertices, i.e, actors (A), movies (M), directors (D) and writers (W). E.g., actors  $a_1$  and  $a_2$  acted in a movie  $m_1$  directed by a director  $d_1$ .

Due to the prevalence of HINs[\[34,](#page-12-10) [55,](#page-12-11) [61,](#page-12-12) [62\]](#page-12-13) and the practicality of vertices' BC[\[16,](#page-12-14) [30,](#page-12-15) [36,](#page-12-16) [39,](#page-12-17) [42,](#page-12-18) [57,](#page-12-19) [60\]](#page-12-20), in this paper, we are interested in computing a specifc type of vertices' BC on HINs, e.g., find which vertices with type  $A$  are important bridges to the communication between other vertices also with type  $A$ . A simple idea to formulate the BC of a vertex  $a_v$  with type A on an HIN

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit<https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org.](mailto:info@vldb.org) Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097. [doi:10.14778/3681954.3682006](https://doi.org/10.14778/3681954.3682006)

(called coarse-grained BC, cBC for short) is to extend the original BC defnition on conventional homogeneous networks to HINs, i.e., the cBC of  $a_v$  is the fraction of the shortest paths between all pairs of vertices with type A that pass through  $a_v$ .

The key of BC computation is counting shortest paths. However, unlike conventional homogeneous networks, two vertices with the same type in an HIN are usually not directly linked, but can be indirectly connected through diferent paths where each path is a sequence of other types of vertices and edges, and carries a unique communication semantics. To distinguish the communication semantics of those paths, a well-known concept of the meta path has been introduced [\[45\]](#page-12-21), which is a sequence of vertex types and edge types. For example, in a movie network, a meta path (AMDMA) represents that two actors acted in movies which were directed by the same director, while a meta path  $(AMWMA)$  represents that two actors acted in movies which were written by the same writer.

Vertices communicated under diferent semantics form completely diferent communication network topologies. E.g., Fig. [1\(](#page-0-0)c) and Fig. [1\(](#page-0-0)d) show different topologies under meta paths  $(AMDMA)$ and (AMWMA) respectively, which leads to completely different BC results (verifed by our experiments in Subsection [6.1\)](#page-9-0). In this paper, while fnding important communication bridges between vertices, we focus on one specifc communication semantics behind a given meta path  $P$ . That is, we only count concrete shortest paths based on  $P$  (called **shortest**  $P$ -paths) to compute cBC.  $P$ can be chosen by users' needs or machine learning. Such a meta path framework has been widely used in HIN mining tasks like vertices' similarity measure[\[45\]](#page-12-21), community search[\[20,](#page-12-22) [53\]](#page-12-23), PageRank computation[\[32\]](#page-12-24), and clustering[\[46\]](#page-12-25).

To show how to count shortest  $P$ -paths and compute cBC, we use an example in Fig. [1](#page-0-0) where there exist no directed links between any two vertices both with type A. Given  $P = (AMDMA)$ in Fig. [1\(](#page-0-0)c), two vertices both with type  $A$  (e.g.,  $a_1$  and  $a_2$ ) form a  $P$ -pair if they are connected by a path instance of  $P$  (e.g.,  $(a_1m_1d_1m_1a_2)$ ). A path connecting  $a_s$  and  $a_t$  (both with type A), which is formed by a series of sequentially concatenated path instances of  $P$ , is called as a  $P$ -path from  $a_s$  to  $a_t$  (e.g., two path instances  $(a_1m_1d_1m_1a_2)$  and  $(a_2m_5d_4m_8a_5)$  are concatenated as a P-path  $(a_1m_1d_1m_1a_2m_5d_4m_8a_5)$  from  $a_1$  to  $a_5$ ). A shortest P-path from  $a_s$  to  $a_t$  is a  $\mathcal P$ -path which contains the smallest number of vertices. From  $a_1$  to  $a_5$ , there are 7 shortest  $P$ -paths, of which three pass through  $a_2$  ( $a_3$  resp.), and one passes through  $a_4$ . Thus, the cBC of  $a_2$  ( $a_3$  resp.) is  $3/7 + 3/7 = 6/7$ , and that of  $a_4$  is  $1/7 + 1/7 = 2/7$ (considering shortest  $P$ -paths from  $a_1$  to  $a_5$  and from  $a_5$  to  $a_1$ ).

Comparing the cBC of  $a_2$  and  $a_3$ , the conclusion is: for the communication between  $a_1$  and  $a_5$ , the "bridge" importance of  $a_2$  and  $a_3$ is equal. However, this would be challenged if we consider shortest  $P$ -paths from a more fine-grained perspective: capture the influence of other types of vertices to the communication between vertices with type  $A$  in the HIN. Specifically, on a shortest  $P$ -path, there exists not a direct link but a path instance of  $P$  between each  $P$ -pair. After that, among the  $P$ -pairs of several shortest  $P$ -paths, the path instances often pass through the same vertex whose type is not  $A$ so that they interfere with one another. For example, in Fig. [1\(](#page-0-0)c), all path instances between the  $P$ -pair ( $a_3$ ,  $a_5$ ) pass through the same vertex  $d_7$ . As a result, once  $d_7$  is removed, those path instances would all be broken, then  $a_3$  would fail to communicate with  $a_5$ . In

contrast, each of the path instances between  $(a_2, a_5)$  respectively passes through  $d_4$ ,  $d_5$  or  $d_6$ . Once one of  $d_4$ ,  $d_5$  or  $d_6$  is removed, only one of those path instances would be broken, and  $a_2$  could still communicate with  $a_5$ . It indicates that the communication between  $(a_2, a_5)$  is more robust than that between  $(a_3, a_5)$ , because the path instances between  $(a_2, a_5)$  are more independent from one another and not easy to be broken by some vertex with type  $D$ . Similarly, the communication between  $a_1$  and  $a_5$  along the shortest P-paths which contain  $(a_2, a_5)$  is also more robust than that along the shortest  $P$ -paths which contain  $(a_3, a_5)$ . In other words, as for the communication between  $a_1$  and  $a_5$ , the shortest  $P$ -paths which contain  $(a_2, a_5)$  are more important than the shortest  $\mathcal{P}$ paths which contain  $(a_3, a_5)$ . Thus, the "bridge" importance of  $a_2$ to the communication between  $a_1$  and  $a_5$  is greater than that of  $a_3$ , since  $a_2$  lies on more important shortest  $P$ -paths.

To fill the gap of cBC which ignores that shortest  $P$ -paths would have diferent importance to the communication between a pair of vertices due to the infuence of other types of vertices on HINs, we propose a novel definition of the *fine-grained BC* (*fBC* for short). The main idea of formulating fBC is: firstly, give each shortest  $P$ path a weight so that a shortest  $P$ -path from  $a_s$  to  $a_t$  would have a larger weight if along this shortest  $P$ -path, the communication from  $a_s$  to  $a_t$  is more robust; secondly, the fBC of  $a_v$  with type A is the fraction of the sum of weights of the shortest  $P$ -paths between all pairs of vertices with type A that pass through  $a_v$ . That is, a vertex with higher fBC would pass through more shortest  $P$ -paths with larger weights. Actually, cBC is a special case of fBC which considers that the weight of each shortest  $P$ -path is 1.

Applications. The cBC and fBC on HINs can be widely used in many applications. Here are two typical examples. (1) In a movie network, by the meta path  $(AMDMA)$ , the vertices with type  $A$ that have high cBC or fBC are famous actors who play important "bridge" roles in promoting cooperation between other unfamiliar actors with the help of directors. Moreover, if the fBC ranking of an important "bridge" actor (e.g.,  $a_x$ ) is significantly higher than the cBC ranking, more diferent directors would involve in promoting other actors' cooperation, i.e., there contain more diferent directors in the cooperation network between actors which  $a_x$  promotes. From our experimental results, the actors with higher fBC rankings are more likely to appear in numerous flms spanning various genres which are directed by many different directors. (2) In a biomolecular reaction network of Glycolysis/ Gluconeogenesis, by a meta path  $(CGC)$  representing that two compounds  $(C)$  react under the catalysis of a gene product (G), the vertices with type  $C$  that have high cBC or fBC (e.g., Phosphoenolpyruvate) are key intermediate compounds involved in many metabolic pathways of Glycolysis/Gluconeogenesis. Moreover, if a compound  $c<sub>u</sub>$  has higher fBC ranking than cBC ranking, on the metabolic pathways which  $c_y$  participates, each gene product often catalyzes only one reaction between two compounds rather than catalyze multiple reactions between several compounds. The advantage is that if a gene product occurred abnormal changes by gene mutations, most of metabolic pathways which  $c<sub>y</sub>$  participates would not be affected. Challenges and Contributions. As far as we known, we are the frst to focus on a specifc type of vertices' BC on HINs. Considering both coarse-grained and fne-grained perspectives, we propose a

meta path-based BC framework on HINs. To compute BC on HINs, it needs three steps: (1) find all shortest  $P$ -paths; (2) weight each shortest  $P$ -path; (3) find each  $a_v$  lying on which shortest  $P$ -paths to compute its BC.

Firstly, while finding all shortest  $P$ -paths, each path instance would be visited many times. To reduce the time of visiting each path instance, we project the HIN  $\mathbb G$  to a  $\mathcal P$ -multigraph  $G_{\mathcal P}$  so that each path instance between a  $P$ -pair  $(a_v, a_u)$  on G corresponds to an edge between those two vertices on  $G_{\mathcal{P}}$ . Furthermore, computing BC for all vertices with type  $A$  on  $\mathbb G$  is equivalent to computing BC for all vertices on  $G_{\mathcal{P}}$ .

Secondly, we give two elegant properties for the weight of the shortest  $P$ -path. After that, we formulate cBC and fBC measures under our meta path-based BC framework respectively from coarsegrained and fne-grained perspectives.

Thirdly, we propose a generalized basic algorithm which can compute not only cBC and fBC but also their variants in more complex cases (discussed in Section [5\)](#page-8-0). Let  $n\varphi$  be the number of vertices on  $G_{\mathcal{P}}$ , and  $\overline{m}_{\mathcal{P}}$  be the number of vertex pairs which have at least one edge between them on  $G_{\mathcal{P}}$ . The basic algorithm integrates the three steps into performing BFS and reverse BFS on totally  $n_p$ breadth-frst search directed acyclic graphs (BFS DAGs, e.g., Fig. [3\)](#page-5-0) on  $G_{\mathcal{P}}$ . There are two main parameters which affect the total time of the basic algorithm: firstly,  $n\varphi$ ; secondly, the worst time cost of performing BFS and reverse BFS on a BFS DAG, i.e.,  $O(\overline{m}_{\varphi})$ . Thus, to further speed up BC computation but not cause loss to BC results, we develop the network compression strategy to reduce  $n \rho$ and  $\overline{m}_{\mathcal{P}}$ , and the BFS DAG sharing strategy to further reduce  $n_{\mathcal{P}}$ .

Extensive experiments have been conducted on several realworld HINs to verify the great signifcance of cBC and fBC, and show the great efectiveness of our proposed optimization strategies. Related Work. BC is frst proposed by Anthonisse and Freeman[\[7,](#page-12-26) [21\]](#page-12-0). Brandes[\[12\]](#page-12-27) gives the best known BC computation algorithm with  $O(|V||E|)$  ( $O(|V||E| + |V|^2 log|V|)$  resp.) time for unweighted (weighted resp.) graphs. Besides, lots of eforts have been made to further accelerate BC computation[\[43,](#page-12-28) [47\]](#page-12-29). Moreover, as computing exact BC values is time consuming, many researchers have focused on estimating approximate BC values [\[16,](#page-12-14) [18,](#page-12-30) [56\]](#page-12-31). In addition, BC on diferent types of graphs like hypergraphs [\[44\]](#page-12-32), bipartite graphs [\[19\]](#page-12-33), valued graph[\[13\]](#page-12-34), and dynamic graphs [\[22,](#page-12-35) [29,](#page-12-36) [31\]](#page-12-5) has been extensively studied. Some researchers have also given BC variants like edge BC, distance-scaled BC, and group BC[\[8,](#page-12-37) [13\]](#page-12-34). Whereas, none of the above works focus on a specifc type of vertices' BC on HINs. The key of BC computation is counting shortest paths. The problem of counting shortest paths between  $s$  and  $t$  is  $\#P$ -complete [\[52\]](#page-12-38). Many studies have reduced the time of counting shortest paths on simple graphs [\[37,](#page-12-39) [38,](#page-12-40) [59\]](#page-12-41), dynamic graph[\[50\]](#page-12-42), probabilistic graphs[\[40\]](#page-12-43), planar graphs[\[10\]](#page-12-44) and labeled graphs[\[9,](#page-12-45) [11,](#page-12-46) [41,](#page-12-47) [58\]](#page-12-48). However, they don't consider how to reduce the time of fnding each vertex lying on which shortest paths (an essential but pretty costly step in BC computation). Thus, the existing work for counting shortest paths can't apply to speed up cBC or fBC computation. Various metrics have been proposed to rank vertices based on their importance in a network. Diferent metrics have their own unique meaningness and cannot be replaced by each other. For example, PageRank[\[14\]](#page-12-49) focuses on importance vertices that are linked by many other important vertices, infuence maximization[\[27\]](#page-12-50) fnds

a seed set of vertices to maximize the spread of infuence in a network, structural diversity[\[51\]](#page-12-51) measures the number of connected components (each represents a distinct social context) in vertices' neighbourhood, while cBC and fBC cares about the importance of vertices to act as bridges along shortest paths between other vertices. Our experiments compare the top ranked vertices as for PathRank[\[32\]](#page-12-24) (extended PageRank on HINs), infuence spread[\[15\]](#page-12-52), structural diversity[\[23\]](#page-12-53), cBC and fBC, the results verify that those metrics are quite diferent, and cBC and fBC are indispensable.

# <span id="page-2-0"></span>2 PROBLEM DESCRIPTION

An HIN[\[45\]](#page-12-21), denoted as  $\mathbb{G} = (V, E, \mathcal{A}, \mathcal{R}, \phi_V, \phi_E)$  ( $|\mathcal{A}| > 1$  or  $|\mathcal{R}| >$ 1), is an undirected graph with a vertex type mapping function  $\phi_V$ :  $V \to \mathcal{A}$  and an edge type mapping function  $\phi_E : E \to \mathcal{R}$ , where each vertex  $v \in V$  belongs to one particular vertex type  $\phi_V(v) \in \mathcal{A}$ , and each edge  $e \in E$  belongs to one particular edge type  $\phi_E(e) \in \mathcal{R}$ . The **schema**[\[45\]](#page-12-21) of an HIN G is an undirected graph  $T_{\mathbb{G}} = (\mathcal{A}, \mathcal{R})$ defined over the vertex types  $\mathcal A$  and edge types  $\mathcal R$  of  $\mathbb G$ . The schema shows all allowable edge types between vertex types on G. If there is an edge type  $R$  from a vertex type  $A$  to a vertex type  $B$ , denoted as ARB, then the inverse edge type  $R^{-1}$  naturally exists from B to A, denoted as  $BR^{-1}A$ . A **meta path**[\[45\]](#page-12-21)  $P$  is a path defined on  $T_{\mathbb{G}} = (\mathcal{A}, \mathcal{R})$ , and is denoted as  $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_l} A_{l+1}$ , where  $l = |\mathcal{P}|$  is the length of  $\mathcal{P}, A_i \in \mathcal{A}$   $(1 \le i \le l+1)$ , and  $R_j \in \mathcal{R}$  $(1 \le j \le l)$ . Given  $\mathcal{P} = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_l} A_{l+1}$ , we call a path  $\mathit{pins} = (v_1, v_2, \ldots, v_{l+1})$  as a **path instance** of  $\overline{P}$ , if  $\forall i$ , vertex  $\overline{v_i}$  and edge  $e_i = (v_i, v_{i+1})$  satisfy  $\phi_V(v_i) = A_i$  and  $\phi_E(e_i) = R_i$ . If two vertices  $a_u$  and  $a_v$  can be connected by a path instance of  $\mathcal{P}$ , we call  $(a_u, a_v)$  as a P-pair (or  $a_u$  and  $a_v$  are P-neighbors). For simplicity, we also use vertex type names to denote a meta path (i.e.,  $\mathcal{P} = (A_1 A_2 \dots A_{(l+1)/2} \dots A_l A_{l+1})$ , if there exist no multiple edge types between the same pair of vertex types. If  $P$  is symmetric about  $A_{(l+1)/2}$ , we say  $P$  is symmetric and  $A_{(l+1)/2}$  is the symmetry point type. In this paper, we aim to find important bridges to the communication between the vertices which all have the same target type (supposing  $A$ ). To connect two vertices both with type A, both the starting and ending vertex types of  $P$  should be A, i.e.,  $A_1 = A_{l+1} = A$ . Thus, in the rest of this paper, the mentioned meta paths all satisfy the form  $P = (A, \ldots, A)$ . For ease of presentation, we frstly adopt symmetric meta paths in our models and algorithms, then show how to expand to unsymmetric meta paths in Section [5.](#page-8-0) Meta Path-based BC Framework on HINs. Two path instances of P, pins<sub>1</sub> = ( $v_1, v_2, ..., v_{l+1}$ ) and pins<sub>1</sub> = ( $v'_1, v'_2, ..., v'_{l+1}$ ), are concatenable iff  $v_{l+1} = v'_1$ . The concatenated path of pins<sub>1</sub> and *pins*<sub>2</sub> is written as  $p = (pins_1 \circ pins_2)$ . For two vertices  $a_s$  and  $a_t$ both with type A on an HIN  $\mathbb G$ , if a series of path instances of  $\mathcal P$  (i.e.,  $pins_1 = (a_s = a_{x_1},..., a_{x_2}), \text{ } pins_2 = (a_{x_2},..., a_{x_3}),..., \text{ } pins_{n-1} =$  $(a_{x_{n-1}},..., a_{x_n} = a_t)$  can be sequentially concatenated as a path  $p = (pins_1 \circ pins_2 \circ \cdots \circ pins_{n-1}),$  we call p as a P-path from  $a_s$  to  $a_t$ . For any  $i \in [1, n-1]$ ,  $(a_{x_i}, a_{x_{i+1}})$  forms a  $\mathcal P$ -pair. A shortest  $\mathcal P$ -path from  $a_s$  to  $a_t$  is a P-path from  $a_s$  to  $a_t$  which contains the smallest number of vertices. In Fig. [1\(](#page-0-0)c),  $p^1 = (a_1, m_1, d_1, m_1, a_2, m_5, d_4, m_8, a_5)$ is a shortest  $P$ -path from  $a_1$  to  $a_5$ .

DEFINITION 1. (Weight of Shortest  $P$ -Path) Given a shortest  $P$ path  $p^i$  from  $a_s$  to  $a_t$ , the weight of  $p^i$  denoted by  $\beta_{a_s a_t}$   $[p^i]$ , measures that along  $p^i$ , how robust the communication from  $a_s$  to  $a_t$  is.

Table 1: Frequently Used Notions.

<b>Notion</b>	Meaning
$C_p^{\mathcal{P}}(a_n)$	meta path-based BC of $a_n$ on an HIN G
$\beta^{\mathcal{P}}_{a_{s}a_{t}}(\text{or }\beta^{\mathcal{P}}_{a_{s}a_{t}}(a_{v}))$ $\beta^{\mathcal{P}}_{a_{s}a_{t}}[a_{x},a_{x+1}]$	sum of weights of all shortest $P$ -paths from $a_s$ to $a_t$ (which $a_n$ lies on)
	the weight of a $\mathcal{P}$ -pair $(a_x, a_{x+1})$
$\Gamma_{a_{\nu},a_{\nu+1}}$	the set of path instances between a $\mathcal{P}$ -pair $(a_x, a_{x+1})$
	number of path instances that interfere pins while $a_s$ communicating to other vertices
$\frac{ I_{a_s}^d(pins) }{ D_{a_x,a_{x+1}}^p }$	number of all vertices with type D which the path instances in $\Gamma_{a_1,a_{n+1}}$ pass through
$G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$	the $P$ -multigraph built from an HIN G based on a meta path $P$
$F_{\mathcal{P}}[i, j] (= M_{\mathcal{P}}[i, j])$	number of path instances of $P$ between $(a_i, a_j)$ on $\mathbb G$ (edges between $(a_i, a_j)$ on $G_P$ )
$\delta_{a_{n}}(a_{n})$	the source dependency of $a_s$ on $G_p$
$Pred_{a_0}(a_n)$	all predecessors of a vertex $a_v$ on the BFS DAG of $a_s$ on $G_p$
$EI(a_u, a_v)$	packaged information of a $P$ -pair $(a_u, a_v)$ for computing $\beta_{a-a}^{G_p}$ [ $a_u, a_v$ ]
$l(\mathcal{P}) = (A_1 A_2 \dots A_{(l+1)/2})$	the left half meta path of $\mathcal{P} = (A_1 A_2 \dots A_{(l+1)/2} \dots A_l A_{l+1})$

Let  $\Psi_{a_{s}a_{t}}$  be the set of all shortest  $\mathcal{P}$ -paths from  $a_{s}$  to  $a_{t}, \beta_{a_{s}a_{t}}^{\mathcal{P}}$  be the sum of weights of all shortest  $P$ -paths in  $\Psi_{a_{s}a_{t}}$  which measures along those shortest  $P$ -paths, how robust the communication from  $a_s$  to  $a_t$  is. Let  $\beta_{a_s a_t}^{\mathcal{P}} [a_x, a_{x+1}]$  be the weight of a  $\mathcal{P}$ -pair  $(a_x, a_{x+1})$ which measures along all path instances between  $(a_x, a_{x+1})$ , how robust the communication from  $a_s$  to  $a_t$  is.

Assuming there is at least one shortest  $P$ -path from  $a_s$  to  $a_t$ . If  $a_s$  and  $a_t$  form a  $\mathcal{P}$ -pair,  $\beta_{a_s a_t}^{\mathcal{P}} = \beta_{a_s a_t}^{\mathcal{P}} [a_s, a_t]$ . Otherwise (i.e., connecting  $a_s$  and  $a_t$  needs at least two sequentially concatenated path instances of  $P$ ), the weight of the shortest  $P$ -path has the following two elegant properties.

**Properties 1. (Additivity)** If all shortest  $P$ -paths in  $\Psi_{a_s a_t}$  can be divided into r groups ( $r \geq 1$ ) so that each shortest P-path is only in a unique group  $pg^j$ , let  $\beta_{a_s a_t} [pg^j]$  be the sum of weights of the group of shortest  $P$ -paths in  $pg<sup>j</sup>$  which measures along this group of shortest  $P$ -paths, how robust the communication from  $a_s$  to  $a_t$ is. We have  $\beta^{\hat{\mathcal{P}}}_{a_s a_t} = \sum_{p^i \in \Psi_{a_s a_t}} \beta_{a_s a_t} [p^i] = \sum_{j=1}^r \beta_{a_s a_t} [pq^j].$ Properties 2. (Multiplicativity) Given a group of shortest  $P$ -paths  $pq^{j}$  from  $a_s$  to  $a_t$ , if all  $p$ -pairs on each shortest  $p$ -path in  $pq^{j}$  are

identical (assuming they are  $(a_s = a_1, a_2), (a_2, a_3), ..., (a_{k_i}, a_{k_i+1})$ =  $a_t$ ) sequentially), we say  $pg^j$  contains  $k_j$   $\mathcal P$ -pairs where each P-pair is  $(a_x, a_{x+1})$  ( $1 \le x \le k_j$ ). Since each P-pair in pg<sup>j</sup> independently affects the communication robustness from  $a_s$  to  $a_t$ , we have  $\beta_{a_s a_t} [pg^j] = \prod_{x=1}^{k_j} \beta_{a_s a_t}^p [a_x, a_{x+1}].$ 

We observe that all shortest P-paths from  $a_s$  to  $a_t$  always can be divided into r groups ( $r \geq 1$ ) so that (1) each shortest  $P$ -path is only in a unique group; (2) in any a group, all  $P$ -pairs on each shortest  $P$ -path are identical. For example, in Fig. [1\(](#page-0-0)c), there are three groups of shortest  $P$ -paths from  $a_1$  to  $a_5$ :  $pg_{a_1a_5}^1 = (a_1, a_2, a_5)$  (including three shortest  $\mathcal{P}$ -paths),  $pg_{a_1a_5}^2 = (a_1, a_3, a_5)$  (including three shortest P-paths),  $pq_{a_1a_5}^3 = (a_1, a_4, a_5)$  (including one shortest P-path), where each  $p_{a_{s}a_{t}}^{j}$  is represented by the sequence of all its vertices with type A from  $a_s$  to  $a_t$ . Then  $\beta_{a_1 a_5}^{\mathcal{P}} = \sum_{j=1}^3 \beta_{a_1 a_5} [pg^j]$ . And  $pg_{a_1a_5}^1$  contains two  $\mathcal{P}$ -pairs  $(a_1, a_2)$  and  $(a_2, a_5)$ , then  $\beta_{a_1a_5}[pg_1] =$  $\beta_{a_1 a_5}^{\mathcal{P}}[a_1, a_2] \times \beta_{a_1 a_5}^{\mathcal{P}}[a_2, a_5].$ 

For ease of calculation, to compute  $\beta_{a_{s}a_{t}}^{\mathcal{P}}$  (except that  $a_{s}$  and  $a_{t}$ form a  $P$ -pair), instead of computing the weight of each shortest  $\mathcal{P}$ -path in  $\Psi_{a_{s}a_{t}}$  and then summing all the weights, we firstly divide  $\Psi_{a_{s}a_{t}}$  into groups based on the above observation, and then by the above two properties, we have  $\beta_{a_s a_t}^{\mathcal{P}} = \sum_{j=1}^r (\prod_{x=1}^{l_j} \beta_{a_s a_t}^{\mathcal{P}} [a_x, a_{x+1}]).$ 

Based on the above, the base of computing  $\beta^{\mathcal{P}}_{a_{s}a_{t}}$  is the formal definition of  $\beta_{a_{s}a_{t}}^{\mathcal{P}}[a_{x}, a_{x+1}]$ . Next, we firstly give a meta pathbased BC framework on HINs. Then we will respectively propose the coarse-grained and fne-grained formal defnitions of

<span id="page-3-1"></span>

Figure 2: Example of Computing Fine-grained  $\beta^{\mathcal{P}}_{a_{\text{s}},a_{t}}[a_{\text{x}},a_{\text{x+1}}].$ 

 $\beta_{a_{s}a_{t}}^{\mathcal{P}}[a_{x},a_{x+1}]$  so as to introduce the  $\boldsymbol{0}$  coarse-grained BC (cBC for short) and the *fine-grained BC* (*fBC* for short) on HINs.

DEFINITION 2. (Meta Path-based BC Framework on HINs) Given an HIN  $\mathbb{G} = (V, E, \mathcal{A}, \mathcal{R}, \phi_V, \phi_E)$  and a meta path  $\mathcal{P}$  (where  $A_1 = A_{l+1} = A$ , the meta path-based BC framework of a vertex  $a_v$ with type A (i.e.,  $\phi_V(a_v) = A$ ) is defined as follows:

<span id="page-3-0"></span>
$$
C_B^{\mathcal{P}}(a_v) = \sum_{a_s \neq a_0 \neq a_t \in V, \phi_V(a_s) = \phi_V(a_v) = \phi_V(a_t) = A}^{\mathcal{P}_{a_s a_t}^{\mathcal{P}}(a_v)} \qquad (1)
$$

where  $\beta_{a_{s}a_{t}}^{\mathcal{P}}$  is the sum of weights of the shortest  $\mathcal{P}$ -paths from  $a_{s}$  to  $a_t, \beta^{\mathcal{P}}_{a_s a_t}(a_v)$  is the sum of weights of the shortest  $\mathcal{P}$ -paths from  $a_s$ to  $a_t$  which  $a_v$  lies on. By convention,  $\beta_{a_s a_s}^{\mathcal{P}} = 1$ .

Coarse-grained vs Fine-grained BC on HINs. Let  $\Gamma_{a_x, a_{x+1}}$  be the set of path instances between a  $\mathcal{P}\text{-pair }(a_x, a_{x+1})$ . Recall that  $\beta^{\mathcal{P}}_{a_{s}a_{t}}[a_{x}, a_{x+1}]$  measures along all path instances between  $(a_{x}, a_{x+1})$ , how robust the communication from  $a_s$  to  $a_t$ .

From a coarse-grained perspective, the infuence of the vertices whose type are not A to the communication from  $a_s$  to  $a_t$  is ignored. If there are more path instances between a  $\mathcal{P}$ -pair  $(a_x, a_{x+1})$ (i.e., more path instances in  $\Gamma_{a_x,a_{x+1}}$ ), then the number of shortest P-paths from  $a_s$  to  $a_t$  which contains  $(a_x, a_{x+1})$  is larger, so the communication from  $a_s$  to  $a_t$  is more robust, i.e.,  $\beta_{a_s a_t}^{\mathcal{P}}[a_x, a_{x+1}] \propto$  $|\Gamma_{a_x,a_{x+1}}|$ . We define coarse-grained  $\beta^{\mathcal{P}}_{a_s a_t}[a_x,a_{x+1}] = |\Gamma_{a_x,a_{x+1}}|$ . In such a case, we call the BC in Eq.  $(1)$  as the cBC. In Fig. [2\(](#page-3-1)a), coarse-grained  $\beta_{a_{u_1}a_v}^{\mathcal{P}}[a_{u_1}, a_v] = \beta_{a_s a_v}^{\mathcal{P}}[a_{u_1}, a_v] = |\Gamma_{a_{u_1}a_v}| = 3.$ 

From a fine-grained perspective, the influence of the vertices whose type are not A to the communication from  $a_s$  to  $a_t$  is considered. The infuence is greater if more path instances pass through the same vertex whose type is not  $A$ , it often occurs when this vertex type is the symmetry point type (supposing  $D$ ) of  $\mathcal{P}$ .

For ease of presentation, we frstly only consider the infuence of the vertices with type  $D$  (more complex situations are discussed in Section [5\)](#page-8-0), then the communication from  $a_s$  to  $a_t$  based on the path instances between a  $\mathcal P\text{-pair }(a_x, a_{x+1})$  is more robust if: i) More path instances in  $\Gamma_{a_x,a_{x+1}}$ , i.e.,  $\beta_{a_sa_t}^{\mathcal{P}}[a_x,a_{x+1}] \propto |\Gamma_{a_x,a_{x+1}}|$ . ii) Each path instance *pins*  $\in \Gamma_{a_x, a_{x+1}}$  is more independent and not easy to be interfered by other path instances while  $\mathit{pins}$  is participating in the communication from  $a_s$  to  $a_t$ . When  $a_s$  communicates to  $a_t$  along a shortest P-path  $p^i$ , each pins contained in  $p^i$  is interfered by the following path instances (Let  $I_{a_s}^d(pins)$  be the set of all such path instances, including  $pins$ ) : such path instances are contained in other shortest  $P$ -paths whose sources are all  $a_s$ , and such path instances and  $p$ ins pass through the same vertex  $d$ with type  $D$ . For example, in Fig. [2\(](#page-3-1)a), when the communication starts from  $a_{u_1}$  ( $a_s$  resp.) to other vertices,  $pins_1$  would be interfered by two path instances  $(a_{u_1}, m_6, d_3, m_5, a_{u_2})$  and  $(a_{u_1}, m_6, d_3, m_6, a_v)$ (three path instances  $(a_{u_1}, m_6, d_3, m_6, a_v)$ ,  $(a_{u_2}, m_5, d_3, m_5, a_v)$  and

 $(a_{u_2}, m_5, d_3, m_6, a_v)$  resp.), marked in green in Fig. [2\(](#page-3-1)b) (Fig. 2(c) resp.). Thus,  $|I_{a_{u_1}}^{d_3}(pin_1)| = 3$ ,  $|I_{a_3}^{d_3}(pin_1)| = 4$ . W.l.o.g., assume that the interference of the path instances in  $I_{a_s}^d(pins)$  on each other is equivalent, so the independence of each  $\emph{pins}$  is denoted by  $1/|I_{a_s}^d(pins)|$ . Then we have  $\beta_{a_s a_t}^{\hat{p}}[a_x, a_{x+1}] \propto 1/|I_{a_s}^d(pins)|$ .

iii) The path instances in  $\Gamma_{a_x, a_{x+1}}$  pass through more different vertices with type D (let  $|D_{a_x,a_{x+1}}^{\mathcal{P}}|$  be the number of those vertices with type D), i.e.,  $\beta_{a_s a_t}^{\mathcal{P}} [a_x, a_{x+1}] \propto |D_{a_x, a_{x+1}}^{\mathcal{P}}|$ .

Let  $|V_D|$  be the number of all vertices with type D on G, then  $\sum_{\text{pins} \in \Gamma_{a_x, a_{x+1}}} 1/|I_{a_s}^d(\text{pins})| \in (0, |V_D|],$  and  $|D_{a_x, a_{x+1}}^{\phi} | \in [1, |V_D|].$ Thus, we define fine-grained  $\beta_{a_{s}a_{t}}^{\mathcal{P}}[a_{x}, a_{x+1}]$  as follows.

<span id="page-4-0"></span>
$$
\beta^{\mathcal{P}}_{a_{s}a_{t}}[a_{x}, a_{x+1}] = \Big(\sum_{\text{pins} \in \Gamma_{a_{x}, a_{x+1}}} \frac{1}{|I_{a_{s}}^{d}(\text{pins})|}\Big) + |D^{\mathcal{P}}_{a_{x}, a_{x+1}}| \qquad (2)
$$

In such a case, we call the BC in Eq. [\(1\)](#page-3-0) as the fBC. Note that the definition of  $\beta_{a_{s}a_{t}}^{\mathcal{P}}[a_{x}, a_{x+1}]$  in Eq. [\(2\)](#page-4-0) is a straightforward form based on intuitive observations. In the future, we will continue to analyze whether there are other factors affecting  $\beta^{\mathcal{P}}_{a_{s}a_{t}}[a_{x},a_{x+1}]$ and improve Eq. [\(2\)](#page-4-0) accordingly. Even though the form of Eq. [\(2\)](#page-4-0) is changed, our proposed algorithms and optimization strategies could still be applicable only with simple modifcations. For cBC,  $\beta_{a_8a_1}^{\mathcal{P}}[a_x, a_{x+1}] = \beta_{a_1a_8}^{\mathcal{P}}[a_x, a_{x+1}],$  so  $\beta_{a_8a_1}^{\mathcal{P}} = \beta_{a_1a_8}^{\mathcal{P}}(a_s \neq a_1)$ ; for fBC,  $\beta_{a_{s}a_{t}}^{\mathcal{P}}[a_{x}, a_{x+1}] \neq \beta_{a_{t}a_{s}}^{\mathcal{P}}[a_{x}, a_{x+1}],$  so  $\beta_{a_{s}a_{t}}^{\mathcal{P}} \neq \beta_{a_{t}a_{s}}^{\mathcal{P}}(a_{s} \neq a_{t}).$ Problem 1 (Meta Path-based BC Computation, MBCC for **short).** Given an HIN G, a meta path  $P$  (where  $A_1 = A_{l+1} = A$ ), return the cBC or fBC for all vertices with type A.

## 3 MBCC ALGORITHM

To solve the MBCC problem, the frst key step is to fnd all shortest  $P$ -paths. However, each path instance of  $P$  is often contained in many shortest  $P$ -paths. Thus, while finding all shortest  $P$ -paths in an HIN G, each path instance will be visited many times. To reduce the time of visiting each path instance from  $O(|P|)$  to  $O(1)$ , we build a P-multigraph  $G_{\mathcal{P}}$  from  $\mathbb G$  in advance so that each path instance between a  $P$ -pair  $(a_v, a_u)$  in  $\mathbb G$  corresponds to an edge between those two vertices on  $G_{\mathcal{P}}$ . After that, finding all shortest P-paths on G is equivalent to finding all shortest paths on  $G_{\mathcal{P}}$ . Furthermore, computing cBC or fBC for all vertices with type  $A$  on G is equivalent to computing cBC or fBC for all vertices on  $G_{\mathcal{P}}$ .

Our **Basic** algorithm for MBCC has two steps: (1) build the  $P$ multigraph  $G_{\mathcal{P}}$  from the HIN G; (2) compute cBC or fBC for all vertices on  $G_{\mathcal{P}}$ .

DEFINITION 3. (P-multigraph) Given an HIN  $G$  and a meta path P, the P-multigraph  $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$  is a multigraph (where every two vertices may be connected by more than one edge) such that the vertices in  $V_P$  are all vertices with type A on  $\mathbb G$ , and each edge in E<sub>P</sub> between two vertices  $a_v$  and  $a_u$  on G<sub>P</sub> corresponds to a path instance between a P-pair  $(a_v, a_u)$  on G.

Building a  $P$ -Multigraph from an HIN. The key to build a  $P$ multigraph  $G_{\mathcal{P}}$  from an HIN  $\mathbb G$  is to compute the number of path instances of  $P$  between each  $P$ -pair on G. The step (1) of Basic handles this with the commuting matrix [\[45\]](#page-12-21).

DEFINITION 4. (Commuting Matrix[\[45\]](#page-12-21)) Given an HIN G and a meta path  $P = (A_1 A_2 ... A_{l+1})$ , the commuting matrix  $F_{\mathcal{P}}$  for  $\mathcal P$  is defined as  $F_{\mathcal P} = W_{A_1A_2}W_{A_2A_3}\ldots W_{A_lA_{l+1}}$ , where  $W_{A_iA_j}$  is the adjacency matrix between vertex type  $A_i$  and  $A_j$ .

To build  $G_{\mathcal{P}}$  from G, we firstly compute the commuting matrix  $F_{\mathcal{P}}$  for  $\mathcal{P}$  on  $\mathbb{G}$ .  $F_{\mathcal{P}}$  is just the adjacency matrix  $M_{\mathcal{P}}$  of  $G_{\mathcal{P}}$ , since  $F_{\mathcal{P}}[i, j] = M_{\mathcal{P}}[i, j]$  represents the number of path instances of  $\mathcal{P}$ between  $a_i$  and  $a_j$  on  $\mathbb G$  (i.e., the number of edges between  $a_i$  and *a<sub>j</sub>* on  $G_{\mathcal{P}}$ ). Since  $\mathcal{P}$  is symmetric,  $F_{\mathcal{P}} = F_{l(\mathcal{P})} F_{l(\mathcal{P})}^T$  where  $F_{l(\mathcal{P})}$  is the commuting matrix for  $l(P)$  (i.e., the left half meta path of  $P$ ).

After building  $G_{\mathcal{P}}$ , let  $Nei_{G_{\mathcal{P}}}(a_v)$  be the set of neighbors of  $a_v$ on  $G_{\mathcal{P}}$ , then  $a_v$  and each of those vertices in  $Nei_{G_{\mathcal{P}}}(a_v)$  form a P-pair on the HIN G. A path from  $a_s$  to  $a_t$  on  $G_p$ , denoted by  $a_s \rightarrow a_t$ , is a sequence of vertices such that each two adjacent vertices are linked by at least one edge on  $G_{\mathcal{P}}$ . A shortest  $a_s \leadsto a_t$ path is the  $a_s \rightarrow a_t$  path which contains the smallest number of vertices on  $G_{\mathcal{P}}$ . Obviously, each shortest  $\mathcal{P}$ -path from  $a_s$  to  $a_t$  on G corresponds to a shortest  $a_s \leadsto a_t$  path on  $G_p$ . Thus,  $\beta_{a_3a_t}^{\mathcal{P}} = \beta_{a_3a_t}^{G_{\mathcal{P}}} \beta_{a_3a_t}^{\mathcal{P}}(a_v) = \beta_{a_3a_t}^{G_{\mathcal{P}}}(a_v)$ , and  $C_B^{\mathcal{P}}(a_v) = C_B^{G_{\mathcal{P}}}(a_v)$ . That is, computing cBC or fBC for all vertices with type A on  $\mathbb G$  is equivalent to computing cBC or fBC for all vertices on  $G_{\mathcal{P}}$ .

Computing cBC or fBC on a  $P$ -Multigraph. Let  $\delta_{a_s a_t}(a_v)$  =  $\overline{\beta_{a_{s}a_{t}}^{G_{\mathcal{P}}}(a_{v})/\beta_{a_{s}a_{t}}^{G_{\mathcal{P}}}$  be the **pair dependency** of  $a_{s}$  and  $a_{t}$  on  $G_{\mathcal{P}},$  then Eq. [\(1\)](#page-3-0) is equivalently written as  $C_B^{G_p}(a_v) = \sum_{a_s \neq a_v \neq a_t \in V_p} \delta_{a_s a_t}(a_v)$ . To compute cBC or fBC for vertices on  $G_{\mathcal{P}}$ , the key is to compute  $\delta_{a_s a_t}(a_v)$  for all  $a_s \in V_{\mathcal{P}}$ ,  $a_t \in V_{\mathcal{P}}$  and  $a_v \in V_{\mathcal{P}}$ . A simple method that totally needs  $O(n_a^3)$  $\mathcal{P}_{\mathcal{P}}$ ) time is: (1) find all shortest paths on  $G_{\mathcal{P}}$ , (2) find which shortest paths pass through  $a_v$  for each  $a_v \in V_{\mathcal{P}}$ .

To reduce the time, we compute all pair dependencies in groups where each group of pair dependencies have the same  $a_s$ : let  $\delta_{a_s \bullet}(a_v)$  $=\sum_{a_t \in V} \delta_{a_s a_t}(a_v)$  be the **source dependency** of  $a_s$  on  $G_p$ . Then Eq. [\(1\)](#page-3-0) is equivalently written as  $C_B^{\tilde{G}_p}(a_v) = \sum_{a_s \neq a_v \in V_p} \delta_{a_s \bullet}(a_v)$ . Now the key is given a  $a_s \in V_{\mathcal{P}}$ , compute  $\delta_{a_s \bullet}(a_v)$  for all  $a_v \in V_{\mathcal{P}}$ . The step (2) of Basic handles this by performing BFS and reverse BFS on a breadth-frst search directed acyclic graph (BFS DAG) of  $a_s$  on  $G_p$  (the correctness is held by Theorem [1](#page-4-1) and Theorem [2\)](#page-5-1).

DEFINITION 5. (BFS DAG) Given a P-multigraph  $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ and a vertex  $a_s \in V_p$ , the BFS DAG of  $a_s$  on  $G_p$  denoted by  $B_{a_s}^{G_p} =$  $(V_{a_s}^B, E_{a_s}^B)$   $(V_{a_s}^B \subseteq V_{\mathcal{P}}, E_{a_s}^B \subseteq E_{\mathcal{P}})$  is a directed acyclic graph which represents all shortest paths from  $a<sub>s</sub>$  to all other reachable vertices as discovered by conducting the BFS algorithm on  $G_{\mathcal{P}}$ , where  $V_{a_s}^B$ includes all vertices reachable from  $a_s$  on  $G_p$ , and  $E_{a_s}^B$  consists of all edges that are part of the shortest paths from  $a_s$  to each vertex on  $G_{\mathcal{P}}$ .

Specifically, the step (2) of Basic contains two phases. Phase I: take each vertex on  $G_{\mathcal{P}}$  as a source  $a_s,$  compute  $\delta_{a_s \bullet}(a_v)$  for each  $a_v \in V_{\mathcal{P}}$ : (1) Conduct BFS from  $a_s$  (i.e., build the BFS DAG of  $a_s$ ) to compute  $\beta_{a_s a_v}^{G_p}$  for each  $a_v \in V_p$  by Eq. [\(3\)](#page-4-2), and record the BFS DAG of  $a_s$  using the set  $Pred_{a_s}(a_v)$  which contains all predecessors of a vertex  $a_v$  on the BFS DAG, and the stack  $S$  which records the BFS sequence of vertices. (2) Perform reverse BFS (i.e., from the bottom to the top of the BFS DAG of  $a_s$ ) to compute each  $\delta_{a_s \bullet}(a_v)$ by Eq. [\(4\)](#page-5-2). **Phase II**: for each  $a_v \in V_{\mathcal{P}}$ , compute its BC by adding the source dependencies  $\delta_{a_s \bullet}(a_v)$  of all  $a_s \in V_{\mathcal{P}}(a_s \neq a_v)$ .

<span id="page-4-1"></span>THEOREM 1. While conducting BFS from  $a_s$  on  $G_p$  (i.e., building the BFS DAG of  $a_s$ ),  $\beta_{a_s a_v}^{G_p}$  ( $a_s \neq a_v \in V_{\mathcal{P}}$ ) can be computed recursively as follows (initially,  $\beta_{a_s a_s}^{G_p} = 1$ ):

<span id="page-4-2"></span>
$$
\beta_{a_s a_v}^{G_P} = \sum_{a_u \in Pred_{a_s}(a_v)} \beta_{a_s a_u}^{G_P} \times \beta_{a_s a_v}^{G_P}[a_u, a_v]
$$
(3)

Proof.  $pg_{a_{s}a_{u}}^{j} \circ (a_{u}, a_{v})$  means that each shortest  $\mathcal{P}\text{-}path$  in  $pg_{a_sa_u}^j$  correspondingly concatenates to each path instance from  $a_u$  to  $a_v$ . By the additivity and multiplicativity properties in Subsection [2,](#page-2-0)  $\beta_{as}^{G_p} = \sum_{pg^j \subseteq \Psi_{asap}} \beta_{asap}^{G_p} [pg^j]$ 

$$
= \sum_{a_{u} \in Pred_{s}(a_{v})} \left( \sum_{pg^{j} \subseteq \Psi_{a_{s}a_{u}}} \beta_{a_{s}a_{v}}^{G_{p}} [pg^{j} \circ (a_{u}, a_{v})] \right)
$$
  
\n
$$
= \sum_{a_{u} \in Pred_{s}(a_{v})} \left( \sum_{pg^{j} \subseteq \Psi_{a_{s}a_{u}}} \beta_{a_{s}a_{v}}^{G_{p}} [pg^{j}] \times \beta_{a_{s}a_{v}}^{G_{p}} [a_{u}, a_{v}] \right)
$$
  
\n
$$
= \sum_{a_{u} \in Pred_{a_{s}}(a_{v})} \left( \beta_{a_{s}a_{u}}^{G_{p}} \times \beta_{a_{s}a_{v}}^{G_{p}} [a_{u}, a_{v}] \right)
$$

To compute  $\beta_{a_s a_v}^{G_p}$  by Eq. [\(3\)](#page-4-2),  $\beta_{a_s a_v}^{G_p}$  [ $a_u, a_v$ ] for each  ${\cal P}$ -pair  $(a_u, a_v)$ on the BFS DAG of  $a_s$  should be acquired in advance. For cBC,  $\beta_{a_5a_v}^{G_P}[a_u, a_v] = F_{\mathcal{P}}[a_u, a_v]$  (directly acquired while building  $G_{\mathcal{P}}$ in the step (1) of Basic). For fBC, in the step(1) of Basic, while building  $G_{\mathcal{P}}$  from the HIN, we simultaneously establish the data structure  $EI(a_u, a_v)$  for each P-pair  $(a_u, a_v)$  to package the information which will be used for computing  $\beta_{a_3a_v}^{G_p}[a_u, a_v]$  in the step (2) of Basic, including  $|D_{a_u,a_v}^{\mathcal{P}}|$ , and between  $a_u$  and  $a_v$ , each path instance passes through which vertex with type  $D$ . Then in the step (2) of Basic, we conduct BFS twice in Phase I-(1): frstly, check  $EI(a_u, a_v)$  to compute  $I_{a_s}^d(pins)$ ; secondly, compute  $\beta_{a_s a_v}^{G_p}[a_u, a_v]$ with  $|D_{a_u,a_v}^{\mathcal{P}}|$  and  $I_{a_s}^d$  (*pins*) by Eq. [\(2\)](#page-4-0), and compute  $\beta_{a_s a_v}^{\hat{G}_p}$  by Eq. [\(3\)](#page-4-2).

<span id="page-5-1"></span>Theorem 2. While conducting reverse BFS from the bottom to the top of the BFS DAG of  $a_s$  on  $G_{\mathcal{P}}$ ,  $\delta_{a_s \bullet}(a_v)$   $(a_s \neq a_v \in V_{\mathcal{P}})$  can be computed recursively from each leaf node  $a_v$  ( $\delta_{a_s \bullet}(a_v) = 0$ ) of the BFS DAG as follows:

<span id="page-5-2"></span>
$$
\delta_{a_s \bullet}(a_v) = \sum_{a_w : a_v \in Pred_{a_s}(a_w)} \frac{\beta_{a_s a_v}^{G_p} \times \beta_{a_s a_w}^{G_p}[a_v, a_w]}{\beta_{a_s a_w}^{G_p}} \times (1 + \delta_{a_s \bullet}(a_w))
$$
\n(4)

PROOF. On the BFS DAG of  $a_s$ , if  $a_t$  is not a descendant of  $a_v$ , then all shortest  $P$ -paths from  $a_s$  to such an  $a_t$  would not pass through  $a_v$ , i.e.,  $\beta_{asat}^{G_p}(a_v) = 0$ . Let  $Dc(a_v)$  be the set of all descendants of  $a_v$ on the BFS DAG of  $a_s$ , then  $\delta_{a_s \bullet}(a_v) = \sum_{a_f \in Dc(a_v)} \frac{\beta_{a_s a_f}^{G_p}(a_v)}{\beta_{\varphi}}$  $\frac{\frac{\log a_t(uv)}{G\varphi}}{\beta_{asat}}$ . Divide  $Dc(a_n)$  into two subsets, one is the set of all children of  $a_n$  (denoted by  $Cl(a_v)$ ), the other is the set of the remaining descendants but not the children of  $a_v$  (denoted by  $Dc(a_v) \setminus Cl(a_v)$ ), so  $\sum_{a_f \in Dc(a_v)} \frac{\beta_{a_s a_f(a_v)}^{\overline{G_p}}}{\overline{G_p}}$  $\frac{\beta_{s}^{G}(\mu)}{\beta_{as}^{G}(\mu)}} =$  $\sum_{a_w \in Cl(a_v)} \frac{\beta_{a_s a_w}^{G_p}(a_v)}{\sqrt{G_p}}$  $\frac{\frac{\partial \varphi}{\partial s a_w}(a_v)}{\beta_{a_s a_w}} + \sum_{a_t \in Dc(a_v) \setminus Cl(a_v)} \frac{\beta_{a_s a_t}^{G \varphi}(a_v)}{\beta_{a_s a_t}^{G \varphi}}$  $\frac{\int_{g_{s}^{G_{p}}} G_{p}}{\int_{g_{s}^{G_{p}}} G_{q}}$ 

Based on the additivity and multiplicativity properties,  $\frac{\beta_{asat}^{G_p}(a_v)}{G_p}$  $\frac{\beta_{a}^{G}(\mu)}{\beta_{a_{s}a_{t}}} =$ 

$$
\frac{\sum_{a_{w} \in Cl(a_{v})} \beta_{as_{d}}^{G_{p}}}{\beta_{as_{d}}^{G_{p}}} = \frac{\sum_{a_{w} \in Cl(a_{v})} \beta_{as_{d}}^{G_{p}} \beta_{a_{v}a_{u}}^{G_{p}}}{\beta_{as_{d}}^{G_{p}} \beta_{a_{v}a_{u}}^{G_{p}}}
$$
\n
$$
= \sum_{a_{w} \in Cl(a_{v})} \frac{\beta_{as_{d}}^{G_{p}} \beta_{a_{v}a_{w}}^{G_{p}}}{\beta_{as_{d}}^{G_{p}}} \times \frac{\beta_{as_{d}w}^{G_{p}} \beta_{a_{w}a_{u}}^{G_{p}}}{\beta_{as_{d}a_{u}}^{G_{p}}}.
$$
\nThen we have  $\delta_{a_{s}}(a_{v}) = \sum_{a_{w}:a_{v} \in Pred_{s}(a_{w})} \frac{\beta_{as_{d}w}^{G_{p}} \beta_{a_{v}a_{u}}^{G_{p}}}{\beta_{as_{d}a_{w}}^{G_{p}}}.$ \n
$$
+ \sum_{a_{t} \in DC(a_{v}) \setminus Cl(a_{v})} \sum_{a_{w}:a_{v} \in Pred_{s}(a_{w})} \frac{\beta_{as_{d}a_{w}}^{G_{p}} \beta_{as_{d}a_{w}}^{G_{p}}}{\beta_{as_{d}a_{w}}^{G_{p}} \beta_{a_{v}a_{u}}^{G_{p}}} \times \frac{\beta_{as_{d}a_{v}}^{G_{p}} \beta_{a_{v}a_{u}}^{G_{p}}}{\beta_{a_{v}a_{u}}^{G_{p}}}.
$$
\n
$$
= \sum_{a_{w}:a_{v} \in Pred_{s}(a_{w})} \frac{\beta_{as_{d}a_{w}}^{G_{p}} \beta_{a_{v}}^{G_{p}}}{\beta_{as_{d}a_{w}}^{G_{p}}} \times (1 + \sum_{a_{t} \in DC(a_{v}) \setminus Cl(a_{v})} \frac{\beta_{as_{d}a_{v}}^{G_{p}} \beta_{a_{w}a_{u}}^{G_{p}}}{\beta_{as_{d}a_{u}}^{G_{p}}}).
$$
\n
$$
= \sum_{a_{w}:a_{v} \in Pred_{s}(a_{w})} \frac{\beta_{as_{d}a_{w}}^{G_{p}} \beta_{a_{v}}^{G_{p}}}{\beta_{as_{d}a_{w}}^{G_{p}}} \times (1 + \delta_{a_{s}}(a_{
$$

Fig. [3](#page-5-0) gives the process of computing  $\beta_{a_s a_v}^{G_p}$  and  $\delta_{a_s \bullet}(a_v)$  on the BFS DAG of  $a_s$  (the HIN topolopy is in Fig. [2\(](#page-3-1)a)). The pseudo-code

<span id="page-5-3"></span>

<span id="page-5-0"></span>

Figure 3: Example of Computing  $\beta_{a_{s}a_{v}}^{G_{\!P}}$  and  $\delta_{a_{s}\bullet}(a_{v}).$ 

of Basic for computing fBC is shown in Alg. [1.](#page-5-3) For computing cBC, it just needs to remove the lines about computing  $\beta_{a_s a_v}^{G_p} [a_u, a_v]$ . **Complexity Analysis.** In the step (1) of Basic, for  $\overrightarrow{P} = (A_1A_2 \dots$  $\overline{A_{(l+1)/2} \dots A_l A_{l+1}}$ , it costs the time of  $O(|A_1||A_2| \dots |A_{(l+1)/2}| + \dots)$  $|A_1||A_{(l+1)/2}| + |A_1||A_{(l+1)/2}||A_1|$  to compute the commuting matrix for building  $G_{\mathcal{P}}$ , note that EI can be simultaneously recorded while computing the commuting matrix so no extra time cost is required. In the step (2) of Basic, for each  $a_s \in V_{\mathcal{P}}$ , a BFS DAG of  $a_s$  is built, which corresponds to conduct BFS once or twice and conduct reverse BFS once based on the BFS DAG. Let  $n_p = |A_1| (m_p$  resp.) be the number of vertices (edges resp.) on  $G_{\mathcal{P}}$ , and  $\overline{m}_{\mathcal{P}}$  be the number of vertex pairs which have at least one edge between them on  $G_{\mathcal{P}}$  (also the number of  $\mathcal{P}$ -pairs on the HIN), obviously  $\overline{m}_{\mathcal{P}} < m_{\mathcal{P}}$ . Storing  $Nei_{G_{\mathcal{P}}}(a_v)$  and  $Pred_{a_s}(a_v)$  for each  $a_v$  with lists, the worst time complexity for the step (2) of Basic is  $O(n_P \times \overline{m}_P)$  for cBC

and  $O(n_P \times \overline{m}_P \times l_{max})$  for fBC respectively, where  $l_{max}$  is the maximum number of path instances between each  $P$ -pair.

# 4 OPTIMIZATION STRATEGIES

The bottleneck of Basic is step (2). There are two main parameters which afect the CPU time of step (2): frstly, the number of BFS DAGs actually built, i.e.,  $n_p$ ; secondly, the worst time cost of performing BFS and reverse BFS on a BFS DAG, i.e.,  $O(\overline{m}_{\mathcal{P}})$ . Thus, to greatly speed up BC computation but not cause loss to BC results, we develop the  $P$ -Multigraph compression strategy to reduce  $n_p$  and  $\overline{m}_p$  (Subsections [4.1\)](#page-6-0), and the BFS DAG sharing strategy to further reduce  $n_p$  (Subsections [4.2\)](#page-7-0). Since the definitions of  $\beta_{a_s a_t}^{\mathcal{P}}[a_x, a_{x+1}]$  for cBC and fBC are different, the optimization strategies for cBC and fBC are also diferent. So in each subsection, we introduce optimization strategies for cBC and fBC separately.

# <span id="page-6-0"></span>4.1 Compressing P-Multigraph

<span id="page-6-6"></span>4.1.1 Compressing  $G_{\varphi}$  for cBC. There are two basic ideas to compress  $G_{\mathcal{P}}$  for cBC. Firstly, for a vertex  $a_v$  on  $G_{\mathcal{P}}$ , if no shortest paths pass through  $a_v$ , then  $a_v$  can be removed from  $G_p$  since  $C_B^{G_p}(a_v) = 0$ . We call such  $a_v$  as a side vertex and give two kinds of defnitions of side vertices (1-side vertex and 2-side vertex) for cBC. By removing all side vertices,  $G_{\mathcal{P}}$  would be compressed. Secondly, for two vertices  $a_u$  and  $a_v$  on  $G_p$ , if their neighborhood information is the same, then  $a_u$  and  $a_v$  can be merged as one vertex  $a_u$  (or  $a_v$ ) since  $C_B^{G_p}(a_u) = C_B^{G_p}(a_v)$ . We call such  $a_u$  and  $a_v$ as identical vertices and give two kinds of defnitions of identical vertices (1-identical vertices and 2-identical vertices) for cBC. By merging each group of identical vertices as one vertex,  $G_{\phi}$ would also be compressed.

Graph Compression with Side Vertices. Firstly, we give the definition of the 1-side vertex by extending the defnition of the side vertex from an homogeneous network[\[43\]](#page-12-28) to the  $P$ -multigraph. Secondly, to reduce the time of identifying side vertices for cBC, we propose a relaxed defnition of the 1-side vertex, i.e., the 2-side vertex, which is defned directly on the HIN.

<span id="page-6-2"></span>DEFINITION 6. (1-Side Vertex) Given the P-multigraph  $G_{\varphi}$  =  $(V_{\mathcal{P}}, E_{\mathcal{P}})$ , a vertex  $a_i \in V_{\mathcal{P}}$  is a 1-side vertex on  $G_{\mathcal{P}}$ , iff there exist at least one edge between each two vertices in  $Nei_{G_P}(a_i) \bigcup \{a_i\}.$ 

<span id="page-6-3"></span>DEFINITION 7. (2-Side Vertex) Given an HIN G and a symmetric meta path  $P$  (whose symmetry point type is D), a vertex  $a_i$  with type A is a 2-side vertex on  $\mathbb G$ , iff there is only one vertex with type  $D$  which forms a  $l(P)$ -pair with  $a_i$  on  $\mathbb{G}$  ( $l(P)$  is the left half meta path of  $P$ ).

For example, in Fig. [4\(](#page-6-1)a),  $a_{x_1}$ ,  $a_{x_2}$  and  $a_{x_3}$  are 1-side vertices. In Fig. [4\(](#page-6-1)b),  $a_{x_1}$  and  $a_{x_2}$  are 2-side vertices.

Remark 1. Comparing Def. [6](#page-6-2) with Def. [7,](#page-6-3) the set of 2-side vertices is a subset of the set of 1-side vertices because of relaxing. However, according to extensive experiments on many real-world HINs (Section [6\)](#page-9-1), it indicates that in most cases, using 2-side vertices is better, because identifying 2-side vertices is faster, and the compression efects of 1-side vertices and 2-side vertices are very close.

**Observation 1.** If those 2-side vertices which have the same  $l(P)$ neighbor  $d_j$  ( $\phi_V(d_j) = A_{(l+1)/2}$ ) are divided into the same group (called a same\_side\_set), then  $Nei_{G_{\cal P}}(a_i) \cup \{a_i\}$  for each side vertex  $a_i$  in a same\_side\_set is the same. For example, in Fig. [4\(](#page-6-1)b),  $a_{x_1}$  and  $a_{x_2}$  forms a same\_side\_set.

<span id="page-6-4"></span><span id="page-6-1"></span>

Figure 5: Example of Identical Vertices for cBC.

<span id="page-6-5"></span> $\begin{array}{l} (1) \ E I(a_{x1},a_{y1}) = E I(a_{x2},a_{y1}) = E I(a_{x3},a_{y1}) \\ E I(a_{x1},a_{y2}) = E I(a_{x2},a_{y2}) = E I(a_{x3},a_{y2}) \\ (2) \ pr_1 = (a_{x1},a_{x2}) \ pr_2 = (a_{x1},a_{x3}) \ pr_3 = (a_{x2},a_{x3}) \\ E I(pr_1) = E I(pr_2) \end{array}$  $EI(pr_1) = EI(pr_3)$  $EI(pr_2)=EI(pr_3)$ 

Figure 6: Example of Identical Vertices for fBC.

Graph Compression with Identical Vertices. Firstly, we introduce the definition of the 1-identical vertices which is extended from an homogeneous network[\[43\]](#page-12-28) to the  $P$ -multigraph. Secondly, to speed up the time of fnding identical vertices, we give a relaxed defnition of the 1-identical vertices, called the 2-identical vertices, which is defned on the HIN directly.

DEFINITION 8. (1-Identical Vertices) Given the P-multigraph  $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ , two vertices  $a_1 \in V_{\mathcal{P}}$  and  $a_2 \in V_{\mathcal{P}}$  are type-I (or type-II) 1-identical on  $G_{\mathcal{P}}$  iff  $Nei_{G_{\mathcal{P}}}(a_1) = Nei_{G_{\mathcal{P}}}(a_2)$  (or  $Nei_{G_p}(a_1) \bigcup \{a_1\} = Nei_{G_p}(a_2) \bigcup \{a_2\}$ , and for any  $a_u \in Nei_{G_p}(a_1)$  $\bigcap Nei_{G_{\mathcal{P}}}(a_2)$   $(a_u \neq a_1, a_2)$ ,  $M_{\mathcal{P}}[a_1, a_u] = M_{\mathcal{P}}[a_2, a_u]$ .

DEFINITION 9. (2-Identical Vertices) Given an HING and a symmetric meta path  $P$  (whose symmetry point type is D), two vertices  $a_1$  and  $a_2$  both with type A are 2-identical on G iff  $F_{\ell(\mathcal{P})}[a_1, d_i] =$  $F_{l}(\varphi)$  [a<sub>2</sub>, d<sub>i</sub>] for each  $d_i$  ( $d_i \in D$ ).

For example, in Fig. [5\(](#page-6-4)a),  $a_{x_1}$  and  $a_{x_2}$  are type-I 1-identical ver-tices. In Fig. [5\(](#page-6-4)b),  $a_{x_3}$  and  $a_{x_4}$  are type-II 1-identical vertices. In Fig. [5\(](#page-6-4)c),  $a_{x_5}$  and  $a_{x_6}$  are 2-identical vertices.

Remark 2. Due to relaxing, the set of 2-identical vertices is a subset of the set of type-II 1-identical vertices. By lots of experiments (Section [6\)](#page-9-1), it indicates only identifying type-II 1-identical vertices is the best choice. Since the time of identifying type-II 1-identical vertices and the time of identifying 2-identical vertices are similar, but type-II 1-identical vertices contain all 2-identical vertices.

<span id="page-6-8"></span>4.1.2 Compressing  $G_{\mathcal{P}}$  for fBC. For fBC,  $\beta_{a_s a_t}^{\mathcal{P}} \neq \beta_{a_t a_s}^{\mathcal{P}}$ , so the idea of side vertices for cBC can't apply to fBC. However, the idea of identical vertices still works for fBC. Thus, we propose the new defnition of identical vertices for fBC which can be merged as one vertex on  $G_{\varphi}$  so that  $G_{\varphi}$  would be compressed.

Graph Compression with Identical Vertices. In the following, we give the new definition of identical vertices for fBC on  $G_{\mathcal{P}}$ .

<span id="page-6-7"></span>DEFINITION 10. (Identical Vertices) Given the P-multigraph  $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ , two vertices  $a_1 \in V_{\mathcal{P}}$  and  $a_2 \in V_{\mathcal{P}}$  are identical on  $G_{\mathcal{P}}$  iff the following two conditions are satisfied simultaneously: (1)  $Nei_{G_P}(a_1) \bigcup \{a_1\} = Nei_{G_P}(a_2) \bigcup \{a_2\}$ , and for any a neighbor  $a_u \in Nei_{G_p}(a_1) \cap Nei_{G_p}(a_2)$   $(a_u \neq a_1, a_2)$ ,  $EI(a_1, a_u) = EI(a_2, a_u)$ .

<span id="page-7-1"></span>

<span id="page-7-2"></span>

Figure 7: Sharing a BFS DAG for a same\_side\_set  $(a_{x_1}, a_{x_2}, a_{x_3})$ .

 $(2)$  For a set of identical vertices (i.e., iden\_set), when the number of identical vertices in the iden\_set is larger than 2, for any two  $P$ -pairs  $pr_i$  and  $pr_j$  whose vertices are in the iden\_set,  $EI(pr_i) = EI(pr_i)$ .

# For example, in Fig. [6,](#page-6-5)  $a_{x_1}$ ,  $a_{x_2}$  and  $a_{x_3}$  are identical vertices.

## <span id="page-7-0"></span>4.2 Sharing BFS DAGs

To further reduce the number of BFS DAGs actually built, the basic idea is that if a group of vertices have the same neighborhood information, then they can build and share only one BFS DAG while computing their source dependencies.

<span id="page-7-3"></span>4.2.1 Sharing BFS DAGs for cBC. By Observation 1 in Sub-section [4.1.1,](#page-6-6) all 2-side vertices in a same\_side\_set have the same neighborhood information, so we propose the side vertices-based

advanced algorithm for cBC computation (SdAdvCBC) which can build and share only one BFS DAG for all 2-side vertices in a same\_side\_set while computing their source dependencies, then remove those 2-side vertices from  $G_{\varphi}$  in batch.

**SdAdvCBC Algorithm.** For each vertex  $a_w$  on  $G_p = (V_p, E_p)$ , SdAdvCBC computes its cBC in two parts: (1) the sum of source dependencies  $\delta_{a_i \bullet}(a_w)$  for all 2-side vertex  $a_i \in V_{\mathcal{P}}$ ; (2) the sum of source dependencies  $\delta_{a_{i}\bullet}(a_{w})$  for all non-2-side vertex  $a_{i} \in V_{\varphi}$ . The pseudo-code of SdAdvCBC is in Alg. [2.](#page-7-1)

SdAdvCBC contains four steps. Firstly, identify all 2-side vertices and divide them into different same\_side\_sets based on their  $l(P)$ neighbors (i.e., the grouping strategy mentioned in Observation 1). Secondly, for each same\_side\_set, invoke our CBC\_SameSide algorithm to compute the above cBC part (1) for each  $a_w$  on  $G_p$ , then remove the whole same side set from  $G_{\mathcal{P}}$ , repeat the second step until all same\_side\_sets are removed and get compressed  $G_{\mathcal{P}}$ . Thirdly, invoke our Basic algorithm to compute the above cBC part (2) for each  $a_w$  on compressed  $G_p$ . Fourthly, sum the results of parts (1) and (2) to get the final cBC for each  $a_w \in V_{\mathcal{P}}$ .

Take a same\_side\_set  $(a_{x_1}, a_{x_2}, a_{x_3})$  as an example, we show how CBC\_SameSide builds and shares only one BFS DAG for all 2-side vertices in a same side set while computing their source dependencies. Assuming no BFS DAG sharing, for each  $a_{x_i}$  (*i* = 1, 2, or 3), it firstly perform BFS from  $a_{x_i}$  (i.e., build a BFS DAG of  $a_{x_i}$ ) to compute  $\beta^{\mathcal{P}}_{a_{x_i}a_v}$ , secondly perform reverse BFS on the BFS DAG of  $a_{x_i}$  to compute  $\delta_{a_{x_i}}(a_v)$ . In total, three BFS DAGs would be built (in Fig. [7\(](#page-7-2)a)), which is time wasting. Fortunately, for each source  $a_{x_1}$ ,  $a_{x_2}$  or  $a_{x_3}$  of those three BFS DAGs, after dividing its children into two groups (one is  $B_1$  whose vertices don't belong to the same\_side\_set, another is  $B_2$  whose vertices belong to the same\_side\_set), the following two observations are naturally acquired: (1) all  $B_1$ s as well as their descendants are completely the same; (2) all  $B_2$ s have no descendants so that  $\delta_{a_{x_i}}(a_v)$   $(a_v \in B_2,$  $i = 1, 2,$  or 3) is 0, thus, it doesn't need to perform BFS from each  $a_{x_i}$ to its  $B_2$ . Based on the above, CBC\_SameSide merges those three BFS DAGs in Fig. [7\(](#page-7-2)a) into one shared BFS DAG in Fig. [7\(](#page-7-2)b) while computing source dependencies. Specifically, by choosing  $a_{x_1}$  as a proxy for  $(a_{x_1}, a_{x_2}, a_{x_3})$ , CBC\_SameSide only needs to perform one BFS from  $a_{x_1}$  (i.e., build only one shared BFS DAG which contains  $a_{x_1}$ , a  $\overline{B_1}$  and the descendants of the  $B_1$ ), but can compute  $\beta_{a_{x_i}a_v}^{\mathcal{P}}$  for all  $a_{x_i}$  (*i* = 1, 2, or 3). Similarly, CBC\_SameSide only needs to perform one reverse BFS on the shared BFS DAG, but can compute  $\delta_{a_{x_i}}(a_v)$  of all  $a_{x_i}$  (*i* = 1, 2, or 3).

Remark 3. When the third step of SdAdvCBC computes the cBC part (2) for each  $a_w \in V_{\mathcal{P}}$  by Basic, the aggregated pair dependencies  $\sum_{a_j \neq a_w} \delta_{a_j a_i}(a_w)$  ( $a_i$  is a 2-side vertex,  $a_j$  is a non-2-side vertex) are missed, because the second step of SdAdvCBC has removed each 2-side vertex  $a_i$ . To compensating such pair dependencies, in the second step of SdAdvCBC, when CBC\_SameSide computes the cBC part (2) for each  $a_w$ , each source dependency  $\delta_{a_i \bullet}(a_w)$  times 2, because  $\delta_{a_i \bullet}(a_w) = \sum_{a_j \neq a_w} \delta_{a_j a_i}(a_w)$  ( $a_i$  is a 2-side vertex,  $a_j$ is a non-2-side vertex).

<span id="page-7-4"></span>4.2.2 Sharing BFS DAGs for fBC. By Def. [10,](#page-6-7) each set of identical vertices (called an *iden\_set*) have the same neighborhood information, so we present the identical vertices-based advanced



<span id="page-8-2"></span>1: the queue  $Q \leftarrow \emptyset$ , the stack  $S \leftarrow \emptyset$ ; Enqueue  $a_S \rightarrow Q$ ; 2:  $\beta_{a_s}[a_t] \leftarrow 0$ ,  $Pred[a_t] \leftarrow \text{empty list}, Dist[a_t] \leftarrow \infty, a_t \in V_{\mathcal{P}};$ 3:  $\beta_{a_S}[a_S] \leftarrow 1$ ,  $Dist[a_S] \leftarrow 0$ ;  $|I_{a_S}^d| \leftarrow 0, d \in V_D$ ; 4: while  $O$  is not empty do Dequeue  $a_u \leftarrow Q$ ; push  $a_u \rightarrow S$ ; 6: **The same as lines 9-14 in Alg. [1](#page-5-3)** (replace  $|I_{aS}^d|$  in line14 in Alg. 1 with  $|I_{a_S}^d|$  + = 1× (1 + *ident* [a<sub>v</sub>])); 7: if  $a_{\mu} = a_{\rm s}$  then for  $a_{ui} \in iden\_set[a_u]$  and  $a_{ui} \neq a_u$  do 9: **for** each pins  $\in$   $(a_u, a_{ui})$  do 10: Get *d* of *pins* from  $EI(a_u, a_{ui}); |I_{a_s}^d|$  + +; 11: The same as lines 15-24 in Alg. [1](#page-5-3) (replace line23 in Alg. [1](#page-5-3) with  $\beta_{a_S}[a_v]$  + =  $\beta_{a_S}[a_u] \times \beta[a_u, a_v] \times (1 + ident[a_u]),$ 12:  $\delta_{a_S}[a_v] \leftarrow 0, a_v \in V_{\mathcal{P}};$ 13: while  $S$  is not empty do 14: Pop  $a_w \leftarrow S$ ; 15: for  $(a_v, \beta[a_v, a_w]) \in Pred[a_w]$  do 16:  $\delta a_S [a_v] + \frac{\beta a_S [a_v] \times \beta [a_v, a_w]}{\beta a_S [a_w]}$  $\times (1+\delta_{a_{\mathcal{S}}}[a_{\mathcal{W}}])\times (1+ident[a_{\mathcal{W}}]);$ 17: if  $a_w \neq a_s$  then 18: for  $a_{wi} \in iden\_set[a_w]$  do<br>  $C'_B[a_{wi}] + \in \delta_{a_S}[a_w] \times (1 + ident[a_s])$ ; 20: **return**  $C'_B$ ;

<span id="page-8-1"></span>

algorithm for fBC computaion (IdAdvFBC) which can build and share only one BFS DAG for all identical vertices in an *iden\_set* while computing their source dependencies.

IdAdvFBC Algorithm. IdAdvFBC contains four steps to compute the fBC for each  $a_w \in V_{\mathcal{P}}$ . Firstly, identify all identical vertices by Def. [10,](#page-6-7) merge all vertices in an iden\_set as a proxy vertex. Let *ident*  $[a_v]$  be the number of identical vertices (excluding  $a_v$ ) for the vertex  $a_v \in V_{\mathcal{P}}$ . Secondly, for each proxy of an *iden\_set*, use our FBC\_Identical algorithm to compute the sum of source dependencies  $\delta_{a_i \bullet}(a_w)$  of all  $a_i$  where  $a_i$  is the proxy or other vertices in the iden\_set, repeat the second step until all proxies are considered. Thirdly, use our Basic algorithm to compute the sum of source dependencies  $\delta_{a_{i}\bullet}(a_{w})$  of all  $a_{i}$  where  $a_{i}$  doesn't belong to any iden\_set. Fourthly, sum the results of the second and third steps to get the final FBC for  $a_w$ .

Take an *iden\_set* ( $a_{x_1}, a_{x_2}, a_{x_3}$ ) as an example.  $a_{x_1}$  is the proxy after merging this *iden\_set*, and *ident*[ $a_{x_1}$ ] = 2. **FBC\_Identical** merges three BFS DAGs in Fig. [8\(](#page-8-1)a) into one shared BFS DAG in Fig. [8\(](#page-8-1)b) while computing source dependencies. Note that all  $B_2$ s are not on the shared BFS DAG since each  $a_{x_i}$  ( $i = 1, 2,$  and 3) and its  $B_2$  have been merged as the proxy  $a_{x_1}$  in the first step of IdAdvFBC. Specifcally, FBC\_Identical works as follows (the pseudo-code in Alg. [3\)](#page-8-2): (1) Perform BFS from the proxy  $a_{x_1}$  to its  $B_1$ , and then visit the *iden\_set* of  $a_{x_1}$  to check  $EI(a_{x_1}, a_{x_1})$  ( $j = 2$  or 3), so as to compute  $I_{a_{x_i}}(\text{pins})$  (each  $\text{pins}$  is between the first two levels of the shared BFS DAG). Continue to perform BFS from  $a_{x_1}$ 's  $B_1$  to its descendants to compute  $I_{a_{x_1}}(\text{pins})$  (each pins is between the

<span id="page-8-3"></span>

Figure 9: HINs vs. Weighted Homogeneous Networks

remaining levels of the shared BFS DAG). (2) Perform one BFS on the shared BFS DAG to compute  $\beta_{a_{x_1}a_v}^{G_P}[a_u,a_v]$  and  $\beta_{a_{x_1}a_v}^{\mathcal{P}}$ . (3) Perform one reverse BFS on the shared BFS DAG to compute  $\delta_{a_{x_1}}(a_v)$ . **Remark 4.** Based on Def. [10,](#page-6-7)  $\beta_{a_{x_1}a_v}^{\mathcal{P}} = \beta_{a_{x_2}a_v}^{\mathcal{P}} = \beta_{a_{x_3}a_v}^{\mathcal{P}}, \delta_{a_{x_1}}(a_v) =$  $\delta_{a_{x_2}}(a_v) = \delta_{a_{x_3}}(a_v)$ . Thus, FBC\_Identical only needs to compute  $\beta_{a_{x_1}a_v}^{\mathcal{P}}$  and  $\delta_{a_{x_1}\bullet}(a_v)$  on the shared BFS DAG, but can naturally acquire  $\beta^{\mathcal{P}}_{a_{x_2}a_v}$ ,  $\beta^{\mathcal{P}}_{a_{x_3}a_v}$  and  $\delta_{a_{x_2}\bullet}(a_v)$  and  $\delta_{a_{x_3}\bullet}(a_v)$ . Note that in fact, any vertex  $a_f$  on the shared BFS DAG might be a proxy of  $iden\_set[a_f]$ . Thus, FBC\_Identical multiplies some temporary results by  $\vec{ident}[a_f] + 1$  when necessary (in lines 6, 11, 16 and 19). **Remark 5.** For fBC, the neighborhood information of a vertex  $a_v$ on a BFS DAG contains two parts: (1) the neighbors of  $a_v$ , (2) the EI between  $a_v$  and its neighbors. A set of identical vertices have both the same neighbors and  $EI$  information. We also introduce a concept of similar vertices[\[5\]](#page-12-54) for fBC which only requires each set of similar vertices have the same neighbors, and develop the similar vertices-based advanced algorithm (called SmAdvFBC) which can build and share only one BFS DAG for all similar vertices in a similar\_set while computing their source dependencies. We

of SmAdvFBC and IdAdvFBC are compared in our experiments. Theorem 3. Network compression and BFS DAG sharing strategies do not cause any loss to cBC or fBC of vertices on  $G_{\mathcal{P}}$  (Proof in [\[5\]](#page-12-54)). Complexity Analysis. (1) Suppose there are  $q$  same\_side\_sets on  $G_{\mathcal{P}}$ . After removing the ith same\_side\_set from  $G_{\mathcal{P}}$ , there left  $n_i$ vertices and  $\overline{m}_i$  vertex pairs which have at least one edge between them. The worst time complexity of SdAdvCBC (Alg. [2\)](#page-7-1) reduces to  $O(\overline{m}_{\mathcal{P}} + \sum_{i=1}^{q-1} \overline{m}_i + n_q \times \overline{m}_q)$  compared with the step (2) of Basic. (2) Suppose  $G'_{\mathcal{P}}$  is generated by compressing  $G_{\mathcal{P}}$  with identical vertices. Let  $n''_{\mathcal{P}}$  ( $\overline{m}''_{\mathcal{P}}$  resp.) be the number of vertices (vertex pairs which have at least one edge between them resp.) on  $G_{\bm{\mathcal{P}}}^{\prime\prime}$ , then the worst time complexity of FBC\_Identical (Alg. [3\)](#page-8-2) is  $O(\overline{m}_{\mathcal{P}}^{\prime\prime} \times l_{max})$ , and the worst time complexity of IdAdvFBC reduces to  $O(n_p'' \times m_p'' \times l_{max})$ compared with the step (2) of Basic.

omit the similar details here for space limitation. The performance

#### <span id="page-8-0"></span>5 DISCUSSION

Firstly, HINs capture complex semantic relationships between different types of vertices, but some of which cannot be expressed by weighted homogeneous networks. For example, Fig. [9\(](#page-8-3)a) and Fig. [9\(](#page-8-3)b) show two diferent movie HINs, but would be transformed to the same weighted homogeneous network in Fig. [9\(](#page-8-3)c) where vertices are actors, a vertex weight represents the number of directors an actor cooperated with, an edge weight represents the cooperative times of two actors in movies which are directed by the same director. Considering fBC, in Fig.  $9(a)$ ,  $a_4$  is more important bridge than  $a_2$  or  $a_3$  to the communication between  $a_1$  and  $a_5$ , while in Fig. [9\(](#page-8-3)b),  $a_2$ ,  $a_3$  and  $a_4$  are equally important. However, the diference between Fig. [9\(](#page-8-3)a) and Fig. [9\(](#page-8-3)b) is missed when considering BC in Fig. [9\(](#page-8-3)c), since Fig. [9\(](#page-8-3)c) loses the topology information which the vertices with type  $D$  participate. Secondly, for a symmetric meta path  $P$ , when we consider the influence of the vertices whose type (supposing  $Q$ ) is neither  $A$  nor the symmetry point type D to the communication from  $a_s$  to  $a_t$ , the definition of  $\beta^{\mathcal{P}}_{a_{s}a_{t}}[a_{x},a_{x+1}]$  for fBC should be modified. Since  $\mathcal P$  is symmetric,  $Q$  would appears multiple times on  $P$ . Let  $c$  be the number of times which  $Q$  appears on  $P$ , then we just need to modify Eq. [\(2\)](#page-4-0) into  $\beta^{\mathcal{P}}_{a_{s}a_{t}}[a_{x}, a_{x+1}] = \left( \sum_{pins \in \Gamma_{a_{x}, a_{x+1}}} \frac{1}{\sum_{i=1}^{c} (1/c) \times I_{a_{s}}^{qi}(pins)} \right) + |Q^{\mathcal{P}}_{a_{x}, a_{x+1}}|,$ where  $q_i$  is the *i*th type-Q vertex on the path instance *pins*. When  $P$  is asymmetric, if Q only appears once on  $P$ , Eq. [\(2\)](#page-4-0) still holds, while if Q appears multiple times on  $P$  (supposing c times), Eq. [\(2\)](#page-4-0) is modifed into the same form as above. When we simultaneously consider the infuence of multiple diferent types of vertices whose types are not A to the communication from  $a_s$  to  $a_t$ , the modification to Eq. [\(2\)](#page-4-0) is similar as above. Note that our Basic algorithm does not need any changes but can be directly applicable to all the above cases. Moreover, our optimization strategies only need to simply adjust the information in  $EI$  then can apply to all the above cases, except for the strategies based on 2-side and 2-identical vertices whose definitions depend on the symmetry point type  $D$ .

# <span id="page-9-1"></span>6 EXPERIMENTS

All experiments are implemented with C++, on a machine with an Intel(R) Xeon(R) Gold 6226R CPU 2.9 GHz and 32GB main memory. Algorithms. For efectiveness testing, we compare PathRank[\[32\]](#page-12-24), infuence spread[\[15\]](#page-12-52), or structural diversity[\[23\]](#page-12-53) rankings with cBC or fBC rankings for vertices on HINs. For efficiency testing, we compare our Basic algorithms with diferent optimization strategies for cBC and fBC (Table [2\)](#page-9-2).

Datasets. We use four real datasets: Movies[\[4,](#page-12-55) [48\]](#page-12-56), Yelp[\[6\]](#page-12-57), DBLP[\[2,](#page-12-58) [49\]](#page-12-59) and IMDb[\[3\]](#page-12-60). Table [3\(](#page-9-3)a) shows the statistics of the datasets. Movies records relationships among movies, actors, directors and writers from Wikipedia. Yelp records relationships among users, reviews, businesses and cities on a restaurant review website. DBLP records relationships among authors, papers and venues on an online bibliographic database. IMDb records relationships among movies, actors, directors and writers on a movie website. Table [3\(](#page-9-3)b) shows the meta path  $P$  used in each dataset for efficiency evaluation. For IMDb, we extract four sub datasets with diferent sizes to variously test the efficiency of our algorithms.

# <span id="page-9-0"></span>6.1 Efectiveness Evaluation

Case Study on Movies. On Movies, we randomly extract a sub dataset, with 1628 actors (A), 701 movies (M) and 200 directors (D). Given  $P = (AMDMA)$ , Fig. [10\(](#page-9-4)a) shows the P-multigraph of the sub dataset. Computing cBC and fBC for all vertices with type A, it verifes that a vertex with higher cBC or fBC is really an important "bridge" in the communication among other vertices. For example, the yellow vertex 117 whose cBC ranks at top-1, is the bridge between two well-connected communities (each marked with a yellow dashed circle), i.e., once vertex 117 is removed, the communication between most of those two communities' members would be broken. Similarly, the red vertex 300 whose fBC ranks at top-1 is the bridge among four communities (each marked with

Table 2: Summary of Algorithms.

<span id="page-9-2"></span>

<b>Strategy</b>		<b>Description</b>					
<b>BA</b>		Graph splitting by bridge removing and articulation vertex cloning[43],					
		proposed for homogeneous networks, but can be directly used for HINs.					
SD1		Acceleration with 1-Side Vertices (Sec. 4.1.1)					
	Acceleration with 2-Side Vertices (Sec. 4.1.1, Sec. 4.2.1) SD2 (SdAdvCBC)						
ID1				Acceleration with type-I (T1) and type-II (T2) 1-identical Vertices (Sec. 4.1.1)			
ID2			Acceleration with 2-Identical Vertices (Sec. 4.1.1)				
SL (SmAdvFBC)			Acceleration with Similar Vertices (Sec. 4.2.2)				
ID (IdAdvFBC)		Acceleration with Identical Vertices (Sec. 4.1.2, Sec.4.2.2)					
Algorithm(cBC)							
	BasC (Basic for cBC)		$BasC + BA$	BasC+BA+SD1	BasC+BA+SD2		
	BasC+BA+ID1 T1 T2		BasC+BA+ID1 T2	BasC+BA+ID2	BasC+BA+SD2+ID1 T2		
Algorithm(fBC)							
	BasF (Basic for fBC)		$BasF+BA$	$BasF + BA + SI$ .	$BasF + BA + \Pi$		

#### Table 3: Statistics of Datasets.

<span id="page-9-3"></span>

<span id="page-9-4"></span>

Figure 10: Case Study on Movies.

<span id="page-9-5"></span>

(b) Academic Development Trajectory of Hector Garcia-Molina Figure 11: Case Study on DBLP.

a red dashed circle). We also compute edge BC[\[13\]](#page-12-34) of all edges and mark the edges whose edge BC rank at top-1 and top-2 in green in Fig. [10\(](#page-9-4)a). We can see that an edge has the highest edge BC, it doesn't mean that the endpoints of this edge also have the highest vertex BC, and vice versa. Thus, vertex BC and edge BC[\[8\]](#page-12-37) are substantially diferent. One of our future work is to redefne and efficiently compute edge BC on HINs. Moreover, we compute PathRank, infuence spread and structural diversity for vertices with type A. Vertices rankings under PathRank, influence spread or structural diversity are completely diferent from their cBC or fBC rankings. Furthermore, Fig. [10\(](#page-9-4)b) shows the heterogeneous subgraph that contains the common  $P$ -neighbors of vertex 117 (Eli Wallach, short for EW) and vertex 300 (Michael Caine, short for MC). Compared with EW, the shortest  $P$ -paths from actors in  $S_1$  to  $\mathcal{S}_2$  which pass through MC (marked with a red box) contain more directors. Suppose that MC lost contact with  $d_5$ , MC still could be a cooperation bridge between actors in  $S_1$  and  $S_2$  with the help of  $d_6$ ,  $d_7$  or  $d_8$ . However, once EW lost contact with  $d_4$ , EW would not be a cooperation bridge between actors in  $S_1$  and  $S_2$ . It is consistent with the fact that MC has higher fBC rank than EW. In reality, MC acted in numerous flms spanning various genres which were directed by many directors. We asked ChatGPT[\[1\]](#page-12-61) "Michael Caine and Eli Herschel Wallach, which of these two actors is more famous?" The answer is "Both Michael Caine and Eli Wallach were renowned actors, but if we were to gauge global recognition and infuence, Michael Caine is generally considered the more famous of the two." Finally, given  $\mathcal{P} = (AMWMA)$ , cBC and fBC rankings both occur changes, since the communication network topologies are diferent. Case Study on DBLP. On DBLP, we extract a sub dataset of DBLP from 6 research areas. To observe the variation of cBC or fBC for vertices with type  $A$  over time, from 1970-2009, we divide every five years' data into a snapshot. Given  $P = (APVPA)$  which represents two authors (A) published papers (P) in the same venues (V), we compute  $cBC$  and  $fBC$  for vertices with type  $A$  on each snapshot respectively. As shown in Fig. [11,](#page-9-5) there are two interesting fndings: (1) When the cBC or fBC of an author grows over time, his/her academic achievements are also gradually increasing. This can help to fnd rising stars. (2) When the cBC of an author remains high over time, but his/her fBC suddenly increases (or decreases) in a snapshot, we manually verify that during this period, the author began to explore multiple research areas (or deeply devoted to a specifc research area). This can help to fnd multi-feld researchers.

# 6.2 Efficiency Evaluation

Analysis of Side Vertices & Identical Vertices for cBC. Firstly, we compare 1-side and 2-side vertices for cBC by testing the number of side vertices (SD\_Num), the number of same\_side\_sets for all 2-side vertices (Set\_Num), the number of edges which would be subsequently removed after removing side vertices (E\_rmv\_Num), as well as the CPU time of identifying (ident\_Time) and removing (rmv\_Time) side vertices on each dataset. As shown in Table [4,](#page-10-0) to reduce the cBC computation time, generally, using 2-side vertices is better than 1-side vertices. Since in most cases (except for Yelp with no 2-side vertices), SD\_Num (E\_rmv\_Num resp.) for 2-side vertices is just slightly less than that for 1-side vertices, which means that the network compression efects of 1-side and 2-side vertices are close. Whereas, both ident\_Time and rmv\_Time for 2-side vertices are greatly smaller than those for 1-side vertices.

Secondly, we compare 1-identical and 2-identical vertices for cBC by testing the number of identical vertices (ID\_Num), the number of iden\_sets (Set\_Num), the number of edges which would be subsequently removed after merging identical vertices (E\_rmv\_Num), and the total CPU time of identifying and merging identical vertices (Time) on each dataset. We also compare type-I and type-II

<span id="page-10-0"></span>Table 4: Statistics of Side Vertices & Identical Vertices for cBC.



<span id="page-10-1"></span>Figure 12: Comparison of Optimization Strategies as for BasC.

<span id="page-10-2"></span>

Figure 13: Evaluating Optimization Strategies for fBC.

<span id="page-10-3"></span>

Figure 14: Scalability Test and Impact of Meta Path Length.

1-identical vertices for cBC by testing their ID\_Num, Set\_Num and Time. As shown in Table [4,](#page-10-0) to reduce cBC computation time with identical vertices, only using type-II 1-identical vertices is the best choice. Because ID\_Num for type-II vertices is greatly larger than that for type-I vertices, but the time of identifying and merging type-II vertices is much smaller than that of type-I vertices in most cases (in DBLP, there only exist type-II vertices). Besides, ID\_Num and E\_rmv\_Num (which means the network compression effect) of type-II 1-identical vertices is a little larger than that of 2-identical vertices, while the identifying and merging time of them are close. Comparison of Optimization Strategies as for BasC. In Fig. [12,](#page-10-1) to evaluate the efects of diferent optimization strategies (listed in Table [2\)](#page-9-2), we respectively integrate BA, BA+SD1, BA+SD2, BA+ID1 \_T1\_T2, BA+ID1\_T2, BA+ID2, or BA+SD2+ID1\_T2 to BasC (our Basic algorithm for cBC) to test their CPU time. Firstly, SD2 has better acceleration effect than SD1 in general, except for Yelp (because there are only a few 1-side vertices and no 2-side vertices in Yelp). In particular, on DBLP, SD2 reduces cBC computation time from 20.45 hours to 1.5 hours. Secondly, in each dataset, ID1\_T2 has the best acceleration efect on speeding up cBC calculation compared with ID1\_T1\_T2 or ID2. Particularly, on DBLP, both ID1 and ID2 work extremely well, reducing cBC computation time from 20.45 hours to about 0.5 hours. Thirdly, on most datasets, BA has great efect on speeding up cBC calculation. However, it is still quite meaningful to develop other strategies, because sometimes there exist no bridges and articulation vertices[\[43\]](#page-12-28) on a dataset like DBLP, then BA would lose the acceleration efect. In sum, whether a strategy works depends on the characteristic of each dataset (e.g., the number of side vertices, identical vertices). Based on our experimental results, using BA+ID1\_T2 would be a good choice since it has the best acceleration efect in most cases.

Analysis of Similar Vertices & Identical Vertices for fBC. We compare similar vertices with identical vertices for fBC by testing the number of similar vertices (SL\_Num), the number of similar\_sets (SL Set Num), the CPU time of identifying similar vertices (ident Time), the number of identical vertices (ID\_Num), the number of iden\_sets (ID\_Set\_Num), the number of edges which would be subsequently removed after merging identical vertices (E\_rmv\_Num), and the CPU time of identifying and merging identical vertices (Time). As shown in Fig. [13\(](#page-10-2)a), in most cases, only with very little ident\_Time (Time resp.), lots of similar vertices (identical vertices resp.) can be identifed (identifed and merged resp.). Moreover, on each dataset, SL\_Num (ID\_Num resp.) is large but SL\_Set\_Num (ID\_Set\_Num resp.) is small, it indicates that our SL (ID resp.) strategy would have good efect on speeding up fBC calculation due to BFS DAG sharing. Note that Yelp has only a few similar vertices and identical vertices, and DBLP spends much time on merging 17436 identical vertices into 315 proxy vertices, however, our SL and ID strategies still work very well as shown in Fig. [13\(](#page-10-2)b).

Comparison of Optimization Strategies as for BasF. In Fig. [13](#page-10-2) (b), to evaluate the efects of diferent optimization strategies (listed in Table [2\)](#page-9-2), we respectively integrate BA, BA+SL or BA+ID to BasF (our Basic algorithm for fBC) to test their CPU time. Compared with BA, our SL and ID strategies have remarkably better acceleration efect for fBC computation. Moreover, ID is superior to SL. Because ID greatly reduces the graph size by merging each  $iden\_set$  into a proxy, while for SL, similar vertices can't be removed or merged on  $G_{\mathcal{P}}$ . In addition, for a *iden\_set*, ID only needs to compute the proxy's source dependency, while for a similar\_set, SL still needs to compute all similar vertices' source dependencies. All in all, based on our experimental results, using BA+ID would be a good choice since it has the best acceleration efect in all used datasets.

Scalability Test. We generate five sub datasets by randomly selecting 20%, 40%, 60%, 80%, and 100% of vertices with type A (B resp.) for IMDb(3) and IMDb(1) (Yelp resp.). Then we conduct two advanced cBC (fBC resp.) computation algorithms on each sub dataset. As shown in Fig. [14\(](#page-10-3)a)-(d), generally, those algorithms all scale well with the number of vertices with type  $A$  (B resp.). We also add edges to 1.2, 1.4, 1.6 and 1.8 times the original size for Movies and Yelp. Fig. [14\(](#page-10-3)e) and (f) show our basic and advanced algorithms all scale well with the number of edges, and our optimization strategies still work well when the HINs get denser.

Impact of Meta Path length. Firstly, we group all meta paths in Movies based on their lengths, then conduct our algorithms for each group. Fig. [14\(](#page-10-3)g) shows the average CPU Time of each algorithm for each group (the meta paths with length 6 are asymmetric, so SD2 and ID1\_T2 cannot be used). As the length of meta paths grows, each algorithm costs more CPU time. Since longer meta paths lead to more edges on the  $P$ -multigraph. Secondly, for  $P =$  $(AMDMA)$  in Movies, by repeating  $P$  different times (e.g.,  $P^2 =$  $(AMDMAMDMA)$ , we compute  $\frac{\overline{m_p}}{m_c}$  on each  $\mathcal{P}^k$ -multigraph ( $k \in$ [1, 8]), where  $m_C = \sum_{c \in C} \frac{n_c \times (n_c - 1)}{2}$  (*C* is the set of connected components on the  $\mathcal{P}^k$ -multigraph,  $n_c$  is the number of vertices in  $c \in C$ ). In Fig. [14\(](#page-10-3)h), as  $\mathcal{P}^k$  gets longer,  $\frac{\overline{m_p}}{m_C}$  gradually becomes 1 which means that each two vertex pairs have at least one edge between them on the  $\mathcal{P}^k$ -multigraph, then it makes no sense to  $\overline{k}$ fnd shortest paths since the BC of all vertices is 0. After that, we test the CPU time of each algorithm when the meta path is  $\mathcal{P}^k$  $(k \in [1, 4])$ . As shown in Fig. [14\(](#page-10-3)i), when k grows, the CPU time of 5 algorithms all becomes larger. However, the growth trends of those algorithms are slowing down when  $k$  is larger than 3, since in such cases, each vertex pairs have edges between them on the  $\mathcal{P}^k$ -multigraph so that  $\overline{m}_{\mathcal{P}}$  remains unchanged.

# 7 CONCLUSIONS

In this paper, we are the frst to focus on a specifc type of vertices' BC on HINs. We propose a meta path-based BC framework on HINs and formalize cBC and fBC measures under the framework. We develop a generalized basic BC computation algorithm for cBC, fBC and their variants, and several optimization strategies which can greatly speed up cBC or fBC computation. Our experimental results on several real-life HINs reveal the great signifcance of both cBC and fBC, and verify that our optimization strategies are practically efective, and our algorithms have good scalability. In the future, we will try to integrate our cBC and fBC algorithms into existing graph databases (e.g, Neo4j), and study how to perform cBC or fBC computation on distributed computing platform (e.g., Spark).

# ACKNOWLEDGMENTS

This work was supported by (i) NSFC Grants 62202277, U22A2025, U2241211, U20B2046; (ii) the RGC of Hong Kong, China, No.14205520; (iii) Major Basic Research Program of Shandong Provincial NSF Grants ZR2022ZD02. Dongxiao Yu is the corresponding author.

#### **REFERENCES**

- <span id="page-12-61"></span>[1] ChatGPT. last accessed:01/06/2024.<https://chatgpt.com/>
- <span id="page-12-58"></span>[2] DBLP. last accessed:01/06/2024.<https://www.aminer.org/billboard/citation>
- <span id="page-12-60"></span>[3] IMDb. last accessed:01/06/2024.<https://www.IMDb.com/interfaces/>
- <span id="page-12-55"></span><span id="page-12-54"></span>Movies. last accessed:01/06/2024.<https://www.aminer.cn/data-sna#Movie> [5] Technical report(password:HINbcVLDB2024). last accessed:01/06/2024. [https:](https://github.com/1ran/BccH) [//github.com/1ran/BccH](https://github.com/1ran/BccH)
- <span id="page-12-57"></span>[6] Yelp. last accessed:01/06/2024.<https://www.yelp.com/dataset/>
- <span id="page-12-26"></span>[7] Jac M. Anthonisse. 1971. The rush in a directed graph. J. Comput. Phys. (1971),  $1 - 10$ .
- <span id="page-12-37"></span>[8] Phiradet Bangcharoensap, Tsuyoshi Murata, Hayato Kobayashi, and Nobuyuki Shimizu. 2016. Transductive Classifcation on Heterogeneous Information Networks with Edge Betweenness-based Normalization. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. 437–446.
- <span id="page-12-45"></span>[9] Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. 2012. Expressive Languages for Path Queries over Graph-Structured Data. ACM Trans. Database Syst. 37, 4 (2012), 31:1–31:46.
- <span id="page-12-44"></span>[10] Ivona Bezáková and Andrew Searns. 2018. On Counting Oracles for Path Problems. In ISAAC, Vol. 123. 56:1–56:12.
- <span id="page-12-46"></span>[11] Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Antti Ukkonen. 2014. Distance oracles in edge-labeled graphs. In EDBT. 547–558.
- <span id="page-12-27"></span>[12] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. Journal of mathematical sociologyJ Math Sociol 25, 2 (2001), 163–177.
- <span id="page-12-34"></span>[13] Ulrik Brandes. 2008. On variants of shortest-path betweenness centrality and their generic computation. Soc. Networks 30, 2 (2008), 136–145.
- <span id="page-12-49"></span>[14] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. Comput. Networks 30, 1-7 (1998), 107–117.
- <span id="page-12-52"></span>[15] Suqi Cheng, Huawei Shen, Junming Huang, Wei Chen, and Xueqi Cheng. 2014. Imrank: infuence maximization via fnding self-consistent ranking. In SIGIR. 475–484.
- <span id="page-12-14"></span>[16] Cyrus Cousins, Chloe Wohlgemuth, and Matteo Riondato. 2021. Bavarian: Betweenness Centrality Approximation with Variance-Aware Rademacher Averages. In SIGKDD. 196-206.
- <span id="page-12-2"></span>[17] Elizabeth M. Daly and Mads Haahr. 2007. Social network analysis for routing in disconnected delay-tolerant MANETs. In MobiHoc. 32–40.
- <span id="page-12-30"></span>[18] Martin Everett and Stephen P Borgatti. 2005. Ego network betweenness. Soc. Networks 27, 1 (2005), 31–38.
- <span id="page-12-33"></span>[19] Martin G Everett and Stephen P Borgatti. 2005. Extending centrality. Models and methods in social network analysis 35, 1 (2005), 57–76.
- <span id="page-12-22"></span>[20] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. Proc. VLDB Endow. 13, 6 (2020), 854–867.
- <span id="page-12-0"></span>[21] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. Sociometry (1977), 35–41.
- <span id="page-12-35"></span>[22] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2015. Fully Dynamic Betweenness Centrality Maintenance on Massive Networks. Proc. VLDB Endow. 9, 2 (2015), 48–59.
- <span id="page-12-53"></span>[23] Xin Huang, Hong Cheng, Rong-Hua Li, Lu Qin, and Jefrey Xu Yu. 2013. Top-k structural diversity search in large networks. Proc. VLDB Endow. 6, 13 (2013), 1618–1629.
- <span id="page-12-3"></span>[24] Yanan Jiang and Hui Xia. 2024. Adversarial attacks against dynamic graph neural networks via node injection. High-Confdence Computing 4, 1 (2024), 100185.
- <span id="page-12-8"></span>[25] Ayoub Jibouni, Dounia Lotfi, and Ahmed Hammouch. 2023. Link prediction using betweenness centrality and graph neural networks. Soc. Netw. Anal. Min. 13, 1 (2023), 5.
- <span id="page-12-7"></span>[26] Shuangshuang Jin, Zhenyu Huang, Yousu Chen, Daniel Chavarría-Miranda, John Feo, and Pak Chung Wong. 2010. A novel application of parallel betweenness centrality to power grid contingency analysis. In IPDPS. 1–7.
- <span id="page-12-50"></span>[27] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of infuence through a social network. In SIGKDD. 137–146.
- <span id="page-12-4"></span>[28] Dirk Koschützki and Falk Schreiber. 2008. Centrality analysis methods for biological networks and their application to gene regulatory networks. Gene regulation and systems biology 2 (2008), GRSB–S702.
- <span id="page-12-36"></span>[29] Nicolas Kourtellis, Gianmarco De Francisci Morales, and Francesco Bonchi. 2015. Scalable Online Betweenness Centrality in Evolving Graphs. ACM Trans. Knowl. Discov. Data 27, 9 (2015), 2494–2506.
- <span id="page-12-15"></span>[30] Tomás Lagos, Oleg A Prokopyev, and Alexander Veremyev. 2024. Finding groups with maximum betweenness centrality via integer programming with random path sampling. *J. Glob. Optim.* 88, 1 (2024), 199-232.
- <span id="page-12-5"></span>[31] Min-Joong Lee, Jungmin Lee, Jaimie Yejean Park, Ryan Hyun Choi, and Chin-Wan Chung. 2012. QUBE: a quick algorithm for updating betweenness centrality. In WWW. 351–360.
- <span id="page-12-24"></span>[32] Sangkeun Lee, Sungchan Park, Minsuk Kahng, and Sang-goo Lee. 2013. PathRank: Ranking nodes on a heterogeneous graph for fexible hybrid recommender systems. Expert Syst. Appl. 40, 2 (2013), 684–697.
- <span id="page-12-6"></span>[33] Loet Leydesdorff. 2007. Betweenness centrality as an indicator of the interdisciplinarity of scientifc journals. J. Assoc. Inf. Sci. Technol. 58, 9 (2007), 1303–1319.
- <span id="page-12-10"></span>[34] Jongmin Park, Seunghoon Han, Soohwan Jeong, and Sungsu Lim. 2024. Hyperbolic Heterogeneous Graph Attention Networks. In WWW. 561–564.
- <span id="page-12-1"></span>[35] Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. 2019. Estimating Node Importance in Knowledge Graphs Using Graph Neural Networks. In SIGKDD. 596–606.
- <span id="page-12-16"></span>[36] Leonardo Pellegrina and Fabio Vandin. 2024. Silvan: estimating betweenness centralities with progressive sampling and non-uniform rademacher bounds. ACM Trans. Knowl. Discov. Data 18, 3 (2024), 52:1–52:55.
- <span id="page-12-39"></span>You Peng, Jeffrey Xu Yu, and Sibo Wang. 2023. PSPC: Efficient Parallel Shortest Path Counting on Large-Scale Graphs. In ICDE. 896–908.
- <span id="page-12-40"></span>[38] Yu-Xuan Qiu, Dong Wen, Lu Qin, Wentao Li, Ronghua Li, and Ying Zhang. 2022. Efficient Shortest Path Counting on Large Road Networks. Proc. VLDB Endow. 15, 10 (2022), 2098–2110.
- <span id="page-12-17"></span>[39] Sai Charan Regunta, Sai Harsh Tondomker, Kshitij Shukla, and Kishore Kothapalli. 2023. Efficient parallel algorithms for dynamic closeness-and betweenness centrality. Concurr. Comput. Pract. Exp. 35, 17 (2023), e6650.
- <span id="page-12-43"></span>[40] Yuanfang Ren, Ahmet Ay, and Tamer Kahveci. 2018. Shortest path counting in probabilistic biological networks. BMC Bioinform. 19, 1 (2018), 465:1–465:19.
- <span id="page-12-47"></span>[41] Michael N. Rice and Vassilis J. Tsotras. 2010. Graph Indexing of Road Networks for Shortest Path Queries with Label Restrictions. Proc. VLDB Endow. 4, 2 (2010), 69–80.
- <span id="page-12-18"></span>[42] Diego Santoro and Ilie Sarpe. 2022. Onbra: Rigorous estimation of the temporal betweenness centrality in temporal networks. In WWW. 1579–1588.
- <span id="page-12-28"></span>[43] Ahmet Erdem Sariyüce, Kamer Kaya, Erik Saule, and Ümit V. Çatalyürek. 2017. Graph Manipulations for Fast Centrality Computation. ACM Trans. Knowl. Discov. Data 11, 3 (2017), 26:1–26:25.
- <span id="page-12-32"></span>[44] Julian Shun. 2020. Practical parallel hypergraph algorithms. In PPoPP. 232–249.
- <span id="page-12-21"></span>[45] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. Path-Sim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. Proc. VLDB Endow. 4, 11 (2011), 992–1003.
- <span id="page-12-25"></span>[46] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S Yu, and Xiao Yu. 2013. Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. ACM Trans. Knowl. Discov. Data 7, 3 (2013), 1–23.
- <span id="page-12-29"></span>[47] Guangming Tan, Dengbiao Tu, and Ninghui Sun. 2009. A Parallel Algorithm for Computing Betweenness Centrality. In ICPP. 340–347.
- <span id="page-12-56"></span>[48] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social infuence analysis in large-scale networks. In SIGKDD. 807–816.
- <span id="page-12-59"></span>[49] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In SIGKDD. 990–998.
- <span id="page-12-42"></span>[50] Vianney Kengne Tchendji and Jerry Lacmou Zeutouo. 2019. An Efficient CGM-Based Parallel Algorithm for Solving the Optimal Binary Search Tree Problem Through One-to-All Shortest Paths in a Dynamic Graph. Data Sci. Eng. 4, 2 (2019), 141–156.
- <span id="page-12-51"></span>[51] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. 2012. Structural diversity in social contagion. Proc. Natl. Acad. Sci. 109, 16 (2012), 5962–5966.
- <span id="page-12-38"></span>[52] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. SIAM J. Comput. 8, 3 (1979), 410–421.
- <span id="page-12-23"></span>[53] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Efective and Efficient Truss Computation over Large Heterogeneous Information Networks. In ICDE. 901–912.
- <span id="page-12-9"></span>[54] Vahan Yoghourdjian, Tim Dwyer, Karsten Klein, Kim Marriott, and Michael Wybrow. 2018. Graph Thumbnails: Identifying and Comparing Multiple Graphs at a Glance. IEEE Trans. Vis. Comput. Graph. 24, 12 (2018), 3081–3095.
- <span id="page-12-11"></span>[55] Xingtong Yu, Yuan Fang, Zemin Liu, and Xinming Zhang. 2024. Hgprompt: Bridging homogeneous and heterogeneous graphs for few-shot prompt learning. In AAAI, Vol. 38. 16578–16586.
- <span id="page-12-31"></span>[56] Qi Zhang, Rong-Hua Li, Minjia Pan, Yongheng Dai, Guoren Wang, and Ye Yuan. 2022. Efficient Top-k Ego-Betweenness Search. In ICDE. 380-392.
- <span id="page-12-19"></span>[57] Tianming Zhang, Yunjun Gao, Jie Zhao, Lu Chen, Lu Jin, Zhengyi Yang, Bin Cao, and Jing Fan. 2024. Efficient Exact and Approximate Betweenness Centrality Computation for Temporal Graphs. In  $W\!\bar W\!\bar W$ . 2395–2406.
- <span id="page-12-48"></span>[58] Xiaofei Zhang and M. Tamer Özsu. 2019. Correlation Constraint Shortest Path over Large Multi-Relation Graphs. Proc. VLDB Endow. 12, 5 (2019), 488–501.
- <span id="page-12-41"></span>Yikai Zhang and Jeffrey Xu Yu. 2020. Hub Labeling for Shortest Path Counting. In SIGMOD. 1813–1828.
- <span id="page-12-20"></span>[60] Zhigao Zheng, Bo Du, Chen Zhao, and Peichen Xie. 2023. Path Merging Based Betweenness Centrality Algorithm in Delay Tolerant Networks. IEEE J. Sel. Areas Commun. 41, 10 (2023), 3133–3145.
- <span id="page-12-12"></span>[61] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Infuential community search over large heterogeneous information networks. Proc. VLDB Endow. 16, 8 (2023), 2047–2060.
- <span id="page-12-13"></span>[62] Ying Zou, Zihan Fang, Zhihao Wu, Chenghui Zheng, and Shiping Wang. 2024. Revisiting multi-view learning: A perspective of implicitly heterogeneous Graph Convolutional Network. Neural Networks 169 (2024), 496–505.