Efficient Betweenness Centrality Computation over Large Heterogeneous Information Networks

Xinrui Wang
ShanDong University
xrwang@sdu.edu.cn

Yiran Wang
ShanDong University
yr_wang@mail.sdu.edu.cn

Xuemin Lin
Shanghai Jiaotong University
xuemin.lin@gmail.com

Jeffrey Xu Yu
The Chinese University of Hong Kong
yu@se.cuhk.edu.hk

Hong Gao
Zhejiang Normal University
honggao@zjnu.edu.cn

Xiuzhen Cheng
ShanDong University
xzcheng@sdu.edu.cn

Dongxiao Yu✉
ShanDong Universityy
dxyu@sdu.edu.cn

ABSTRACT

Betweenness centrality (BC), a classic measure which quantifies 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 first to study a specific 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? We advocate a meta path-based BC framework on HINs and formalize both coarse-grained and fine-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-first search directed acyclic graph (BFS DAG) sharing. Experiments on several real-world HINs show the significance of cBC and fBC, and the effectiveness of our proposed optimization strategies.

PVLDB Reference Format:
Xinrui Wang, Yiran Wang, Xuemin Lin, Jeffrey 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

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at 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 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]. 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 reflect their abilities to control the communication between other vertices in the network[21, 35]. Vertices with high BC can facilitate, impede or bias the transmission of messages in the network[21], which are greatly meaningful in practice [17, 24, 28, 31, 33]. For example, in social networks, users with high BC can promote communication between other unfamiliar users[31]. In power transmission networks, the failures of power grid components with high BC would lead to widespread power outage incidents[26]. Furthermore, vertices' BC has widely been used for improving the quality of other graph analysis tasks, such as link prediction[25] and graph visualisation[54].

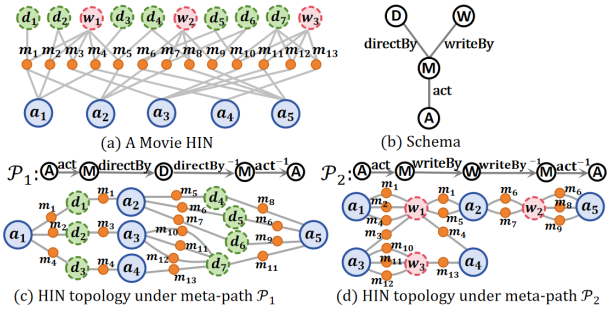


Figure 1: Example of a Movie HIN.

In reality, many information networks are inherently heterogeneous, containing multiple types of vertices and multiple types of edges which represent different 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(a) shows an example of a movie HIN which describes the relationships among different 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, 55, 61, 62] and the practicality of vertices' BC[16, 30, 36, 39, 42, 57, 60], in this paper, we are interested in computing a specific 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

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3682006

(called ***coarse-grained BC***, ***cBC*** for short) is to extend the original BC definition 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 different 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], 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 different semantics form completely different communication network topologies. E.g., Fig. 1(c) and Fig. 1(d) show different topologies under meta paths ($AMDMA$) and ($AMWMA$) respectively, which leads to completely different BC results (verified by our experiments in Subsection 6.1). In this paper, while finding important communication bridges between vertices, we focus on one specific communication semantics behind a given meta path $\mathcal{P}$. That is, we only count concrete shortest paths based on $\mathcal{P}$ (called **shortest $\mathcal{P}$-paths**) to compute cBC. $\mathcal{P}$ 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], community search[20, 53], PageRank computation[32], and clustering[46].

To show how to count shortest $\mathcal{P}$-paths and compute cBC, we use an example in Fig. 1 where there exist no directed links between any two vertices both with type A. Given $\mathcal{P} = (AMDMA)$ in Fig. 1(c), two vertices both with type A (e.g., a_1 and a_2) form a **$\mathcal{P}$-pair** if they are connected by a **path instance** of $\mathcal{P}$ (e.g., $(a_1 m_1 d_1 m_1 a_2)$). A path connecting a_s and a_t (both with type A), which is formed by a series of sequentially concatenated path instances of $\mathcal{P}$, is called as a **$\mathcal{P}$-path** from a_s to a_t (e.g., two path instances $(a_1 m_1 d_1 m_1 a_2)$ and $(a_2 m_5 d_4 m_8 a_5)$ are concatenated as a $\mathcal{P}$-path $(a_1 m_1 d_1 m_1 a_2 m_5 d_4 m_8 a_5)$ from a_1 to a_5). A shortest $\mathcal{P}$-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 $\mathcal{P}$-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 $\mathcal{P}$-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 $\mathcal{P}$-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 $\mathcal{P}$-path, there exists not a direct link but a path instance of $\mathcal{P}$ between each $\mathcal{P}$-pair. After that, among the $\mathcal{P}$-pairs of several shortest $\mathcal{P}$-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(c), all path instances between the $\mathcal{P}$-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 $\mathcal{P}$-paths which contain (a_2, a_5) is also more robust than that along the shortest $\mathcal{P}$-paths which contain (a_3, a_5). In other words, as for the communication between a_1 and a_5, the shortest $\mathcal{P}$-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 $\mathcal{P}$-paths.

To fill the gap of cBC which ignores that shortest $\mathcal{P}$-paths would have different importance to the communication between a pair of vertices due to the influence 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 $\mathcal{P}$-path a weight so that a shortest $\mathcal{P}$-path from a_s to a_t would have a larger weight if along this shortest $\mathcal{P}$-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 $\mathcal{P}$-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 $\mathcal{P}$-paths with larger weights. Actually, cBC is a special case of fBC which considers that the weight of each shortest $\mathcal{P}$-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 different directors would involve in promoting other actors' cooperation, i.e., there contain more different 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 films 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_y 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_y participates would not be affected.

Challenges and Contributions. As far as we known, we are the first to focus on a specific type of vertices' BC on HINs. Considering both coarse-grained and fine-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 $\mathcal{P}$-paths; (2) weight each shortest $\mathcal{P}$-path; (3) find each a_v lying on which shortest $\mathcal{P}$-paths to compute its BC.

Firstly, while finding all shortest $\mathcal{P}$-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 $\mathcal{P}$-pair (a_v, a_u) on $\mathbb{G}$ 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 $\mathcal{P}$-path. After that, we formulate cBC and fBC measures under our meta path-based BC framework respectively from coarse-grained and fine-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). Let $n_{\mathcal{P}}$ 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 $n_{\mathcal{P}}$ **breadth-first search directed acyclic graphs** (**BFS DAGs**, e.g., Fig. 3) on $G_{\mathcal{P}}$. There are two main parameters which affect the total time of the basic algorithm: firstly, $n_{\mathcal{P}}$; secondly, the worst time cost of performing BFS and reverse BFS on a BFS DAG, i.e., $O(\overline{m}_{\mathcal{P}})$. Thus, to further speed up BC computation but not cause loss to BC results, we develop the network compression strategy to reduce $n_{\mathcal{P}}$ and $\overline{m}_{\mathcal{P}}$, and the BFS DAG sharing strategy to further reduce $n_{\mathcal{P}}$.

Extensive experiments have been conducted on several real-world HINs to verify the great significance of cBC and fBC, and show the great effectiveness of our proposed optimization strategies.

Related Work. BC is first proposed by Anthonisse and Freeman[7, 21]. Brandes[12] 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 efforts have been made to further accelerate BC computation[43, 47]. Moreover, as computing exact BC values is time consuming, many researchers have focused on estimating approximate BC values [16, 18, 56]. In addition, BC on different types of graphs like hypergraphs [44], bipartite graphs [19], valued graph[13], and dynamic graphs [22, 29, 31] has been extensively studied. Some researchers have also given BC variants like edge BC, distance-scaled BC, and group BC[8, 13]. Whereas, none of the above works focus on a specific 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]. Many studies have reduced the time of counting shortest paths on simple graphs [37, 38, 59], dynamic graph[50], probabilistic graphs[40], planar graphs[10] and labeled graphs[9, 11, 41, 58]. However, they don't consider how to reduce the time of finding 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. Different metrics have their own unique meaningness and cannot be replaced by each other. For example, PageRank[14] focuses on importance vertices that are linked by many other important vertices, influence maximization[27] finds a seed set of vertices to maximize the spread of influence in a network, structural diversity[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] (extended PageRank on HINs), influence spread[15], structural diversity[23], cBC and fBC, the results verify that those metrics are quite different, and cBC and fBC are indispensable.

2 PROBLEM DESCRIPTION

An **HIN**[45], 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 $\phi_V : V \rightarrow \mathcal{A}$ and an edge type mapping function $\phi_E : E \rightarrow \mathcal{R}$, 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] of an HIN $\mathbb{G}$ 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 $\mathbb{G}$. 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] $\mathcal{P}$ 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 $pins = (v_1, v_2, \dots, v_{l+1})$ as a **path instance** of $\mathcal{P}$, if $\forall i$, vertex 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 **$\mathcal{P}$-pair** (or a_u and a_v are **$\mathcal{P}$-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 $\mathcal{P}$ is symmetric about $A_{(l+1)/2}$, we say $\mathcal{P}$ 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 $\mathcal{P}$ 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 $\mathcal{P} = (A, \dots, A)$. For ease of presentation, we firstly adopt symmetric meta paths in our models and algorithms, then show how to expand to unsymmetric meta paths in Section 5.

Meta Path-based BC Framework on HINs. Two path instances of $\mathcal{P}$, $pins_1 = (v_1, v_2, \dots, v_{l+1})$ and $pins_1 = (v'_1, v'_2, \dots, v'_{l+1})$, are concatenable iff $v_{l+1} = v'_1$. The concatenated path of $pins_1$ and $pins_2$ 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., $pins_1 = (a_s = a_{x_1}, \dots, a_{x_2}), pins_2 = (a_{x_2}, \dots, a_{x_3}), \dots, pins_{n-1} = (a_{x_{n-1}}, \dots, a_{x_n} = a_t)$) can be sequentially concatenated as a path $p = (pins_1 \circ pins_2 \circ \dots \circ pins_{n-1})$, we call p as a **$\mathcal{P}$-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 $\mathcal{P}$-path from a_s to a_t which contains the smallest number of vertices. In Fig. 1(c), $p^1 = (a_1, m_1, d_1, m_1, a_2, m_5, d_4, m_8, a_5)$ is a shortest $\mathcal{P}$-path from a_1 to a_5.

Definition 1. ***(Weight of Shortest $\mathcal{P}$-Path)*** *Given a shortest $\mathcal{P}$-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.

Notion	Meaning
$C_B^{\mathcal{P}}(a_v)$	meta path-based BC of a_v on an HIN $\mathbb{G}$
$\beta^{\mathcal{P}}_{a_s a_t}$ (or $\beta^{\mathcal{P}}_{a_s a_t}(a_v)$)	sum of weights of all shortest $\mathcal{P}$-paths from a_s to a_t (which a_v lies on)
$\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$	the weight of a $\mathcal{P}$-pair (a_x, a_{x+1})
$\Gamma_{a_x, a_{x+1}}$	the set of path instances between a $\mathcal{P}$-pair (a_x, a_{x+1})
$\lvert I^d_{a_x}(pins)\rvert$	number of path instances that interfere *pins* while a_s communicating to other vertices
$\lvert D^{\mathcal{P}}_{a_x, a_{x+1}}\rvert$	number of all vertices with type D which the path instances in $\Gamma_{a_x, a_{x+1}}$ pass through
$G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$	the $\mathcal{P}$-multigraph built from an HIN $\mathbb{G}$ based on a meta path $\mathcal{P}$
$F_{\mathcal{P}}[i, j]$ $(= M_{\mathcal{P}}[i, j])$	number of path instances of $\mathcal{P}$ between (a_i, a_j) on $\mathbb{G}$ (edges between (a_i, a_j) on $G_{\mathcal{P}}$)
$\delta_{a_s \bullet}(a_v)$	the source dependency of a_s on $G_{\mathcal{P}}$
$Pred_{a_s}(a_v)$	all predecessors of a vertex a_v on the BFS DAG of a_s on $G_{\mathcal{P}}$
$EI(a_u, a_v)$	packaged information of a $\mathcal{P}$-pair (a_u, a_v) for computing $\beta^{G_{\mathcal{P}}}_{a_s a_v}[a_u, a_v]$
$I(\mathcal{P}) = (A_1 A_2 \dots A_{(l+1)/2})$	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, $\beta^{\mathcal{P}}_{a_s a_t}$ be the sum of weights of all shortest $\mathcal{P}$-paths in $\Psi_{a_s a_t}$ which measures along those shortest $\mathcal{P}$-paths, how robust the communication from a_s to a_t is. Let $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$ 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 $\mathcal{P}$-path from a_s to a_t. If a_s and a_t form a $\mathcal{P}$-pair, $\beta^{\mathcal{P}}_{a_s a_t} = \beta^{\mathcal{P}}_{a_s a_t}[a_s, a_t]$. Otherwise (i.e., connecting a_s and a_t needs at least two sequentially concatenated path instances of $\mathcal{P}$), the weight of the shortest $\mathcal{P}$-path has the following two elegant properties.

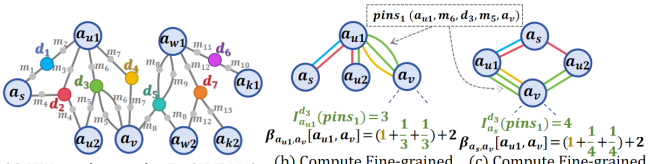
Properties 1. (Additivity) If all shortest $\mathcal{P}$-paths in $\Psi_{a_s a_t}$ can be divided into r groups ($r \geq 1$) so that each shortest $\mathcal{P}$-path is only in a unique group pg^j, let $\beta_{a_s a_t}[pg^j]$ be the sum of weights of the group of shortest $\mathcal{P}$-paths in pg^j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

Properties 2. (Multiplicativity) Given a group of shortest $\mathcal{P}$-paths pg^j from a_s to a_t, if all $\mathcal{P}$-pairs on each shortest $\mathcal{P}$-path in pg^j are identical (assuming they are $(a_s = a_1, a_2), (a_2, a_3), \dots, (a_{k_j}, a_{k_j+1} = a_t)$ sequentially), we say pg^j contains k_j $\mathcal{P}$-pairs where each $\mathcal{P}$-pair is $(a_x, a_{x+1})(1 \leq x \leq k_j)$. Since each $\mathcal{P}$-pair in pg^j independently affects the communication robustness from a_s to a_t, we have $\beta_{a_s a_t}[pg^j] = \prod_{x=1}^{k_j} \beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$.

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

For ease of calculation, to compute $\beta^{\mathcal{P}}_{a_s a_t}$ (except that a_s and a_t form a $\mathcal{P}$-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 $\beta^{\mathcal{P}}_{a_s a_t} = \sum_{j=1}^{r} (\prod_{x=1}^{l_j} \beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}])$.

Based on the above, the base of computing $\beta^{\mathcal{P}}_{a_s a_t}$ is the formal definition of $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$. Next, we firstly give a meta path-based BC framework on HINs. Then we will respectively propose the coarse-grained and fine-grained formal definitions of



(a) HIN topology under P=(AMDMA) (b) Compute Fine-grained $\beta_{a_{u1}, a_v}[a_{u1}, a_v]$ (c) Compute Fine-grained $\beta_{a_s, a_v}[a_{u1}, a_v]$

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

$\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$ so as to introduce the ***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:*

$$C_B^{\mathcal{P}}(a_v) = \sum_{a_s \neq a_v \neq a_t \in V, \phi_V(a_s) = \phi_V(a_v) = \phi_V(a_t) = A} \frac{\beta^{\mathcal{P}}_{a_s a_t}(a_v)}{\beta^{\mathcal{P}}_{a_s a_t}} \quad (1)$$

where $\beta^{\mathcal{P}}_{a_s a_t}$ 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, $\beta^{\mathcal{P}}_{a_s a_s} = 1$.

Coarse-grained vs Fine-grained BC on HINs. Let $\Gamma_{a_x, a_{x+1}}$ be the set of path instances between a $\mathcal{P}$-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 influence 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 $\mathcal{P}$-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., $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}] \propto |\Gamma_{a_x, a_{x+1}}|$. 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(a), coarse-grained $\beta^{\mathcal{P}}_{a_{u_1} a_v}[a_{u_1}, a_v] = \beta^{\mathcal{P}}_{a_s a_v}[a_{u_1}, a_v] = |\Gamma_{a_{u_1} a_v}| = 3$.

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 influence 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 firstly only consider the influence of the vertices with type D (more complex situations are discussed in Section 5), then the communication from a_s to a_t based on the path instances between a $\mathcal{P}$-pair (a_x, a_{x+1}) is more robust if:
i) More path instances in $\Gamma_{a_x, a_{x+1}}$, i.e., $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}] \propto |\Gamma_{a_x, a_{x+1}}|$.
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 $pins$ is participating in the communication from a_s to a_t. When a_s communicates to a_t along a shortest $\mathcal{P}$-path p^i, each $pins$ contained in p^i is interfered by the following path instances (Let $I^d_{a_s}(pins)$ be the set of all such path instances, including $pins$) : such path instances are contained in other shortest $\mathcal{P}$-paths whose sources are all a_s, and such path instances and $pins$ pass through the same vertex d with type D. For example, in Fig. 2(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(b) (Fig. 2(c) resp.). Thus, $|I^{d_3}_{a_{u_1}}(pins_1)| = 3$, $|I^{d_3}_{a_s}(pins_1)| = 4$. W.l.o.g., assume that the interference of the path instances in $I^d_{a_s}(pins)$ on each other is equivalent, so the independence of each $pins$ is denoted by $1/|I^d_{a_s}(pins)|$. Then we have $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}] \propto 1/|I^d_{a_s}(pins)|$.

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

Let $|V_D|$ be the number of all vertices with type D on $\mathbb{G}$, then $\sum_{pins\in\Gamma_{a_x,a_{x+1}}} 1/|I^d_{a_s}(pins)| \in (0, |V_D|]$, and $|D^{\mathcal{P}}_{a_x,a_{x+1}}| \in [1, |V_D|]$. Thus, we define fine-grained $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$ as follows.

$$\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}] = \Big(\sum_{pins\in\Gamma_{a_x,a_{x+1}}} \frac{1}{|I^d_{a_s}(pins)|}\Big) + |D^{\mathcal{P}}_{a_x,a_{x+1}}| \quad (2)$$

In such a case, **we call the BC in Eq. (1) as the fBC**. Note that the definition of $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$ in Eq. (2) 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) accordingly. Even though the form of Eq. (2) is changed, our proposed algorithms and optimization strategies could still be applicable only with simple modifications. For cBC, $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}] = \beta^{\mathcal{P}}_{a_t a_s}[a_x, a_{x+1}]$, so $\beta^{\mathcal{P}}_{a_s a_t} = \beta^{\mathcal{P}}_{a_t a_s}$ $(a_s \neq a_t)$; for fBC, $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}] \neq \beta^{\mathcal{P}}_{a_t a_s}[a_x, a_{x+1}]$, so $\beta^{\mathcal{P}}_{a_s a_t} \neq \beta^{\mathcal{P}}_{a_t a_s}$ $(a_s \neq a_t)$.

Problem 1 (Meta Path-based BC Computation, MBCC for short). Given an HIN $\mathbb{G}$, a meta path $\mathcal{P}$ (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 first key step is to find all shortest $\mathcal{P}$-paths. However, each path instance of $\mathcal{P}$ is often contained in many shortest $\mathcal{P}$-paths. Thus, while finding all shortest $\mathcal{P}$-paths in an HIN $\mathbb{G}$, each path instance will be visited many times. To reduce the time of visiting each path instance from $O(|\mathcal{P}|)$ to $O(1)$, we build a **$\mathcal{P}$-multigraph** $G_{\mathcal{P}}$ from $\mathbb{G}$ in advance so that each path instance between a $\mathcal{P}$-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 $\mathcal{P}$-paths on $\mathbb{G}$ is equivalent to finding all shortest paths on $G_{\mathcal{P}}$. Furthermore, computing cBC or fBC for all vertices with type A on $\mathbb{G}$ is equivalent to computing cBC or fBC for all vertices on $G_{\mathcal{P}}$.

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

Definition 3. ***($\mathcal{P}$-multigraph)*** *Given an HIN $\mathbb{G}$ and a meta path $\mathcal{P}$, the $\mathcal{P}$-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 $V_{\mathcal{P}}$ are all vertices with type A on $\mathbb{G}$, and each edge in $E_{\mathcal{P}}$ between two vertices a_v and a_u on $G_{\mathcal{P}}$ corresponds to a path instance between a $\mathcal{P}$-pair (a_v, a_u) on $\mathbb{G}$.*

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

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

To build $G_{\mathcal{P}}$ from $\mathbb{G}$, 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_j on $G_{\mathcal{P}}$). Since $\mathcal{P}$ is symmetric, $F_{\mathcal{P}} = F_{l(\mathcal{P})} F^T_{l(\mathcal{P})}$ where $F_{l(\mathcal{P})}$ is the commuting matrix for $l(\mathcal{P})$ (i.e., the left half meta path of $\mathcal{P}$).

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 $\mathcal{P}$-pair on the HIN $\mathbb{G}$. A path from a_s to a_t on $G_{\mathcal{P}}$, denoted by $a_s \rightsquigarrow a_t$, is a sequence of vertices such that each two adjacent vertices are linked by at least one edge on $G_{\mathcal{P}}$. A shortest $a_s \rightsquigarrow a_t$ path is the $a_s \rightsquigarrow a_t$ 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 $\mathbb{G}$ corresponds to a shortest $a_s \rightsquigarrow a_t$ path on $G_{\mathcal{P}}$. Thus, $\beta^{\mathcal{P}}_{a_s a_t} = \beta^{G_{\mathcal{P}}}_{a_s a_t}$, $\beta^{\mathcal{P}}_{a_s a_t}(a_v) = \beta^{G_{\mathcal{P}}}_{a_s a_t}(a_v)$, and $C^{\mathcal{P}}_B(a_v) = C^{G_{\mathcal{P}}}_B(a_v)$. 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 $\mathcal{P}$-Multigraph. Let $\delta_{a_s a_t}(a_v) = \beta^{G_{\mathcal{P}}}_{a_s a_t}(a_v)/\beta^{G_{\mathcal{P}}}_{a_s a_t}$ be the **pair dependency** of a_s and a_t on $G_{\mathcal{P}}$, then Eq. (1) is equivalently written as $C^{G_{\mathcal{P}}}_B(a_v) = \sum_{a_s\neq a_v\neq a_t\in V_{\mathcal{P}}} \delta_{a_s a_t}(a_v)$. 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 $O(n^3_{\mathcal{P}})$ 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 $G_{\mathcal{P}}$. Then Eq. (1) is equivalently written as $C^{G_{\mathcal{P}}}_B(a_v) = \sum_{a_s\neq a_v\in V_{\mathcal{P}}} \delta_{a_s\bullet}(a_v)$. 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-first search directed acyclic graph (**BFS DAG**) of a_s on $G_{\mathcal{P}}$ (the correctness is held by Theorem 1 and Theorem 2).

Definition 5. ***(BFS DAG)*** *Given a $\mathcal{P}$-multigraph $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ and a vertex $a_s \in V_{\mathcal{P}}$, the BFS DAG of a_s on $G_{\mathcal{P}}$ denoted by $B^{G_{\mathcal{P}}}_{a_s} = (V^B_{a_s}, E^B_{a_s})$ $(V^B_{a_s} \subseteq V_{\mathcal{P}}, E^B_{a_s} \subseteq E_{\mathcal{P}})$ is a directed acyclic graph which represents all shortest paths from a_s to all other reachable vertices as discovered by conducting the BFS algorithm on $G_{\mathcal{P}}$, where $V^B_{a_s}$ includes all vertices reachable from a_s on $G_{\mathcal{P}}$, and $E^B_{a_s}$ 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 $\beta^{G_{\mathcal{P}}}_{a_s a_v}$ for each $a_v \in V_{\mathcal{P}}$ by Eq. (3), 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). **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 $a_s \in V_{\mathcal{P}}(a_s \neq a_v)$.

Theorem 1. *While conducting BFS from a_s on $G_{\mathcal{P}}$ (i.e., building the BFS DAG of a_s), $\beta^{G_{\mathcal{P}}}_{a_s a_v}$ $(a_s \neq a_v \in V_{\mathcal{P}})$ can be computed recursively as follows (initially, $\beta^{G_{\mathcal{P}}}_{a_s a_s} = 1$):*

$$\beta^{G_{\mathcal{P}}}_{a_s a_v} = \sum_{a_u\in Pred_{a_s}(a_v)} \beta^{G_{\mathcal{P}}}_{a_s a_u} \times \beta^{G_{\mathcal{P}}}_{a_s a_v}[a_u, a_v] \quad (3)$$

Proof. $pg^j_{a_sa_u} \circ (a_u, a_v)$ *means that each shortest* $\mathcal{P}$*-path in* $pg^j_{a_sa_u}$ *correspondingly concatenates to each path instance from* a_u *to* a_v*. By the additivity and multiplicativity properties in Subsection 2,*

$$\beta^{G_\mathcal{P}}_{a_sa_v} = \sum_{pg^j \subseteq \Psi_{a_sa_v}} \beta^{G_\mathcal{P}}_{a_sa_v}[pg^j]$$
$$= \sum_{a_u \in Pred_s(a_v)} \left(\sum_{pg^j \subseteq \Psi_{a_sa_u}} \beta^{G_\mathcal{P}}_{a_sa_v}[pg^j \circ (a_u, a_v)] \right)$$
$$= \sum_{a_u \in Pred_s(a_v)} \left(\sum_{pg^j \subseteq \Psi_{a_sa_u}} \beta^{G_\mathcal{P}}_{a_sa_v}[pg^j] \times \beta^{G_\mathcal{P}}_{a_sa_v}[a_u, a_v] \right)$$
$$= \sum_{a_u \in Pred_{a_s}(a_v)} \left(\beta^{G_\mathcal{P}}_{a_sa_u} \times \beta^{G_\mathcal{P}}_{a_sa_v}[a_u, a_v] \right) \quad \blacksquare$$

To compute $\beta^{G_\mathcal{P}}_{a_sa_v}$ by Eq. (3), $\beta^{G_\mathcal{P}}_{a_sa_v}[a_u, a_v]$ for each $\mathcal{P}$-pair (a_u, a_v) on the BFS DAG of a_s should be acquired in advance. For cBC, $\beta^{G_\mathcal{P}}_{a_sa_v}[a_u, a_v] = F_\mathcal{P}[a_u, a_v]$ (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 $\mathcal{P}$-pair (a_u, a_v) to package the information which will be used for computing $\beta^{G_\mathcal{P}}_{a_sa_v}[a_u, a_v]$ in the step (2) of Basic, including $|D^\mathcal{P}_{a_u,a_v}|$, 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): firstly, check $EI(a_u, a_v)$ to compute $I^d_{a_s}(pins)$; secondly, compute $\beta^{G_\mathcal{P}}_{a_sa_v}[a_u, a_v]$ with $|D^\mathcal{P}_{a_u,a_v}|$ and $I^d_{a_s}(pins)$ by Eq. (2), and compute $\beta^{G_\mathcal{P}}_{a_sa_v}$ by Eq. (3).

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:*

$$\delta_{a_s\bullet}(a_v) = \sum_{a_w : a_v \in Pred_{a_s}(a_w)} \frac{\beta^{G_\mathcal{P}}_{a_sa_v} \times \beta^{G_\mathcal{P}}_{a_sa_w}[a_v, a_w]}{\beta^{G_\mathcal{P}}_{a_sa_w}} \times (1 + \delta_{a_s\bullet}(a_w)) \tag{4}$$

Proof. *On the BFS DAG of* a_s*, if* a_t *is not a descendant of* a_v*, then all shortest* $\mathcal{P}$*-paths from* a_s *to such an* a_t *would not pass through* a_v*, i.e.,* $\beta^{G_\mathcal{P}}_{a_sa_t}(a_v) = 0$*. Let* $Dc(a_v)$ *be the set of all descendants of* a_v *on the BFS DAG of* a_s*, then* $\delta_{a_s\bullet}(a_v) = \sum_{a_t \in Dc(a_v)} \frac{\beta^{G_\mathcal{P}}_{a_sa_t}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_t}}$*. Divide* $Dc(a_v)$ *into two subsets, one is the set of all children of* a_v *(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* $\sum_{a_t \in Dc(a_v)} \frac{\beta^{G_\mathcal{P}}_{a_sa_t}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_t}} =$

$$\sum_{a_w \in Cl(a_v)} \frac{\beta^{G_\mathcal{P}}_{a_sa_w}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_w}} + \sum_{a_t \in Dc(a_v) \setminus Cl(a_v)} \frac{\beta^{G_\mathcal{P}}_{a_sa_t}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_t}}.$$

Based on the additivity and multiplicativity properties, $\frac{\beta^{G_\mathcal{P}}_{a_sa_t}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_t}} =$

$$\frac{\sum_{a_w \in Cl(a_v)} \beta^{G_\mathcal{P}}_{a_sa_t}(a_v, a_w)}{\beta^{G_\mathcal{P}}_{a_sa_t}} = \frac{\sum_{a_w \in Cl(a_v)} \beta^{G_\mathcal{P}}_{a_sa_v} \cdot \beta^{G_\mathcal{P}}_{a_va_w} \cdot \beta^{G_\mathcal{P}}_{a_wa_t}}{\beta^{G_\mathcal{P}}_{a_sa_t}}$$
$$= \sum_{a_w \in Cl(a_v)} \frac{\beta^{G_\mathcal{P}}_{a_sa_v} \cdot \beta^{G_\mathcal{P}}_{a_va_w}}{\beta^{G_\mathcal{P}}_{a_sa_w}} \times \frac{\beta^{G_\mathcal{P}}_{a_sa_w} \cdot \beta^{G_\mathcal{P}}_{a_wa_t}}{\beta^{G_\mathcal{P}}_{a_sa_t}}.$$

Then we have $\delta_{a_s\bullet}(a_v) = \sum_{a_w : a_v \in Pred_s(a_w)} \frac{\beta^{G_\mathcal{P}}_{a_sa_w}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_w}}$

$$+ \sum_{a_t \in Dc(a_v) \setminus Cl(a_v)} \sum_{a_w : a_v \in Pred_s(a_w)} \frac{\beta^{G_\mathcal{P}}_{a_sa_v} \cdot \beta^{G_\mathcal{P}}_{a_va_w}}{\beta^{G_\mathcal{P}}_{a_sa_w}} \times \frac{\beta^{G_\mathcal{P}}_{a_sa_w} \cdot \beta^{G_\mathcal{P}}_{a_wa_t}}{\beta^{G_\mathcal{P}}_{a_sa_t}}$$
$$= \sum_{a_w : a_v \in Pred_s(a_w)} \frac{\beta^{G_\mathcal{P}}_{a_sa_w}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_w}} \times (1 + \sum_{a_t \in Dc(a_v) \setminus Cl(a_v)} \frac{\beta^{G_\mathcal{P}}_{a_sa_w} \cdot \beta^{G_\mathcal{P}}_{a_wa_t}}{\beta^{G_\mathcal{P}}_{a_sa_t}})$$
$$= \sum_{a_w : a_v \in Pred_s(a_w)} \frac{\beta^{G_\mathcal{P}}_{a_sa_w}(a_v)}{\beta^{G_\mathcal{P}}_{a_sa_w}} \times (1 + \delta_{a_s\bullet}(a_w)). \quad \blacksquare$$

Fig. 3 gives the process of computing $\beta^{G_\mathcal{P}}_{a_sa_v}$ and $\delta_{a_s\bullet}(a_v)$ on the BFS DAG of a_s (the HIN topology is in Fig. 2(a)). The pseudo-code

Algorithm 1: Basic($\mathbb{G}$, $\mathcal{P}$, A, D)

```
Build the P-multigraph G_P = (V_P, E_P) from the HIN G, and record EI(a_u, a_v) for
  each P-pair (a_u, a_v);
C_B[a_v] ← 0, a_v ∈ V_P;
for a_s ∈ V_P do
    the queue Q ← ∅, the stack S ← ∅; Enqueue a_s → Q;
    β[a_t] ← 0, Pred(a_t) ← empty list, Dist[a_t] ← ∞, a_t ∈ V_P;
    β[a_s] ← 1, Dist[a_s] ← 0; |I^d_{a_s}| ← 0, d ∈ V_D;
    while Q is not empty do // first BFS
        Dequeue a_u ← Q; push a_u → S;
        for each a_v ∈ Nei_{G_P}(a_u) do
            if Dist[a_v] = ∞ then
                Dist[a_v] ← Dist[a_u] + 1; Enqueue a_v → Q;
            if Dist[a_v] = Dist[a_u] + 1 then
                for each pins ∈ (a_u, a_v) do
                    Get d of pins from EI(a_u, a_v); |I^d_{a_s}| + +;
    while Q is not empty do // second BFS
        Dequeue a_u ← Q;
        for each a_v ∈ Nei_{G_P}(a_u) do
            if Dist[a_v] = Dist[a_u] + 1 then
                for each pins ∈ (a_u, a_v) do
                    Get d of pins from EI(a_u, a_v);
                    β[a_u, a_v] += 1/|I^d_{a_s}|;
                β[a_u, a_v] ← β[a_u, a_v] + |D_{a_u,a_v}|;
                β[a_v] += β[a_u]×β[a_u, a_v];
                Append (a_u, β[a_u, a_v]) → Pred(a_v);
    δ[a_v] ← 0, a_v ∈ V_P;
    while S is not empty do // reverse BFS
        Pop a_w ← S;
        for (a_v, β[a_v, a_w]) ∈ Pred(a_w) do
            δ[a_v] += β[a_v]×β[a_v,a_w] / β[a_w] (1 + δ[a_w]);
        if a_w ≠ a_s then
            C_B[a_w] += δ[a_w];
return C_B;
```

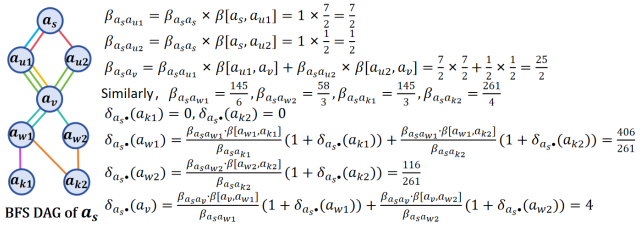


Figure 3: Example of Computing $\beta^{G_\mathcal{P}}_{a_sa_v}$ and $\delta_{a_s\bullet}(a_v)$.

of Basic for computing fBC is shown in Alg. 1. For computing cBC, it just needs to remove the lines about computing $\beta^{G_\mathcal{P}}_{a_sa_v}[a_u, a_v]$.

Complexity Analysis. In the step (1) of Basic, for $\mathcal{P} = (A_1A_2 \ldots A_{(l+1)/2} \ldots A_lA_{l+1})$, it costs the time of $O(|A_1||A_2| \ldots |A_{(l+1)/2}| + |A_1||A_{(l+1)/2}| + |A_1||A_{(l+1)/2}||A_1|)$ 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 $n_\mathcal{P} = |A_1|$ ($m_\mathcal{P}$ 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 $O(n_\mathcal{P} \times \overline{m}_\mathcal{P})$ for cBC

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

4 OPTIMIZATION STRATEGIES

The bottleneck of Basic is step (2). There are two main parameters which affect the CPU time of step (2): firstly, the number of BFS DAGs actually built, i.e., $n_{\mathcal{P}}$; 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 $\mathcal{P}$-Multigraph compression strategy to reduce $n_{\mathcal{P}}$ and $\overline{m}_{\mathcal{P}}$ (Subsections 4.1), and the BFS DAG sharing strategy to further reduce $n_{\mathcal{P}}$ (Subsections 4.2). Since the definitions of $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}]$ for cBC and fBC are different, the optimization strategies for cBC and fBC are also different. So in each subsection, we introduce optimization strategies for cBC and fBC separately.

4.1 Compressing $\mathcal{P}$-Multigraph

4.1.1 **Compressing $G_{\mathcal{P}}$ 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 $G_{\mathcal{P}}$ since $C_B^{G_{\mathcal{P}}}(a_v) = 0$. We call such a_v as a side vertex and give two kinds of definitions 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 $G_{\mathcal{P}}$, 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 $C_B^{G_{\mathcal{P}}}(a_u) = C_B^{G_{\mathcal{P}}}(a_v)$. We call such a_u and a_v as identical vertices and give two kinds of definitions of identical vertices (**1-identical vertices** and **2-identical vertices**) for cBC. By merging each group of identical vertices as one vertex, $G_{\mathcal{P}}$ would also be compressed.

Graph Compression with Side Vertices. Firstly, we give the definition of the **1-side vertex** by extending the definition of the side vertex from an homogeneous network[43] to the $\mathcal{P}$-multigraph. Secondly, to reduce the time of identifying side vertices for cBC, we propose a relaxed definition of the 1-side vertex, i.e., the **2-side vertex**, which is defined directly on the HIN.

Definition 6. ***(1-Side Vertex)*** *Given the $\mathcal{P}$-multigraph $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$, 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 $Nei_{G_{\mathcal{P}}}(a_i) \bigcup \{a_i\}$.*

Definition 7. ***(2-Side Vertex)*** *Given an HIN $\mathbb{G}$ and a symmetric meta path $\mathcal{P}$ (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 $l(\mathcal{P})$-pair with a_i on $\mathbb{G}$ ($l(\mathcal{P})$ is the left half meta path of $\mathcal{P}$).*

For example, in Fig. 4(a), a_{x_1}, a_{x_2} and a_{x_3} are 1-side vertices. In Fig. 4(b), a_{x_1} and a_{x_2} are 2-side vertices.

Remark 1. Comparing Def. 6 with Def. 7, 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), it indicates that in most cases, using 2-side vertices is better, because identifying 2-side vertices is faster, and the compression effects of 1-side vertices and 2-side vertices are very close.

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

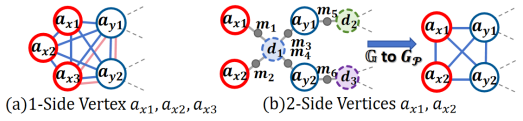


Figure 4: Example of Side Vertices for cBC.

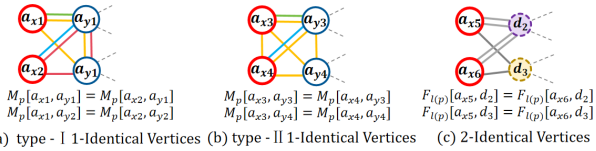


Figure 5: Example of Identical Vertices for cBC.

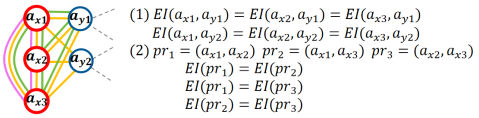


Figure 6: Example of Identical Vertices for fBC.

Graph Compression with Identical Vertices. Firstly, we introduce the definition of **the 1-identical vertices** which is extended from an homogeneous network[43] to the $\mathcal{P}$-multigraph. Secondly, to speed up the time of finding identical vertices, we give a relaxed definition of the 1-identical vertices, called **the 2-identical vertices**, which is defined on the HIN directly.

Definition 8. ***(1-Identical Vertices)*** *Given the $\mathcal{P}$-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 $Nei_{G_{\mathcal{P}}}(a_1) \bigcup \{a_1\} = Nei_{G_{\mathcal{P}}}(a_2) \bigcup \{a_2\}$), and for any $a_u \in Nei_{G_{\mathcal{P}}}(a_1) \bigcap Nei_{G_{\mathcal{P}}}(a_2)$ $(a_u \neq a_1, a_2)$, $M_{\mathcal{P}}[a_1, a_u] = M_{\mathcal{P}}[a_2, a_u]$.*

Definition 9. ***(2-Identical Vertices)*** *Given an HIN $\mathbb{G}$ and a symmetric meta path $\mathcal{P}$ (whose symmetry point type is D), two vertices a_1 and a_2 both with type A are 2-identical on $\mathbb{G}$ iff $F_{l(\mathcal{P})}[a_1, d_i] = F_{l(\mathcal{P})}[a_2, d_i]$ for each $d_i (d_i \in D)$.*

For example, in Fig. 5(a), a_{x_1} and a_{x_2} are type-I 1-identical vertices. In Fig. 5(b), a_{x_3} and a_{x_4} are type-II 1-identical vertices. In Fig. 5(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), 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.

4.1.2 **Compressing $G_{\mathcal{P}}$ for fBC.** For fBC, $\beta^{\mathcal{P}}_{a_s a_t} \neq \beta^{\mathcal{P}}_{a_t a_s}$, 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 definition of **identical vertices** for fBC which can be merged as one vertex on $G_{\mathcal{P}}$ so that $G_{\mathcal{P}}$ would be compressed.

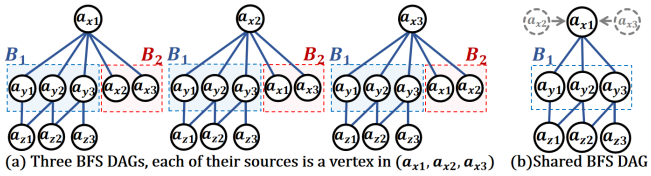
Graph Compression with Identical Vertices. In the following, we give the new definition of identical vertices for fBC on $G_{\mathcal{P}}$.

Definition 10. ***(Identical Vertices)*** *Given the $\mathcal{P}$-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) $Nei_{G_{\mathcal{P}}}(a_1) \bigcup \{a_1\} = Nei_{G_{\mathcal{P}}}(a_2) \bigcup \{a_2\}$, and for any a neighbor $a_u \in Nei_{G_{\mathcal{P}}}(a_1) \bigcap Nei_{G_{\mathcal{P}}}(a_2)$ $(a_u \neq a_1, a_2)$, $EI(a_1, a_u) = EI(a_2, a_u)$.

Algorithm 2: SdAdvCBC($\mathbb{G}$, $\mathcal{P}$, A)

Identify all 2-side vertices and divide them into different *same_side_sets*;
The same as lines 1-2 in Alg. 1;
for *each same_side_set* **do**
  CBC_SameSide(*same_side_set*);
  $G_{\mathcal{P}} \leftarrow$ Delete *same_side_set* from $G_{\mathcal{P}}$;
Basic($G_{\mathcal{P}}$, $\mathcal{P}$, A);
return $(C_B[a_w] + C'_B[a_w])$ for each $a_w \in V_{\mathcal{P}}$;

procedure CBC_SameSide(*same_side_set*)
the queue $Q \leftarrow \emptyset$, the stack $S \leftarrow \emptyset$;
$Pred(a_t) \leftarrow$ empty list, $Dist[a_t] \leftarrow \infty$, $a_t \in V_{\mathcal{P}}$;
for $a_i \in$ *same_side_set* **do**
  for $a_t \in V_{\mathcal{P}}$ **do**
    $\beta_{a_i}[a_t] \leftarrow 0$; $\beta_{a_i}[a_i] \leftarrow 1$; $Dist[a_i] \leftarrow 0$;
Proxy $a_s \leftarrow$ *same_side_set*[0]; Push $a_s \rightarrow S$;
for *each* $a_v \in Nei_{G_{\mathcal{P}}}(a_s)$ *but* $a_v \notin$ *same_side_set* **do**
  The same as lines 10-11 in Alg. 1 (replace a_u with a_s);
  if $Dist[a_v] = Dist[a_s] + 1$ **then**
    for $a_i \in$ *same_side_set* **do**
      $\beta_{a_i}[a_v] += \beta_{a_i}[a_i] \times F_{\mathcal{P}}[a_i, a_v]$;
    Append $a_s \rightarrow Pred(a_v)$;
while Q *is not empty* **do**
  Dequeue $a_u \leftarrow Q$; push $a_u \rightarrow S$;
  for *each* $a_v \in Nei_{G_{\mathcal{P}}}(a_u)$ **do**
    The same as lines 10-11 in Alg. 1;
    if $Dist[a_v] = Dist[a_u] + 1$ **then**
      for $a_i \in$ *same_side_set* **do**
        $\beta_{a_i}[a_v] += \beta_{a_i}[a_u] \times F_{\mathcal{P}}[a_u, a_v]$;
      Append $a_u \rightarrow Pred(a_v)$;
for $a_i \in$ *same_side_set* **do**
  $\delta_{a_i}[a_v] \leftarrow 0$, $a_v \in V_{\mathcal{P}}$;
while S *is not empty* **do**
  Pop $a_w \leftarrow S$;
  for $a_v \in Pred(a_w)$ **do**
    for $a_i \in$ *same_side_set* **do**
      $\delta_{a_i}[a_v] += \frac{\beta_{a_i}[a_v] \times F_{\mathcal{P}}[a_v, a_w]}{\beta_{a_i}[a_w]} (1 + \delta_{a_i}[a_w])$;
  if $a_w \neq a_s$ **then**
    for $a_i \in$ *same_side_set* **do**
      $C'_B[a_w] \leftarrow C'_B[a_w] + \delta_{a_i}[a_w] \times 2$;
return C'_B;



(a) Three BFS DAGs, each of their sources is a vertex in (a_{x1}, a_{x2}, a_{x3}) (b)Shared BFS DAG

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 $\mathcal{P}$-pairs pr_i and pr_j whose vertices are in the iden_set, $EI(pr_i) = EI(pr_j)$.

For example, in Fig. 6, a_{x_1}, a_{x_2} and a_{x_3} are identical vertices.

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.

4.2.1 **Sharing BFS DAGs for cBC.** By Observation 1 in Subsection 4.1.1, 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 $G_{\mathcal{P}}$ in batch.

SdAdvCBC Algorithm. For each vertex a_w on $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$, 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 $\delta_{a_j \bullet}(a_w)$ for all non-2-side vertex $a_j \in V_{\mathcal{P}}$. The pseudo-code of SdAdvCBC is in Alg. 2.

SdAdvCBC contains four steps. Firstly, identify all 2-side vertices and divide them into different *same_side_sets* based on their $l(\mathcal{P})$-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 $G_{\mathcal{P}}$, 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 $G_{\mathcal{P}}$. 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 $\delta_{a_{x_i} \bullet}(a_v)$. In total, three BFS DAGs would be built (in Fig. 7(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_1s as well as their descendants are completely the same; (2) all B_2s have no descendants so that $\delta_{a_{x_i} \bullet}(a_v)$ ($a_v \in B_2$, 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(a) into one shared BFS DAG in Fig. 7(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 B_1 and the descendants of the B_1), but can compute $\beta^{\mathcal{P}}_{a_{x_i} a_v}$ 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 $\delta_{a_{x_i} \bullet}(a_v)$ 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).

4.2.2 **Sharing BFS DAGs for fBC.** By Def. 10, each set of identical vertices (called an *iden_set*) have the same neighborhood information, so we present the identical vertices-based advanced

Algorithm 3: FBC_Identical(a_s)

1: the queue $Q \leftarrow \emptyset$, the stack $S \leftarrow \emptyset$; Enqueue $a_s \rightarrow Q$;
2: $\beta_{a_s}[a_t] \leftarrow 0$, $Pred[a_t] \leftarrow$ empty list, $Dist[a_t] \leftarrow \infty$, $a_t \in V_{\mathcal{P}}$;
3: $\beta_{a_s}[a_s] \leftarrow 1$, $Dist[a_s] \leftarrow 0$; $|I^d_{a_s}| \leftarrow 0, d \in V_D$;
4: **while** Q *is not empty* **do**
5: Dequeue $a_u \leftarrow Q$; push $a_u \rightarrow S$;
6: **The same as lines 9-14 in Alg. 1** (replace $|I^d_{a_s}|$ in line14 in Alg. 1 with $|I^d_{a_s}| += 1 \times (1 + ident[a_v])$);
7: **if** $a_u = a_s$ **then**
8: **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 $EI(a_u, a_{ui})$; $|I^d_{a_s}|$++;
11: **The same as lines 15-24 in Alg. 1** (replace line23 in Alg. 1 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: $\delta_{a_s}[a_v] += \frac{\beta_{a_s}[a_v] \times \beta[a_v, a_w]}{\beta_{a_s}[a_w]} \times (1 + \delta_{a_s}[a_w]) \times (1 + ident[a_w])$;
17: **if** $a_w \neq a_s$ **then**
18: **for** $a_{wi} \in iden\_set[a_w]$ **do**
19: $C'_B[a_{wi}] += \delta_{a_s}[a_w] \times (1 + ident[a_s])$;
20: **return** C'_B;

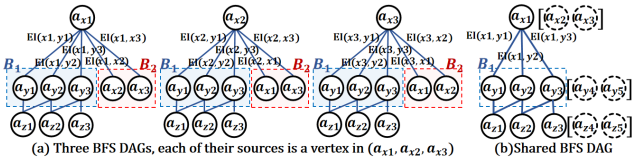


Figure 8: Sharing a BFS DAG for an *iden_set* $(a_{x_1}, a_{x_2}, a_{x_3})$.

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, 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 $\delta_{a_j \bullet}(a_w)$ of all a_j where a_j 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(a) into one shared BFS DAG in Fig. 8(b) while computing source dependencies. Note that all B_2s 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. Specifically, FBC_Identical works as follows (the pseudo-code in Alg. 3): (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 $EI(a_{x_1}, a_{x_j})$ ($j = 2$ or 3), so as to compute $I_{a_{x_j}}(pins)$ (each *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 $I_{a_{x_1}}(pins)$ (each *pins* is between the remaining levels of the shared BFS DAG). (2) Perform one BFS on the shared BFS DAG to compute $\beta^{G_{\mathcal{P}}}_{a_{x_1} a_v}[a_u, a_v]$ and $\beta^{\mathcal{P}}_{a_{x_1} a_v}$. (3) Perform one reverse BFS on the shared BFS DAG to compute $\delta_{a_{x_1} \bullet}(a_v)$.

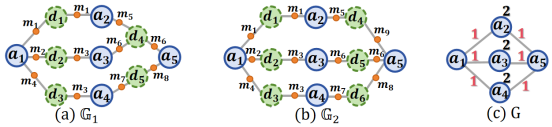


Figure 9: HINs vs. Weighted Homogeneous Networks

Remark 4. Based on Def. 10, $\beta^{\mathcal{P}}_{a_{x_1} a_v} = \beta^{\mathcal{P}}_{a_{x_2} a_v} = \beta^{\mathcal{P}}_{a_{x_3} a_v}$, $\delta_{a_{x_1} \bullet}(a_v) = \delta_{a_{x_2} \bullet}(a_v) = \delta_{a_{x_3} \bullet}(a_v)$. Thus, FBC_Identical only needs to compute $\beta^{\mathcal{P}}_{a_{x_1} a_v}$ 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 $ident[a_f] + 1$ 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] 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 omit the similar details here for space limitation. The performance 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]).*

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) 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 $G''_{\mathcal{P}}$, then the worst time complexity of FBC_Identical (Alg. 3) is $O(\overline{m}''_{\mathcal{P}} \times l_{max})$, and the worst time complexity of IdAdvFBC reduces to $O(n''_{\mathcal{P}} \times \overline{m}''_{\mathcal{P}} \times l_{max})$ compared with the step (2) of Basic.

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(a) and Fig. 9(b) show two different movie HINs, but would be transformed to the same weighted homogeneous network in Fig. 9(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(b), a_2, a_3 and a_4 are equally important. However, the difference between Fig. 9(a) and Fig. 9(b) is missed when

considering BC in Fig. 9(c), since Fig. 9(c) loses the topology information which the vertices with type D participate. Secondly, for a symmetric meta path $\mathcal{P}$, 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 $\mathcal{P}$. Let c be the number of times which Q appears on $\mathcal{P}$, then we just need to modify Eq. (2) into $\beta^{\mathcal{P}}_{a_s a_t}[a_x, a_{x+1}] = \left(\sum_{pins \in \Gamma_{a_x, a_{x+1}}} \frac{1}{\sum_{i=1}^{c} (1/c) \times I^{q_i}_{a_s}(pins)}\right) + |Q^{\mathcal{P}}_{a_x, a_{x+1}}|$, where q_i is the ith type-Q vertex on the path instance $pins$. When $\mathcal{P}$ is asymmetric, if Q only appears once on $\mathcal{P}$, Eq. (2) still holds, while if Q appears multiple times on $\mathcal{P}$ (supposing c times), Eq. (2) is modified into the same form as above. When we simultaneously consider the influence of multiple different types of vertices whose types are not A to the communication from a_s to a_t, the modification to Eq. (2) 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.

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 effectiveness testing, we compare PathRank[32], influence spread[15], or structural diversity[23] rankings with cBC or fBC rankings for vertices on HINs. For efficiency testing, we compare our Basic algorithms with different optimization strategies for cBC and fBC (Table 2).

Datasets. We use four real datasets: Movies[4, 48], Yelp[6], DBLP[2, 49] and IMDb[3]. Table 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(b) shows the meta path $\mathcal{P}$ used in each dataset for efficiency evaluation. For IMDb, we extract four sub datasets with different sizes to variously test the efficiency of our algorithms.

6.1 Effectiveness 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 $\mathcal{P} = (AMDMA)$, Fig. 10(a) shows the $\mathcal{P}$-multigraph of the sub dataset. Computing cBC and fBC for all vertices with type A, it verifies 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 a red dashed circle). We also compute edge BC[13] of all edges and mark the edges whose edge BC rank at top-1 and top-2 in green in Fig. 10(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] are substantially different. One of our future work is to redefine and efficiently compute edge BC on HINs. Moreover, we compute PathRank, influence spread and structural diversity for vertices

Table 2: Summary of Algorithms.

Strategy	Description
BA	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)
SD2 (SdAdvCBC)	Acceleration with 2-Side Vertices (Sec. 4.1.1, Sec.4.2.1)
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+SL	BasF+BA+ID

Table 3: Statistics of Datasets.

(a)

Dataset	Vertices	Edges	Vertex types	Edge types	Num of frequently used meta paths
Movies	34,283	56,094	4	3	10
Yelp	9,129,970	15,039,821	4	4	10
DBLP	2,677,139	5,898,161	3	2	6
IMDb	1,586,997	2,602,969	4	3	10

(b)

Dataset	$\mathcal{P}$	$n_{\mathcal{P}}$	$\overline{m_{\mathcal{P}}}$
Movies	AMDMA	10,128	352,600
Yelp	BRURB	16,000	1,321,978
DBLP	APVPA	18,275	32,247,867
IMDb	AMDMA	(1) 15,868 (2) 19,225 (3) 32,426 (4) 100,037	(1) 2,193,938 (2) 2,607,706 (3) 3,881,039 (4) 6,550,895

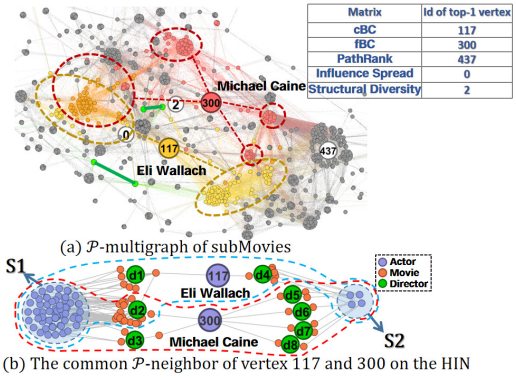


Figure 10: Case Study on Movies.

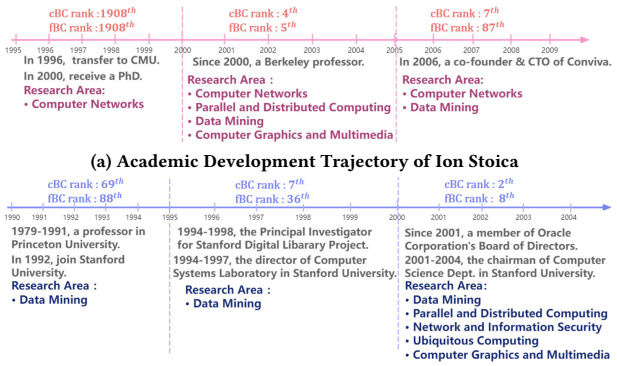


Figure 11: Case Study on DBLP.

with type A. Vertices rankings under PathRank, influence spread or structural diversity are completely different from their cBC or fBC rankings. Furthermore, Fig. 10(b) shows the heterogeneous subgraph that contains the common $\mathcal{P}$-neighbors of vertex 117 (Eli Wallach, short for EW) and vertex 300 (Michael Caine, short for MC). Compared with EW, the shortest $\mathcal{P}$-paths from actors in S_1 to 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 films spanning various genres which were directed by many directors. We asked ChatGPT[1] "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 influence, 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 different.

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 $\mathcal{P}$ = ($APVPA$) 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, there are two interesting findings: (1) When the cBC or fBC of an author grows over time, his/her academic achievements are also gradually increasing. This can help to find 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 specific research area). This can help to find multi-field 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, 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 effects 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 1-identical vertices for cBC by testing their ID_Num, Set_Num and Time. As shown in Table 4, to reduce cBC computation time with

Table 4: Statistics of Side Vertices & Identical Vertices for cBC.

		Movies	IMDb(1)	Yelp	IMDb(3)	IMDb(4)	DBLP
SD1	SD_Num	3,209	9,324	525	18,489	66,409	15,675
	E_rmv_Num	72,016	661,815	6,007	1,049,859	2,177,980	30,060,433
	ident_Time (Sec.)	1.354	39.452	0.729	61.929	148.665	6918.62
	rmv_Time (Sec.)	49.01	755.092	40.868	2882.5	18597.9	33275.2
SD2	SD_Num	2,936	8,577	0	17,366	61,801	15,675
	Set_Num	1,097	1,070	0	2,817	11,962	9
	E_rmv_Num	64,557	656,580	0	1,039,920	2,147,938	30,060,433
	ident_Time (Sec.)	0.001	0.002	0.001	0.004	0.005	0.001
	rmv_Time (Sec.)	17.005	122.043	0	587.17	3976.51	4511.75
ID1 type-I	ID_Num	0	0	0	2	33	0
	Set_Num	0	0	0	1	11	0
	E_rmv_Num	0	0	0	1	34	0
	Time (Sec.)	8.603	168.101	89.944	348.147	467.45	18.49
ID1 type-II	ID_Num	2,207	7,335	8	14,265	49,778	17436
	Set_Num	890	1,212	6	2,800	10,625	315
	E_rmv_Num	66,908	608,699	385	933,596	1,934,527	32,054,019
	Time (Sec.)	0.178	4.947	0.293	6.951	39.22	1692.52
ID2	ID_Num	2,166	7,302	0	14,190	49,052	17,436
	Set_Num	908	1,222	0	2,822	10,813	315
	E_rmv_Num	66,100	608,025	0	931,715	1,926,716	32,054,019
	Time (Sec.)	0.159	4.25	0	6.544	7.677	1685.8

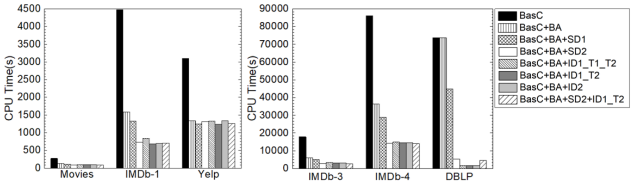
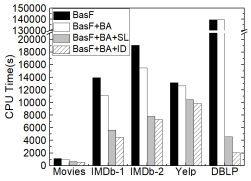


Figure 12: Comparison of Optimization Strategies as for BasC.

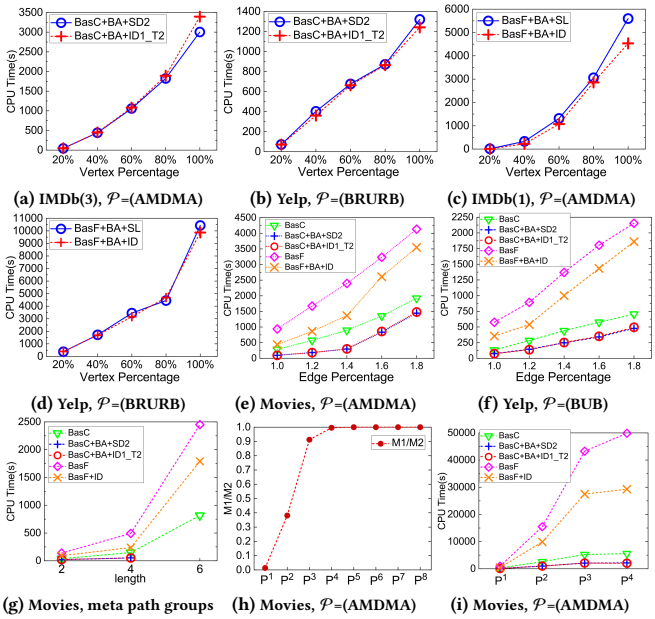
		Movies	IMDb(1)	IMDb(2)	Yelp	DBLP
SL	SL_Num	3292	9289	11337	17	18239
	SL_Set_Num	872	1160	1488	7	72
	ident_Time (Sec.)	0.014	0.08	0.089	0.048	4.601
ID	ID_Num	2205	7332	8840	8	17436
	ID_Set_Num	890	1214	1547	6	315
	E_rmv_Num	66882	608686	696482	385	32054019
	Time (Sec.)	0.219	4.402	5.659	0.055	1854.79

(a) Statistics of Similar Vertices & Identical Vertices for fBC



(b) Comparison of Optimization Strategies as for BasF.

Figure 13: Evaluating Optimization Strategies for fBC.



(a) IMDb(3), $\mathcal{P}$=(AMDMA) **(b) Yelp, $\mathcal{P}$=(BRURB)** **(c) IMDb(1), $\mathcal{P}$=(AMDMA)**

(d) Yelp, $\mathcal{P}$=(BRURB) **(e) Movies, $\mathcal{P}$=(AMDMA)** **(f) Yelp, $\mathcal{P}$=(BUB)**

(g) Movies, meta path groups **(h) Movies, $\mathcal{P}$=(AMDMA)** **(i) Movies, $\mathcal{P}$=(AMDMA)**

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

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, to evaluate the effects of different optimization strategies (listed in Table 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 effect 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 effect 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] on a dataset like DBLP, then BA would lose the acceleration effect. 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 effect 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(a), in most cases, only with very little ident_Time (Time resp.), lots of similar vertices (identical vertices resp.) can be identified (identified 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 effect 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(b).

Comparison of Optimization Strategies as for BasF. In Fig. 13 (b), to evaluate the effects of different optimization strategies (listed in Table 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 effect 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 effect 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(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(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(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 $\mathcal{P}$-multigraph. Secondly, for $\mathcal{P} = (AMDMA)$ in Movies, by repeating $\mathcal{P}$ different times (e.g., $\mathcal{P}^2 = (AMDMAMDMA)$), we compute $\frac{\overline{m}_{\mathcal{P}}}{m_C}$ 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(h), as $\mathcal{P}^k$ gets longer, $\frac{\overline{m}_{\mathcal{P}}}{m_C}$ 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 find 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(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 first to focus on a specific 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 significance of both cBC and fBC, and verify that our optimization strategies are practically effective, 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

[1] ChatGPT. last accessed:01/06/2024. https://chatgpt.com/
[2] DBLP. last accessed:01/06/2024. https://www.aminer.org/billboard/citation
[3] IMDb. last accessed:01/06/2024. https://www.IMDb.com/interfaces/
[4] Movies. last accessed:01/06/2024. https://www.aminer.cn/data-sna#Movie
[5] Technical report(password:HINbcVLDB2024). last accessed:01/06/2024. https://github.com/1ran/BccH
[6] Yelp. last accessed:01/06/2024. https://www.yelp.com/dataset/
[7] Jac M. Anthonisse. 1971. The rush in a directed graph. *J. Comput. Phys.* (1971), 1–10.
[8] Phiradet Bangcharoensap, Tsuyoshi Murata, Hayato Kobayashi, and Nobuyuki Shimizu. 2016. Transductive Classification 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.
[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.
[10] Ivona Bezáková and Andrew Searns. 2018. On Counting Oracles for Path Problems. In *ISAAC*, Vol. 123. 56:1–56:12.
[11] Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Antti Ukkonen. 2014. Distance oracles in edge-labeled graphs. In *EDBT*. 547–558.
[12] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of mathematical sociologyJ Math Sociol* 25, 2 (2001), 163–177.
[13] Ulrik Brandes. 2008. On variants of shortest-path betweenness centrality and their generic computation. *Soc. Networks* 30, 2 (2008), 136–145.
[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.
[15] Suqi Cheng, Huawei Shen, Junming Huang, Wei Chen, and Xueqi Cheng. 2014. Imrank: influence maximization via finding self-consistent ranking. In *SIGIR*. 475–484.
[16] Cyrus Cousins, Chloe Wohlgemuth, and Matteo Riondato. 2021. Bavarian: Betweenness Centrality Approximation with Variance-Aware Rademacher Averages. In *SIGKDD*. 196–206.
[17] Elizabeth M. Daly and Mads Haahr. 2007. Social network analysis for routing in disconnected delay-tolerant MANETs. In *MobiHoc*. 32–40.
[18] Martin Everett and Stephen P Borgatti. 2005. Ego network betweenness. *Soc. Networks* 27, 1 (2005), 31–38.
[19] Martin G Everett and Stephen P Borgatti. 2005. Extending centrality. *Models and methods in social network analysis* 35, 1 (2005), 57–76.
[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.
[21] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.
[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.
[23] Xin Huang, Hong Cheng, Rong-Hua Li, Lu Qin, and Jeffrey Xu Yu. 2013. Top-k structural diversity search in large networks. *Proc. VLDB Endow.* 6, 13 (2013), 1618–1629.
[24] Yanan Jiang and Hui Xia. 2024. Adversarial attacks against dynamic graph neural networks via node injection. *High-Confidence Computing* 4, 1 (2024), 100185.
[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.
[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.
[27] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *SIGKDD*. 137–146.
[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.
[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.
[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.
[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.
[32] Sangkeun Lee, Sungchan Park, Minsuk Kahng, and Sang-goo Lee. 2013. PathRank: Ranking nodes on a heterogeneous graph for flexible hybrid recommender systems. *Expert Syst. Appl.* 40, 2 (2013), 684–697.
[33] Loet Leydesdorff. 2007. Betweenness centrality as an indicator of the interdisciplinarity of scientific journals. *J. Assoc. Inf. Sci. Technol.* 58, 9 (2007), 1303–1319.
[34] Jongmin Park, Seunghoon Han, Soohwan Jeong, and Sungsu Lim. 2024. Hyperbolic Heterogeneous Graph Attention Networks. In *WWW*. 561–564.
[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.
[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.
[37] You Peng, Jeffrey Xu Yu, and Sibo Wang. 2023. PSPC: Efficient Parallel Shortest Path Counting on Large-Scale Graphs. In *ICDE*. 896–908.
[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.
[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.
[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.
[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.
[42] Diego Santoro and Ilie Sarpe. 2022. Onbra: Rigorous estimation of the temporal betweenness centrality in temporal networks. In *WWW*. 1579–1588.
[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.
[44] Julian Shun. 2020. Practical parallel hypergraph algorithms. In *PPoPP*. 232–249.
[45] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 4, 11 (2011), 992–1003.
[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.
[47] Guangming Tan, Dengbiao Tu, and Ninghui Sun. 2009. A Parallel Algorithm for Computing Betweenness Centrality. In *ICPP*. 340–347.
[48] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In *SIGKDD*. 807–816.
[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.
[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.
[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.
[52] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.
[53] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and Efficient Truss Computation over Large Heterogeneous Information Networks. In *ICDE*. 901–912.
[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.
[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.
[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.
[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 *WWW*. 2395–2406.
[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.
[59] Yikai Zhang and Jeffrey Xu Yu. 2020. Hub Labeling for Shortest Path Counting. In *SIGMOD*. 1813–1828.
[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.
[61] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential community search over large heterogeneous information networks. *Proc. VLDB Endow.* 16, 8 (2023), 2047–2060.
[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.