Butterfly-Core Community Search over Labeled Graphs

Zheng Dong[1], Xin Huang[2]*, Guorui Yuan[1], Hengshu Zhu[1]*, Hui Xiong[3]
[1]Baidu Talent Intelligence Center, Baidu Inc.
[2]Hong Kong Baptist University [3]Rutgers University
{dongzheng01, yuanguorui, zhuhengshu}@baidu.com, xinhuang@comp.hkbu.edu.hk, xionghui@gmail.com

ABSTRACT

Community search aims at finding densely connected subgraphs for query vertices in a graph. While this task has been studied widely in the literature, most of the existing works only focus on finding homogeneous communities rather than heterogeneous communities with different labels. In this paper, we motivate a new problem of cross-group community search, namely Butterfly-Core Community (BCC), over a labeled graph, where each vertex has a label indicating its properties and an edge between two vertices indicates their cross relationship. Specifically, for two query vertices with different labels, we aim to find a densely connected cross community that contains two query vertices and consists of butterfly networks, where each wing of the butterflies is induced by a k-core search based on one query vertex and two wings are connected by these butterflies. We first develop a heuristic algorithm achieving 2-approximation to the optimal solution. Furthermore, we design fast techniques of query distance computations, leader pair identifications, and index-based BCC local explorations. Extensive experiments on seven real datasets and four useful case studies validate the effectiveness and efficiency of our BCC and its multi-labeled extension models.

PVLDB Reference Format:
Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, Hui Xiong. Butterfly-Core Community Search over Labeled Graphs. PVLDB, 14(11): 2006-2018, 2021.
doi:10.14778/3476249.3476258

1 INTRODUCTION

Graphs are extensively used to represent real-life network data, e.g., social networks, academic collaboration networks, expertise networks, professional networks, and so on. Indeed, most of these networks can be regarded as labeled graphs, where vertices are usually associated with attributes as labels (e.g., roles in IT professional networks). A unique topological structure of labeled graph is the cross-group community, which refers to the subgraph formed by two knit-groups with close collaborations but different labels. For example, a cross-role business collaboration naturally forms a cross-group community in the IT professional networks.

In the literature, numerous community models have been proposed for community search based on various kinds of dense subgraphs, e.g. quasi-clique [15], k-core [16, 33, 44], k-truss [26], and densest subgraph. For example, in the classical k-core based community model, a subgraph of k-core requires that each vertex has at least k neighbors within k-core [7, 43]. The cohesive structure of k-core ensures that group members are densely connected with at least k members. However, most of the existing studies only focus on finding homogeneous communities [19, 24], which treat the semantics of all vertices and edges without differences.

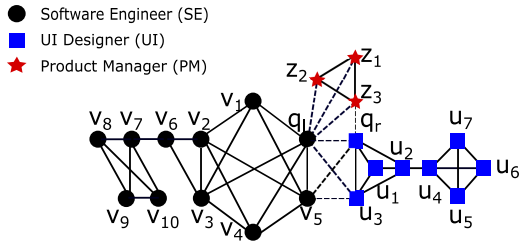


Figure 1: An example of labeled graph G in IT professional networks with three labels of SE, UI and PM. The collaborations between two employees of the same role (across over different roles) denote by the solid edges (dashed edges).

Motivating example. Figure 1 shows an example of IT professional networks G, where each vertex represents an employee and an edge represents the collaboration relationship between two employees. The vertices have three shapes and colors, which represent three different roles: "Software Engineer (SE)", "UI Designer (UI)", and "Product Manager (PM)". The edges have two types of solid and dashed lines. A solid edge represents a collaboration between two employees of the same role. A dashed edge represents a collaboration across over different roles, e.g., the dashed edge (q_l, q_r) represents the collaboration between two SE and UI employees. Our motivation is how to effectively find communities formed by these cross-group collaborations given two employees with different roles. Interestingly, considering a search for cross-group communities containing two query vertices $Q = \{q_l, q_r\}$, we find that conventional community search models cannot discover satisfactory results:

- **Structural community search**. This kind algorithms find communities containing all query vertices over a simple graph, which ignores the vertex labels. W.l.o.g. we select k-core [43] as an example, the maximum core value of q_l, q_r are 4 and 3 respectively, one limitation of this model is the smaller vertex coreness dominates k value to contain all query vertices. Each vertex on G has a degree of at least 3, thus the whole graph of G is returned as the answer. However, the model suffers from several disadvantages: (1) it fails to capture different community densities of two teams. (2) it treats the semantics of all edges equally, which not only ignores the semantics of different

*Corresponding authors.

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476258

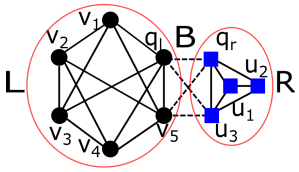


Figure 2: An example of butterfly-core community H. A bow tie B is formed by all dashed edges across two labeled groups.

edges but also mixes different teams. Other community search models add graph size constraints such as the minimum size of k-core [30] or the minimum diameter [26]. However, such improved models find the community answer of $\{q_l, q_r, v_5, u_3\}$, which suffers from missing many group members with no cross-group edges.

- **Attributed community search**. The studies [19, 24] focus on identifying the communities that have cohesive structure and share homogeneous labels. For instance, using query vertices $Q = \{q_l, q_r\}$, keywords $W_q = \{SE, UI\}$ as input, [19] returns an empty result, as the keyword cohesiveness is always 0.
- **Ours**. The expected answer of our proposed *Butterfly-Core Community (BCC) search*, aims to find cross-group communities using two query vertices, as shown in Figure 2. The BCC has three key parts. The first L is the induced subgraph formed by the vertices with the label SE, which is a 4-core. The second R is a 3-core with UI as vertex label. The third part B is the bipartite graph (the induced subgraph of G by vertices $\{q_l, q_r, v_5, u_3\}$ with only dashed edges) across over two groups of SE and UI vertices containing a butterfly, i.e., a complete 2×2 biclique.

Motivated by the above example, in this paper, we study a novel problem of cross-group community search in labeled graphs, namely BCC Search. Specifically, given a labeled graph G, two query vertices with different labels $Q = \{q_l, q_r\}$ and integers $\{k_1, k_2, b\}$, we define the (k_1, k_2, b)-BCC search problem as to find out a densely connected cross-group community that is related to query vertices Q. However, the efficient extraction of BCCs raises significant challenges, due to its problem NP-hardness. Therefore, we develop a greedy algorithmic framework, which first finds a BCC containing Q and then iteratively maintains BCC by removing the farthest vertices to Q from the graph. The method can achieve a 2-approximation to the optimal BCC answer, obtaining no greater than twice of the smallest diameter. We observe that the computation bottleneck of online BCC search lies in the query distance computation and butterfly degree counting. To tackle these technical challenges, we first present a new algorithm for fast query distance computation to prune unnecessary distance updates. Moreover, we develop novel algorithms for leader pair identification, which finds and updates leader vertices to admit butterfly degrees locally. In addition, we further develop a fast algorithm L^{2}P-BCC, integrating several optimization strategies including the bulk deletion of removing multiple vertices each time, and the local exploration to generate a small candidate graph using an offline butterfly-core index. We further discuss how to extend the BCC model to handle queries with multiple vertex labels. To summarize, we make the following contributions.

- We study and formulate a novel problem of BCC search over a labeled graph, which finds a (k_1, k_2, b)-butterfly-core community containing two query vertices q_l and q_r with different labels. We give a detailed analysis of BCC-problem and illustrate several useful applications. (Section 3).
- We show the BCC problem is NP-hard and cannot be approximated in polynomial time within a factor $(2-\varepsilon)$ of the optimal diameter for any small $\varepsilon \in (0, 1)$ unless $P = NP$ (Section 4).
- We develop a greedy algorithm for online BCC search, which achieves a 2-approximation to an optimal BCC answer with the smallest diameter. (Section 5).
- We develop several improved strategies for fast BCC search. First, we optimize the shortest path computations; Second, we develop leader pair identification algorithms for butterfly degree maintenance; Finally, we propose an index-based local search method (Section 6).
- We extend the BCC model to handle cross-group communities with multiple vertex labels and leverage our 2-labeled BCC search techniques to develop an efficient solution (Section 7).
- We conduct extensive experiments on seven real-world datasets with ground-truth cross-group communities. Four interesting case studies on real-world *global flight networks*, *international trade networks*, *complex fiction networks*, and *academic collaboration networks*, show the usefulness of our model and algorithms against other approaches (Section 8).

We discuss related work in Section 2, and conclude the paper with a summary in Section 9.

2 RELATED WORK

Attributed community detection. The attributed community detection task is to find all communities in an entire attributed graph [10, 23, 37, 60], which differs from our problem in terms of community properties and input data. A survey of clustering on attributed graphs can be found in [10]. The talent and management computing studies the organizational network analysis in business environment such as [38, 47, 48, 56]. Heterogeneous communities [46] are defined based on meta patterns, which are different from our communities across over two labeled groups.

Community search. Community search finds the query-dependent communities in a graph [11, 20, 25, 55, 57]. Community search models can be categorized based on different dense subgraphs including k-core [6, 16, 33, 34, 44, 50], k-truss [26], quasi-clique [15], and densest subgraph [53]. Recently, several complex community models have been studied for various graph data [8, 28], such as directed graphs [21, 31, 35], weighted graphs [45, 59], spatial-social networks [5, 12, 13, 29] and so on. Most attributed community search studies aim at finding the communities that have a dense structure and similar attributes [19, 24, 36, 58]. Most recently, two studies investigate community search on heterogeneous information networks [22, 27], where vertices belong to multiple labeled types. Fang et al. [22] leveraged meta-path patterns to find communities where all vertices have the same label of a query vertex and close relationships following the given meta-paths. Jian et al. [27] proposed the relational constraint to require connections between labeled

vertices in a community. They developed heuristic solutions for detecting and searching relational communities due to the hard-to-approximate problem. Both studies are different from our BCC search model that takes two query vertices with different labels and finds a leader pair based community integrating two cross-over groups. Our problem is NP-hard but can be approximately tackled.

Butterfly counting. In bipartite graph analytics [42, 52], the butterfly is a cohesive structure of 2×2 biclique. Butterfly counting calculates the number of butterflies in a bipartite graph [32, 40, 41, 51, 54]. Sanei et al. [40] proposed exact butterfly counting and approximation solutions using randomized strategies. Wang et al. [51] further optimized the butterfly counting by assigning high degree vertex with high priority to visit wedges, expected the lower degree vertex to be the middle-vertex. However, these studies focus on the *global butterfly counting* to compute the butterfly number over an entire graph. While our BCC search algorithms aim at finding a few vertices with large butterfly degrees as the leaders of cross-group communities. Moreover, our algorithms can dynamically update such leader vertices to admit butterfly degrees when the graph structure changes. Overall, our proposed butterfly solutions are efficient to find leader vertices and update butterfly degrees *locally*, which efficiently reduce the use of *global* butterfly counting.

3 PROBLEM FORMULATION

In this section, we introduce the definition and our problem.

3.1 Labeled Graph

Let $G = (V, E, \ell)$ be a labeled graph, where V is a set of vertices, $E \subseteq V \times V$ is a set of undirected edges, and $\ell : V \to \mathcal{A}$ is a vertex label function mapping from vertices V to labels $\mathcal{A}$. For each vertex $v \in V$, v is associated with a label $\ell(v) \in \mathcal{A}$. The edges have two types, i.e., given two vertices $\{u, v\} \in V$, if with same label $\ell(u) = \ell(v)$, (u, v) is a homogeneous edge; otherwise, if $\ell(u) \neq \ell(v)$, (u, v) is a heterogeneous (cross) edge. For example, consider G in Figure 1, G has three labels: SE, UI and PM. The vertex v_1 has a label of SE. The edge (v_1, v_2) is a homogeneous edge for (SE-SE). The edge (v_5, u_3) is a heterogeneous edge for (SE-UI).

Given a subgraph $H \subseteq G$, the degree of a vertex v in H is denoted as $deg_H(v) = |N_H(v)|$, where $N_H(v)$ is the set of v's neighbors in H. For two vertices u, v, we denote $dist_H(u, v)$ as a length of the shortest path between u and v in H, where $dist_H(u, v) = \infty$ if u and v are disconnected. The diameter of H is defined as the maximum length of the shortest path in H, i.e., $diam(H) = \max_{u,v \in V(H)} dist_H(u, v)$ [26].

3.2 K-Core and Butterfly

We give two definitions of k-core [43] and butterfly [40, 51].

Definition 1 (k-core). *Given a subgraph $H \subseteq G$ and an integer k, H is a k-core if each vertex v has at least k neighbors within H, i.e., $deg_H(v) \geq k$.*

The coreness $\delta(v)$ of a vertex $v \in V$ is defined as the largest number k such that there exists a connected k-core containing v. In Figure 2, L is a 4-core as each vertex has at least 4 neighbors within L. Next, we define the butterfly [40, 51] in a bipartite graph.

Definition 2 (Butterfly). *Given a bipartite graph $B = (V_L, V_R, E)$ where $E \subseteq V_L \times V_R$, a butterfly H is a 2×2 biclique of G induced by four vertices $v_{l_1}, v_{l_2} \in V_L$, $v_{r_1}, v_{r_2} \in V_R$ such that all four edges (v_{l_1}, v_{r_1}), (v_{l_1}, v_{r_2}), (v_{l_2}, v_{r_1}) and (v_{l_2}, v_{r_2}) exist in H.*

Definition 3 (Butterfly Degree). *Given a bipartite graph $B = (V_L, V_R, E)$, the butterfly degree of vertex v is the number of butterfly subgraphs containing v in B, denoted by $\chi(v)$.*

Example 1. *In Figure 2, the subgraph B is a butterfly since it is a 2×2 biclique formed by four vertices $\{q_l, v_5, q_r, u_3\}$. There exists a unique butterfly B containing the vertex q_r. Thus, the butterfly degree of q_r is $\chi(q_r) = 1$.*

3.3 Butterfly-Core Community Model

We next discuss a few choices to model the cross-group relationships between two groups V_L and V_R with different labels in the community H, and analyze their pros and cons. To quantify the strength of cross-group connections, we use the number of butterflies between two groups, denoted as b.

- First, we consider that $\chi(v) \geq b$ for each vertex $v \in H$. It requires that each vertex's butterfly count is at least b, i.e., $\chi(v) \geq b$. This constraint is too strict, which may miss some vertices without heterogeneous edges. Take Figure 1 as an example, some vertices act like leaders or liaisons who are in charge of communications across the groups, i.e., $\{q_l, q_r, v_5, u_3\}$, while some vertices mostly link within their own group with less interactions across the groups such as $\{v_1, v_2, v_3, v_4\}$. If we model in this way, an input $Q = \{v_1, q_r\}$ requires that v_1 exists in at least one butterfly, which is impossible.
- Second, we alternatively consider $\sum_{v \in V(H)} \chi(v) \geq b$, which requires that the total butterfly count in H is at least $\lceil b/4 \rceil$. However, it is hard to set b as we cannot estimate a qualified number of butterflies in community H, which is a global criterion varying significantly over different kinds of graphs.
- Finally, we consider a constraint in V_L and V_R that $\exists v_l \in V_L$ and $\exists v_r \in V_R$ to make $\chi(v_l) \geq b$ and $\chi(v_r) \geq b$ hold. It is motivated by real applications. Generally, one collaboration community has at least one leader in each group, so we require that there exists at least one vertex in each group whose butterfly count is at least b. In this setting, no matter the input query vertices are leaders biased (e.g., $Q = \{q_l, q_r\}$) or juniors biased (e.g., $Q = \{v_1, u_1\}$), the underlying community is identical.

In view of these considerations, we define the BCC as follows.

Definition 4 (Butterfly-Core Community). *Given a labeled graph G, a (k_1, k_2, b)-butterfly-core community (BCC) $H \subseteq G$ satisfies the following conditions:*

1. Two labels: there exist two labels $A_l, A_r \in \mathcal{A}$, $V_L = \{v \in H : \ell(v) = A_l\}$ and $V_R = \{v \in H : \ell(v) = A_r\}$ such that $V_L \cap V_R = \emptyset$ and $V_L \cup V_R = V_H$;

2. Left core: the induced subgraph of H by V_L is k_1-core;

3. Right core: the induced subgraph of H by V_R is k_2-core;

4. Cross-group interactions: $\exists v_l \in V_L$ and $\exists v_r \in V_R$ such that butterfly degree $\chi(v_l) \geq b$ and $\chi(v_r) \geq b$ hold.

In terms of vertex labels, condition (1) requires that the BCC contains exactly two labels for all vertices. In terms of homogeneous groups, conditions (2) and (3) ensure that each homogeneous group

satisfies the cohesive structure of k-core, in which community members are internally densely connected. In terms of cross-group interactions, condition (4) targets two representative vertices of two homogeneous groups, which have a required number of butterflies with densely cross-group interactions. Moreover, we call the two vertices v_l and v_r with $\chi(v_l) \geq b$ and $\chi(v_r) \geq b$ as a *leader pair*.

Example 2. *Figure 2 shows a* $(4, 3, 1)$*-BCC. The subgraphs L and R are respectively the left 4-core group and the right 3-core group, respectively. The subgraph B is a butterfly across over two groups L and R, and* $\chi(q_l) = \chi(q_r) = 1$.

3.4 Problem Formulation

We formulate the BCC-Problem studied in this paper.

Problem 1. *(BCC-Problem) Given* $G(V, E, \ell)$*, two query vertices* $Q = \{q_l, q_r\} \subseteq V$ *and three integers* $\{k_1, k_2, b\}$*, the BCC-Problem finds a BCC* $H \subseteq G$*, such that:*

1. Participation & Connectivity: H is a connected subgraph containing Q;

2. Cohesiveness: H is a (k_1, k_2, b)*-BCC.*

3. Smallest diameter: H has the smallest diameter, i.e., $\nexists H' \subseteq G$*, such that* $diam(H') < diam(H)$*, and* H' *satisfies the above conditions* 1 *and* 2.

The BCC-Problem prefers a tight BCC with the smallest diameter such that group members have a small communication cost, to remove query unrelated vertices. In addition, we further study and generalize BCC-problem for multiple query vertices with different labels in Section 7.

Example 3. *Consider the graph G in Figure 1. Assume that the inputs* $Q = \{q_l, q_r\}$*,* $k_1 = 4$*,* $k_2 = 3$*, and* $b = 1$*. The answer is the* $(4, 3, 1)$*-butterfly-core community containing Q as shown in Figure 2.*

3.5 Why Butterfly-Core Community Model?

Why butterfly. A butterfly is a complete bipartite subgraph of 2×2 vertices, which serves as a fundamental motif in bipartite graphs. For two groups V_L and V_R with different labels, we model the collaborative interactions between two groups V_L and V_R using the butterfly model [9, 17, 39]. More butterflies indicate 1) stronger connections between two groups and 2) similar properties sharing within the same group members, which are validated in many application scenarios. For instance, in the users-items bipartite graph B, a user $x_1 \in V_L$ buy an item $y_1 \in V_R$, then we have an edge (x_1, y_1) in B. Thus, two users x_1 and x_2 buy the same two items y_1 and y_2, which forms a butterfly in B. Two users purchase the more same items, indicating the more similar purchasing preferences of them and more butterflies in the community. Similar cases happen in the common members of *the board of directors* between *two different companies* and also the common members of *the steering committee* in *two conference organizations*. Moreover, in the email communication networks, threads of emails are delivered between *two partner groups* and also cc's to superiors on both sides. The superiors of two partner groups receive the most emails and play the *leader pair* positions of our BCC model. Overall, butterfly is a basis higher-order motif in bipartite graphs, which can be regarded as an implicit connection measure between two same labeled vertices.

Why BCC model. The BCC model inherits several good structural properties and efficient computations. First, the community structure enjoys high computational efficiency. The k-core is a natural and cohesive subgraph model of communities in real applications, requiring that every person has at least k neighbors in social groups, which can be computed faster than k-clique. In addition, butterfly listing takes a polynomial time complexity and enjoys an efficient enumeration, which could be optimized by assigning the wedge visiting priority based on vertex degrees [39, 51]. Second, two labeled groups in BCC model admit practical cases of different group densities in real-world applications. Our BCC model crosses over two labeled groups, which may have different group sizes and densities. Thus, two different k-core parameters, i.e., k_1 and k_2 are greatly helpful to capture different community structures of two groups. One simple way for parameter setting is to automatically set k_1 and k_2 with the coreness of two queries q_l and q_r respectively. Third, the automatic identification of leader pair in BCC discovery. The constraint of cross-group interactions is motivated by real-world scenarios that leaders or liaisons in each group always take most interactions with the other group.

3.6 Applications

In the following, we illustrate useful applications of BCC search.

- **Interdisciplinary collaboration search**. Two principal investigators from different departments in the university, intend to form a team to apply for an interdisciplinary research grant. The team is better formed by two cohesive groups with good inner-group communications. Moreover, the principal investigators or liaisons should also have cross-group communications to bridge two groups together.
- **Professional team discovery**. In high-tech companies, there are usually many cross-department projects between two teams with different sizes of employees. Moreover, the technical leader and product manager of each team always take charge of the cross-group communications and information sharing, which naturally form a butterfly, i.e., 2×2 biclique.
- **Various real-world cross-group mining tasks**. We can apply the BCC search on various real-world labeled graphs, e.g., *global flight networks*, *international trade networks*, *complex fiction networks*, and *academic collaboration networks*, as reported in four interesting case studies in Section 8.

4 HARDNESS AND APPROXIMATION

In this section, we analyze the hardness and non-approximability of the BCC-Problem.

Theorem 1. *The BCC-Problem is NP-hard.*

Proof. The complete proof is available in the article [18]. □

Approximation. For $\alpha \geq 1$, we say that an algorithm achieves an α-approximation to BCC-Problem if it outputs a connected (k_1, k_2, b)-BCC $H \subseteq G$ such that $Q \subseteq H$ and $diam(H) \leq \alpha \cdot diam(H^*)$, where H^* is the optimal BCC. That is, H^* is a connected (k_1, k_2, b)-BCC s.t. $Q \subseteq H^*$, and diam(H^*) is the minimum among all such BCCs containing Q.

Algorithm 1 BCC Online Search (G, Q)

Require: $G = (V, E, \ell)$, $Q = \{q_l, q_r\}$, three integers $\{k_1, k_2, b\}$.
Ensure: A connected (k_1, k_2, b)-BCC O with a small diameter.
1: Find a maximal connected (k_1, k_2, b)-BCC containing Q as G_0; //see Algorithm 2
2: $l \leftarrow 0$;
3: **while** $connect_{G_l}(Q)$ = **true do**
4: Compute $\text{dist}_{G_l}(q, u)$, $\forall q \in Q$ and $\forall u \in V(G_l)$;
5: $u^* \leftarrow argmax_{u \in V(G_l)} dist_{G_l}(u, Q)$;
6: $dist_{G_l}(G_l, Q) \leftarrow dist_{G_l}(u^*, Q)$;
7: Delete u^* and its incident edges from G_l;
8: Maintain G_l as a (k_1, k_2, b)-BCC; //see Algorithm 4
9: $G_{l+1} \leftarrow G_l$; $l \leftarrow l + 1$;
10: **return** $O \leftarrow \arg\min_{G' \in \{G_0, \ldots, G_{l-1}\}} dist_{G'}(G', Q)$;

Theorem 2. *Unless $P = NP$, for any small $\epsilon \in (0, 1)$ and given parameters $\{k_1, k_2, b\}$, the BCC-Problem cannot be approximated in polynomial time within a factor $(2 - \varepsilon)$ of the optimal.*

Proof. We prove it by contradiction. Assume that there exists a $(2 - \varepsilon)$-approximation algorithm for the BCC-Problem in polynomial time complexity, no matter how small the $\epsilon \in (0, 1)$ is. This algorithm can distinguish between the YES and NO instances of the maximum clique decision problem. That is, if an approximate answer of the reduction problem has a diameter of 1, it corresponds to the Yes-instance of maximum clique decision problem; otherwise, the answer with a diameter value of no less than 2 corresponds to the No-instance of the maximum clique decision problem. This is impossible unless $P = NP$. □

5 BCC ONLINE SEARCH

In this section, we present a greedy algorithm for the BCC-problem, which online searches a BCC.

5.1 BCC Online Search Algorithm

We begin with a definition of query distance as follows.

Definition 5 (Query Distance). *Given a graph G, a query set Q, and a set of vertices X, the query distance of X is the maximum length of the shortest path from $v \in X$ to a query vertex $q \in Q$, i.e., $dist_G(X, Q) = max_{v \in X, q \in Q} dist_G(v, q)$.*

For simplicity, we use $dist_G(H, Q)$ and $dist_G(v, Q)$ to represent the query distance for the vertex set V_H in $H \subseteq G$ and a vertex $v \in V$. Motivated by [26], we develop a greedy algorithm to find a BCC with the smallest diameter. Here is an overview of the algorithm. First, it finds a maximal connected (k_1, k_2, b)-BCC containing $Q = \{q_l, q_r\}$, denoted as G_0. As the diameter of G_0 may be large, it then iteratively removes from G_0 the vertices far away to Q, meanwhile it maintains the remaining graph as a (k_1, k_2, b)-BCC.

Algorithm. Algorithm 1 outlines a greedy algorithmic framework for finding a BCC. The algorithm first finds G_0 that is a maximal connected (k_1, k_2, b)-BCC containing Q (line 1). Then, we set $l = 0$. For all $u \in V(G_l)$ and $q \in Q$, we compute the shortest distance between q and u, and obtain the vertex query distance $dist_{G_l}(u, Q)$ (line 4). Among all vertices, we pick up a vertex u^* with the maximum distance $dist_{G_l}(u^*, Q)$, which equals to $dist_{G_l}(G_l, Q)$ (lines 5-6). Next, we remove the vertex u^* and its incident edges from G_l and also delete vertices/edges to maintain G_l as a (k_1, k_2, b)-BCC (lines 7-8). Then, we repeat the above steps until G_l is disqualified to be a BCC containing Q (lines 3-9). Finally, the algorithm terminates and returns a BCC O, where O is one of the graphs $G' \in \{G_0, ..., G_{l-1}\}$ with the smallest query distance $dist_{G'}(G', Q)$ (line 10). Note that each intermediate graph $G' \in \{G_0, ..., G_{l-1}\}$ is a (k_1, k_2, b)-BCC.

Algorithm 2 Find $G_0(G, Q)$

Require: $G = (V, E, \ell)$, $Q = \{q_l, q_r\}$, three integers $\{k_1, k_2, b\}$.
Ensure: A connected $\{k_1, k_2, b\}$-BCC G_0 containing Q.
1: $V_L \leftarrow \{v \in V \mid \ell(v) = \ell(q_l)\}$; $V_R \leftarrow \{v \in V \mid \ell(v) = \ell(q_r)\}$;
2: Let L be a k_1-core induced subgraph of G by V_L;
3: Let R be a k_2-core induced subgraph of G by V_R;
4: $B = \{V_B, E_B\}$, where $V_B = V_L \cup V_R$ and $E_B = \{V_L \times V_R\} \cap E$;
5: Butterfly Counting(B); // See Algorithm 3
6: $max_l \leftarrow \max_{u \in V_L} \chi(u)$;
7: $max_r \leftarrow \max_{u \in V_R} \chi(u)$;
8: **if** $max_l < b$ **or** $max_r < b$ **then**
9: **return** $\emptyset$;
10: **return** $G_0 \leftarrow L \cup B \cup R$;

5.2 Butterfly-Core Discovery and Maintenance

We present two important procedures for BCC online search algorithm: finding G_0 (line 1 in Algorithm 1) and butterfly-core maintenance (line 8 in Algorithm 1).

5.2.1 Finding G_0. As an essentially important step, finding G_0 is to identify a maximal connected (k_1, k_2, b)-BCC containing Q in graph G. The challenge lies in finding a butterfly-core structure, which needs to shrink the graph by vertex removals. However, deleting vertices may trigger off the change of vertex coreness and butterfly degree for vertices in the remaining graph. To address it, our algorithm applies the k-core decomposition algorithm and then runs the butterfly counting method once. The *general idea* is to first identify a candidate subgraph formed by two groups of vertices sharing the same labels with q_l and q_r. Then, it shrinks the graph by applying core decomposition algorithm, which deletes disqualified vertices to identify k_1-core and k_2-core, denoted by L and R respectively. Then, it counts the butterfly degree for all vertices and checks whether there exists two vertices $v_l \in V_L$ and $v_r \in V_R$ such that $\chi(v_l) \geq b$ and $\chi(v_r) \geq b$ hold.

Algorithm 2 presents the details of finding G_0. For query vertices Q, first we pick out all vertices with the same labels with query vertices (line 1). Each vertex set in V_L and V_R constructs the subgraph. We apply the k-core algorithm respectively to find the connected component graph L and R containing query vertices q_l and q_r (lines 2-3). Next we construct a bipartite graph B to find cross-group butterfly structures in the community. V_B consists of the vertex set V_L and V_R. E_B is the set of cross-group edges (line 4). Then, we compute the number of butterflies for each vertex in B using a butterfly counting method [40, 51] in Algorithm 3 (line 5), which calculates the butterfly degree for v as $\chi(v) = \sum_{w \in N_v^2} \binom{|N(v) \cap N(w)|}{2}$. Here N_v^2 is the vertices set within v's 2-hop neighborhood (excluding v). Algorithm 3 returns the butterfly degree of all the vertices, maintaining two values max_l and max_r to record the maximum butterfly degree on each side. Then we check if there exists at least one vertex

Algorithm 3 Butterfly Counting [40, 51]

Require: $B = (V_B, E_B)$.
Ensure: $\chi(v)$ for all vertices $v \in V_B$.
1: **for** $v \in V_B$ **do**
2: Initialize a hashmap P as empty;
3: **for** $u \in N(v)$ **do**
4: **for** $w \in N(u)$ **do**
5: $P[w] \leftarrow P[w] + 1$; //the number of 2-hop paths
6: **for** $w \in P$ **do**
7: $\chi(v) \leftarrow \chi(v) + \binom{P[w]}{2}$;
8: **return** $\{\chi(v) | \forall v \in V_B\}$;

Algorithm 4 Butterfly Core Maintenance(G, S)

Require: $G = L \cup B \cup R$, vertex set S to be removed, three integers $\{k_1, k_2, b\}$.
Ensure: A (k_1, k_2, b)-BCC graph.
1: Split S into S_L and S_R according to their labels;
2: Remove vertices S_L from L and maintain L as k_1-core, update B;
3: Remove vertices S_R from R and maintain R as k_2-core, update B;
4: Apply butterfly counting on B and check there exists one vertex on each side with butterfly degree larger than b;
5: **return** G;

whose butterfly degree is no less than b in each side, i.e., $max_l \geq b$ and $max_r \geq b$, otherwise return $\emptyset$ (lines 8-9). Finally, we merge three subgraph parts to form G_0 (line 10).

5.2.2 (k_1, k_2, b)-Butterfly-Core Maintenance. Algorithm 4 describes the procedure for maintaining G as a (k_1, k_2, b)-BCC after the deletion of vertices S from G, where the removal set $S = \{u^*\}$ (line 8 of Algorithm 1). Generally speaking, after removing vertices S and their incident edges from G, G may not be a (k_1, k_2, b)-BCC any more, or Q may be disconnected. Thus, Algorithm 4 iteratively deletes vertices having degree less than k_1 (k_2), until G becomes a connected (k_1, k_2, b)-BCC containing Q. We outline the procedure in Algorithm 4.

5.3 Approximation and Complexity Analysis

We first analyze the approximation of Algorithm 1.

Theorem 3. *Algorithm 1 achieves 2-approximation to an optimal solution H^* of the BCC-problem, that is, the obtained $\{k_1, k_2, b\}$-BCC O has $diam(O) \leq 2diam(H^*)$.*

Proof. The complete proof is available in the article [18]. □

Complexity analysis. We analyze the complexity of Algorithm 1. Let t be the number of iterations and $t \leq |V|$. We assume that $|V| - 1 \leq |E|$, w.l.o.g., considering that G is a connected graph.

Theorem 4. *Algorithm 1 takes $O(t(\sum_{u \in V} d_u^2 + |E|))$ time and $O(|E|)$ space.*

Proof. Let d_u denote the maximal vertex degree in the bipartite graph B. Finding G_0 in Algorithm 2 takes $O(\sum_{u \in B} d_u^2 + |E|)$ which runs k-core computation once in $O(|E|)$ and calls Algorithm 3 once in $O(\sum_{u \in B} d_u^2)$[40]. To shrink the community diameter, the computation of shortest distances by a BFS traversal starting from queries Q takes $O(t|Q||E|) = O(t|E|)$ time for t iterations and $|Q| = 2$.

Algorithm 5 Fast Query Distance Computation(G_i, q, D_i)

Require: A graph G_i, query vertex q, a set of removal vertices D_i.
Ensure: The updated distance $dist(v, q)$ for all vertices v.
1: Remove all vertices D_i and their incident edges from G_i;
2: $d_{min} \leftarrow min_{v \in D_i} dist_{G_i}(v, q)$;
3: $S_s = \{v \in V(G_i) \setminus D_i \mid dist_{G_i}(v, q) = d_{min}\}$;
4: $S_u = \{v \in V(G_i) \setminus D_i \mid dist_{G_i}(v, q) > d_{min}\}$;
5: Apply the BFS traverse on G_i starting from S_s to update the query distance of vertices in S_u;
6: **return** all updated distance $dist(u, q)$ for $u \in S_u$;

Algorithm 4 takes $O(t \sum_{u \in B} d_u^2)$ time for butterfly counting in t iterations. Moreover, the core maintenance takes $O(|E|)$ time for t iterations in total. As a result, the time complexity of Algorithm 1 is $O(t(\sum_{u \in V} d_u^2 + |E|))$. Next, we analyze the space complexity. It takes $O(|V| + |E|)$ space to store graph G and $O(|V|)$ space to keep the all vertices' corenesses, butterfly degrees, and query distances. Overall, Algorithm 1 takes $O(|V| + |E|) = O(|E|)$ space, due to a connected graph G with $|V| - 1 \leq |E|$. □

6 L^{2}P-BCC: LEADER PAIR BASED LOCAL BCC SEARCH

Based on our greedy algorithmic framework in Algorithm 1, we propose three methods for fast BCC search in this section. The first method is the fast computation of query distance, which only updates a partial of vertices with new query distances in Section 6.1. The second method fast identifies a pair of leader vertices, which both have the large enough butterfly degrees in Section 6.2. The leader pair tends to have large butterfly degrees even after the phase of graph removal, which can save lots of computations in butterfly counting. Section 6.3 presents an index-based local BCC search.

6.1 Fast Query Distance Computation

Here, we present a fast algorithm to compute the query distance for vertices in G. In line 4 of Algorithm 1, it needs to compute the query distance for all vertices, which invokes expensive computation costs. However, we observe that a majority of vertices keep the query distance unchanged after each phase of graph removal. This suggests that a partial update of query distances may ensure the updating exactness and speed up the efficiency.

The key idea is to identify the vertices whose query distances need to update. Given a set of vertices D_i deleted in graph G_i, we denote the distance $d_{min} = \min_{v \in D_i} dist_{G_i}(v, q)$. Let the vertex set $S_u = \{v \in V(G_i) \setminus D_i \mid dist_{G_i}(v, q) > d_{min}\}$. We have two useful observations as follows.

- For each vertex $v \in S_u$, we need to recompute the query distance $dist_{G_{i+1}}(v, q)$. It is due to the fact that the vertex u with $dist_{G_i}(u, q) \leq d_{min}$ keeps the query distance unchanged in G_{i+1}. This is because no vertices that are along with the shortest paths between u and q, are deleted in graph G_i.
- For vertex $v \in S_u$, $dist_{G_{i+1}}(v, q) \geq dist_{G_i}(v, q)$ always holds, due to $G_{i+1} \subseteq G_i$. Thus, instead of traversing from query vertex q, we update the query distance $dist_{G_{i+1}}(v, q)$ by traversing from S_s, where $S_s = \{v \in V(G_i) \setminus D_i \mid dist_{G_i}(v, q) = d_{min}\}$.

The fast query distance computation method is presented in Algorithm 5. First, we remove a set of vertices D_i from graph G_i

Table 1: The shortest distances of queries to other vertices. The symbol "–" represents the vertex set keeps unchanged.

query	1	2	3	4
q_l	$\{v_1, v_2, v_3\}$	$\{u_2, u_3, u_5, u_6\}$	$\{q_r, u_1, u_4, u_7\}$	$\{u_9\}$
q_r	$\{u_1, u_2, u_3, u_9\}$	$\{v_1, v_3, u_4, u_5, u_7\}$	$\{q_l, v_2, u_6\}$	$\emptyset$
after the deletion of u_9				
q_l	–	–	–	$\emptyset$
q_r	$\{u_1, u_2, u_3\}$	$\{v_1, v_3, u_5\}$	$\{q_l, v_2, u_6, \boldsymbol{u_4}, \boldsymbol{u_7}\}$	$\emptyset$

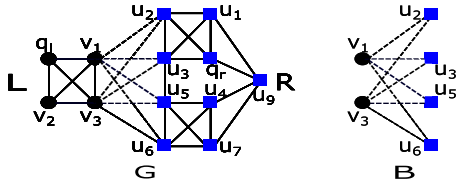


Figure 3: An example of G and its bipartite subgraph B.

(line 1). We then calculate the minimum distance d_{min}, which is the minimum length of the shortest path from D_i to q (line 2). We select vertices S_s whose shortest path equals to d_{min} as the BFS starting points (line 3). The set of vertices to be updated is S_u whose shortest path is larger than d_{min} (line 4). Then, we apply the BFS algorithm starting from vertices in S_s, we treat S_u as unvisited and all other vertices as visited (line 5). The algorithm terminates until S_u are visited or the BFS queue becomes empty. Finally, we return the shortest path to q for all vertices (line 6). In each iteration of Algorithm 1 it always keeps one query vertex's distance to others unchanged, only needs to update the distances of another query vertex, because $S_u = \emptyset$ always holds for one of the query vertices.

Example 4. *Consider the graph G in Figure 3 and $Q = \{q_l, q_r\}$. From Table 1 we know the vertex u_9 has the maximum distance to Q, i.e, $D_i = \{u_9\}$ to remove. (1) For q_l, the vertex u_9 is the farthest vertex to q_l, thus $S_u = \emptyset$, which indicates no vertices' distances to q_l need to update. (2) For q_r, $d_{min} = dist_G(u_9, q_r) = 1$. The vertex set to update is $S_u = \{v \in V(G) \mid dist_G(v, q_r) > 1\}$. The BFS starting vertex set is $S_s = \{v \in V(G) \setminus \{u_9\} \mid dist_G(v, q_r) = 1\} = \{u_1, u_2, u_3\}$. The updated distances are also shown in Table 1.*

6.2 Leader Pair Identification

Next, we present an efficient algorithm for leader pair identification, which improves the efficiency of the butterfly counting part.

Leader pair identification. The butterfly counting in Algorithm 3 needs to be invoked once a graph G_i is updated in Algorithm 1. This may lead to a large number of butterfly counting, which is very time costly. Recall that the definition of BCC requires a pair of vertices (v_l, v_r) such that $\chi(v_l) \geq b$ and $\chi(v_r) \geq b$. This motivates us to find a good pair of leader vertices, whose butterfly degrees are large enough w.r.t. b in two groups L and R, even after a number of graph removal iterations. Thus, it can avoid finding a new leader pair and save time cost. In the following, we show two key observations to find a good leader pair (v_l, v_r) where $v_l \in V_L$ and $v_r \in V_R$.

Algorithm 6 Leader Pair Identification (G, q, ρ, b)

Require: A graph $G = L$ (or R), a query vertex $q = q_l$ (or q_r), ρ, b.
Ensure: A lead vertex $p = v_l$ (or v_r).
1: $p \leftarrow q$; //Initiate as query vertex.
2: $b_{max} \leftarrow max_{v \in V(G)} \chi(v)$;
3: $b_p \leftarrow b_{max}/2$;
4: **if** $\chi(p) \geq b_p$ **then**
5: **return** p; //Query vertex has a large enough butterfly degree.
6: **else**
7: **while** $b_p \geq b$ **do**
8: $d \leftarrow 1$; //Search from the 1-hop neighbors of q.
9: **while** $d \leq \rho$ **do**
10: $S \leftarrow \{u \mid dist_G(u, q) = d, u \in G\}$;
11: **if** $\exists s \in S$ such that $\chi(s) \geq b_p$ **then**
12: **return** s;
13: **else**
14: $d \leftarrow d + 1$; //Increase the search distance d.
15: $b_p \leftarrow b_p/2$;
16: **return** p;

Observation 1. *The leader pair (v_l, v_r) should have large butterfly degrees $\chi(v_l)$ and $\chi(v_r)$, which do not easily violate the constraint of cross-over interactions.*

Observation 2. *The leader pair (v_l, v_r) should have small query distances $dist(v_l, Q)$ and $dist(v_r, Q)$, which are close to query vertices Q and not easily deleted by graph removal.*

We present the algorithm of leader pair identification to find good leader vertex candidates in Algorithm 6. Here, we use the graph G to represent a graph L/R. The parameter ρ denotes to search leaders within ρ-hops neighbors of the query vertex q. Our aim is to find a leader vertex v_l/v_r with large enough butterfly degree. We first initiate p as the query vertex q since it is the closest with distance 0 (line 1). This is especially effective when the input query vertex is leader biased who contains the largest butterfly degree. If the degree number is large enough, i.e., greater than $b_{max}/2$, we return p as the leader vertex (lines 2-5); otherwise, we find the leader vertex p in the following manner. We first increase d from 1 to ρ (line 14), and decrease b_p in $\{b_{max}/2, b_{max}/4, ..., b\}$ (lines 7-15). We get the set of vertices S whose distance to the query is d (line 10). Then, we identify one vertex $s \in S$ with $\chi(s) \geq b_p$ and return s as the leader vertex; otherwise, we increase d by 1 to search the next hop (lines 9-14). Note that we return an initial p if no better answer is identified (line 16).

Example 5. *Consider G in Figure 3 and query $Q = \{q_l, q_r\}$, $\rho = 3$. It has $\chi(v) = 6, \forall v \in \{v_1, v_3\}$ and $\chi(u) = 3, \forall u \in \{u_2, u_3, u_5, u_6\}$. For L, it initializes p with the query q_l. Since $b_{max} = 6$ and $b_p = 3$, $\chi(p) = 0$ then it starts from $d = 1$ to search q_l's 1-hop neighbors, i.e., $S = \{v_1, v_2, v_3\}$, it returns v_1 as the leader vertex since $\chi(v_1) \geq b_p$. For R, it follows a similar process to return u_2 as the leader vertex.*

Butterfly degree update for leader pair. Here, we consider how to efficiently update the butterfly degrees of leader pair vertices $\chi(v_l)$ and $\chi(v_r)$ after graph removal in Algorithm 1.

The algorithm of updating the butterfly degrees of the leader pair is outlined in Algorithm 7. First, we check the labels of p and v (line 1). If $\ell(p) = \ell(v)$, we find the common neighbors $N(v) \cap N(p)$

Algorithm 7 Butterfly Degree Update for Leader Pair (B, p, v)

Require: A graph $B = (V_B, E_B)$, a leader vertex p, a deletion vertex v.
Ensure: Butterfly degree $\chi(p)$.
1: **if** $\ell(p) = \ell(v)$ **then**
2: $\alpha \leftarrow |N(v) \cap N(p)|$; //The number of common neighbors of p, v.
3: $\chi(p) \leftarrow \chi(p) - \binom{\alpha}{2}$;
4: **else**
5: **if** $v \in N(p)$ **then**
6: $\beta \leftarrow 0$;
7: **for** $u \in N(v)$ **and** $u \neq p$ **do**
8: $\beta \leftarrow \beta + |N(u) \cap N(p)| - 1$; //Add the count of removed butterflies.
9: $\chi(p) \leftarrow \chi(p) - \beta$;
10: **return** $\chi(p)$;

shared by p and v, its number denoted as α, then the number of butterflies containing p and v is $\binom{\alpha}{2}$. Then, $\chi(p)$ decreases by $\binom{\alpha}{2}$ (lines 2-3). For $\ell(p) \neq \ell(v)$, if p and v do not connect, then there exists no butterflies (line 5); otherwise, we enumerate each vertex u, i.e., v's neighbors, and check their common neighbors with p, i.e., $N(u) \cap N(p)$. Note that β keeps the number of butterflies involving v, so we directly update $\chi(p)$ by decreasing β (lines 6-9).

Example 6. *Assume that the leader pair is $\{v_1, u_2\}$ and to delete u_6 on B in Figure 3. (1) For u_2, $\ell(u_2) = \ell(u_6)$, since their common neighbors are $\{v_1, v_3\}$ and $\alpha = 2$, $\chi(u_2)$ changes to 2 from 3. (2) For v_1, $\ell(v_1) \neq \ell(u_6)$, we enumerate u_6's neighbors except v_1, i.e., $\{v_3\}$. Since $\beta = |N(v_3) \cap N(v_1)| - 1 = 3$, $\chi(v_1)$ changes to 3 from 6.*

Complexity analysis. Next, we analyze the time and space complexity of leader pair identification and update in Algorithms 6 and 7. First, Algorithm 6 takes $O(|E| \log (b_{max} - b))$ time and $O(|E|)$ space, which identifies the leader pair using a binary search of suitable butterfly degree $b_p \in [b, b_{max}]$ within the query vertex's neighborhood. Next, we analyze the leader pair update in Algorithm 7.

Theorem 5. *One run of butterfly degree update in Algorithm 7 takes $O(d_u^2)$ time and $O(|E|)$ space, where $d_u = \max_{v \in B} d(v)$.*

Proof. Consider a leader vertex p and a deletion vertex v. Algorithm 7 takes $O(\min(d_p, d_v))$ time for $\ell(p) = \ell(v)$ and $O(d_p \cdot d_v)$ time for $\ell(p) \neq \ell(v)$. The maximum degree in bipartite graph B is d_u. Thus, the time complexity of Algorithm 7 is $O(d_u^2)$. Algorithm 7 takes $O(|V| + |E|) = O(|E|)$ space to store the vertices and their incident edges in $B \subseteq G$. □

In summary, a successful leader pair identification by Algorithm 6 can significantly reduce the calling times of butterfly counting in Algorithm 3. It only needs to *locally* update the butterfly degree for leader vertices using Algorithm 7, but not recalculating the butterfly degree for all vertices *globally* in the entire graph. Our proposed butterfly computing strategy for leader pair identification and update is very efficient for BCC search, as validated in Exp-4 in Section 8.

6.3 Index-based Local Exploration

In this section, we develop a local algorithm which detects a small densely connected candidate subgraph around Q. First, we construct the BCindex for all vertices in G, which consists of two components:

Algorithm 8 Index-based Local Exploration

Require: $G = L \cup B \cup R$, $Q = \{q_l, q_r\}$, three integers $\{k_1, k_2, b\}$.
Ensure: A connected (k_1, k_2, b)-BCC R with a small diameter.
1: Compute a shortest path P connecting Q using the butterfly-core path weight;
2: $k_l \leftarrow \min_{v \in V_L} \delta(v)$; $k_r \leftarrow \min_{v \in V_R} \delta(v)$;
3: Iteratively expand P into graph $G_t = \{v \in L \mid \delta(v) \geq k_l\} \cup \{v \in R \mid \delta(v) \geq k_r\}$ by adding adjacent vertices v, until $|V(G_t)| > \eta$;
4: Compute a connected (k_1, k_2, b)-BCC containing Q of G_t with the largest coreness on each side;
5: Remove disqualified subgraphs on G_t to return the final BCC;

the coreness [7] and butterfly degree. The offline k-core index could efficiently find k-core from G, meanwhile reduce the size of bipartite graph B. Moreover, we keep the butterfly degree number index of each vertex on the bipartite graph with different labels using Algorithm 3. Based on the obtained BCindex, we present our method of index-based local exploration, which is briefly presented in Algorithm 8. The algorithm starts from the query vertices and finds the shortest path connecting two query vertices. A naive method of shortest path search is to find a path with the minimum number of edges, while this may produce a path along with the vertices of small corenesses and butterfly degrees. To this end, we give a new definition of butterfly-core path weight as follows.

Definition 6 (Butterfly-Core Path Weight). *Given a path P between two vertices s and t in G, the butterfly-core weight of path P is defined as $\underline{dist}(s,t) = dist_G(s,t) + \gamma_1(\delta_{max} - \min_{v \in P} \delta(v)) + \gamma_2(\chi_{max} - \min_{v \in P} \chi(v))$, where $\delta(v)$ is the coreness of vertex v, $\chi(v)$ is the butterfly degree of v, δ_{max} and χ_{max} are the maximum coreness and the maximum butterfly degree in G respectively.*

The value of $(\delta_{max} - \min_{v \in P} \delta(v))$ and $(\chi_{max} - \min_{v \in P} \chi(v))$ respectively measures the shortfall in the coreness and butterfly degree of vertices in path P w.r.t. the corresponding maximum value in G. We then expand the extracted path P to a large graph G_t as a candidate BCC by involving the local neighborhood of query vertices, as outlined in Algorithm 8. We start from vertices in P, split the vertices by their labels into V_L and V_R, obtain the minimum coreness of vertices in each side as $k_l = \min_{v \in V_L} \delta(v)$ and $k_r = \min_{v \in V_R} \delta(v)$ (line 2). Due to the different density of two groups, we expand P in different core values. We then expand the path by iteratively inserting adjacent vertices with coreness no less than k_l and k_r respectively, in a BFS manner into G_t, until the vertex size exceeds a threshold η, i.e., $|V(G_t)| > \eta$, where η is empirically tuned. After that, for each vertex $v \in V(G_t)$, we add all its adjacent edges into G_t (line 3). Since G_t is a local expansion, the coreness of L, R will be at most k_l and k_r. Based on G_t, we extract the connected (k_1, k_2, b)-BCC containing Q (lines 4-5). Although Algorithm 8 does not preserve 2-approximation guarantee of optimal answers, it achieves the results of good quality fast in practice, as validated in our experiments.

7 HANDLE MULTI-LABELED BCC SEARCH

In this section, we generalize the BCC model from 2 group labels to multiple m group labels where $m \geq 2$. Then, we discuss how to extend our existing techniques to handle multi-labeled BCC search.

Algorithm 9 Multi-labeled BCC Search Framework (G, Q)

Require: $G = (V, E, \ell)$, multi-labeled query $Q = \{q_1, \ldots, q_m\}$, group core parameters $\{k_i : 1 \le i \le m\}$, butterfly degree parameter b.
Ensure: A connected mBCC O with a small diameter.
1: Find a maximal connected subgraph of multi-labeled BCC containing Q as G_0 by Def. 8; //using Algorithm 2
2: $l \leftarrow 0$;
3: **while** $connect_{G_l}(Q)$ = **true do**
4: Delete vertex $u^* \in V(G_l)$ with the largest $dist_{G_l}(u^*, Q)$ from G_l; //using Algorithm 5
5: Maintain each labeled group as a k_i-core, $\forall i \in \{1, 2, ..., m\}$;
6: Update the leader pairs and check the cross-group connectivity among m labeled groups by Def. 7; //Algorithms 3 & 4, optimized by Algorithms 6 & 7
7: $G_{l+1} \leftarrow G_l$; $l \leftarrow l + 1$;
8: **return** $O \leftarrow \arg\min_{G' \in \{G_0, \ldots, G_{l-1}\}} dist_{G'}(G', Q)$;

Multi-labeled BCC search. First, we extend the *cross-group interaction* in Def. 4 to a new definition of *cross-group connectivity*. For two groups labeled with A_l and A_r, they have cross-group interaction *iff* $\exists v_l \in V_L$ and $\exists v_r \in V_R$ such that butterfly degree $\chi(v_l) \ge b$ and $\chi(v_r) \ge b$. We say that two labels A_l and A_r have cross-group interaction, denoted as (A_l, A_r).

Definition 7 (Cross-group Connectivity). *Two labels A_l and A_r have cross-group connectivity if and only if there exists a* cross-group path $P = \{A_1, \ldots, A_j\}$ *where $A_1 = A_l, A_j = A_r$, and (A_i, A_{i+1}) has cross-group interaction for any $1 \le i < j$.*

To model group connection in a multi-labeled BCC, it requires that for any two labels A_l and A_r, there exists either a direct *cross-group interaction* or a *cross-group path* between two group core structures, implying the strong connection between two different groups within a BCC community. Specifically, we extend the definition of multi-labeled BCC (mBCC) model as follows.

Definition 8 (Multi-labeled Butterfly-Core Community). *Given a labeled graph G, an integer $m \ge 2$, group core parameters $\{k_i : 1 \le i \le m\}$, a multi-labeled butterfly-core community (mBCC) $H \subseteq G$ satisfies the following conditions:*

1. Multiple labels: there exist m different labels $A_1, A_2, ..., A_m \in \mathcal{A}$ such that $\forall 1 \le i \le m$, $V_i = \{v \in V(H) : \ell(v) = A_i\}$ and $V_i \cap V_j = \emptyset$ for $i \ne j$, and $V_1 \cup V_2 \cup ... \cup V_m = V_H$;

2. Core groups: the induced subgraph of H by V_i is k_i-core where $1 \le i \le m$;

3. Cross-group connectivity: for $1 \le i, j \le m$, any two labels A_i and A_j have cross-group connectivity in H.

For $m = 2$, this definition is equivalent to our BCC model in Def. 4. From the conditions (1) and (2), the mBCC has exactly m labeled groups, and each group is a k_i-core. The condition (3) requires that these m groups could be connected by the cross-group interactions. To ensure the cross-group connectivity, each group should have at least one leader vertex v with $\chi(v) \ge b$. Note that we only count on the butterflies among two-labeled bipartite graph, but not the accumulated number of butterflies over multiple bipartite graphs.

mBCC-search problem. Let us consider a multi-labeled query $Q = \{q_1, \ldots, q_m\}$ where each query vertex q_i has a distinct label A_i, the group core parameters $\{k_i : 1 \le i \le m\}$, and the butterfly degree parameter b, the *mBCC-search problem* is to find a connected multi-labeled BCC containing Q with the smallest diameter.

Table 2: Network statistics (K = 10^3 and M = 10^6).

Network	$\lvert V\rvert$	$\lvert E\rvert$	$Labels$	k_{max}	d_{max}
Baidu-1	30**K**	508**K**	383	43	12
Baidu-2	41**K**	2**M**	346	189	13
Amazon	335**K**	926**K**	2	6	549
DBLP	317**K**	1**M**	2	113	342
Youtube	1.1**M**	3**M**	2	51	28,754
LiveJournal	4**M**	35**M**	2	360	14,815
Orkut	3.1**M**	117**M**	2	253	33,313

Algorithm. We propose a mBCC search framework in Algorithm 9, which leverages our previous techniques of the search framework in Algorithm 1 and fast strategies in Section 6. The algorithm first finds a maximal connected mBCC containing all query vertices Q as G_0 (line 1) and then iteratively removes the farthest vertex u^* from the graph (lines 3-7). The mechanism is to maintain each labeled group as a k_i-core (line 5) and update the leader pairs (line 6). Based on the identified leader pairs and cross-group interactions, it checks the cross-group connectivity for m labeled groups in G_l by Def. 7 as follows. We first construct a new graph H_m with m isolated vertices, where each vertex represents a labeled group. Then, we insert into H_m an edge between two vertices if two labeled groups have a cross-group interaction. Finally, the cross-group connectivity of G_l is equivalent to the graph connectivity of H_m. In addition, our fast query distance computation in Algorithm 5 and local search strategies in Algorithms 6 and 7 can be extended for efficiency.

Complexity analysis. We analyze the complexity of Algorithm 9. At each iteration, the shortest path computation takes $O(|Q| \cdot (|V| + |E|)) = O(m|E|)$ time, due to $|V| - 1 \le |E|$. Moreover, it takes $O(|E|)$ time to find and maintain all k_i-cores for m different labels. To identify the leader pairs, it runs the butterfly counting over the whole graph once in $O(\sum_{u \in V} d_u^2)$ time. The extra cost of checking cross-group connectivity takes $O(m^2) \subseteq O(m|E|)$ time, due to $Q \subseteq V$ and $|V| - 1 \le |E|$. Actually, the step of checking cross-group connectivity can be further optimized in $O(m)$ time using the union-find algorithm in any conceivable application [14]. Thus, our multi-labeled BCC search in Algorithm 9 takes $O(t(m|E| + \sum_{u \in V} d_u^2))$ time for t iterations and $O(|E|)$ space.

8 EXPERIMENTS

Datasets. We use seven real datasets as shown in Table 2. Two new real-world datasets of labeled graphs are collected from Baidu which is a high tech company in China. They are IT professional networks with the ground-truth communities of joint projects between two department teams, denoted as *Baidu-1* and *Baidu-2*. In both graphs, each vertex represents an employee and the label represents the working department. An edge exists between two employees if they have communication through the company's internal instant messaging software. Baidu-1 and Baidu-2 are generated based on data logs for three months and one whole year, respectively. In addition, we use five graphs with ground-truth communities from SNAP, namely Amazon, DBLP, Youtube, LiveJournal and Orkut, by randomly adding synthetic vertex labels into them. Specifically,

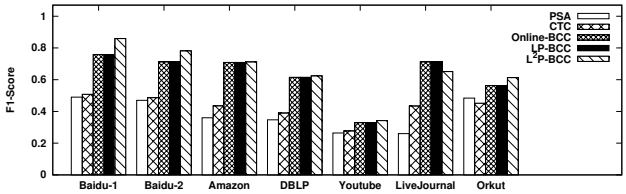


Figure 4: Quality evaluation on ground-truth networks

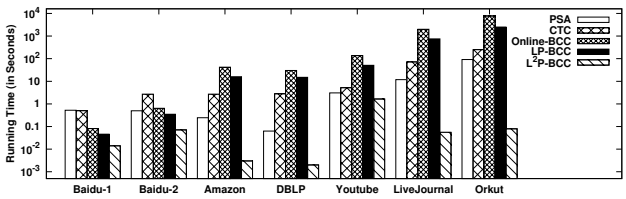


Figure 5: Efficiency evaluation on ground-truth networks

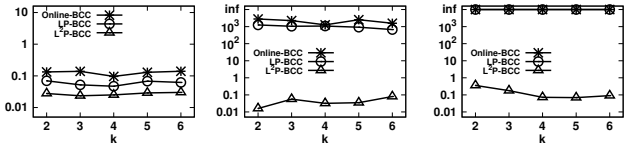
we split the vertices based on communities into two parts, and assign all vertices in each part with one label. We also generate the query pairs by picking any two vertices with different labels. To add cross edges within communities, we randomly assign vertices with 10% cross edges to simulate the collaboration behaviors between two communities. Moreover, we add 10% noise data of cross edges globally on the whole graph.

Compared methods. We compare our three BCC search approaches with two community search competitors as follows.

- CTC: finds a closest k-truss community containing a set of query vertices [26].
- PSA: progressively finds a minimum k-core containing a set of query vertices [30].
- Online-BCC: our online BCC search in Algorithm 1.
- LP-BCC: our Online-BCC method equipped with two accelerate strategies, i.e., fast query distance computation in Algorithm 5 and leader pair identification in Algorithms 6 & 7.
- L^{2}P-BCC: our local leader pair BCC search described in Algorithm 8, which is also equipped with Algorithm 5 and Algorithms 6 & 7.

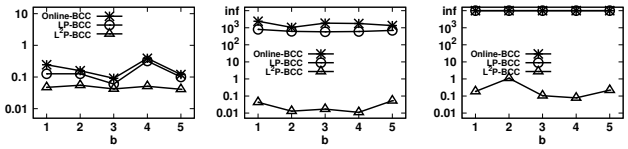
Note that all our methods use bulk deletion that removes a batch of vertices with the farthest distances from the graph. We use the default parameter setting of CTC [26] and PSA [30]. For L^{2}P-BCC, we set the parameters $\gamma_1 = 0.5$, $\gamma_2 = 0.5$, and $\rho = 3$.

Queries and evaluation metrics. We generate a BCC query $Q = \{q_1, \ldots, q_m\}$ and set $m = 2$ by default. For a 2-labeled query $Q = \{q_l, q_r\}$, we set k_1 and k_2 as the coreness value of query vertices q_l and q_r, and $b = 1$, respectively. To evaluate the efficiency, we report the running time in seconds and treat a query search as infinite if it exceeds 30 minutes. To evaluate the community quality, we use the F1-score metric to measure the alignment between a discovered community C and a ground-truth community $\hat{C}$.



(a) Baidu-1 (b) LiveJournal (c) Orkut

Figure 6: Varying core value k: Query Time.



(a) Baidu-1 (b) LiveJournal (c) Orkut

Figure 7: Varying butterfly value b: Query Time.

Table 3: A comparison of Online-BCC and LP-BCC on DBLP, in terms of query distance calculation and butterfly computation. The "#butterfly counting" denotes the calling times of the butterfly counting procedure in Algorithm 3.

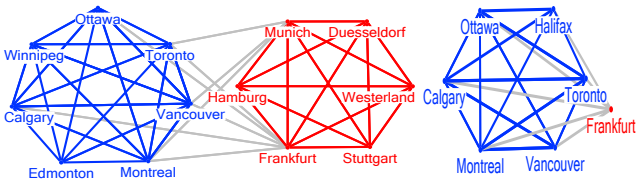
Methods	Online-BCC	LP-BCC	Speedup
Query distance calculation	1.58s	0.75s	2.1**x**
Leader pair update	4.98s	0.46s	10.8**x**
#butterfly counting	34.86	1.21	28.8**x**
Total time	11.5s	4.04s	2.8**x**

8.1 Quality and Efficiency Evaluations

Exp-1: Quality evaluation with ground-truth communities. We evaluate the effectiveness of different community search models over labeled graphs. Figure 4 reports the averaged F1-scores of all methods over 1,000 random queries on seven networks. We observe that our approaches achieve the highest F1-score on all networks against CTC [26] and PSA [30]. L^{2}P-BCC is better than Online-BCC and LP-BCC on most datasets except LiveJournal. All methods cannot have good results on Youtube.

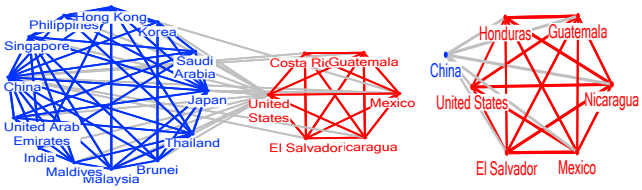
Exp-2: Efficiency evaluation of all methods. We evaluate the efficiency performance of different community search models. Figure 5 shows the running time results of all methods. All methods finish the query processing within 30 minutes, except Online-BCC and LP-BCC on Orkut as they generate a large candidate graph G_0. Most other methods use the local search strategy and run much faster than LP-BCC. Interestingly, Online-BCC and LP-BCC run slightly faster than CTC and PSA on Baidu-1 and Baidu-2. Overall, L^{2}P-BCC achieves the best efficiency performance, which can deal with one BCC search query within 1 second on most datasets.

Exp-3: Parameter sensitivity evaluation. We evaluate the parameter sensitivity of k_1, k_2, b on efficiency performance. When we test one parameter, the other two parameters are fixed. Moreover, we test one parameter of k_1 and k_2 as k due to their symmetry property. Figure 6 shows the running time by varying the core value k.



(a) Our Butterfly-Core Community (b) CTC [26]

Figure 8: Case study on flight networks for query Q = {"Toronto", "Frankfurt"}.



(a) Our Butterfly-Core Community (b) CTC [26]

Figure 9: Case study on trade networks for query Q ={"United State", "China"}.

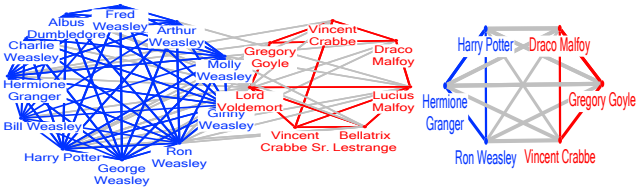
The larger k results in less running time since it generates a smaller G_0 for a larger k. Figure 7 shows the running time by varying the butterfly degree b. We observe that our approach achieves a stable efficiency performance on different values b.

Exp-4: Efficiency evaluations of query distance and butterfly computations. We evaluate the efficiency of our fast query distance computation in Algorithm 5, and leader pair identification in Algorithms 6 and 7. Table 3 reports the running time of LP-BCC and Online-BCC on DBLP. LP-BCC using Algorithm 5 achieves 2.1**x** speedups than the baseline for query distance computation. Moreover, LP-BCC using Algorithms 6 and 7 greatly improves the leader pair update process, achieves 28.8**x** and 10.8**x** speedups on the calling times of butterfly counting and running time, respectively. This validates the proposed leader pair identification can target stable leaders with large butterfly degrees for graph removals.

8.2 Three Real-world Case Study Comparisons

We conduct case studies on three real-world networks, including a global flight network [1], an international trade network [2] and the J. K. Rowling's Harry Potter series fiction network [3]. We compare our method LP-BCC with the closest truss community search (CTC [26]). We set the parameter $b = 3$ for LP-BCC.

Exp-5: A case study on flight networks. We use a labeled graph of flight network with 238 different labels, where each vertex represents a city with a label of the home country. The flight network has 3,334 vertices and 19,205 edges. Note that this graph is an undirected single-edge graph generated from the source data [1]. We generate an edge between two cities if there exits more than one airline between them. Edges between two vertices with the same/different labels are domestic/international airlines. We set the



(a) Our Butterfly-Core Community (b) CTC [26]

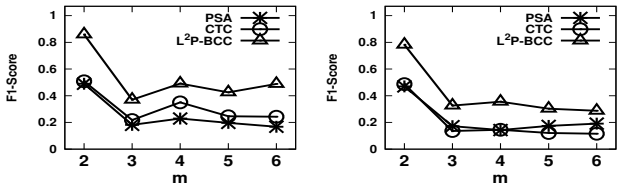
Figure 10: Case study on Harry Potter fiction networks for query Q ={"Ron Weasley", "Draco Malfoy"}.

query vertices Q ={"Toronto", "Frankfurt"}. Figure 8(a) depicts our BCC result. The flight community consists of a 6-core in blue, the butterflies in gray, and a 5-core in red. A butterfly forms by four cities "Toronto", "Vancouver", "Frankfurt", and "Munich", which are the transport hubs for transnational airlines. The core subgraphs in blue and red reflect the complex and dense networks of domestic airlines in Canada and Germany, respectively. Figure 8(b) shows the community result of CTC [26]. Unfortunately, most discovered vertices are the cities in Canada, which fails to find the international airline community as our BCC model in Figure 8(a).

Exp-6: A case study on international trade networks. The international trade network describes trade relations between countries/regions. Each vertex represents a country/region with its located continent as a label, e.g., the label of "China" is "Asia". The trade labeled graph has seven labels. We add an edge between two countries/regions if one is the top-5 import or export trade partners to the other in 2019. We set query vertices Q ={"United State", "China"}. Figure 9(a) reports our BCC community result. It consists of dense trade subnetworks in "Asia" and "North America", and also the trade leaders, i.e., "United State" and "China" have the most transcontinental trades. Figure 9(b) shows the CTC community [26] fails to find the other major trade partners in "Asia".

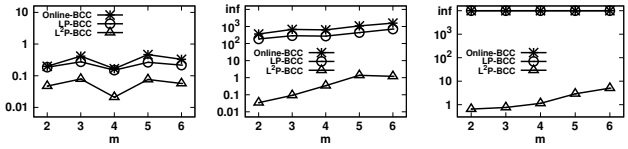
Exp-7: A case study on Harry Potter fiction networks. The Harry Potter network is a 2-labeled graph, where each vertex represents a character. Each character has a label representing his camp justice or evil. An edge is added between two characters if they have intersections in the fiction. We set the query Q ={"Ron Weasley", "Draco Malfoy"}. Figure 10(b) shows that the CTC community only finds "Ron Weasley" with his best friends of Harry and Hermione in the Gryffindor house, Malfoy and his two cronies belonging in the Slytherin house. However, it misses the main leader in the evil camp, i.e., "Lord Voldemort". What's more, it also fails to find Ron's huge family, including his parents and brothers, as our BCC result shown in Figure 10(a).

In summary, existing community models [26, 30] which ignore the graph labels, are hard to discover those communities across over two different labeled groups. This makes a huge difference from our BCC model in three aspects: (1) they ignore the different dense level of two labeled groups; (2) they confuse the semantics of different edge types (e.g., international and domestic airlines); (3) they may miss some members of one group due to the strong structural required by models. On other hand, our BCC model aims



(a) Baidu-1 (b) Baidu-2

Figure 11: Quality evaluation of multi-labeled BCC search.



(a) Baidu-1 (b) LiveJournal-M (c) Orkut-M

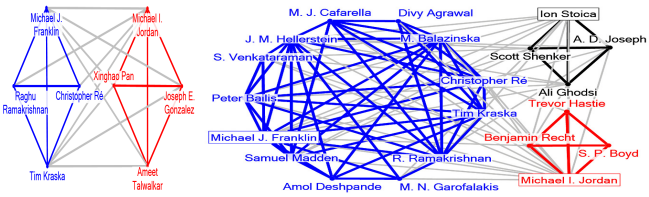
Figure 12: Varying the number of query vertices with different labels m: Query Time.

to mine out the whole dense teams including the leader pairs, inner edges within a group, and cross edges between two groups.

8.3 Multi-labeled BCC Search Evaluation

Exp-8: Quality evaluation of multi-labeled BCC search with ground-truth communities. We evaluate on two datasets Baidu-1 and Baidu-2 with multi-labeled ground-truth communities, representing joint projects among multiple department teams in the enterprise. We vary the number of query labels m and report F1-score on the average of 100 queries. Figure 11 reports the results of CTC [26], PSA [30], and our local method L^{2}P-BCC for multi-labeled BCC search in Algorithm 9. We observe that all methods achieve worse performance with the increased query labels m, indicating a more challenging task for identifying multi-labeled BCC communities. Nevertheless, our approach L^{2}P-BCC consistently performs better than the competitors CTC and PSA for all parameters m on Figures 11(a) and 11(b).

Exp-9: Efficiency evaluation of multi-labeled BCC search. We implement three extension approaches Online-BCC, LP-BCC and L^{2}P-BCC for multi-labeled BCC search, based on the framework in Algorithm 9. Besides Baidu-1, we use two other large graphs LiveJournal and Orkut, assign six vertex labels for all vertices randomly over the entire graph, denoted as LiveJournal-M and Orkut-M respectively. Figure 12 reports the running time of three mBCC methods varying by m. Figures 12(a) shows that all methods achieve stable efficiency performance for different m on Baidu-1. On the other hand, all methods takes a longer running time slightly with the increased m on large graphs in Figures 12(b)-12(c). The reason is that mBCC search takes more cost of shortest path computation for m queries, confirming the time complexity analysis of Algorithm 9 in Section 7. Importantly, our local method L^{2}P-BCC once again runs fastest among all the three methods, validating the effectiveness of our fast strategies even for multi-labeled BCC search.



(a) 2-labeled BCC for Q_1 (b) 3-labeled BCC for Q_2

Figure 13: Case study of mBCC-search on collaboration networks for Q_1 = {"Tim Kraska", "Michael I. Jordan"} and Q_2 = {"Michael J. Franklin", "Michael I. Jordan", "Ion Stoica"}.

Exp-10: Case study of multi-labeled interdisciplinary collaboration groups. We construct a real-world dataset of research collaboration network based on the "DBLP-Citation-network V12" at Aminer [4, 49]. It has 144, 334 vertices, 1, 821, 930 edges, and a total of 7 vertex labels, which is publicly available[1]. Each vertex represents an author, an edge exists if they have at least one co-authored paper. The *vertex label* is the research field of his/her most published papers, e.g., "DB", "ML", and so on. We set the parameters $b = 3$ and $k_i = 3$ for all q_i. First, we query the 2-labeled BCC community containing Q_1 ={"Tim Kraska", "Michael I. Jordan"}, which is shown in Figure 13(a). Homogeneous 3-core groups are densely connected in red and blue. In terms of butterfly structure, two query vertices "Tim" and "Michael" have a butterfly degree of 6 and 3 respectively. This is an interdisciplinary research group crossover two fields of "DB" and "ML", which works on the machine learning techniques for database systems and vice versa (a.k.a. ML4DB and DB4ML). Second, we conduct 3-labeled BCC by querying Q_2 ={"Michael J. Franklin", "Michael I. Jordan", "Ion Stoica"}, who are from the research fields of "DB", "ML", and "Systems and Networking (Sys)", respectively. The result of 3-labeled cross-discipline community is shown in Figure 13(b). There exist two *cross-group interactions* ("ML", "DB") and ("DB", "Sys") for a given $b = 3$. Thus, two 3-core groups "ML" and "Sys" has the *cross-group connectivity* via the *cross-group path* P ={"ML", "DB", "Sys"} in Figure 13(b).

9 CONCLUSION

In this paper, we study a new problem of butterfly-core community search over a labeled graph. To tackle it efficiently, we propose a 2-approximation solution for BCC search. To further improve the efficiency, we develope a fast local method L^{2}P-BCC to quickly calculate query distances and identify leader vertices. We further extend 2-labeled BCC model to multi-labeled BCC model and propose efficient solutions. Extensive experiments on real-world networks with ground-truth communities validate the effectiveness and efficiency of our BCC models and algorithms.

ACKNOWLEDGMENTS

This work was supported by the Natural Science Foundation of China under Grant No. 61836013 and HK RGC Grant No. 22200320.

[1] https://github.com/zhengdongzd/butterfly-core

REFERENCES

[1] https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.dat.
[2] https://wits.worldbank.org/datadownload.aspx?lang=en.
[3] https://github.com/efekarakus/potter-network.
[4] https://www.aminer.cn/citation.
[5] Ahmed Al-Baghdadi and Xiang Lian. 2020. Topic-based community search over spatial-social networks. *PVLDB* 13, 12 (2020), 2104–2117.
[6] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *DMKD* 29, 5 (2015), 1406–1433.
[7] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).
[8] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2018. An Optimal and Progressive Approach to Online Search of Top-K Influential Communities. *PVLDB* 11, 9 (2018), 1056–1068.
[9] Stephen P Borgatti and Martin G Everett. 1997. Network analysis of 2-mode data. *Social networks* 19, 3 (1997), 243–270.
[10] Cécile Bothorel, Juan David Cruz, Matteo Magnani, and Barbora Micenkova. 2015. Clustering attributed graphs: models, measures and methods. *Network Science* 3, 3 (2015), 408–444.
[11] Lu Chen, Chengfei Liu, Kewen Liao, Jianxin Li, and Rui Zhou. 2019. Contextual community search over large social networks. In *ICDE*. 88–99.
[12] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum co-located community search in large scale social networks. *PVLDB* 11, 10 (2018), 1233–1246.
[13] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, Jeffrey Xu Yu, and Jianxin Li. 2020. Finding Effective Geo-social Group for Impromptu Activities with Diverse Demands. In *KDD*. 698–708.
[14] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press. 1–1313 pages.
[15] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *SIGMOD*. 277–288.
[16] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *SIGMOD*. 991–1002.
[17] Tyler Derr, Cassidy Johnson, Yi Chang, and Jiliang Tang. 2019. Balance in signed bipartite networks. In *CIKM*. 1221–1230.
[18] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-Core Community Search over Labeled Graphs. *arXiv preprint arXiv:2105.08628* (2021).
[19] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *PVLDB* (2016), 1233–1244.
[20] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2019. A survey of community search over big graphs. *VLDBJ* (2019), 1–40.
[21] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2018. Effective and efficient community search over large directed graphs. *TKDE* 31, 11 (2018), 2093–2107.
[22] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *PVLDB* 13, 6 (2020), 854–867.
[23] Fangda Guo, Ye Yuan, Guoren Wang, Xiangguo Zhao, and Hao Sun. 2021. Multi-attributed Community Search in Road-social Networks. *arXiv preprint arXiv:2101.09668* (2021), 109–120.
[24] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *PVLDB* 10, 9 (2017), 949–960.
[25] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2019. *Community Search over Big Graphs*. Morgan & Claypool Publishers. 1–206 pages.
[26] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *PVLDB* (2015), 276–287.
[27] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and Efficient Relational Community Detection and Search in Large Dynamic Heterogeneous Information Networks. *PVLDB* 13, 10 (2020), 1723–1736.
[28] Yuli Jiang, Xin Huang, Hong Cheng, and Jeffrey Xu Yu. 2018. Vizcs: Online searching and visualizing communities in dynamic graphs. In *ICDE*. IEEE, 1585–1588.
[29] Junghoon Kim, Tao Guo, Kaiyu Feng, Gao Cong, Arijit Khan, and Farhana M Choudhury. 2020. Densely connected user community and location cluster search in location-based social networks. In *SIGMOD*. 2199–2209.
[30] Conggai Li, Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2019. Efficient progressive minimum k-core search. *PVLDB* 13, 3 (2019), 362–375.
[31] Jianxin Li, Xinjue Wang, Ke Deng, Xiaochun Yang, Timos Sellis, and Jeffrey Xu Yu. 2017. Most influential community search over large social networks. In *ICDE*. 871–882.
[32] Rundong Li, Pinghui Wang, Peng Jia, Xiangliang Zhang, Junzhou Zhao, Jing Tao, Ye Yuan, and Xiaohong Guan. 2021. Approximately Counting Butterflies in Large Bipartite Graph Streams. *TKDE* (2021).
[33] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential Community Search in Large Networks. *PVLDB* 8, 5 (2015), 509–520.
[34] Zhe Lin, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Zhihong Tian. 2021. Hierarchical core maintenance on large dynamic graphs. *PVLDB* 14, 5 (2021), 757–770.
[35] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based Community Search over Large Directed Graphs. In *SIGMOD*. 2183–2197.
[36] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. VAC: Vertex-Centric Attributed Community Search. In *ICDE*. 937–948.
[37] Jiehuan Luo, Xin Cao, Xike Xie, Qiang Qu, Zhiqiang Xu, and Christian S Jensen. 2020. Efficient Attribute-Constrained Co-Located Community Search. In *ICDE*. 1201–1212.
[38] Chuan Qin, Hengshu Zhu, Tong Xu, Chen Zhu, Liang Jiang, Enhong Chen, and Hui Xiong. 2018. Enhancing person-job fit for talent recruitment: An ability-aware neural network approach. In *SIGIR*. 25–34.
[39] Garry Robins and Malcolm Alexander. 2004. Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory* 10, 1 (2004), 69–94.
[40] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks. In *KDD*. 2150–2159.
[41] Seyed-Vahid Sanei-Mehri, Yu Zhang, Ahmet Erdem Sariyüce, and Srikanta Tirthapura. 2019. FLEET: butterfly estimation from a bipartite graph stream. In *CIKM*. 1201–1210.
[42] Ahmet Erdem Sarıyüce and Ali Pinar. 2018. Peeling bipartite networks for dense subgraph discovery. In *WSDM*. 504–512.
[43] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.
[44] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *KDD*. 939–948.
[45] Longxu Sun, Xin Huang, Rong-Hua Li, and Jianliang Xu. 2019. Fast Algorithms for Intimate-Core Group Search in Weighted Graphs. In *WISE*. 728–744.
[46] Yizhou Sun and Jiawei Han. 2013. Mining heterogeneous information networks: a structural analysis approach. *ACM SIGKDD Explorations Newsletter* (2013), 20–28.
[47] Ying Sun, Fuzhen Zhuang, Hengshu Zhu, Xin Song, Qing He, and Hui Xiong. 2019. The impact of person-organization fit on talent management: A structure-aware convolutional neural network approach. In *KDD*. 1625–1633.
[48] Ying Sun, Fuzhen Zhuang, Hengshu Zhu, Qi Zhang, Qing He, and Hui Xiong. 2021. Market-oriented job skill valuation with cooperative composition neural network. *Nature communications* 12, 1 (2021), 1–12.
[49] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD*. 990–998.
[50] Chaokun Wang and Junchao Zhu. 2019. Forbidden nodes aware community search. In *AAAI*, Vol. 33. 758–765.
[51] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex priority based butterfly counting for large-scale bipartite networks. *PVLDB* 12, 10 (2019), 1139–1152.
[52] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient bitruss decomposition for large-scale bipartite graphs. In *ICDE*. 661–672.
[53] Yubao Wu, Ruoming Jin, Jing Li, and Xiang Zhang. 2015. Robust local community detection: on free rider effect and its elimination. *PVLDB* 8, 7 (2015), 798–809.
[54] Yixing Yang, Yixiang Fang, Maria E Orlowska, Wenjie Zhang, and Xuemin Lin. 2021. Efficient bi-triangle counting for large bipartite networks. *PVLDB* 14, 6 (2021), 984–996.
[55] Kai Yao and Lijun Chang. 2021. Efficient Size-Bounded Community Search over Large Networks. *PVLDB* 14, 8 (2021), 1441–1453.
[56] Yuyang Ye, Hengshu Zhu, Tong Xu, Fuzhen Zhuang, Runlong Yu, and Hui Xiong. 2019. Identifying high potential talent: A neural network based dynamic social profiling approach. In *ICDM*. IEEE, 718–727.
[57] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2017. Index-based densest clique percolation community search in networks. *TKDE* 30, 5 (2017), 922–935.
[58] Zhiwei Zhang, Xin Huang, Jianliang Xu, Byron Choi, and Zechao Shang. 2019. Keyword-Centric Community Search. In *ICDE*. 422–433.
[59] Dong Zheng, Jianquan Liu, Rong-Hua Li, Cigdem Aslay, Yi-Cheng Chen, and Xin Huang. 2017. Querying intimate-core groups in weighted graphs. In *ICSC*. 156–163.
[60] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *PVLDB* (2009), 718–729.