P2PG: A Generic Framework for Graph Evaluation on Public-Private Graphs

Jiaxin Jiang, Jingjie Zhan, Lyu Xu, Byron Choi, Qiange Wang, Ning Liu, Bingsheng He, Jianliang Xu

Abstract—In many graph applications (e.g., social networks or transaction networks), users may prefer to hide parts or all of their personal data graphs (e.g., private friendships) from the public. This leads to a recent graph model, namely the *public-private graph* (PP-Graph) model, in which each user has his/her own private graph. While there have been studies analysis on PP-Graph, efficient frameworks for general graph application on PP-Graph have not yet been well studied. For example, the answers for querying on the public graphs with and without users' private graphs may differ a lot. In this paper, we propose a generic graph evaluation framework, called *public-private graph evaluation* (P2PG). P2PG consists of two modes, namely P2PG-Q and P2PG-A, which can support the graph query and the graph analysis on PP-Graphs, respectively. To achieve these two tasks, P2PG contains a practical solution that consists of two major steps, namely *partial evaluation* (PEVAL) and *incremental evaluation* (INCEVAL). We have verified through experiments that on top of P2PG, the algorithms of graph querying and analysis on PP-Graph run up to four orders of magnitude faster than the original algorithms do without P2PG.

1 INTRODUCTION

As reported in a recent study [14], users may possess private graphs, such as private knowledge bases or social networks. For example, 52.6% of 1.4 million New York City Facebook users conceal their friends lists. This propensity for users' personal privacy has given rise to the *public-private graph* (PP-Graph) model [7], [1], [39], which comprises a public graph visible to all, alongside numerous private graphs that are accessible only to their respective users. In this model, every user perceives the combination of the public graph and their individual private graph as the graph data, which may lead to potentially unique views for different users. This model necessitates a reevaluation of existing graph algorithms for two primary reasons: firstly, the substantial size of the PP-Graph—for instance, the latest iteration of the semantic knowledge base YAGO encompassing 4.5 million entities and 24 millions facts, makes it impractical to apply existing indexing techniques [30], [20] to each user's PP-Graph; secondly, given the diverse available graph algorithms, there is a clear need for a unified framework capable of optimizing the efficiency for graph querying and analysis on PP-Graphs.

Example 1.1. *Consider a public collaboration network, G, as illustrated in Figure 1 (e.g., [22]), where each node represents an academic labeled with keywords reflecting their research interests, and each edge signifies a collaborative effort between academics in research publications. A specific professor, referred to here as Bob, maintains a private collaboration network, G'_4, depicted in Figure 1 (e.g., encompassing grants, conference collaborations, and organizational affiliations). Both G and G'_4 are accessible to Bob, whereas G'_1, G'_2, and G'_3 are always invisible to him as they are privately owned by Alice, Dave, and Carol, respectively. The networks G and G'_4 are integrated through a set of common nodes, termed portal nodes (indicated by the concentric circles in Figure 1). In pursuit of launching a novel interdisciplinary project titled "DB-AI-CV," Bob initially seeks to identify close collaborators within a two-hop distance in his private network G'_4. However, querying Bob's network with {"DB","AI","CV"} yields no results. Contrarily, analyzing solely the public graph G reveals a subtree with Bob at the root and {"Dave","Carol"} as leaf vertices, who, notably, lack proximal association. An examination on the combined graph, encompassing both G and G'_4, furnishes Bob with a more relevant subtree, again rooted at Bob, but this time with {"Alice","Carol"} as the leaf vertices, indicating a more intimate collaborative relationship.*

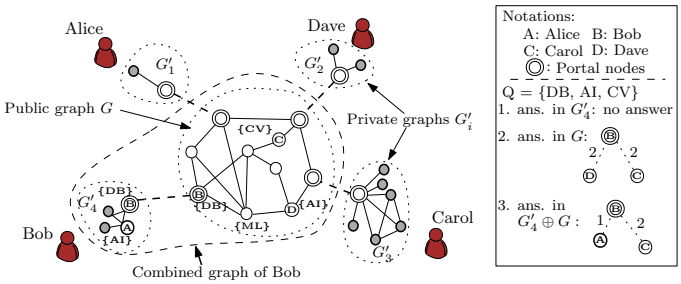


Fig. 1: An example of the public-private graph model (G is a public graph, and G'_1, G'_2, G'_3 and G'_4 are private graphs)

Challenges in Public-Private Graph Models. The discussion highlights three significant challenges inherent to the public-private graph model. Firstly, the variability of outcomes in graph tasks on private graphs and PP-Graphs underscores the complexity of achieving accurate results when both public and private graph data are involved. Secondly, the requirement of continuous update and management of indexes for each user's PP-Graph will lead to a labor-

- *Jiaxin Jiang, Qiange Wang, Bingsheng He are with School of computing, National University of Singapore, Singapore. E-mail: {jiangjx, wang.qg, hebs}@comp.nus.edu.sg*
- *Lyu Xu, Byron Choi, Jianliang Xu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. E-mail: {cslyuxu, bchoi, xujl}@comp.hkbu.edu.hk*
- *Jingjie Zhan, Ning Liu are with the Department of Computer Science, Shandong University, China. E-mail: {202300620107X, liun21cs}@sdu.edu.cn*

intensive task due to the highly personalized nature of these graphs. Lastly, the strategies of implementing constraints, conducting queries, and employing indexing techniques for graph querying or analysis must be tailored to fit the specific needs of the detailed tasks, requiring specific approaches for different algorithms.

Despite previous studies that addressed certain issues on PP-Graphs—for example, i) [7] focused on sketching and sampling algorithms; and ii) [49] and [15] dealt with the k-core and k-truss problems, respectively-a comprehensive framework that effectively tackles these challenges in a unified manner has not been developed yet. Our prior work [26] proposed a framework called PPKWS that addressed a general set of graph query tasks, that is, three kinds of keyword search queries. However, PPKWS did not support graph analysis on PP-Graphs. A straightforward approach would be to combine the public and each private graph, then compute each PP-Graph from scratch. This method is computationally costly. In this work, we propose the first unified framework called P2PG that can support both the graph query and the graph analysis on PP-Graph, which can overcome the above challenges in the PP-Graph model.

Overview of P2PG. P2PG contains two computational paradigms for the PP-Graph model: i) P2PG-Q, tailored for graph querying, necessitating the inclusion of structures from the private graph in the results; and ii) P2PG-A, aimed at graph analysis, yielding results from the combined graph. Within the P2PG framework, we developed a set of expressive APIs, facilitating users in effortlessly crafting algorithms specific to the P2PG environment. Moreover, to cater to the unique demand of incremental computation on PP-Graph, we designed protocols for incremental computation alongside a universal index for the public graph, ensuring efficient graph computing.

Contributions. This paper introduces several significant contributions of P2PG towards the advancement of graph analysis and querying in the PP-Graph model. Specifically:

- We establish a comprehensive and generic framework for evaluating public-private graphs (PP-Graphs), termed P2PG. P2PG includes two key steps, namely *Partial Evaluation* (PEVAL) and *Incremental Evaluation* (INCEVAL). We demonstrate that numerous graph queries (*e.g.*, keyword searches of semantics r-clique [30], Blinks [20], knk [29]) and analysis algorithms (*e.g.*, dense subgraph discovery DG [6], DW [19], FD [21]) can be seamlessly adapted to operate within the P2PG framework with minimal modifications.
- We furnish a comprehensive suite of expressive APIs that enable developers to intricately design their algorithms tailored for the PP-Graph model.
- We introduce two efficient indexes for the public graph, namely the *PageRank-based All Distance Sketch* (PADS) and the *PageRank-based Keyword Distance Sketch* (KPADS). These indexes enhance the conventional *All Distance Sketches* (ADS) by integrating PageRank values into their construction. PADS and KPADS maintain theoretical accuracy guarantees similar to ADS while achieving significantly higher practical precision.
- Through comprehensive experimentation, we validate the efficiency and effectiveness of P2PG. Results illustrate remarkable performance improvements for various graph algorithms on public-private graphs, with Blinks witnessing an average speedup of 202 times, r-clique improving by 12 times on average, and knk by 120 times. Additionally, the accuracy of PADS is validated to be 99.7%.
- Through case studies on the DBLP dataset, we demonstrate the practical effectiveness of the PP-Graph model in identifying potential collaborations and key collaborators using keyword searches in real-world scenarios. Additionally, we show that P2PG can effectively detect graph anomalies, such as potential conflicts of interest (COI), through these case studies.

Organization. This paper is organized as follows: Section 2 presents the background and the problem statement. Section 3 introduces the overview of the P2PG framework. Section 4 presents the procedures to implement keyword search semantics on top of P2PG. Section 5 presents how to implement the graph analysis algorithms on top of P2PG. Section 6 presents the indexes of P2PG and Section 7 reports the experimental evaluation. Section 8 discusses the related work. In Section 9, we conclude the paper.

2 Background and Problem Statement

2.1 Background

Graphs. In this study, we define a graph as a *labeled*, *weighted*, and *undirected* graph, denoted by $G = (V, E, \Sigma, L)$, where i) V is the set of vertices; ii) E: $V \times V$ is the set of edges; iii) Σ is the set of labels that can be assigned to vertices of V; and iv) L: $V \rightarrow 2^{\Sigma}$ is a mapping function that assigns each vertex $v \in V$ a subset of labels from Σ. Additionally, each edge $e = (u, v) \in E$ is associated with a positive weight, represented by $w(e)$. For simplicity, we omit L and Σ when they are irrelevant to the discussion. Moreover, we note a nuanced deviation in terminology where the size of the graph, $|G|$, is expressed as the sum of the numbers of its vertices and edges, *i.e.*, $|G|= |V|+|E|$. For two vertices u and v of a graph G, the *shortest* distance between u and v is denoted by $d(u, v)$.

As elucidated in Section 1, not all graphs are readily accessible or intended for public view [26], [7], [49]. To accommodate this distinction, we introduce some concepts for the PP-Graph model as follows.

Private Graph. Given a *public graph* $G = (V, E, \Sigma, L)$ that is accessible to each user in V and a user $u \in V$, a *private graph* of u, denoted by $G'(u) = (V', E', \Sigma', L')$, is a graph accessible to user u only. Note that L and Σ may differ from L' and Σ' to reflect the distinct nature of users' private graphs. For simplicity, we omit user u and use G' to represent $G'(u)$ when it is clear from the context.

Public-Private Graph (PP-Graph). Given a public graph $G = (V, E, \Sigma, L)$ and a private graph $G' = (V', E', \Sigma', L')$, the *public-private graph* is the combined graph $G \oplus G'$ of G and G', denoted by $G_c = (V_c, E_c)$, such that i) $V_c = V \cup V'$; and ii) $E_c = E \cup E'$. Formally, we assume that $V \cap V' \neq \emptyset$.

Definition 2.1 (Portal Node)**.** *Let $G' = (V', E')$ be a private graph and $G = (V, E)$ be a public graph. A vertex v is considered a* portal node, *if and only if $v \in V \cap V'$. Therefore, the set of portal nodes $\mathbb{P}$ is defined by the intersection $\mathbb{P} = V \cap V'$.*

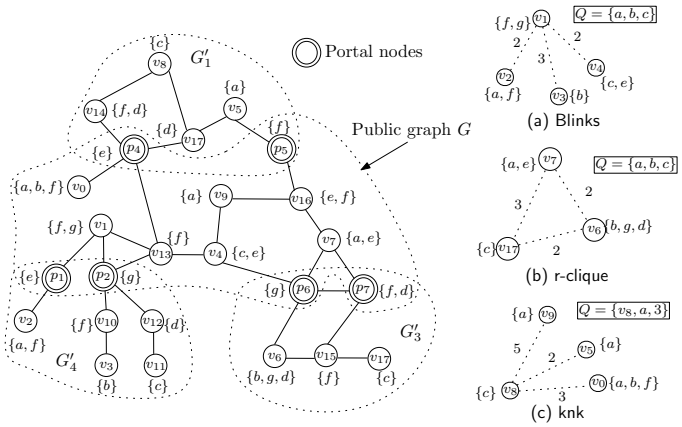


Fig. 2: An example of a public graph, three private graphs, and three popular graph query algorithms

Graph computing generally falls into two categories: graph query and graph analysis. For graph query, we take keyword search as an example. For graph analysis, we consider dense subgraph discovery as a representative case. They are chosen as examples because they illustrate the broad applicability of graph computing techniques.

2.1.1 Graph Queries

Keyword Search. Several keyword search semantics and algorithms have been proposed [4], [20], [30], [50], [24], [25] (see Figure 2). We review the following three queries as their semantics and underlying algorithms are diverse and they are driven by various interesting applications.

Blinks. He et al. [20] propose that a keyword query is a 2-ary tuple (Q, τ) which contains a set of keywords Q=$\{q_1,\ldots,q_n\}$ and a distance bound τ. Given a graph $G = (V, E, \Sigma, L)$, an answer to Q in G is a subgraph of G, denoted as $T = \{r, v_1, \ldots v_n\}$, such that (i) T is a tree rooted at r; (ii) v_i is a leaf vertex of T and $q_i \in L(v_i)$; and (iii) $d(r, v_i) \leq \tau$.

r-clique. Kargar et al. [30] propose r-clique, which determines the subgraph that all pairs of vertices that contain the query keywords are reachable to each other within r hops. That is, $d(v_i, v_j) < r$, where v_i and v_j are a pair of vertices that contain the query keywords in an answer subgraph.

knk. Jiang et al. [29] propose knk, which determines the top-k vertices R that contain a query keyword which is the nearest to a given query vertex q, *i.e.*, there does not exist $u \notin R$, but u contains the query keyword such that $d(q, u) < \max_{v \in R} d(q, v)$.

Example 2.1. *Consider a public graph G and several private graphs (G'_1, G'_3 and G'_4) shown in Figure 2. All the edge weights are 1. The answer to the query ($Q = \{a, b, c\}$, $\tau = 3$) under the* Blinks *semantic on $G \oplus G'_4$ is shown in Figure 2(a). Figure 2(b) and 2(c) respectively show the answers of the queries ($Q = \{a, b, c\}$, $r = 4$) and $Q = \{v_8, a, 3\}$ under the* r-clique *and* knk *semantics.*

2.1.2 Graph Analysis

Dense Subgraph Discovery (DSD). DSD is pivotal in graph anomaly detection, serving as a key tool for uncovering spam, fraud, and network intrusions [21], [44], [41]. The methodology behind DSD encompasses various adaptations to suit different contexts, including DG [6] for unweighted graphs, DW [19] for weighted graphs, and FD [21] for fraud detection.

Induced Subgraph. For a subset S within the vertex set V, the induced subgraph is represented as $G[S] = (S, E[S])$. Here, $E[S]$ consists of all edges (u, v) such that both vertices u and v belong to S, formally expressed as $E[S] = (u, v)|(u, v) \in E \wedge u, v \in S$. The cardinality of the subset S is denoted by $|S|$.

Density Metrics g**.** This study incorporates the density metrics g, as utilized in prior research [21], [19], [6], defined as $g(S) = \frac{f(S)}{|S|}$, where f represents the aggregate weight of the subgraph $G[S]$. This encompasses both the total vertex weight in S and the edge weights in $E[S]$:

$$f(S) = \sum_{u_i \in S} a_i + \sum_{u_i, u_j \in S \ (u_i, u_j) \in E} c_{ij} \tag{1}$$

Here, a_i represents the weight of a vertex u_i ($a_i \geq 0$), and c_{ij} represents the weight of an edge (u_i, u_j) ($c_{ij} > 0$).

Dense Subgraphs (DG) *[6].* The concept of DG is critical for measuring how closely knit certain parts of a graph are, which is instrumental in revealing underlying patterns of connections. This metric is particularly useful across a wide range of fields, including but not limited to the identification of artificial comments on digital platforms[34] and the detection of deceptive practices within social networks [3]. For a given vertex subset $S \subseteq V$, the density of a DG, denoted by $g(S)$, is calculated as $g(S) = \frac{|E[S]|}{|S|}$. Here, $|E[S]|$ is the count of edges within the subgraph induced by S, and $|S|$ is the cardinality of the subset S.

Dense Weighted Subgraph (DW) *[19].* Graphs often feature weighted edges, with weights reflecting values such as transaction volumes. The DW metric advances the dense subgraph concept to include these weighted connections. For any subset of vertices $S \subseteq V$, the DW density is calculated as $g(S) = \frac{\sum_{(u_i, u_j) \in E[S]} c_{ij}}{|S|}$, where c_{ij} is the weight of the edge $(u_i, u_j) \in E$.

Fraudar (FD) *[21].* To address fraudsters concealing their actions, Hooi et al. [21] proposed the FD algorithm. The FD density metric for a vertex subset $S \subseteq V$ is:

$$g(S) = \frac{\sum_{u_i \in S} a_i + \sum_{(u_i, u_j) \in E[S]} c_{ij}}{|S|} \tag{2}$$

where a_i represents vertex u_i's weight, and c_{ij}, the weight for edge $(u_i, u_j) \in E$, is a logarithmic function: $c_{ij} = \frac{1}{\log(x+c)}$ with x the edge's vertex degree and c a positive constant, showcasing FD's analytical approach to identifying suspicious activities [21].

2.2 Problem Statement

Definition 2.2 (PP-Answer)**.** *Given an answer $a = (V_a, E_a) \in A$, a is a public-private answer (PP-Answer) iff i) $\bigcup L(v'_i) \cap Q \neq \emptyset$, where $v'_i \in V_a$ and $v'_i \in G'.V$, and ii) $\bigcup L(v_i) \cap Q \neq \emptyset$, where $v_i \in V_a$ and $v_i \in G.V$.*

PP-Graph Graph Computation. The computation of a graph algorithm f on the PP-Graph is represented by

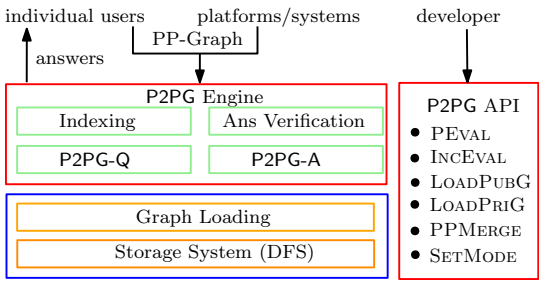


Fig. 3: Architecture of P2PG

eval$(G \oplus G', f, \mathcal{M})$, where $G \oplus G'$ represents the PP-Graph and $\mathcal{M}$ denotes the computational mode in which f operates, whether as a graph query or graph analysis algorithm. It is important to note that the outcome of a graph query algorithm applied to PP-Graph may yield public answers, private answers, or PP-Answer (refer to Definition 2.2). Previous studies can be readily adapted to address the first two scenarios. However, the PP-Answer poses the most significant technical challenge, prompting this study to concentrate on exploring solutions for generating PP-Answer during graph query processes.

Problem Statement. Given a public graph G, a private graph G', and a graph algorithm f, our objective is to devise a framework that enhances the efficiency of executing eval$(G \oplus G', f, \mathcal{M})$, focusing on optimizing the performance of processing f on PP-Graph $G \oplus G'$ under the computational mode $\mathcal{M}$.

Remark. G is accessable to all the users. Each G' is possibly disconnected and only accessable to a single user. In real-world applications, $|G'|$ is often relatively smaller than $|G|$.

3 Framework of P2PG

In this section, we present an overview and sample APIs of P2PG, and then demonstrate the characteristic of the algorithms on P2PG.

3.1 Overview of P2PG and APIs

We follow two design goals to satisfy operational demands.

- *Programmability.* We provide a set of simple user-defined APIs for developers to develop their algorithms on PP-Graph. P2PG can recasting their algorithms with slight modifications.
- *Efficiency.* P2PG allows efficient and scalable graph evaluation on PP-Graph.

Architecture of P2PG (Figure 3). Figure 3 illustrates the architecture of P2PG, which incorporates user-defined functions. P2PG provides APIs that allow developers to implement their algorithms on a PP-Graph. Initially, P2PG loads the public graph, accessible to all users, and the private graph, which is visible only to individual users. Subsequently, users can tailor their approach by choosing either graph query or graph analysis based on their operational needs. Additionally, users can integrate their own indexing functions within the framework. For graph query, users can verify whether the answers are PP-Answer.

APIs of P2PG. P2PG simplifies the process of defining the algorithms on PP-Graphs by allowing users to specify a few simple functions. The key APIs encompass several components designed to facilitate the customization and optimization of the algorithm implementation. These components provide a robust framework for users to implement their algorithms. Details are listed below:

Step 1: Partial Evaluation (PEval). The PEval accepts a partial graph (*a.k.a.* a subgraph) G and a graph algorithm f as inputs and produces the set of answers $\mathcal{A}$ based on the subgraph. We denote this process by $\mathcal{A}$=PEval(G, f).

Step 2: Incremental Evaluation (IncEval). The IncEval utilizes the answers $\mathcal{A}$ of PEval, graph increments ΔG, and an incremental graph algorithm f as inputs. Through applying f with the increments ΔG, IncEval refines $\mathcal{A}$ to produce the final answers for the combined graphs. We denote this process by $\mathcal{A}$=IncEval$(G, \Delta G, \mathcal{A}, f)$.

3.2 Characteristic of the Algorithms on P2PG

P2PG supports all graph tasks that simultaneously employ PEval and IncEval. For example, our previous work, PPKWS, has developed methods for both PEval and IncEval for r-clique, Blinks and knk (Section 4). Additionally, algorithms such as Fraudar [21] and its incremental version, Spade [27], demonstrate the application of these concepts by using Fraudar as the PEval and Spade as the IncEval to implement dense subgraph detection algorithms on PP-Graph (Section 5). This approach is utilized for detecting anomalous communities. There are numerous other examples. For instance, in graph simulation, one can use [38] as the PEval and [17] as the IncEval. Similarly, for calculating shortest distances [11] and [5] can serve as PEval and IncEval, respectively. For graph subgraph isomorphism, [48], [12], [43] can serve as PEval and [46], [32], [8] can serve as IncEval.

The two modes of operation, P2PG-Q and P2PG-A, serve distinct purposes. In P2PG-Q, users are primarily interested in information relevant to their private graphs. In this mode, PEval is executed on private graphs, followed by executing IncEval with the public graphs. Conversely, in P2PG-A, users utilize their private graphs to enhance analysis results on public graphs. Thus, PEval is run on public graphs while IncEval utilizes information from private graphs to refine partial answers on public graphs.

4 P2PG-Q: Graph Query

In this section, we present how three representative query semantics (r-clique, Blinks and knk) are implemented on top of P2PG. For each semantic, we first summarize its query evaluation and then present its two steps in P2PG.

4.1 Key Steps

Step 1) Partial Evaluation (PEval). Upon receiving $Q = \{q_1, q_2, \ldots, q_n\}$ and the private graph G' (Line 1), PEval computes the partial answers A' and *refinement indicators* $\mathcal{C}$. Each $a' \in A'$ records the query keywords it contains. $\mathcal{C}$ is used to indicate what to be refined in the partial answers A'. (The definition of $\mathcal{C}$ is discussed with query semantics in Section 4.) For instance, in this section, we denote $C \in \mathcal{C}$ as $\{(e_1, e_2)\}$, where e_1 and e_2 could be either a vertex or a

keyword. Each C records *a set of* (e_1, e_2) *pairs whose distances need to be further refined in a partial answer* $a' \in A'$.

Step 2) Incremental Evaluation (IncEval). Instead of rerunning the keyword search algorithms on the PP-Graph, P2PG refines and completes the partial answers. The shortest distance between any pair of vertex/keyword can be different after attaching the private graph to the public graph. Hence, IncEval takes the query Q, the partial answers A' and the refinement indicators $\mathcal{C}$ as an input, and refines the distance between each pair in C ($C \in \mathcal{C}$) for each $a' \in A'$.

Specifically, consider any pair $(u_1, u_2) \in C$. Since G' is a subgraph of G_c, the shortest path between u_1 and u_2 in G' is obviously a path on the PP-Graph $G_c = G \oplus G'$. The shortest distance between u_1 and u_2 in G' (*i.e.*, $d'(u_1, u_2)$) is a trivial upper bound of that in G_c (*i.e.*, $d_c(u_1, u_2)$). Hence, we index the portal distances of G'. When G' is attached to G, the portal distances are refined and then each $d_c(u_1, u_2)$ is refined by comparing the lengths of the paths that cross the portal nodes. For a partial answer $a' \in A'$, P2PG completes it by using the public graph G. IncEval (a) determines which keywords are missing from the partial answers and (b) completes A' with G to form the final answer set A.

To sum up, a keyword search algorithm f can be implemented on the PP-Graph model with the minor modification by following the above three steps. Firstly, P2PG applies f on the private graph G' to compute partial answers A' and refinement indicators $\mathcal{C}$. Secondly, P2PG refines each answer $a' \in A'$ according to the indicator $C \in \mathcal{C}$. Lastly, P2PG completes A' by retrieving the missing keywords on the public graph G to yield A.

4.2 Distance-based Keyword Search (r-clique) on P2PG

We recall that the r-clique keyword search semantic [30] determines the subgraph that all pairs of the vertices that contain the query keywords are reachable to each other within τ hops, where τ is a user-specified parameter. More specifically, the r-clique semantic is as follows:

Input: A graph G, a query $Q = \{q_1, q_2, \ldots, q_n\}$, τ.

Output: Answer A, where for each $a \in A$, $a = \{v_1, v_2, \ldots, v_n\}$, s.t. $q_i \in L(v_i)$ and $d(v_i, v_j) \leq \tau$.

4.2.1 Overview of r-clique

Kargar et al. [30] introduce an approximation algorithm that efficiently computes the top-k answers in polynomial time (PTIME). We present the key steps of the r-clique algorithm, using our notation for clearer understanding:

Initialization. Keywords q_i are associated with sets of corresponding nodes, V_{q_i}s, forming the search space $SP = (V_{q_1}, \ldots, V_{q_n})$. r-clique initializes the process by inserting a tuple $\langle SP, a \rangle$ into a priority queue $\mathcal{S}$, with SP representing the search space and $a = \{v_1, \ldots, v_n\}$ being an initial approximation of the optimal answer for SP. This priority queue $\mathcal{S}$ is sorted by the weight of a, defined as the total distance among keyword-associated nodes. To derive the optimal answer a for a given SP, r-clique calculates the shortest paths between nodes $v_i \in V_{q_i}$ and sets V_{q_j} for all distinct $q_i, q_j \in Q$. It constructs a potential optimal answer $a_{v_i} = \{u_1, \ldots, v_i, \ldots, u_n\}$, where each u_j is selected by $\arg\min_{\forall v_j \in V_{q_j}} d(v_i, v_j)$, effectively identifying the closest nodes across differing keyword sets (Algorithm 1, Lines 14-18). The optimal answer is then selected as the most compact a from among all these candidate answers.

Algorithm 1: PEval for r-clique

```
   Input: Private graph G', portal nodes ℙ, keyword query Q
   Output: Answer set A', refinement indicators 𝒞
1  Expand match candidates with portal nodes: V'_{q_i} = V_{q_i} ∪ ℙ
2  Construct search space SP = (V'_{q_1}, ..., V'_{q_n})
3  Initialize queues A, 𝒮, and a refinement indicator set 𝒞
4  a' = FindTopAnswer(SP); 𝒮.add(⟨SP, a'⟩)
5  while 𝒮 is not empty do
6  |  ⟨SP, a'⟩ = 𝒮.removeTop(); A.add(a'); 𝒞.insert(a'.C)
7  |  decompose SP and push the subsapces
   |    ⟨SP_i, FindTopAnswer(SP_i)⟩ into 𝒮
8  return (A, 𝒞)
9  Function FindTopAnswer(SP)
10 |  Initialize A' for potential answers
11 |  foreach V'_{q_i} ∈ SP do
12 |  |  foreach v_i ∈ V'_{q_i} do
13 |  |  |  Prepare a with starting node v_i, initiate match
14 |  |  |  foreach V'_{q_j} ∈ SP if q_i ≠ q_j do
15 |  |  |  |  Compute d_j = d(v_i, V'_{q_j})
16 |  |  |  |  Select u_j minimizing d(v_i, v_j)
17 |  |  |  |  Update match for q_j
18 |  |  |  A'.add(a)
19 |  return answer a ∈ A' with minimal aggregate distance
```

Search Space Decomposition. r-clique breaks down the search space through recursion. At each step, it extracts the foremost pair $\langle SP, a \rangle$ from the priority queue $\mathcal{S}$, incorporating $a = \{v_1, \ldots, v_n\}$ into the collection of answers. The algorithm then segments SP into n distinct subspaces, denoted by $SP_i = (V_{q_1}, \ldots, V_{q_i} \setminus v_i, \ldots, V_{q_n})$ for each $q_i \in Q$, effectively excluding the node v_i from the i-th set to create new, narrowed search subspaces (Algorithm 1, Line 7). Following this decomposition, r-clique reinserts these subdivided spaces, SP_i, into $\mathcal{S}$.

Termination. The search procedure terminates when $\mathcal{S}$ is empty or the top-k answers are found.

4.2.2 r-clique on P2PG (PP-r-clique)

Partial Answer $a \in A'$**.** A partial answer a, belonging to the set A', is represented by the tuple $\langle v, \mathsf{mat} \rangle$. In this structure, v serves as the answer's root vertex, while mat functions as a mapping. For each query keyword q, $\mathsf{mat}[q]$ produces a tuple $\langle \mathsf{u}, \mathsf{d} \rangle$, where $\mathsf{mat}[q].\mathsf{u}$ specifies a vertex u linked to q, either by the label $L(u)$ or as a portal node. $\mathsf{mat}[q].\mathsf{d}$, in turn, quantifies the distance between u and the root v.

(1) PEval. In adapting the approach of Kargar et al. [30], P2PG introduces PEval to execute all r-clique computations on G', as detailed in Algorithm 1. To leverage the potential of completing partial answers with information from the public graph, we incorporate portal nodes $\mathbb{P}$ into the search space, resulting in $V'_{q_i} = V_{q_i} \cup \mathbb{P}$. For each $a \in A'$, PEval specifies a collection of vertex pairs for further refinement in $C = (v, \langle u, d \rangle)$ and referred to by $a.C$.

(2) IncEval. IncEval further refines the answers within $a.C$ by comparing the distances $d_c(v, u)$ and $d'(v, u)$, the former measured after integrating the private graph with the public graph (Algorithm 2, Lines 3-5). For a given partial answer $a = \langle v, \mathsf{mat} \rangle$, refinement is achieved by minimizing the distance for each pair $(v, \langle u, d \rangle) \in a.C$. If a refined pair $\langle u, d \rangle = \mathsf{mat}[q]$ includes a portal node u not labeled with the

Algorithm 2: Answer refinement for r-clique

```
Input: Partial answers A'=eval(G', Q, r-clique), d_c, Q
Output: Refined answers A
1 foreach a' ∈ A' do
2     foreach (v, ⟨u, d⟩) in a'.C do
3         foreach (p_i, p_j) ∈ P × P do
4             dist = d'(v, p_i) + d_c(p_i, p_j) + d'(p_j, u)
5             d = min(d, dist)
6 return A'
```

query keyword q, it indicates the missing of the keyword q in the partial answer. To address this, the distance from u to q is computed within the public graph. If this computed distance $d_c(u, q) + d$ exceeds the threshold τ, the partial answer is discarded based on the definition of r-clique.

Moreover, an answer $a \in A$ is considered *qualified* as a PP-Answer *iff* it satisfies two criteria: 1) The distance for each query keyword q in a, denoted as $a.\mathsf{mat}[q].\mathsf{d}$, does not exceed a predefined threshold τ; and 2) the query keywords within a are distributed across both public and private graphs. To facilitate this verification process, we employ a counter for each answer to track the quantity of keywords matched specifically within the private graph.

Theorem 4.1. *Given an answer of* PP-r-clique, $a = \langle v, \mathsf{mat}\rangle$, $a.\mathsf{mat}[q].\mathsf{d} \leq (2c-1)d_c(v, a.\mathsf{mat}[q].\mathsf{u})$.

Proof. The proof is presented in Appx. A.3 of [28]. □

4.3 Keyword Search with Distinct Subtree Answer (Blinks) on P2PG

In the realm of keyword searches on data graphs lacking connectivity indices, a prevalent approach is to initiate traversal from vertices containing the query keywords. Bhalotia et al.[4] introduced the backward keyword search algorithm to tackle such queries. Building on this, He et al.[20] advanced the methodology with Blinks, a strategy designed for backward expansion, to efficiently generate subtree answers. This search paradigm is characterized by the following semantic description:

INPUT: The input consists of a graph G and a set of query keywords $Q = \{q_1, q_2, \ldots, q_n\}$.

OUTPUT: The output is a set of answers A. Each answer $a \in A$ is structured as $\langle r, \{v_1, v_2, \ldots, v_n\}\rangle$, where each q_i belongs to the label set $L(v_i)$ of vertex v_i, and the distance between the root node r and each vertex v_i is within a predefined threshold τ.

4.3.1 Overview of Blinks

Initialization. For a given keyword query $Q = \{q_1, q_2, \ldots, q_n\}$, we define V_{q_i} as the set of vertices containing keyword q_i—termed the search origin. Additionally, V_i represents the set of vertices capable of reaching any vertex within V_{q_i}.

Backward Expansion. During each search iteration, we prioritize the vertex set V_i with the minimum size. From this set, we select a vertex $v \in V_i$ that is closest to V_{q_i} for backward expansion. In this phase, a vertex u, connected to v via an incoming edge (u, v), is considered for addition to V_i. This vertex u is evaluated as a candidate answer root; if u does not qualify, the search recursively expands backward.

Answer Discovery. The process identifies a root vertex r that has access to at least one vertex containing each query keyword q_i, for every q_i in Q. This root r is thus capable of forming a connection to all queried keywords, qualifying as an answer root.

4.3.2 Blinks on P2PG (PP-Blinks)

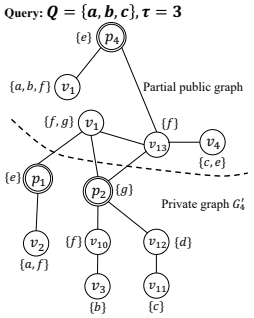
Partial Answer $a \in A'$. A partial answer, a, is structured as a tuple $\langle r, \mathsf{mat}\rangle$, with r serving as a candidate answer root and mat as a mapping function. For each query keyword q, $\mathsf{mat}[q]$ maps q to a tuple $\langle \mathsf{v}, \mathsf{d}\rangle$: $\mathsf{mat}[q].\mathsf{v}$ identifies a vertex v where either $q \in L(v)$ or $v \in \mathbb{P}$, and $\mathsf{mat}[q].\mathsf{d}$ specifies the distance from r to v. Additionally, PEVAL generates a set $C = (r, q)$ for every partial answer a, containing the vertex-keyword pairs that require refinement, symbolized by $a.C$. Since the computation of $\mathsf{mat}[q].\mathsf{v}$ and that of $\mathsf{mat}[q].\mathsf{d}$ are similar, we only show how to compute $\mathsf{mat}[q].\mathsf{d}$ below.

(1) PEVAL. The process begins by setting the search origin to the set of query keywords Q within the private graph G'. As traversal occurs and each vertex $r \in V'$ is encountered, it is recorded as a potential answer, denoted by a. Then we track the keywords absent in the partial answer a. These missing keywords indicate the need for further exploration within the public graph to complete the answer.

(2) INCEVAL. INCEVAL contains three steps: answer refinement, answer completion, and answer qualification.

(a) Answer Refinement (Algorithm 3). INCEVAL refines the partial answers by comparing distances $d_c(r, q)$ and $d'(r, q)$ for each vertex-keyword pair $(r, q) \in a.C$, particularly after integrating the private graph with the public graph. This process aims to refine the distances between answer roots and their corresponding keywords. The refinement leverages the updated portal distances d_c, executing in $O(|C||\mathbb{P}|^2)$. Two critical steps facilitate this refinement: identifying the shortest paths that may involve portal nodes and refining distances between these nodes in the PP-Graph. Then INCEVAL assesses whether the refined portal distances contribute to refining the connection between an answer root and a keyword (Lines 5-6 of Algorithm 3).

(b) Answer Completion (Algorithm 4). The second step involves expanding the search backward on the public graph, especially since the answer root r' may reside within this graph. For every partial answer with r' as its root in the portal nodes ($\mathbb{P}$), INCEVAL employs Breadth-First Traversal ($\mathcal{T}_{r'}$) to explore the public graph from r', extending up to x hops, where $x = \{\tau - \mathsf{mat}[q].\mathsf{d}\}$. When a vertex u, reached after x' hops in traversal $\mathcal{T}_p$, is previously encountered in another traversal $\mathcal{T}_{p'}$ (with $p' \neq p$), P2PG utilizes a flood search strategy (see [50]) to refine the distance (dist) for the already visited answers (Lines 14-19). If u is unvisited, P2PG creates a new partial answer with u as the root (at Line 8). The minimal distance from u to a query keyword q combines x' with the distance from p to q. The subsequent phase focuses on identifying and retrieving any keywords absent from each partial answer. For every answer $a \in A$, the process involves calculating the distance between each query keyword $q \in Q$ and the answer root $a.r$ within the public graph (Lines 20-23). If the distance $d(a.r, q)$ is smaller than the current distance $a.\mathsf{mat}[q].\mathsf{d}$, this distance is refined.



(a) Partial public-private graph

root	a	b	c
p_1	$(v_2, 1)$		
p_2		$(v_3, 2)$	$(v_{11}, 2)$
v_2	$(v_2, 0)$		
v_3		$(v_3, 0)$	
v_{10}		$(v_3, 1)$	$(v_{11}, 3)$
v_{11}			$(v_{11}, 0)$
v_{12}		$(v_3, 3)$	$(v_{11}, 1)$

(b) PEval: Partial Evaluation

root	a	b	c
p_1	$(v_2, 1)$		
p_2	$(v_2, 3)$	$(v_3, 2)$	$(v_{11}, 2)$
v_2	$(v_2, 0)$		
v_3		$(v_3, 0)$	
v_{10}		$(v_3, 1)$	$(v_{11}, 3)$
v_{11}			$(v_{11}, 0)$
v_{12}		$(v_3, 3)$	$(v_{11}, 1)$

(c) IncEval: Answer Refinement

root	a	b	c
p_1	$(v_2, 1)$		
p_2	$(v_2, 3)$	$(v_3, 2)$	$(v_{11}, 2)$
v_2	$(v_2, 0)$		
v_3		$(v_3, 0)$	
v_{10}		$(v_3, 1)$	$(v_{11}, 3)$
v_{11}			$(v_{11}, 0)$
v_{12}		$(v_3, 3)$	$(v_{11}, 1)$
v_1	$(v_2, 2)$	$(v_3, 3)$	$(v_{11}, 3)$
v_{13}	$(v_2, 3)$	$(v_3, 3)$	$(v_{11}, 3)$

(d) IncEval: Answer Completion

root	a	b	c
p_1	$(v_2, 1)$		
p_2	$(v_2, 3)$	$(v_3, 2)$	$(v_{11}, 2)$
v_2	$(v_2, 0)$		
v_3		$(v_3, 0)$	
v_{10}		$(v_3, 1)$	$(v_{11}, 3)$
v_{11}			$(v_{11}, 0)$
v_{12}		$(v_3, 3)$	$(v_{11}, 1)$
v_1	$(v_2, 2)$	$(v_3, 3)$	$(v_{11}, 3)$
v_{13}	$(v_2, 3)$	$(v_3, 3)$	$(v_{11}, 3)$

(e) Answer Verification

Fig. 4: Example of query execution (PEVAL and INCEVAL) of PP-Blinks. Each row in the table represents an answer, with the 'root' column indicating the root vertex. The subsequent columns ('a', 'b', 'c') show the leaf vertices associated with the query keyword at the header, along with the distances from the root vertex.

Algorithm 3: Answer Refinement for Blinks

Input: Partial answers A', d_c, and bound τ
Output: Refined partial answers A'

```
foreach a ∈ A' do
    foreach (r, q) in a.C do
        foreach (p_i, p_j) ∈ P × P do
            dist = d'(r, p_i) + d_c(p_i, p_j) + d'(p_j, q)
            if a'.mat[q].d ≥ dist then
                a'.mat[q].d = dist
return A'
```

(d) Answer Qualification. The answer verification is the same as that of PP-r-clique.

Example 4.1. *In Figure 4(b),* PEVAL *returns 7 partial answers on private graphs. When the private graph is integrated with the public graph, the answer rooted at p_2 is refined. Specifically, the distance between p_2 and the query keyword $'a'$ is refined to 3 by* INCEVAL*, as highlighted in red in Figure 4(c). Subsequently,* INCEVAL *expands backward from two portal nodes, p_1 and p_2, resulting in two additional answers rooted at v_1 and v_{13}, as shown in Figure 4(d). The three answers rooted at p_2, v_1, and v_{13} are returned, while the rest are pruned due to the absence of required query keywords.*

Lemma 4.2. *Consider $a = \langle r, \mathsf{mat}\rangle \in \mathsf{eval}(G \oplus G', Q, \mathsf{Blinks})$ and $a' = \langle r, \mathsf{mat}'\rangle$ returned by* P2PG*:*

- *if* $\mathsf{mat}[q].\mathsf{v} \in G'.V$*, then* $\mathsf{mat}'[q].\mathsf{v} = \mathsf{mat}[q].\mathsf{v}$ *and* $\mathsf{mat}'[q].\mathsf{d} = \mathsf{mat}[q].\mathsf{d}$*; and*
- *if* $\mathsf{mat}[q].\mathsf{v} \notin G'.V$*, then* $\mathsf{mat}'[q].\mathsf{d} \leq (2c-1)\mathsf{mat}[q].\mathsf{d}$.

Proof. The proof is presented in Appx. A.1 of [28]. □

4.4 Top-k Nearest Keyword Search (knk) on P2PG

4.4.1 Overview of knk

A knk query, as defined by Jiang et al. [29], consists of a triple (v, q, k), where v denotes a query vertex, q represents a query keyword, and k specifies the count of the nearest vertices to v that include the keyword q.

INPUT: a query point v, a query keyword q

OUTPUT: top-k vertices $A = \{a = \{\langle u_i, d_i\rangle\}\}$ ranked by d_i, where $q \in L(u_i)$.

4.4.2 knk on P2PG (PP-knk)

(1) PEVAL. Any knk algorithms can be applied on P2PG without any modification. P2PG takes [29] as PEVAL to compute the knk answers on the private graph G'.

Algorithm 4: Answer Completion for Blinks

Input: Public graph G, $\mathbb{P}$, refined answers A', Q, and bound τ
Output: Completed answers A

```
Initialize an empty map A = {}
foreach p ∈ P do
    if A'.ContainsKey(p) then
        foreach u in the x'-th hop BF' traversal starting from p do
            if A.ContainsKey(u) then
                A[u] = UpdateAns(A[u], A'[p], x', Q)
            else
                A[u].mat = {⟨A'[p].mat[q].v, A'[p].mat[q].d + x'⟩}
foreach a ∈ A do
    ComplAns(a, Q)
    if Not a.IsQualified() then
        A.Remove(a)
return A
Function UpdateAns(a_1, a_2, x, Q)
    foreach q ∈ Q do
        if a_2.mat[q].d + x < a_1.mat[q].d then
            a_1.mat[q].d = a_2.mat[q].d + x
            a_1.mat[q].v = a_2.mat[q].v
    return a_1
Function ComplAns(a, Q)
    foreach q ∈ Q do
        Compute the shortest distance d(a.r, q) on the public graph
        a.mat[q].d = min(d(a.r, q), a.mat[q].d)
```

Partial Answer $a' \in A'$. The partial answer a' is a list mat where the i-th element has two attributes $\langle \mathsf{u}, \mathsf{d}\rangle$. $a'.\mathsf{mat}[i].\mathsf{u}$ is a vertex u, such that $u \in V'$ and $q \in L(u)$ or $a'.\mathsf{u} \in \mathbb{P}$ and $a'.\mathsf{mat}[i].\mathsf{d} = d'(v, u)$. We use a boolean variable to record whether u is a portal. For the partial answer a', PEVAL declares $C = \{(v, u)\}$ to indicate what to be refined, where v and u are two vertices on the private graph. More specifically, v is the query point of knk and u is a candidate matched vertex in the private graph.

(2) INCEVAL. The refinement of the vertex pair $(v, u) \in C$ is identical to that discussed in Section 4.2. Given the refined answer a', P2PG completes a' in the public graph. For the i-th element of $a'.\mathsf{mat}$, *i.e.*, $\langle u, d\rangle$, if $q \in L(u)$, u is a candidate match vertex. Moreover, if u is a portal node, P2PG estimates the shortest distance between u and the keyword q in the public graph with the intersection of PADS(u) and KPADS(q). $\hat{d}(u, q) + d$ is appended with the keyword vertex u' at the end of $a'.\mathsf{mat}$, where u' can be obtained by the inverted index of KPADS(q) (For simplicity, we omit the details of this data structure in this paper). It is worth noting that, we maintain a priority queue with a

TABLE 1: Complexity of P2PG-Q

Algorithms	PEVAL	INCEVAL(Answer Refinement; Answer Completion)
PP-r-clique	Same as [30]	$O(\|A'\|\|Q\|\|\mathbb{P}\|^2)$; $O(\|A\|\|Q\|k\ln\|V\|)$
PP-Blinks	Same as [20]	$O(\|A'\|\|Q\|\|\mathbb{P}\|^2)$; $O(m_1\|\mathbb{P}\|\|Q\|+\|A\|\|Q\|k\ln\|V\|)$
PP-knk	Same as [29]	$O(m_2\|\mathbb{P}\|^2)$; $O(\|\mathbb{P}\|k\ln\|V\|)$

fixed size k for a'.mat rather than a list.

Lemma 4.3. *If $u \in V'$ belongs to the answer of a* knk *query (v, q, k) on G_c, where $v \in V'$, then u is returned by* PP-knk.

Proof. We denote the set of vertices containing q as V_q. Given two vertex $u_1, u_2 \in V_q$. Without loss of generality, we assume that $u_1 \in V'$ and $d_c(v, u_1) \leq d_c(v, u_2)$. Then the ranking of u_1 is higher than u_2. It is worth noting that the exact value of $d_c(v, u_1)$ is returned by P2PG. Next, we prove that the ranking is hold in P2PG.

- If $u_2 \in V$, then $d_c(v, u_2) \leq \hat{d}(v, u_2)$ since $\hat{d}(v, u_2)$ is always larger than $d_c(v, u_2)$, returned by P2PG. Naturally, the ranking of u_1 is still higher than that of u_2 since $d_c(v, u_1) \leq d_c(v, u_2) \leq \hat{d}(v, u_2)$.
- If $u_2 \in V'$, since the exact value of $d_c(v, u_2)$ is also returned by P2PG, $d_c(v, u_1) \leq d_c(v, u_2)$ is still hold. The ranking of u_1 is still higher than that of u_2.

Hence, $\forall u \in V'$ is an answer in G_c, u is returned by P2PG since its ranking is hold in the context of P2PG. □

4.5 Complexity Analysis

In this section, we present the complexity of each key step of PP-r-clique, PP-Blinks and PP-knk as shown in Table 1.

Complexity of the PP-r-clique**.** PEVAL applies the keyword search algorithm of [30] with an answer qualification function which can be finished by a linear scanning, bounded by $O(V')$. Hence, PEVAL inherits the complexity of r-clique (cf. [30]). The answer refinement of INCEVAL is in $O(|A'||C||\mathbb{P}^2|)$ since refining each partial answer takes $O(|C||\mathbb{P}|^2)$. It is bounded by $O(|A'||Q||\mathbb{P}|^2)$. The answer completion of INCEVAL is in $O(|A||Q|k\ln|V|)$.

Complexity of the PP-Blinks**.** PEVAL applies the keyword search algorithm of [20] with an answer qualification function which can be finished by a linear scanning, bounded by $O(V')$. Hence, PEVAL inherits the complexity of Blinks (cf. [20]). The answer refinement of INCEVAL is in $O(|A'||C||\mathbb{P}^2|)$ since refining each partial answer takes $O(|C||\mathbb{P}|^2)$. It is bounded by $O(|A'||Q||\mathbb{P}|^2)$. The answer completion of INCEVAL is in $O(m_1|\mathbb{P}||Q|+|A||Q|k\ln|V|)$. The backward expansion on the public graph takes $O(m_1|\mathbb{P}||Q|)$ where m_1 is the average number of the nodes within the x hops of the portals. For each visited node, it takes $O(|Q|)$ to refine the distance. The answer completion of INCEVAL takes $O(|A||Q|k\ln|V|)$ to retrieve the missing keywords of each answer.

Complexity of the PP-knk**.** PEVAL applies the keyword search algorithm of [30] without any changes. Hence, PEVAL inherits the complexity of knk (cf. [29]). INCEVAL is in $O(m_2|\mathbb{P}^2|)$ where $m_2 = |a'.\text{mat}|$ since refining each partial answer takes $O(|\mathbb{P}|^2)$. The answer completion of INCEVAL is in $O(|\mathbb{P}|k\ln|V|)$.

5 P2PG-A: GRAPH ANALYSIS

In this section, we present how peeling-based graph analysis algorithms (PEEL) are implemented on top of P2PG.

Algorithm 5: PEVAL of Peeling Algorithm

Input: A public graph $G = (V, E)$
Output: Peeling sequence O on G
1 Initialize $S_0 = V$
2 **for** $i = 1$ **to** $|V|$ **do**
3 Choose vertex $u \in S_{i-1}$ to maximize $g(S_{i-1} \setminus \{u\})$
4 Update $S_i = S_{i-1} \setminus \{u\}$
5 Append u to sequence O
6 **return** O, *highest density subgraph* $\arg\max_{S_i} g(S_i)$

Algorithm 6: INCEVAL of Peeling Algorithm

Input: Public graph G, Private graph G', Peeling sequence O
Output: Peeling sequence O' on $G \oplus G'$
1 Sort vertices in V' by their indices in O
2 Initialize O', and priority queue T
3 **for** *each vertex* $u_i = O[i]$ *in* V' **do**
4 Insert u_i into T and set neighbors as gray
5 **while** T *not empty* **do**
6 Process vertex removal or insertion in T based on condition and update O'
7 Extend O' with remaining white vertices
8 **return** O' *and optimal subgraph* S_i

(1) PEVAL *of* PEEL *(Algorithm 5).* The set of vertices remaining after the i-th peeling iteration is represented by S_i. Initially, the algorithm sets $S_0 = V$ (Line 1). In each iteration, a vertex u_i is removed from S_{i-1} to maximize $g(S_{i-1} \setminus u_i)$. This selection and removal process continues until the vertex set is depleted. This iterative peeling yields a sequence of vertex sets from S_0 to $S_{|V|}$, progressively decreasing in size from $|V|$ to 0. The algorithm seeks the set S_i, for i ranging from 0 to $|V|$, that achieves the maximum density metric $g(S_i)$, identified as S^P. For efficiency, rather than preserving the entire sequence of sets, the algorithm records the peeling sequence $O = [u_1, \ldots, u_{|V|}]$, where each u_i corresponds to the vertex removed at step i, effectively transitioning from S_{i-1} to S_i.

(2) INCEVAL *of* PEEL *(Algorithm 6).* The Peeling Algorithm processes a PP-Graph to generate a peeling sequence and identify a dense subgraph. It begins by sorting vertices according to their indices in the sequence O, and initializes an empty sequence O' and a priority queue T. Each vertex, denoted as u_i from $O[i]$, is inserted into T, with neighbors marked as gray to handle dependencies and avoid reprocessing. In each iteration, vertices are added to or removed from T based on their peeling weights, updating O' accordingly. After processing all vertices in the queue, O' is extended with any remaining unmarked (white) vertices. The algorithm concludes by returning the complete sequence and an optimal subgraph S_i.

Complexity. The time complexity of PEVAL is $O(|E|\log|V|)$, identical to that of [21]. Meanwhile, the complexity of INCEVAL is also $O(|E|\log|V|)$, identical to that of [27], [23].

6 INDEX IN P2PG

As presented in Section 2, the definitions of keyword search semantics often involve the shortest distances of nodes, *e.g.*, [20], [30], [4], and [50]. Their query algorithms require numerous shortest distance computations. For example, when applying r-clique [30] on the PP-Graph $G_c = G \oplus G'_3$ in Figure 2, finding an answer of $Q = \{a, b, c\}$, as shown in Figure 2(c), requires 12 shortest distance computations in PEVAL for r-clique on G' (Algorithm 1, Line 7), and 8

shortest distance computations on $G \oplus G'$ (Algorithm 2). Hence, we propose indexes for the public graph G and the private graph G' *respectively*, so as to optimize the query processing on the PP-Graph $G \oplus G'$.

Firstly, to avoid the exhaustive search for the distances of shortest paths that *cross the public graph and private graph*, we propose PADS and KPADS for estimating the shortest distances between the vertices in the public graph and those between the keywords and vertices in the public graph (Section 6.1 and Section 6.2). Secondly, we index the portal nodes by precomputing their all-pair shortest distances (Section 6.3). Thirdly, we introduce a portal-keyword distance map to store the shortest distances between the portal nodes and the keywords (Section 6.3).

6.1 PageRank-based All Distance Sketches

In this subsection, we review ADS and then propose our index. It is known that ADS is small in size, accurate, and efficient in answering shortest distance queries. Our main idea is to use PageRank to determine the chance of a node to be included in the sketch (*i.e.*, the index).

All-Distances Sketches (ADS). Recall that in [9], given a graph $G = (V, E)$, each vertex v is associated with a sketch, which is a set of vertices and their corresponding shortest distances from v. To select the vertices in V and put them as the centers in the sketch of v, each vertex is initially assigned a *random* value in $[0, 1]$. If a vertex $u \in V$ has the k-th largest value among the vertices which have been traversed from v in the Dijkstra order, then u is added to the sketch of v. k is a user-defined parameter set by user. A larger k results in larger and more accurate sketches. The shortest distance between u and v can be estimated by the intersection set of ADS(u) and ADS(v) (a.k.a. the common centers).

A drawback of ADS is that it does not consider the relative importance of the vertices when generating the sketch. We observe that vertices with high PageRanks, which roughly estimates the importance of the vertices in a graph, should be added to the sketch to cover the shortest paths. On the contrary, the vertices with low PageRanks are unlikely to be on many shortest paths and should not be added to the sketch.

PageRank. We employ any efficient algorithms to obtain the PageRank of the vertices of a graph G. We use a function pr: $V \rightarrow$ [0,1] to denote the PageRank of a vertex v by $pr(v)$.

Dijkstra rank. We recall that we can efficiently obtain the Dijkstra rank of a vertex v w.r.t a source vertex s as follows. We run the Dijkstra's algorithm starting at s and obtain the order of the visited nodes $[v_1, v_2, \ldots, v_l]$. The Dijkstra rank of v_i w.r.t s is i, denoted as $\pi(s, v_i) = i$.

PageRank-based All-Distances Sketches (PADS**)**. Given a Dijkstra rank π, the PageRank, a vertex v, and a threshold k, the PADS of v is defined as follows:

$$\text{PADS}(v) = \{(u, d(v, u)) \mid pr(u) \geq k(v, u)\}, \tag{3}$$

where $k(v, u)$ is the k-th largest PageRank among the nodes from v to u according to π.

Example 6.1. (PADS *Construction*) *Consider the public graph G in Figure 5. Assume $k = 1$. We compute the PageRank values for all the vertices in the graph, as shown below the vertices' labels. v_{13} covers 41 out of 156 shortest paths in the graph G in total, which is the largest among all the vertices. This shows that the node having a large PageRank value, $pr(v_{13}) = 0.130$, can be an effective center. To determine the* PADS *of v_1, we run the Dijkstra's algorithm by taking v_1 as the source vertex to obtain the Dijkstra ranked list $[v_1, p_1, p_2, v_{13}, v_4, \ldots, p_7]$. Since the PageRank value of v_{13} is the highest among the first four vertices in the ranked list, v_{13} is added to* PADS(v_1) *with its distance to v_1. Similarly, v_1 is added to* PADS(v_1).

Algorithm 7: PADS construction

```
Input: Graph G = (V, E)
Output: PADS
1  compute the PageRank pr of the vertices in G
2  initialize PADS(v) = {(v, 0)} for each vertex v ∈ V
3  sorted the vertices V by the descending order of pr(v)
4  for v ∈ V do
5      for u in the Dijkstra's traversal do
6          if |{(w, d) ∈ PADS(u) | d ≤ d(v, u)}| < k then
7              add (v, d(v, u)) into PADS(u)
8          else
9              continue the traversal on the next vertex
10 return PADS
```

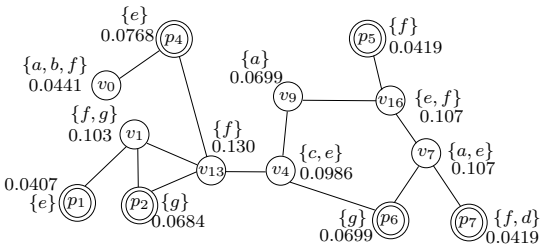


Fig. 5: A Public Graph (Fragment) and the PageRank

Shortest Distance Estimation. Given a shortest distance query (u, v) and the PADS, $\hat{d}(u, v)$ is computed by the intersection of PADS(u) and PADS(v) as follows:

$$\hat{d}(u, v) = \min\{(d_1 + d_2)\}, \tag{4}$$

where $(w, d_1) \in$ PADS(u), $(w, d_2) \in$ PADS(v).

Space Complexity. The expected size of PADS(v) is $O(k \ln n)$, where n is the number of nodes reachable from v, which is bounded by $O(k \ln |V|)$.

Time Complexity. Each iteration of PageRank and Dijkstra rank are both computed in $O(|V|+|E|)$. In Algorithm 7, the times each edge (v, u) has been traversed is bounded by the size of PADS(v) (Line 6). Since the expected size of PADS(v) is bounded by $k \ln |V|$, the time complexity of Algorithm 7 is $O(k|E| \ln |V|)$ (cf. [9]).

Consider the graph G in Figure 5. We set $k = 1$ and compute its ADS shown in Table 2 and its PADS shown in Table 3. There are two advantages of PADS. First, the size of PADS is smaller than that of ADS. Second, the estimation of PADS is much more accurate than that of ADS.

Example 6.2. (*Shortest Distance Estimation.*) *Consider the graph G in Figure 5 and its* PADS*s in Table 3. Given two vertices v_9 and v_7, there are two common centers v_{16} and v_{13} in* PADS(v_9) *and* PADS(v_7). *The shortest distance is estimated by Eq. 4, i.e., $\hat{d}(v_9, v_7) = 2$ (i.e., 0% error). By* ADS*, $\hat{d}(v_9, v_7) = 4$ is returned (i.e., 100% error). More specifically, we compare the estimation accuracy of* ADS *and* PADS *between all pairs of the*

TABLE 2: An ADS label for the public graph in Figure 5

Vertex ID	ADS
v_0	$\{(v_0,0),(p_4,1),(v_1,3),(p_1,4),(p_7,6)\}$
p_4	$\{(p_4,0),(v_1,2),(p_1,3),(p_7,5)\}$
v_{13}	$\{(v_{13},0),(p_4,1),(v_1,1),(p_1,2),(p_7,4)\}$
v_1	$\{(v_1,0),(p_1,1),(p_7,5)\}$
p_1	$\{(p_1,0),(p_7,6)\}$
p_2	$\{(p_2,0),(v_1,1),(p_1,2),(p_7,5)\}$
v_4	$\{(v_4,0),(v_{13},1),(v_9,1),(p_4,2),(v_1,2),(p_1,3),(p_7,3)\}$
v_9	$\{(v_9,0),(p_4,3),(v_1,3),(p_7,3)\}$
p_6	$\{(p_6,0),(v_4,1),(v_{13},2),(v_9,2),(p_7,2)\}$
v_{16}	$\{(v_{16},0),(v_9,1),(p_7,2)\}$
v_7	$\{(v_7,0),(v_{16},1),(p_7,1)\}$
p_5	$\{(p_5,0),(v_9,2),(p_7,3)\}$
p_7	$\{(p_7,0)\}$

TABLE 3: The PADS label for the public graph in Figure 5

Vertex ID	PADS
v_0	$\{(v_0,0),(p_4,1),(v_{13},2)\}$
p_4	$\{(p_4,0),(v_{13},1)\}$
v_{13}	$\{(v_{13},0)\}$
v_1	$\{(v_1,0),(v_{13},1)\}$
p_1	$\{(p_1,0),(v_1,1),(v_{13},2)\}$
p_2	$\{(p_2,0),(v_1,1),(v_{13},1)\}$
v_4	$\{(v_4,0),(v_{13},1)\}$
v_9	$\{(v_9,0),(v_4,1),(v_{16},1),(v_{13},2)\}$
p_6	$\{(p_6,0),(v_4,1),(v_7,1),(v_{13},2)\}$
v_{16}	$\{(v_{16},0),(v_7,1),(v_{13},3)\}$
v_7	$\{(v_7,0),(v_{16},1),(v_{13},3)\}$
p_5	$\{(p_5,0),(v_{16},1),(v_7,2),(v_{13},4)\}$
p_7	$\{(p_7,0),(v_7,1),(v_{16},2),(v_{13},4)\}$

vertices in Figure 5. The average error of PADS *(resp. ADS) is around* 3% *(resp.* 38%*).*

It is worth noting that PADS exhibits the theoretical guarantee of the shortest path estimation stated below.

Lemma 6.1. *The distance between two vertices* u *and* v *is estimated using Eq. 4 with an approximation factor* $(2c-1)$*, where* $c = \lceil \frac{\ln|V|}{\ln k} \rceil$ *with a constant probability, i.e.,* $\hat{d}(u,v) \leq (2c-1)d(u,v)$.

Proof. Let $d = d(u,v)$. Let $N_i(u)$ denote the neighbors of u within id hops. For simple exposition, we denote the intersections of $N_i(u)$ and $N_{i-1}(v)/N_{i+1}(v)$ as $I_{2i-1} = N_i(u) \cap N_{i-1}(v)$ and $I_{2i} = N_i(u) \cap N_{i+1}(v)$, respectively. It is worth noting that $I_{i-1} \subseteq I_i$. Consider the ratio of $\frac{|I_{i-1}|}{|I_i|}$ and a ratio threshold $\frac{\alpha}{k}$. Given the vertices with k largest pr values in I_i, if one of them (say w) hits I_{i-1}, w belongs to both PADS(v) and PADS(u). The distance d can be estimated within $(2i-1)d$. The probability of *at least one of the vertices, which has the k largest PageRank values in* I_i*, hits the* I_{i-1} is $1-(1-\frac{\alpha}{k})^k \approx 1-e^{-\alpha}$. Since there are n vertices in graph G at most, $|I_i| \leq n$. Hence, there exists $i \leq \log_{k/\alpha} n$. □

6.2 PageRank-based Keyword Distance Sketches

We denote the *shortest distance between a vertex* v and *a keyword* t by $d(v,t)$, where $d(v,t) = \min\{d(v,u)|t \in L(u), u \in V\}$. To estimate the distance between a given vertex and a keyword, we propose KPADS, which is constructed by PADS-merging: Given any two vertices u and u' where $t \in L(u)$ and $t \in L(u')$, there may exist common centers in PADS(u) and PADS(u'). Hence, we only keep the smaller one between $\hat{d}(v,u')$ and $\hat{d}(v,u)$, since both of them are the upper bound of $d(v,t)$.

Keyword-PADS **(KPADS).** For each keyword $t \in \Sigma$, we build a sketch KPADS(t). KPADS(t) can be built by merging PADS of those vertices that contain t, *i.e.*, PADS(v) where $t \in L(v)$. More formally, given a center $(w_i,d_i) \in$PADS(v), $(w_i,d_i) \in$ KPADS(t) *iff* $\forall(w_i,d_i') \in$PADS(v') and $t \in L(v')$, $d_i' \geq d_i$.

TABLE 4: The KPADS label for the public graph in Figure 5

Terms	KPADS
a	$\{(v_9,0),(v_4,1),(p_4,1),(v_7,0),(v_{13},2),(v_{16},1),(v_0,0)\}$
b	$\{(v_0,0),(v_{13},2),(p_4,1)\}$
c	$\{(v_{13},1),(v_4,0)\}$
d	$\{(v_{13},4),(v_7,1),(p_7,0),(v_{16},2)\}$
e	$\{(v_{13},1),(v_4,0),(v_1,1),(v_7,0),(p_4,0),(v_{16},0),(p_1,0)\}$
f	$\{(p_5,0),(v_1,0),(v_{13},0),(p_4,1),(v_7,1),(v_{16},0),(v_0,0),(p_7,0)\}$
g	$\{(p_6,0),(v_1,0),(v_4,1),(v_{13},1),(v_7,1),(p_2,0)\}$

Shortest Keyword-Vertex Distance Estimation. Given a vertex v and a keyword t, the shortest distance $\hat{d}(v,t)$ can be computed as follows:

$$\hat{d}(v,t) = \min\{(d_1+d_2)|(w,d_1) \in \text{PADS}(v), (w,d_2) \in \text{KPADS}(t)\} \quad (5)$$

Example 6.3. *Consider the graph* G *in Figure 5 and its* PADS *in Table 3. The KPADS is shown in Table 4. Consider the shortest distance between* a *and* p_4*. The distance is estimated by the intersection of* KPADS(a) *and* PADS(p_4)*. Two common centers are* p_4 *and* v_{13}*.* $\hat{d}(a,p_4) = 1$ *is returned by the common center* p_4*.*

Lemma 6.2. *The distance between a vertex* v *and a keyword* t *derived from Eq. 5 has an approximation factor* $(2c-1)$ *where* $c = \lceil \frac{\ln|V|}{\ln k} \rceil$ *with a constant probability, i.e.,* $\hat{d}(v,t) \leq (2c-1)d(v,t)$.

Proof. Given a vertex v and a keyword t, we denote the vertex which is the closest to v and contains t as u, i.e. for any vertex u' where $t \in L(u')$, $d(v,u') \geq d(v,u)$. And $\hat{d}(v,u)$ can be estimated with the same approximation factor, $(2c-1)$, by PADS(v) and PADS(u) with the same probability, $1-e^{-\alpha}$, of Lemma 6.1. We denote the common center by w_i.

$$d(v,w_i) + d(w_i,u) \leq (2c-1)d(v,u) \quad (6)$$

By the definition of PADS-merging (we compress the common centers while keep smallest distance), we have $(w_i,d_i) \in$ KPADS, and $d_i \leq d(w_i,u)$.

$$d(v,w_i) + d_i \leq d(v,w_i) + d(w_i,u) \leq (2c-1)d(v,u) \quad (7)$$

□

Time Complexity. The time complexity of the shortest distance estimation between a vertex and a keyword (or another vertex) is $O(k \ln|V|)$ (Appendix A.2 of [28]).

Index Size. The size of KPADS(t) is bounded by $\sum|\text{PADS}(v_i)|$. Therefore, the total size of KPADS for all the terms is bounded by $\sum_{v_i \in V}|L(v_i)|\text{PADS}(v_i)$. In practice, $|L(v_i)|$ is often small.

Query Processing with the Indexes. We take Blinks as an example. It takes $O(|E|+|V|\ln|V|)$ to complete an answer of Blinks on the public graph G by Dijkstra's algorithm with Fibonacci heap (Algorithm 4, Line 22). With KPADS, this procedure can be done in $O(|Q|k \ln|V|)$.

6.3 Indexes of Portal Distances

The shortest distance computations on the PP-Graph can be time-consuming. In this subsection, we index the shortest distances of the portal nodes since the number of portal nodes $|\mathbb{P}|$ is small when compared to $|V|$. We then extend the idea to index the distances of portal and keyword nodes.

Portal Distance Maps. We call the shortest distance between two portal nodes *the portal distance*. We precompute all the portal distances of $\mathbb{P}$ on the public graph G (denoted as d) and the private graph G' (denoted as d'), respectively. We

Algorithm 8: Portal Distance Map Construction

Input: All pair portal distance on private graph $d'(p_i, p_j)$, All pair portal distance on public graph $d(p_i, p_j)$
Output: All-Pairs portal distance on the combined graph

1 initialize a priority queue $Queue$
2 **for** $p_i, p_j \in \mathbb{P}$ **do**
3 **if** $d(p_i, p_j) \geq d'(p_i, p_j)$ **then**
4 $d(p_i, p_j) = d'(p_i, p_j)$
5 $Queue$.INSERT($\langle p_i, p_j, d(p_i, p_j)\rangle$)
6 **while** $Queue$ *is not empty* **do**
7 $\langle p_1, p_2, dist\rangle$ = $Queue$.REMOVETOP();
8 **for** $p_i \in \mathbb{P}$ **do**
9 **if** $d(p_i, p_2) \geq d(p_i, p_1) + dist$ **then**
10 $d(p_i, p_2) = d(p_i, p_1) + dist$
11 $Queue$.INSERT($\langle p_i, p_2, d(p_i, p_2)\rangle$)
12 **if** $d(p_i, p_1) \geq d(p_i, p_2) + dist$ **then**
13 $d(p_i, p_1) = d(p_i, p_2) + dist$
14 $Queue$.INSERT($\langle p_i, p_1, d(p_i, p_1)\rangle$)
15 **return** d

index the distances in distance maps, d and d', respectively. We can then efficiently index the portal distances of the PP-Graph G_c as follows.

Step 1. Portal Distance Refinement (Algorithm 8). We first refine the portal distance in the private graph in the presence of those in the public graph (Lines 3-5). We use a priority queue $Queue$ to maintain the refined portal distances. If $d(p_i, p_j) \leq d'(p_i, p_j)$, where $p_i, p_j \in \mathbb{P}$, we refine $d'(p_i, p_j)$ to $d(p_i, p_j)$ (Line 4) and insert the pair with the distance into $Queue$ (Line 5). Next, we pop $\langle p_1, p_2, dist\rangle$ from the head of $Queue$, when $Queue$ is not empty. For each $p_i \in \mathbb{P}$, if the sum of $d(p_i, p_1)$ and the refined portal distance $d(p_1, p_2)$ is smaller than the current portal distance $d(p_i, p_2)$, there is a shorter path between p_i and p_2 via p_1. Then, the portal distance between p_i and p_2 can be refined. Similarly, the distance between p_i and p_1 can be refined by p_2.

Step 2. Shortest Distance Refinement using Portal Distance. We next reduce the refinement of shortest distance via the portal distance maps described above. To index the shortest distances of the combined graph, we compare the shortest distance in the private graph and the length of the paths crossing the portal nodes as follows:

$$d_c(v_1, v_2) = \min \begin{cases} d'(v_1, v_2); \\ d'(v_1, p_i) + d'(p_j, v_2) + d_c(p_i, p_j), \text{where } p_i, p_j \in \mathbb{P}. \end{cases} \quad (8)$$

Portal-Keyword Distance Map. We extend the idea to the distances between the portal nodes and the keywords, and index them in a *portal-keyword distance map*, denoted as PKD. More formally, given a portal node $p \in \mathbb{P}$ and $t \in G'.\Sigma$, $\mathsf{PKD}(p, t)$ is a tuple $\langle \mathsf{v}, \mathsf{d}\rangle$, where 1) $\mathsf{PKD}(p, t).\mathsf{v} \in G'.V$ is the nearest vertex v of p such that $t \in L(v)$ and 2) $\mathsf{PKD}(p, t).\mathsf{d} = d'(p, v)$.

Vertex-Portal Distance Map. We also index the distances between the vertex of the private graph and each portal node, denoted by $d'(v, p)$, where $v \in G'.V$ and $p \in \mathbb{P}$. Hence, the refinement between $v \in G'.V$ and $t \in G'.\Sigma$ can be computed by Formula (9).

$$d_c(v, t) = \min \begin{cases} d'(v, t); \\ d'(v, p_i) + d_c(p_i, p_j) + \mathsf{PKD}(p_j, t).\mathsf{d}, \end{cases} \quad (9)$$

where $p_i, p_j \in \mathbb{P}$.

Query Processing with the Indexes. The indexes proposed in this section significantly improve the performance of answer refinement. For example, the refinement time of each r-clique answer reduces to $O(|Q||\mathbb{P}|^2)$, since the distances (Algo 2, Line 3) have been precomputed. Without the indexes, it takes $O(|Q||\mathbb{P}|^2(|E|+|V|\ln|V|))$ by running the Dijkstra's algorithm on $G \oplus G'$.

TABLE 5: Statistics of real-world datasets

Datasets	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	avg. # of keywords	$\lvert\mathbf{V}'\rvert$	$\lvert\mathbf{E}'\rvert$
YAGO3	2,635,317	5,260,573	3.79	482	501
DBpedia	5,795,123	15,752,299	3.72	538	873
DBLP	1,791,688	5,187,025	10	9.2	27.6

TABLE 6: Characteristics of PADS and ADS

Datasets	Construction Time		Size (# of centers)		Approx. ratio	
	ADS	PADS	ADS	PADS	ADS	PADS
YAGO3	5096s	5066s	28.79M	20.57M	1.08452	1.00001
DBpedia	39237.3s	38757s	103.65M	74.21M	1.13194	1.0059
PP-DBLP	3761s	2770s	20.49M	15.15M	1.06178	1.00284

7 Experimental Study

We used real-life datasets to conduct three sets of experiments to evaluate P2PG for their (1) index characteristics, (2) query performance and (3) optimization performance.

7.1 Experimental Setup

7.1.1 Software and hardware

Our experiments were run on a machine with a 2.93GHz CPU and 64GB memory running CentOS 7.4. The implementation was made memory-resident.

7.1.2 Algorithms

We implemented Blinks and r-clique in C++ and used the same settings as presented in the original works. For Blinks, we adopted METIS for partitioning. For r-clique, we built the neighbor index with $R = 3$, as in [30]. We obtained the code of knk from [29] and used the same setting. We designed the baseline algorithms as follows. 1) For Baseline-Blinks and Baseline-r-clique, we extended Blinks and r-clique with a simple qualification function to verify if an answer is a valid public-private answer and applied them on the combined graph G_c. For Baseline-knk, we directly applied knk on the combined graph G_c.

7.1.3 Datasets and default indexes

Table 5 summarizes the characteristics of the datasets used.

YAGO3.[1] YAGO3 [37] is a large knowledge base, derived from Wikipedia, WordNet and GeoNames. In the experiment, we extracted the entities (vertices) and the corresponding facts (edges) in specific domains (*e.g.*, chemistry, or movies) to form the private graphs. The rest of the entities and facts formed the public graphs.

DBpedia.[2] DBpedia (v3.9) is a knowledge graph with 5.8M vertices and 15.8M edges. It extracts structured content from the information created in various Wikimedia projects. Similar to YAGO3, we derived private graphs of DBpedia from specific domains. The rest of the entities and facts form the public graph.

1. http://www.mpi-inf.mpg.de/yago
2. http://dbpedia.org

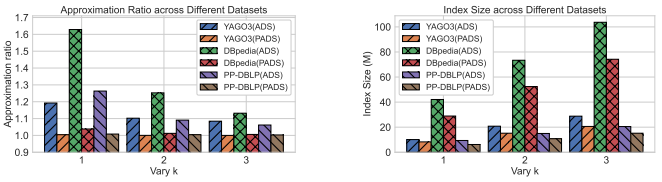


Fig. 6: Comparison between PADS and ADS

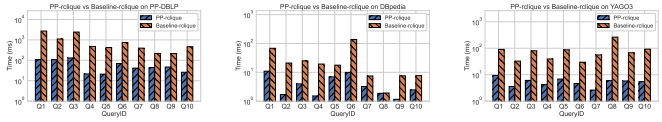


Fig. 7: Comparison of PP-r-clique and Baseline-r-clique.

Intuitively, the information from a specific domain can be kept privately by their owners (*e.g.*, private laboratories, or movie investors). Hence, we extracted the entities in a specific domain by retrieving YAGO3's ontology graph, *i.e.*, all the descendant entities of the domain term (*e.g.*, chemicals and moive information etc) will be returned. All these entities consist of V' and the corresponding induced subgraph of the ontology graph form E'. The portal node set $\mathbb{P} = \{v | v \in V' \text{ and } v \in V\}$. In this section, we present the performance results of the experiments that used the entities in chemistry and movie domains as private graphs.

PP-DBLP.[3] We used public-private graphs from real-world DBLP records, called PP-DBLP [22]. We set the "current" time as 2013. Existing collaborations made the public graph, while ongoing collaborations formed the private graphs, as they were only known by some authors.

7.1.4 Queries

We generated 50 random synthetic keyword queries for the experiments. Some details are given below. For each algorithm, we report the results of 10 queries, including three good, three bad, and four medium cases.

Blinks *and* r-clique. The query keyword $q \in Q$ was randomly picked from the label set $G.\Sigma$ and $G'.\Sigma$. For Blinks, we set the pruning threshold d_{max} (a.k.a. τ_{prune} in [20]) to 5 to ensure keyword nodes were reachable from the root vertex within 5 hops. We remark that if $Q \cap G'.\Sigma = \emptyset$ or $Q \cap G.\Sigma = \emptyset$, Q has no public-private answer. Users obtain the public answers (resp. private answers) by passing the public graph G (resp. private graph G') to PEVAL as input. But P2PG does not offer the performance improvement. As a consequence, Q cannot show the performance of INCEVAL. To make sure the public-private answers exist and investigate the runtimes of the three key steps, we generate Q s.t. $Q \cap G'.\Sigma \neq \emptyset$ and $Q \cap G.\Sigma \neq \emptyset$.

knk. We note that the frequency of a keyword in the private graph is smaller than 64. Again, to study public-private answers, we generated the query (v, q, k), where k was set to 64, v was randomly picked from $G'.V$, and q was selected following the keyword distribution of the combined graph.

7.2 Experimental Evaluation for P2PG-Q

Exp-1: Characteristics of P2PG-Q. We next report the size of the PADS and the time of constructing KPADS. We also present the efficiency and effectiveness of P2PG-Q in Table 6 and Figure 6.

3. https://github.com/samjjx/pp-data

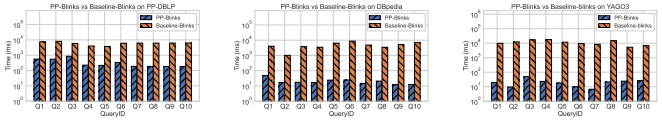


Fig. 8: Comparison of PP-Blinks and Baseline-Blinks.

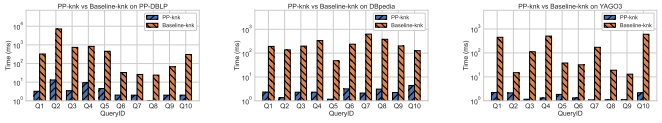


Fig. 9: Comparison of PP-knk and Baseline-knk.

Index Sizes. For comparison, we implemented [10] for the shortest distance estimation. For real-life graphs, PADS is 28.6% (resp. 28.5% and 26.1%) smaller than ADS on YAGO3 (resp. DBpedia and PP-DBLP).

Construction Time. We report the construction times in Table 6. P2PG takes 1.41 hours (resp. 10.8 hours and 46 minutes) to construct PADS for YAGO3 (resp. DBpedia and PP-DBLP). The construction time on PADS and ADS is slightly different except PP-DBLP. The construction time of PADS is 26.4% smaller than that of ADS.

Accuracy. We randomly selected a vertex pair (s, t) from $|V|$. We compared the accuracy of PADS with ADS by computing the shortest distances between each vertices pair, denoted as $\hat{d}(s, t)$. The exact distance between s and t was computed using Dijkstra's algorithm [13], denoted as $d(s, t)$. We denoted the error as $\epsilon = \frac{\hat{d}(s,t) - d(s,t)}{d(s,t)}$. We repeated the above procedure 1 million times and got the average error $\bar{\epsilon}$. As we presented in Figure 6, we varied the parameter k from 1 to 3. On YAGO3, $\bar{\epsilon}$ of PADS reduces from 4.2×10^{-3} to 1×10^{-5}. Similarly, $\bar{\epsilon}$ of PADS also decreases significantly on DBpedia and PP-DBLP when k increases. We set $k = 3$ for the comparison between PADS and ADS. $\bar{\epsilon}$ of PADS is 99.99% (resp. 96.53% and 95.40%) smaller than that of ADS on YAGO3 (resp. DBpedia and PP-DBLP).

Exp-2: Query Performance. To evaluate the efficiency of P2PG, we have tested the performance of Blinks, r-clique and knk with and without P2PG.

r-clique (Figure 7). P2PG is 12.11 times faster on average. (1) On PP-DBLP, the query is at most 24.75 times and at least 4.5 times faster than the baseline algorithm. For all the queries, it is 14.30 times faster on average. (2) On DBpedia, the query is at most 13.79 times faster and at least 2.3 times than the baseline algorithm. For all the queries, it is 6.69 times faster on average. (3) On YAGO3, the query is at most 44.09 times, at least 6.31 times and on average 15.4 times faster than the baseline algorithm. This is because Baseline-r-clique requires exploration of the whole search space derived from the PP-Graph, even the queries have public-private answers.

Blinks (Figure 8). P2PG runs in 202 times faster on average. (1) On PP-DBLP, the query is at most 33.97 times and at least 6.84 times faster than the baseline algorithm. For all the queries, it is 22 times faster on average. (2) On DBpedia, the query is at most 554 times and at least 60 times faster than the baseline algorithm. For all the queries, it is 268

TABLE 7: Query Performance Breakdown for r-clique (ms)

	PP-DBLP		DBpedia		YAGO3	
QueryID	PEVAL	INCEVAL	PEVAL	INCEVAL	PEVAL	INCEVAL
Q1	0.079	108.883	0.775	9.298	0.857	8.571
Q2	0.135	107.623	0.076	1.628	0.147	3.617
Q3	0.467	130.902	0.181	3.786	0.136	5.980
Q4	0.052	21.537	0.058	1.457	0.705	3.529
Q5	0.124	21.010	0.293	6.670	0.893	5.981
Q6	1.899	67.266	0.486	9.458	0.077	4.623
Q7	0.502	41.112	0.165	3.063	0.150	2.493
Q8	0.680	43.029	0.096	1.776	0.400	5.579
Q9	0.617	46.260	0.061	1.100	0.461	5.348
Q10	0.399	25.748	0.141	2.310	0.469	5.015

TABLE 8: Query Performance Breakdown for Blinks (ms)

	PP-DBLP		DBpedia		YAGO3	
QueryID	PEVAL	INCEVAL	PEVAL	INCEVAL	PEVAL	INCEVAL
Q1	0.214	556.015	0.334	47.158	0.392	18.786
Q2	0.208	549.014	2.2	14.76	0.165	9.269
Q3	0.324	851.013	1.35	16.02	0.094	49.075
Q4	0.121	224.014	1.04	15.66	0.345	22.8
Q5	0.191	217.013	0.094	24.002	0.371	17.534
Q6	0.344	331.014	0.151	24.422	0.195	9.563
Q7	0.238	185.014	1.02	13.73	0.308	6.341
Q8	0.235	185.013	0.84	19.94	0.225	21.035
Q9	0.231	184.013	0.58	11.3	0.2734	23.71
Q10	0.236	184.013	0.67	11.62	0.249	25.713

times faster on average. (3) On YAGO3, the query is at most 890 times, at least 77 times and on average 315 times faster than the baseline algorithm.

knk (Figure 9). P2PG runs 120 times faster (on average) than the baseline algorithms. On average, PP-knk is 128 times (resp. 110 times and 120 times) faster than Baseline-knk on PP-DBLP (resp. DBpedia and YAGO3).

Exp-3: Query Performance Breakdown. We next report the query performance breakdown (Table 7-9).

r-clique. For PP-DBLP, our findings reveal that PEVAL, on average, occupies only about 0.97% of the total query time, while INCEVAL accounts for a substantial 99.03%. This is primarily due to the small size of private graphs, which are processed relatively quickly, whereas the retrieval from the more extensive public graphs is more time-consuming. Similar trends are observed in the DBpedia and YAGO3 datasets, indicating a consistent pattern across various semantic databases where INCEVAL dominates the computational effort. Despite the significant processing involved, the query times remain within 200 milliseconds, ensuring rapid response to queries.

Blinks. In our experiments on PP-DBLP, an overwhelming 99.9% of the query time is dedicated to INCEVAL, rendering the time spent on PEVAL negligible. Similar patterns are observed in the YAGO3 and DBpedia datasets. Specifically, in the DBpedia dataset, P2PG allocates approximately 5% of the query time to PEVAL and 95% to INCEVAL. In contrast, on YAGO3, P2PG dedicates 1.7% of the query time to PEVAL and 98.3% to INCEVAL. These variations can be attributed to the structural differences among the datasets: the average number of nodes within x hops of the portal nodes in PP-DBLP significantly exceeds those in YAGO3 and DBpedia. Consequently, as the number of vertices traversed in the public graph increases, so does the time consumed by INCEVAL, reflecting the computational intensity required to manage larger graph traversals.

TABLE 9: Query Performance Breakdown for knk (ms)

	PP-DBLP		DBpedia		YAGO3	
QueryID	PEVAL	INCEVAL	PEVAL	INCEVAL	PEVAL	INCEVAL
Q1	3	0.260	2	0.344	2	0.223
Q2	10	3.184	1	0.362	2	0.153
Q3	3	0.443	2	0.305	1	0.166
Q4	8	1.277	2	0.308	1	0.322
Q5	4	0.588	1	0.170	1	0.186
Q6	2	0.030	1	0.178	1	0.341
Q7	2	0.017	3	0.120	1	0.098
Q8	1	0.020	3	0.110	1	0.109
Q9	2	0.018	2	0.225	1	0.119
Q10	2	0.033	4	0.425	2	0.175

knk. In the PP-DBLP dataset, P2PG allocates 92.2% of the query processing time to PEval, with the remaining 7.8% devoted to INCEVAL. This distribution is echoed in our experiments with the YAGO3 and DBpedia datasets, demonstrating a consistent pattern across these platforms. Specifically, in the DBpedia dataset, P2PG spends 87.5% of the time on PEval and 12.5% on INCEVAL. Similarly, in the YAGO3 dataset, 86.6% of the time is dedicated to PEval, with 13.4% allocated to INCEVAL. These results underscore a characteristic trend in which a substantial majority of the query time is consistently devoted to PEval across diverse datasets, indicating the computational effort concentrated in primary evaluation processes compared to auxiliary reference activities

7.3 Experimental Evaluation for P2PG-A

We designate the "current" year as 2013, 2014, 2015, and 2016, respectively, for our analysis. In this framework, existing collaborations contribute to the formation of the public graph, while ongoing collaborations, known only to certain participants, create the private graphs. The four data sets utilized in this study are denoted by PP-DBLPX, where X represents the respective year of the dataset.

Exp-1: Efficiency. The results on the four PP-DBLP datasets are summarized in Table 10. The average speed-up ratio achieved by PP-DG (respectively, PP-DW and PP-FD) on P2PG-A is approximately 1.45×10^4 (respectively, 1.31×10^4 and 955) times greater than that of DG (respectively, DW and FD). This improvement is primarily attributed to the computation paradigm of P2PG-A, which emphasizes the reuse of previously computed results on public graphs rather than computing from scratch on PP-Graph.

Exp-2: Scalability. From PP-DBLP2013 to PP-DBLP2016, the scale of the public graph consistently increased, with the number of edges growing by approximately 42.24%. The increase in computation time is not monotonic due to the growing size of the public graph over time and the decreasing size of the private graph. Nevertheless, across all datasets, PP-DG, PP-DW, and PP-FD all respond within milliseconds. Notably, PP-FD generally consumes slightly more time than PP-DG and PP-DW. This increased computation time can be attributed to the edge weight function of PP-FD, which makes the dense subgraph structure more sensitive to newly inserted edges, thereby affecting a larger region of the graph.

7.4 Case Study

Exploring Potential Collaborations through Keyword Searches. As depicted in Figure 10, we present a partial

TABLE 10: Peeling time and speed up ratios for PP-DBLP datasets for DG, DW, and FD

Dataset	$\|V\|$	$\|E\|$	DG (s)	PP-DG (ms)	Speed-up Ratio	DW (s)	PP-DW (ms)	Speed-up Ratio	FD (s)	PP-FD (ms)	Speed-up Ratio
PP-DBLP2013	1,791,688	5,187,025	13.49	0.64	21,078	13.43	0.71	18,915	27.09	22.19	1,220
PP-DBLP2014	1,791,688	5,893,083	14.59	1.49	9,792	14.80	2.74	5,401	30.81	25.77	1,195
PP-DBLP2015	1,791,688	6,605,428	16.15	0.92	17,554	16.18	0.94	17,213	34.73	41.19	843
PP-DBLP2016	1,791,688	7,378,090	17.56	1.85	9,492	17.75	1.61	11,025	38.74	69.27	560

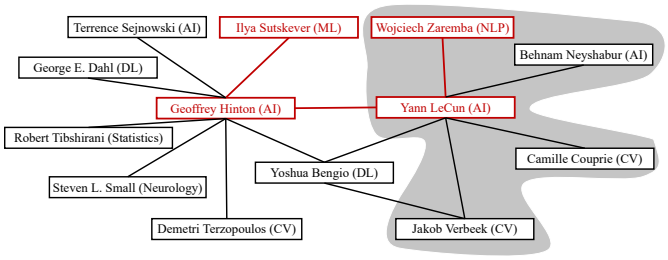
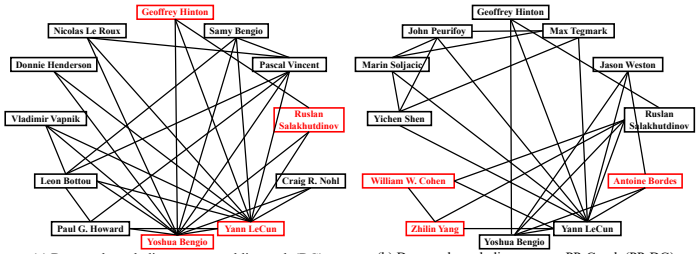


Fig. 10: A partial public collaboration network alongside a private network of Yann LeCun (highlighted in shadow). ML stands for Machine Learning and NLP for Natural Language Processing. A keyword search query {ML,NLP} on the PP-Graph returns the subgraph in red, featuring leaf vertices 'Ilya Sutskever' and 'Wojciech Zaremba', both co-founders of OpenAI.



(a) Dense subgraph discovery on public graph (DG)

(b) Dense subgraph discovery on PP-Graph (PP-DG)

Fig. 11: Dense subgraph discovery on PP-Graph (The public graph and a simulated private graph of Yann LeCun)

public graph from the PP-DBLP dataset with a simulated private graph of Yann LeCun, utilizing 2013 as the temporal boundary for the PP-Graph. Prior to 2013, there was no recorded collaboration between Ilya Sutskever and Wojciech Zaremba. However, by integrating the simulated private collaborations of Yann LeCun into the public graph, and conducting searches using 'ML' and 'NLP' as keywords, we are able to uncover potential collaboration between Ilya Sutskever and Wojciech Zaremba. Notably, both individuals later co-founded OpenAI and established a collaborative relationship. This case illustrates how P2PG can be instrumental in discovering potential social and collaborative connections within the PP-Graph.

Detecting Potential Conflicts of Interest in Scholarly Networks. Another application is conflict of interest (COI) detection. For instance, even if two scholars do not directly collaborate, they may still be part of a tightly-knit community that suggests a potential COI. We conducted a case study on the collaboration network involving Geoffrey Hinton, Yann LeCun, and Yoshua Bengio. We simulated paper COI detection by assuming collaborations before 2013 as part of the public network, while treating those after 2013 as private collaborations (e.g., papers currently under submission to conferences). Applying the DG algorithm to the graph, we identified dense subgraphs as shown in Figure 11(a), detecting existing COIs. Intriguingly, when we applied the PP-DG algorithm to the public network with the simulated private collaboration network post-2013 of Yann LeCun, we observed several interesting phenomena. Although there were no direct collaborations between Antoine Bordes, Zhilin Yang, and William W. Cohen in the public graph, the integration of the private collaborations (involving current submissions) placed them within a closely connected network (Figure 11(b)), still indicating a potential COI. This case study demonstrates how P2PG can effectively uncover potential COIs.

8 Related Work

Keyword Search Semantics. Recently, keyword search has attracted a lot of interest from both industry and research communities (*e.g.*, [20], [30], [4]). He et al. [20] propose an index and search strategies for reducing keyword search time. Kargar et al. [30] propose distance restrictions on keyword nodes, (*i.e.*, the shortest distance between each pair of keyword nodes is smaller than r). Ye et al. [50] propose a search strategy based on a compressed signature to avoid exhaustive search. These studies optimize a specific keyword search semantic. This work improves the performance of different existing keyword search semantics in a generic manner. We propose a P2PG framework for public-private keyword search. Their indexes and search strategies could be adopted in our framework with slight modifications.

Public-Private Graph Model. Some studies on *public-private graph analysis* have been conducted previously. Chierichetti [7] et al. propose two computational paradigms, sketching and sampling, for some key problems on massive public-private graphs. The sketching and sampling are precomputed offline and the online update algorithms are run on the private graphs. Ebadian [15] et al. propose a classification-based hybrid strategy to compute k-truss on public-private graphs, incrementally. Archer [1] et al. propose an approximation algorithm by seeking a set of seeding nodes to solve the reachability query on the public-private graph model. Huang [22] et al. develop a new model of attributed public-private networks by considering the information of vertices. Our work is different from these previous works as P2PG is the first work that studies different keyword search semantics on the public-private graph model.

Dense Subgraph Mining. The problem of finding the densest subgraph aims to identify the subgraph with the highest density within a given directed graph, attracting significant interest due to its potential in detecting fraud, spam, and community structures in social and review networks [6], [36], [18], [35], [2], [31]. While [2] offers algorithms for large-scale graphs using streaming and MapReduce, and [35] accelerates this via a novel convex-programming method, these are generally limited to static graph. Explorations into PP-Graph applications are still nascent; although [16] and [42] provide methods for dynamic graphs, and [45]

identifies dense subtensors in short durations, their scope is restricted to graph analysis and specific dense subgraph metrics. Contrasting these, our P2PG framework not only facilitates graph queries on PP-Graph but also extends to graph analysis, positioning it as a robust solution for deploying algorithms on PP-Graph. This study is pioneering in integrating both graph analysis and query algorithms on PP-Graph, to the best of our knowledge.

9 Conclusions

In this paper, we propose P2PG for supporting efficient graph query and analysis on the PP-Graph model. For graph queries, we implement several keyword search algorithms on P2PG. The proposed indexes, PADS and KPADS, not only offer theoretical guarantees in shortest distance estimation but also demonstrate high accuracy in practice, enhancing the efficiency of graph queries on P2PG. Similarly, we have shown how to implement dense subgraph detection algorithms on P2PG, showcasing its robust computational performance. Additionally, we carefully designed a set of user-friendly APIs and demonstrated how to implement graph algorithms on P2PG. Finally, our case studies illustrate the practical applications of P2PG in real-world scenarios, aiding in the discovery of potential collaborations and conflicts of interest

Acknowledgements. This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-TC-2021-002). This work is also partly supported by HKRGC GRF 12203123.

References

[1] A. Archer, S. Lattanzi, P. Likarish, and S. Vassilvitskii. Indexing public-private graphs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1461–1470, 2017.

[2] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5), 2012.

[3] Y. Ban, X. Liu, T. Zhang, L. Huang, Y. Duan, X. Liu, and W. Xu. Badlink: Combining graph and information-theoretical features for online fraud group detection. *arXiv preprint arXiv:1805.10053*, 2018.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

[5] R. Bramandia, B. Choi, and W. K. Ng. On incremental maintenance of 2-hop labeling of graphs. In *Proceedings of the 17th international conference on World Wide Web*, pages 845–854, 2008.

[6] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.

[7] F. Chierichetti, A. Epasto, R. Kumar, S. Lattanzi, and V. Mirrokni. Efficient algorithms for public-private social networks. In *SIGKDD*, pages 139–148. ACM, 2015.

[8] S. Choudhury, L. Holder, G. Chin, K. Agarwal, and J. Feo. A selectivity based approach to continuous pattern detection in streaming graphs. *arXiv preprint arXiv:1503.00849*, 2015.

[9] E. Cohen. All-distances sketches, revisited: Hip estimators for massive graphs analysis. *IEEE Trans. on Knowl. and Data Eng.*, 27(9):2320–2334, 2015.

[10] E. Cohen, D. Delling, F. Fuchs, A. V. Goldberg, M. Goldszmidt, and R. F. Werneck. Scalable similarity estimation in social networks: Closeness, node labels, and random edge lengths. In *Proceedings of the first ACM conference on Online social networks*, pages 131–142. ACM, 2013.

[11] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[12] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[14] R. Dey, Z. Jelveh, and K. Ross. Facebook users have become much more private: A large-scale study. In *PERCOM Workshops*, pages 346–352, 2012.

[15] S. Ebadian and X. Huang. Fast algorithm for k-truss discovery on public-private graphs. *arXiv preprint arXiv:1906.00140*, 2019.

[16] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th international conference on world wide web*, pages 300–310, 2015.

[17] W. Fan, X. Wang, and Y. Wu. Incremental graph pattern matching. *ACM Transactions on Database Systems (TODS)*, 38(3):1–47, 2013.

[18] A. Gionis and C. E. Tsourakakis. Dense subgraph discovery: Kdd 2015 tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2313–2314, 2015.

[19] N. V. Gudapati, E. Malaguti, and M. Monaci. In search of dense subgraphs: How good is greedy peeling? *Networks*, 77(4):572–586, 2021.

[20] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[21] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 895–904, 2016.

[22] X. Huang, J. Jiang, B. Choi, J. Xu, Z. Zhang, and Y. Song. Pp-dblp: Modeling and generating attributed public-private networks with dblp. In *ICDM*, 2018.

[23] J. Jiang, Y. Chen, B. He, M. Chen, and J. Chen. Spade+: A generic real-time fraud detection framework on dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[24] J. Jiang, B. Choi, X. Huang, J. Xu, and S. S. Bhowmick. Dkws: A distributed system for keyword search on massive graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[25] J. Jiang, B. Choi, J. Xu, and S. S. Bhowmick. A generic ontology framework for indexing keyword search on massive graphs. *IEEE Transactions on Knowledge and Data Engineering*, 33(6):2322–2336, 2019.

[26] J. Jiang, X. Huang, B. Choi, J. Xu, S. S. Bhowmick, and L. Xu. ppkws: An efficient framework for keyword search on public-private networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 457–468. IEEE, 2020.

[27] J. Jiang, Y. Li, B. He, B. Hooi, J. Chen, and J. K. Z. Kang. Spade: A real-time fraud detection framework on evolving graphs. *Proc. VLDB Endow.*, 16(3):461–469, nov 2022.

[28] J. Jiang, H. Xin, B. Choi, J. Xu, S. S. Bhowmick, and L. Xyu. ppkws: An efficient framework for keyword search on public-private networks. *https://www.comp.hkbu.edu.hk/%7Ejxjian/ppkws.pdf*, 2019.

[29] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong. Exact top-k nearest keyword search in large networks. In *SIGMOD*, pages 393–404. ACM, 2015.

[30] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.

[31] S. Khuller and B. Saha. On finding dense subgraphs. In *International colloquium on automata, languages, and programming*, pages 597–608. Springer, 2009.

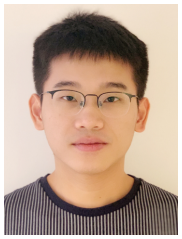
[32] K. Kim, I. Seo, W.-S. Han, J.-H. Lee, S. Hong, H. Chafi, H. Shin, and G. Jeong. Turboflux: A fast continuous subgraph matching system for streaming graph data. In *Proceedings of the 2018 International Conference on Management of Data*, pages 411–426. ACM, 2018.

[33] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *SIGMOD*, pages 173–182, 2006.

[34] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 world wide web conference*, pages 933–943, 2018.

[35] C. Ma, Y. Fang, R. Cheng, L. V. Lakshmanan, and X. Han. A convex-programming approach for efficient directed densest sub-

graph discovery. In *Proceedings of the 2022 International Conference on Management of Data*, pages 845–859, 2022.
[36] C. Ma, Y. Fang, R. Cheng, L. V. Lakshmanan, W. Zhang, and X. Lin. Efficient algorithms for densest subgraph discovery on large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1051–1066, 2020.
[37] F. Mahdisoltani, J. Biega, and F. Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *Seventh Biennial Conference on Innovative Data Systems Research*, 2014.
[38] R. Milner. *Communication and concurrency*, volume 84. Prentice hall Englewood Cliffs, 1989.
[39] B. Mirzasoleiman, M. Zadimoghaddam, and A. Karbasi. Fast distributed submodular cover: Public-private data summarization. In *Advances in Neural Information Processing Systems*, pages 3594–3602, 2016.
[40] M. Qiao, L. Qin, H. Cheng, J. X. Yu, and W. Tian. Top-k nearest keyword search on large graphs. *PVLDB*, 6(10):901–912, 2013.
[41] Y. Ren, H. Zhu, J. Zhang, P. Dai, and L. Bo. Ensemfdet: An ensemble approach to fraud detection based on bipartite graph. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2039–2044. IEEE, 2021.
[42] S. Sawlani and J. Wang. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 181–193, 2020.
[43] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment*, 1(1):364–375, 2008.
[44] K. Shin, T. Eliassi-Rad, and C. Faloutsos. Corescope: Graph mining using k-core analysis—patterns, anomalies and algorithms. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 469–478. IEEE, 2016.
[45] K. Shin, B. Hooi, J. Kim, and C. Faloutsos. Densealert: Incremental dense-subtensor detection in tensor streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1057–1066, 2017.
[46] S. Sun, X. Sun, B. He, and Q. Luo. Rapidflow: An efficient approach to continuous subgraph matching. *Proceedings of the VLDB Endowment*, 15(11):2415–2427, 2022.
[47] Y. Tao, S. Papadopoulos, C. Sheng, and K. Stefanidis. Nearest keyword search in xml documents. In *SIGMOD*, pages 589–600, 2011.
[48] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.
[49] D. Yu, X. Zhang, Q. Luo, L. Zhang, Z. Xie, and Z. Cai. Public-private-core maintenance in public-private-graphs. *Intelligent and Converged Networks*, 2(4):306–319, 2021.
[50] Y. Yuan, X. Lian, L. Chen, J. X. Yu, G. Wang, and Y. Sun. Keyword search over distributed graphs with compressed signature. *IEEE Trans. Knowl. Data Eng.*, 29(6):1212–1225, 2017.



Jiaxin Jiang is a research fellow in the School of Computing, National University of Singapore. He received his BEng degree in computer science and engineering from Shandong University in 2015 and PhD degree in computer science from Hong Kong Baptist University (HKBU) in 2020. His research interests include graph-structured databases, distributed graph computation and fraud detection.

Jingjie Zhan is currently pursuing his Bachelor's degree in Computer Science at Shandong University. He has distinguished himself in competitive programming, earning a gold medal at the Asia-Pacific Informatics Olympiad (APIO) and a first prize at the National Olympiad in Informatics in Provinces (NOIP). Additionally, Jingjie has been selected for the prestigious National Program for Specialized Training of Basic Science Talent. His research focuses on graph-structured databases and fraud detection.

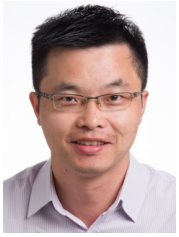
Lyu Xu is a PhD candidate in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in computer science and technology from Southeast University and MSc degree in computer science and technology from Sun Yat-sen University. His research interests include graph-structured databases and privacy preserving computation.

Byron Choi is a Professor in the Department of Computer Science at the Hong Kong Baptist University. He received the bachelor of engineering degree in computer engineering from the Hong Kong University of Science and Technology (HKUST) in 1999 and the MSE and PhD degrees in computer and information science from the University of Pennsylvania in 2002 and 2006, respectively. His research interests include graph data management and time series analysis.

Qiange Wang received the PhD degree in computer science from Northeastern University, China, in 2022. He is currently a postdoctoral research fellow at the National University of Singapore. His research interests include distributed graph processing, learning, and management systems.

Ning Liu received his Ph.D. degree in computer science and technology from Tsinghua university, Beijing, China, in 2021. He is currently an assistant professor in the School of Software, Shandong University, Jinan, China. His research interests mainly include data mining and knowledge-driven applications, especially textural data and sequential data mining. He now is a member of China Computer Federation.

Bingsheng He received the bachelor degree in computer science from Shanghai Jiao Tong University (1999-2003), and the PhD degree in computer science in Hong Kong University of Science and Technology (2003-2008). He is an Professor in School of Computing, National University of Singapore. His research interests are high performance computing, distributed and parallel systems, and database systems.



Jianliang Xu is a Professor in the Department of Computer Science, Hong Kong Baptist University (HKBU). He held visiting positions at Pennsylvania State University and Fudan University. He has published more than 150 technical papers in these areas, most of which appeared in leading journals and conferences including SIGMOD, VLDB, ICDE, TODS, TKDE, and VLDBJ. He has served as a program co-chair/vice chair for a number of major international conferences including IEEE ICDCS 2012, IEEE CPSNA 2015 and WAIM 2016. He is an Associate Editor of IEEE Transactions on Knowledge and Data Engineering (TKDE) and Proceedings of the VLDB Endowment (PVLDB 2018).

APPENDIX A
TOP-k NEAREST KEYWORD SEARCH (knk) ON P2PG

Due to space limitations, we present the knk semantic in this appendix, which is consistent with other previous works [33], [40], [29], [47]. A query of knk is a triple (v, q, k), where v is a query vertex and q is a query keyword. knk aims to find the k nearest vertices from v which contain the keyword q. More specifically, the knk semantic can be described as follows:

- **Input:** a query point v, a query keyword q
- **Output:** top k vertices $A = \{a = \{\langle u_i, d_i\rangle\}\}$ ranked by d_i, where $q \in L(u_i)$

4.4.(I) Overview of knk Any knk algorithms can be applied on the P2PG framework without any modification. Hence, we omit the overview of knk for simplicity.

4.4.(II) knk on P2PG (PP-knk)

(1) PEVAL. P2PG takes [29] as PEVAL to compute the knk answers on the private graph G'.

Partial answer $a' \in A'$. The partial answer a' is a list mat where the i-th element has two attributes $\langle \mathsf{u}, \mathsf{d}\rangle$. $a'.\mathsf{mat}[i].\mathsf{u}$ is a vertex u, such that $u \in V'$ and $q \in L(u)$ or $a'.\mathsf{u} \in \mathbb{P}$ and $a'.\mathsf{mat}[i].\mathsf{d} = d'(v, u)$. We use a boolean variable to record whether u is a portal. For the partial answer a', PEVAL declares $C = \{(v, u)\}$ to indicate what to be refined, where v and u are two vertices on the private graph. More specifically, v is the query point of knk and u is a candidate matched vertex in the private graph.

(2) INCEVAL. The refinement of the vertex pair $(v, u) \in C$ is identical to that discussed in Sec. 4.2. Given the refined answer a', P2PG completes a' in the public graph. For the i-th element of $a'.\mathsf{mat}$, *i.e.*, $\langle u, d\rangle$, if $q \in L(u)$, u is a candidate match vertex. Moreover, if u is a portal node, P2PG estimates the shortest distance between u and the keyword q in the public graph with the intersection of $\mathsf{PADS}(u)$ and $\mathsf{KPADS}(q)$. $\hat{d}(u, q) + d$ is appended with the keyword vertex u' at the end of $a'.\mathsf{mat}$, where u' can be obtained by the inverted index of $\mathsf{KPADS}(q)$ (For simplicity, we omit the details of this data structure in this paper). It is worth noting that, we maintain a priority queue with a fixed size k for $a'.\mathsf{mat}$ rather than a list in INCEVAL.

A.1 Proofs of lemmas

Lemma 6.2. *The distance between a vertex v and a keyword t derived from Eq. 5 has an approximation factor $(2c-1)$, where $c = \lceil \frac{\ln|V|}{\ln k} \rceil$ with a constant probability, and k is a parameter set by user as we introduced in Sec. 6.1.*

Proof. Given a vertex v and a keyword t, we denote the vertex which is the closest to v and contains t as u, i.e. for any vertex u' where $t \in L(u')$, $d(v, u') \geq d(v, u)$. And $\hat{d}(v, u)$ can be estimated with the same approximation factor, $(2c-1)$, by $\mathsf{PADS}(v)$ and $\mathsf{PADS}(u)$ with the same probability, $1 - e^{-\alpha}$, of Lemma 6.1. We denote the common center by w_i.

$$d(v, w_i) + d(w_i, u) \leq (2c-1)d(v, u) \tag{10}$$

By the definition of PADS-merging (we compress the common centers while keep smallest distance), we have $(w_i, d_i) \in \mathsf{KPADS}$, and $d_i \leq d(w_i, u)$.

$$d(v, w_i) + d_i \leq d(v, w_i) + d(w_i, u) \leq (2c-1)d(v, u) \tag{11}$$

□

We denote the answer of Blinks by $a = \langle r, \mathsf{mat}\rangle$, where r is a candidate answer root, mat is a map $\mathsf{mat}[q] = \langle v, d\rangle$ such that $q \in G.L(v)$, and d is the shortest distance between r and v where $q \in Q$. We have the following conclusion.

Lemma 4.2. *The following quality guarantees of the distances hold for $a = \langle r, \mathsf{mat}\rangle \in \mathsf{eval}(G \oplus G', Q, \mathsf{Blinks})$ and $a' = \langle r, \mathsf{mat}'\rangle$ returned by* P2PG*:*

- *if* $\mathsf{mat}[q].\mathsf{v} \in G'.V$, *then* $\mathsf{mat}'[q].\mathsf{v} = \mathsf{mat}[q].\mathsf{v}$ *and* $\mathsf{mat}'[q].\mathsf{d} = \mathsf{mat}[q].\mathsf{d}$; *and*
- *if* $\mathsf{mat}[q].\mathsf{v} \notin G'.V$, *then* $\mathsf{mat}'[q].\mathsf{d} \leq (2c-1)\mathsf{mat}[q].\mathsf{d}$.

Proof. For simplicity, $\mathsf{mat}[q].\mathsf{v}$ (resp. $\mathsf{mat}'[q].\mathsf{v}$) is denoted by v (resp. v').

Case 1: Suppose $r \in G'.V$.

- ***Case 1.1:*** If $v \in G'.V$, due to the definition of Blinks, $\mathsf{mat}[q].\mathsf{d} = d_c(r, q)$. Moreover, INCEVAL refines the distance between of r and q. Hence, $\mathsf{mat}'[q].\mathsf{d} = d_c(r, q)$. Hence, $\mathsf{mat}'[q].\mathsf{d} = \mathsf{mat}[q].\mathsf{d}$. Similarly, we have $v = v'$.
- ***Case 1.2:*** If $v \in G.V$, the shortest path between r and v is denoted by $\mathcal{P}_c(r, \ldots, v)$. Since $v \in G.V$ and $r \in G'.V$, $\mathcal{P}_c(r, \ldots, v)$ contains at least one portal node. We denote them by $\mathbb{P}^c = \{p_1^c, \ldots, p_n^c\}$. We denote the last portal node in $\mathcal{P}_c(r, \ldots, v)$ by p_{last}^c. It is worth noting that the shortest path between p_{last} and v is located on the public graph. Otherwise, there is at least portal node in $\mathcal{P}_c(p_{last}, \ldots, v)$, denoted by p_i.
  1) If $p_i \notin \mathbb{P}^c$, we have $\mathcal{P}_c(r, \ldots, v) = \mathcal{P}_c(r, \ldots, p_1, \ldots, p_{last}, \ldots, p_i, \ldots, r)$, then p_i is the last portal node rather than p_{last}.
  2) If $p_i \in \mathbb{P}^c$, we have $\mathcal{P}_c(r, \ldots, v) = \mathcal{P}_c(r, \ldots, p_1, \ldots, p_i, \ldots, p_{last}, \ldots, p_i, \ldots, r)$, then there is a cycle $\mathcal{P}_c(p_i, \ldots, p_{last}, \ldots, p_i)$ on $\mathcal{P}_c(r, \ldots, v)$ which is contradicted with that $\mathcal{P}_c(r, \ldots, v)$ is the shortest path between r and v.

  Hence, $d_c(p_{last}, v) = d(p_{last}, v)$. Then $d_c(r, v) = d_c(r, p_{last}) + d_c(p_{last}, v)$. Since $r, p_{last} \in G'.V$, $d_c(r, p_{last})$ is returned by INCEVAL. Obviously, since the shortest path of p_{last} and v are all in the public graph, $\hat{d}(p_{last}, v) \leq (2c-1)d(p_{last}, v)$ because of Lemma 6.1. As a consequence,

$$\begin{aligned}\mathsf{mat}'[q].\mathsf{d} &= d_c(r, p_{last}) + \hat{d}(p_{last}, v)\\ &\leq (2c-1)d_c(r, p_{last}) + (2c-1)d(p_{last}, v)\\ &= (2c-1)d_c(r, v)\\ &= (2c-1)\mathsf{mat}[q].\mathsf{d}.\end{aligned}$$

Case 2: Suppose $r \in G.V$.

- ***Case 2.1:*** If $v \in G'.V$, the shortest path between r and v is denoted by $\mathcal{P}_c(r, \ldots, v)$. We denote the first portal node in $\mathcal{P}_c(r, \ldots, v)$ as p_{first}. Then $d_c(r, v) = d_c(r, p_{first}) + d_c(p_{first}, v)$. Since $p_{first}, v \in G'.V$, $d_c(p_{first}, v)$ is returned by INCEVAL. $d_c(r, p_{first}) = d(r, p_{first})$ since the nodes on the shortest path of p_{last}

and v are all in the public graph (otherwise, p_{first} is not the first portal node in $\mathcal{P}_c(r,\ldots,v)$). $d(r,p_{first})$ can be computed by a breadth-first traversal that starts from p_{first} (denoted by $\mathcal{T}$). Consider the breadth-first traversal (denoted by $\mathcal{T}_i$) that starts from the portal node p_i, where, $p_i \neq p_{first}$, which visits r.

- If $p_i \in \mathcal{P}_c(r,\ldots,v)$, $d(r,p_i) \geq d_c(r,p_i)$. Since $p_{first}, p_i \in \mathcal{P}_c(r,\ldots,v)$, we have $d_c(r,p_i) = d_c(r,p_{first}) + d_c(p_{first},p_i)$. Hence, returned by $\mathcal{T}_i$, $d_c(r,v) = d(r,p_i) + d_c(p_i,v)$, which is larger than that returned by $\mathcal{T}$, since $d(r,p_i) + d_c(p_i,v) \geq d(r,p_{first}) + d_c(p_{first},p_i) + d_c(p_i,v)$.
- If $p_i \notin \mathcal{P}_c(r,\ldots,v)$, the subpath between r an p_i returned by $\mathcal{T}_i$, $\mathcal{P}_c(r,\ldots,p_i)$, is not a shorter path between r and v. Otherwise, it is contradicted with that $\mathcal{P}_c(r,\ldots,v)$ is the shortest path between r and v.

Similarly, we have $v = v'$.

- ***Case 2.2:*** If $v \in G.V$, the proof is similar to ***Case 1.2***.

□

Lemma 4.3. *If $u \in V'$ belongs to the answer of a* knk *query (v,q,k) on G_c, where $v \in V'$, then u is returned by* PP-knk.

Proof. We denote the set of vertices containing q as V_q. Given two vertex $u_1, u_2 \in V_q$. Without loss of generality, we assume that $u_1 \in V'$ and $d_c(v,u_1) \leq d_c(v,u_2)$. Then the ranking of u_1 is higher than u_2. It is worth noting that the exact value of $d_c(v,u_1)$ is returned by P2PG. Next, we prove that the ranking is hold in P2PG.

- If $u_2 \in V$, then $d_c(v,u_2) \leq \hat{d}(v,u_2)$ since $\hat{d}(v,u_2)$ is always larger than $d_c(v,u_2)$, returned by P2PG. Naturally, the ranking of u_1 is still higher than that of u_2 since $d_c(v,u_1) \leq d_c(v,u_2) \leq \hat{d}(v,u_2)$.
- If $u_2 \in V'$, since the exact value of $d_c(v,u_2)$ is also returned in the context of P2PG, $d_c(v,u_1) \leq d_c(v,u_2)$ is still hold. The ranking of u_1 is still higher than that of u_2.

Hence, $\forall u \in V'$ is an answer in G_c, u is returned by P2PG since its ranking is hold in the context of P2PG. □

A.2 Complexity analysis

In this section, we analyse the time complexity of estimating the shortest distance in public graph. Moreover, we also present the complexity of each key step of PP-knk, PP-r-clique and PP-knk as shown in Tab. 1.

A.2.1 Complexity of the shortest distance estimation.

Given two vertices v_1 and v_2 and they corresponding PADS, the time of shortest distance estimation is $\min\{|\mathsf{PADS}(v_1)|, |\mathsf{PADS}(v_2)|\}$ on average. Since finding a element in a hashset can be finished in a constant time on average, the estimation cost is $O(k\ln|V|)$, where k is a parameter set by user. A larger k will bring larger sketches as well as more accuracy. Similarly, the time of estimating the shortest distance between a keyword t and a vertex v is $O(\min\{|\mathsf{KPADS}(t)|, |\mathsf{PADS}(v)|\})$. In general, the size of $|\mathsf{KPADS}(t)|$ is much larger than $|\mathsf{PADS}(v)|$. Hence, the cost is $O(k\ln|V|)$, too.

A.3 The qualities of the query answers of P2PG

In this section, we show the quality guarantees of the query answers of various query semantics on P2PG. We show the theoretical bounds of PP-r-clique, PP-Blinks and PP-knk, respectively.

Theorem A.1. PP-r-clique *finds an* r-clique *with $(l-1)$-approximation, where l is the number of the query keywords, i.e., $l = |Q|$.*

Proof. We prove P2PG can return the $(l-1)$-approximate answer of r-clique, $\langle v, \mathsf{mat}\rangle$. We denote the optimal r-clique as a_{opt}, and the greedy r-clique as a_{grdy}. And we use $u_i = a_{opt}.\mathsf{mat}[q_i].\mathsf{u}$ (resp. $v_i = a_{grdy}.\mathsf{mat}[q_i].\mathsf{u}$) to dentoe the keyword nodes in a_{opt} (resp. a_{grdy}). Moreover, we use the symbols $d^{opt}_{i,j} = d_c(u_i,u_j)$ (resp. $d_{i,j} = d_c(v_i,v_j)$) to denote the shortest distance between keyword nodes u_i and u_j (resp. v_i and v_j). We denote the weight of optimal (resp. greedy) r-clique as $W(a_{opt}) = \sum_{i=1}^{l}\sum_{j=1}^{l} d^{opt}_{i,j}$ (resp. $W(a_{grdy}) = \sum_{i=1}^{l}\sum_{j=1}^{l} d_{i,j}$).

Based on the Definition 2.2 of the public-private answers, $\exists v_i$ such that $v_i \in G'.V$ and $\exists u_j$ such that $u_j \in G'.V$. We denote them as v_r and u_r, respectively.

Given any two keyword nodes v_i and v_j, the triangle inequality is kept. More specifically, we have the following formula:

$$d_{i,j} \leq d_{r,i} + d_{r,j} \tag{12}$$

Moreover, we have the weight of a_{grdy} as follows:

$$2 \times W(a_{grdy}) = \sum_{i=1}^{l}\sum_{j=1}^{l} d_{i,j} = 2 \times \sum_{i\neq r} d_{r,i} + \sum_{i\neq r}\sum_{j\neq r, j\neq i} d_{i,j}, \tag{13}$$

where $i,j \in (1,l)$.

Consider the worst case, we have:

$$\sum_{i\neq r}\sum_{j\neq r, j\neq i} d_{i,j} \leq \sum_{i\neq r}\sum_{j\neq r, j\neq i} (d_{r,i} + d_{r,j}) = 2\times(l-2)\sum_{i\neq r} d_{r,i} \tag{14}$$

Consider the Eq. 13 and Eq. 14, we have:

$$2 \times W(a_{grdy}) \leq 2 \times \sum_{i\neq r} d_{r,i} + 2\times(l-2)\sum_{i\neq r} d_{r,i} = 2\times(l-1)\sum_{i\neq r} d_{r,i} \tag{15}$$

Next, we consider the weight of the optimal r-clique ans_{opt}:

$$2\times W(a_{opt}) = \sum_{i=1}^{l}\sum_{j=1}^{l} d^{opt}_{i,j} \geq 2\times\sum_{i\neq r} d^{opt}_{r,i} \geq 2\times\sum_{i\neq r} d_{r,i} \tag{16}$$

Therefore, we have:

$$W(a_{opt}) \geq \sum_{i\neq r} d_{r,i} \tag{17}$$

In this case, a_{opt} and a_{grdy} are considered equal. Hence, Formula 17 is established.

Consider the Eq. 15 and Eq. 17, we have:

$$W(a_{grdy}) \leq (l-1)\sum_{i \neq r} d_{r,i} \leq (l-1) \times W(a_{opt}) \quad (18)$$

Consequently, the $(l-1)$ approximation ratio is satisfied. □

Theorem 4.1. *Given an answer of* PP-r-clique, $a = \langle v, \mathsf{mat} \rangle$, $a.\mathsf{mat}[q].\mathsf{d} \leq (2c-1)d_c(v, a.\mathsf{mat}[q].\mathsf{u})$.

Proof. The proof is the same with Lemma 4.2, ***Case 1.2***. □

Theorem A.2. PP-Blinks *finds* Blinks *answers with* $(2c-1)$*-approximation.*

Proof. The weight of a Blinks answer $a = \langle r, \mathsf{mat} \rangle$ is denoted by $W(a) = \sum \mathsf{mat}[q].\mathsf{d}$. We denote the answer returned by PP-Blinks as a' which is rooted at r. We denote the answer rooted at r and returned by applying Blinks on the combined graph as $a \in \mathsf{eval}(G \oplus G', Q, \mathsf{Blinks})$.

Next, we show that $W(a') \leq (2c-1)W(a)$. As we have proved in Lemma 4.2, $\mathsf{mat}'[q].\mathsf{d} \leq (2c-1)\mathsf{mat}[q].\mathsf{d}$. Hence $W(a') = \sum \mathsf{mat}'[q].\mathsf{d} \leq (2c-1)\sum \mathsf{mat}[q].\mathsf{d} = (2c-1)W(a)$. □

Given a knk query (v, q, k), we denote the top k answers returned by PP-knk as $A' = \{a'\}$. And $a'.\mathsf{mat}$ is sorted by the ascending order of $a'.\mathsf{mat}[i].\mathsf{d}$. And we denote the answers returned by applying knk on the combined graph as $A = \{a\} = \mathsf{eval}(G \oplus G', Q, \mathsf{knk})$. And $a.\mathsf{mat}$ is sorted by the ascending order of $a.\mathsf{mat}[i].\mathsf{d}$.

Theorem A.3. *The distance of* k*-th element of the answer* $a.\mathsf{mat}$ *returned by* PP-knk *is bounded with* $(2c-1)$*-approximation, i.e.,* $a'.\mathsf{mat}[k].\mathsf{d} \leq (2c-1)a.\mathsf{mat}[k].\mathsf{d}$.

Proof. We consider the following two cases.

Case 1: Suppose $\forall i, a.\mathsf{mat}[i].\mathsf{u} \in G'.V$, $a.\mathsf{mat}[i].\mathsf{u}$ will be returned by PP-knk, *i.e.*, $a.\mathsf{mat}[i] \in a'.\mathsf{mat}$, as we have proved in Lemma 4.3. Hence, $a.\mathsf{mat} \subseteq a'.\mathsf{mat}$. Since $|a'.\mathsf{mat}| = |a.\mathsf{mat}| = k$, $a'.\mathsf{mat} = a.\mathsf{mat}$. Hence $a'.\mathsf{mat}[k] = a.\mathsf{mat}[k]$. $a'.\mathsf{mat}[k].\mathsf{d} \leq (2c-1)a.\mathsf{mat}[k].\mathsf{d}$ is satisfied.

Case 2: Suppose $\exists i, a.\mathsf{mat}[i].\mathsf{u} \notin G'.V$. It is worthing noting that $a.\mathsf{mat}[i].\mathsf{d} \leq (2c-1)d_c(v, a.\mathsf{mat}[i].\mathsf{u})$ by PP-knk (the proof is the same with Lemma 4.2, ***Case 1.2***).

Since $a.\mathsf{mat}[i].\mathsf{d} \leq a.\mathsf{mat}[k].\mathsf{d}$, we have

$$(2c-1)a.\mathsf{mat}[i].\mathsf{d} \leq (2c-1)a.\mathsf{mat}[k].\mathsf{d} \quad (19)$$

We prove this theorem by contradiction. If $a'.\mathsf{mat}[k].\mathsf{d} > (2c-1)a.\mathsf{mat}[k].\mathsf{d}$, then

$$a'.\mathsf{mat}[k].\mathsf{u}'.\mathsf{d} > (2c-1)a.\mathsf{mat}[i].\mathsf{d} \quad (20)$$

where $i \in (1, k)$.

Hence $a'.\mathsf{mat}[k]$ is not among the top-k nearest vertices returned by PP-knk. Therefore $a'.\mathsf{mat}[k].\mathsf{d} \leq (2c-1)a.\mathsf{mat}[k].\mathsf{d}$ is established. □