Diversified Subgraph Query Generation with Group Fairness

Hanchao Ma, Sheng Guan, Christopher Toomey, Yinghui Wu
Case Western Reserve University
Cleveland, USA
{hxm382,sxg967,ctt16,yxw1650}@case.edu

ABSTRACT

This paper investigates the problem of subgraph query generation with output that satisfies both diversity and fairness constraints. Given a set of groups with associated cardinality requirements, it is to compute subgraph queries with diversified output that meanwhile covers the groups with the desired cardinality. Such need is evident in web and social search with fairness constraints. We formalize subgraph query generation as a bi-criteria optimization problem on the diversity and fairness properties of queries, and verify its hardness and approximability. We show that the problem is in Σ_2^P, and remains NP-complete even for single-node queries. Despite the hardness, (1) we show that approximations exist whenever a corresponding subset selection process provides good solutions, and provide feasible algorithms with performance guarantees for two practical query generation scenarios. We also present a fast heuristic algorithm for the general problem, which early terminates without enumerating queries. We experimentally verify that our algorithms can efficiently generate queries with desired diversity and coverage properties for targeted groups.

CCS CONCEPTS

• **Information systems** → **Query suggestion**; **Query reformulation**; *Information retrieval diversity*.

KEYWORDS

Attributed graph, Query suggestion, Fairness

ACM Reference Format:
Hanchao Ma, Sheng Guan, Christopher Toomey, Yinghui Wu. 2022. Diversified Subgraph Query Generation with Group Fairness. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3488560.3498525

1 INTRODUCTION

Subgraph queries have been routinely used to retrieve entities from real-world graphs (*e.g.,* social networks [24], knowledge graphs [33]). Several algorithms [7] have been developed that, given a subgraph query $Q(u_o)$ with a designated output node u_o and a graph G, compute a set of nodes (matches) of u_o in G in terms of subgraph isomorphism or its approximate variants [33].

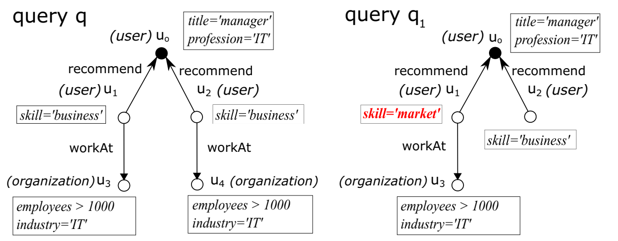

Figure 1: Query suggestion with Fairness: Talent search.

The emerging need for Web search that requires both result diversity and fairness [2, 5, 12, 26] poses new challenges to graph search. In such scenarios, queries are expected to return diversified matches that meanwhile ensure a suitable coverage of designated groups (node sets) of interests from G. Such groups may refer to the population of vulnerable individuals in terms of sensitive attributes (*e.g.,* gender, race, professions) in social networks [15, 16], relevant articles yet with diversified labels for Web exploration [1], or designated columns for query benchmark [4].

Example 1: Consider talent search over a collaboration network G [16], where each node in G denotes a user with attributes such as *title*, *skill*, *profession*, or an organization with attributes such as the number of *employees*. Each edge indicates affiliation (workAt) of a user or recommendation (recommend) between users. A recruiter issues a graph search query q (illustrated in Fig. 1) to find managers u_o who have expertise in managing IT business, and moreover, recommended by two IT managers from large companies. In our test (Section 6), this query returns a set of qualified candidates $q(G)$ with a skewed distribution of 375 male users and 173 female users.

The recruiter pursues the desired gender distribution and diversity of the candidates, and wonders how to query G such that (1) the new answer can equally cover the male and female candidates from $q(G)$, both with 200 candidates ("Equal opportunity" [16]); and (2) the candidates are also more diversified in their majors. A query q_1 with a more desirable answer can be suggested, which contains 202 male and 198 female candidates that spans 30 different majors. The difference between q and q_1 indicates that a relaxed condition on recommendation community (removing the edge from u_2 to u_4) and changed skills (from "business" to "market") help to achieve the desired answer with proper coverage of the gender groups. □

The above examples highlight the need to suggest subgraph queries with diversified matches that can also cover designated groups with desired cardinality. We study a new problem called *subgraph query generation with group fairness* (SQG):

- **Input:** graph G, an initial query $Q(u_o)$, a set of groups $\mathcal{P}$, where each group $P_i \in \mathcal{P}$ is associated with an integer c_i (a cardinality constraint);

- **Output:** a set of subgraph queries $\mathcal{Q}$ obtained by revising $Q(u_o)$, which can retrieve a set of diversified matches ("*Diversity*") from G that also cover each group P_i with desired cardinality c_i ("*Group fairness*").

Such need is evident in social search [16], query benchmark [4], and query optimization [21]. Existing query generation approaches [3, 19, 22, 25, 27, 30] revise queries towards specific properties rather than ensuring both group coverage and answer diversity, thus cannot be directly applied to our problem.

Contributions & organization. This paper formally analyzes the subgraph query generation problem with group fairness constraints. We propose both feasible a approximation scheme as well as practical exact algorithms for large graphs. We refer to subgraph query simply as "query" in the rest of the paper.

A formal analysis. We formalize SQG (Section 3) as a bi-criteria optimization problem defined on a *parameterized query*. A parameterized query $Q(u_o)$ carries variables defined on search predicates and edges yet to be instantiated. Given a parameterized query Q, and targeted groups $\mathcal{P}$ with cardinality constraints from graph G, SQG is to instantiate $Q(u_o)$ with proper search predicates and edge constraints to a set of queries $\mathcal{Q}$, to approach an output (the union of the matches of queries in $\mathcal{Q}$) with maximized diversity and minimized distance to the desired cardinality requirements. We show that SQG is in Σ_2^P and remains to be NP-complete for queries with a single-node pattern. This verifies the hardness of SQG even when the queries can be evaluated in PTIME.

Query generation with performance guarantees (Section 4). Despite the hardness, we present a general approximation scheme for SQG. In a nutshell, our scheme first computes a representative *pivot set* from the groups with exact coverage and maximized diversity. Treating the pivot set as a desired "answer", we then generate $\mathcal{Q}$ with answers that approach the pivot set.

We investigate two specifications of SQG. The first addresses "equal opportunity", by enforcing the coverage of an equal number of nodes from each group. The second specifies cardinality constraint on the query output. For both cases, we develop feasible approximation algorithms with a factor that are only determined by the number of groups $|\mathcal{P}|$ and the size of promising queries.

Fast heuristic with early termination (Section 5). We follow up the analysis by introducing a fast heuristic algorithm. It dynamically relaxes the instances that best improve the coverage at runtime, and early terminates without enumerating all queries.

Using real-life graphs, we verify the effectiveness and efficiency of our algorithms (Section 6). Our algorithms can generate subgraph queries with both desired diversity and small errors in covering designated groups. These algorithms are also feasible. For example, it takes up to 370 seconds to produce instances with desired coverage in real-life graphs with $4.9M$ nodes and $45.6M$ edges.

Related Work. We categorize the related work as follows.

Graph query suggestion. Several methods have been studied to suggest subgraph queries towards answers with various desired properties. Graph query by example [19] induces subgraph queries from subgraphs that contain similar nodes to specified examples. Diversified query suggestion [25] expands an initial query with

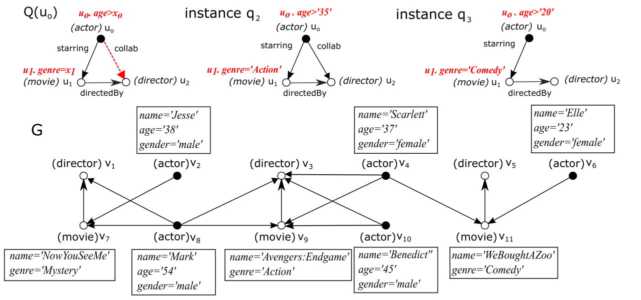

Figure 2: Parameterized Query and Query Instances

new edges that can lead to relevant and diversified matches. Why-questions [27, 30] suggest queries with both relaxation and refinement operators towards exact or similar matches. These methods either cope with diversity or similarity alone or enforce size constraints on the entire match set. In contrast, we study subgraph query generation with both diversity and fairness constraints.

Set selection. Subset selection with diversity and fairness constraints has been studied [26, 31]. Given a universal set and a set of groups (subsets), it computes a diverse subset that can cover each group with individual cardinality constraints. Approximation algorithms have been studied to generate subsets for max-sum and max-min diversification [26]. Although these methods cannot be directly used to suggest queries, we verify that these approximability results provide useful intermediate results that can be leveraged to guide the query generation with bounded errors on the cardinality constraints. Our formal analysis verifies the range of approximation ratios one can expect for the query generation problem, determined by the approximation factor of "yardstick" subset selection problem.

Query generation. Query generation with output cardinality constraints and distribution properties have been investigated for graphs. [3] generates regular path queries from given graph schema that can output node set with different cardinalities over certain attributes. [21] exploits query rewriting to generate SPARQL queries that can cover the answer of given queries with cheaper plans for multi-query optimization. In contrast to these work, (1) we study query generation with fairness constraints on general groups; (2) We consider queries defined in terms of subgraph isomorphism rather than regular path queries; and (3) we do not assume predefined graph schema. Our algorithms can be readily applied to generate subgraph queries for benchmarking graph databases.

2 PRELIMINARIES

Graphs. We consider directed graphs $G = (V, E, L, T)$, where (1) V is a finite set of nodes, (2) $E \subseteq V \times V$ is a set of edges, (3) each node $v \in V$ (resp. edge $e \in E$) carries a label $L(v)$ (resp. $L(e)$); and (4) each node v carries a tuple $T(v) = < (A_1, a_1),\ldots,(A_n, a_n) >$, where each A_i ($i \in [1, n]$) is a distinct node attribute with a value a_i.

We denote the finite set of all the node attributes in G as $\mathcal{A}$. The *active domain* adom(A) of an attribute $A \in \mathcal{A}$ refers to the set of values of $v.A$ as the node v ranges over V.

We next introduce a notion of parameterized queries. A parameterized query allows "placeholders" in search predicates which can be bound to specific values when executed [9]. We extend this notion for graph search to characterize query generation problem.

Paramterized Queries. A *parameterized query* $Q(u_o)$ is a graph (V_Q, E_Q, L_Q, T_Q). (1) V_Q (resp. $E_Q \subseteq V_Q \times V_Q$) is a set of query nodes (resp. query edges). Specifically, $u_o \in V_Q$ is a designated *output node.* (2) Each query node $u \in V_Q$ (resp. query edge $e \in V_E$) has a label $L_Q(u)$ (resp. $L_Q(e)$). (3) For each node $u \in V_Q$, $T_Q(u)$ is a set of literals. A literal l is in the form of $u.A$ op x_l, where op is from $\{>, >=, =, <=, <\}$, and x_l is a *range variable* that can be assigned to a constant. For each edge $e \in E_Q$, $T_Q(e)$ is a binary *edge variable* x_e that can be either "0" or "1". The set of all the variables in $Q(u_o)$ is denoted as $\bar{X} = \bar{X}_L \cup \bar{X}_E$, the union of the range and edge variables.

Query instances. Given a parameterized query $Q(u_o)$, an *instantiation* of $Q(u_o)$ is a function I that maps a set of variables in $\bar{X} = \bar{X}_L \cup \bar{X}_E$ to constants. A *query instance* $q(u_o)$ of $Q(u_o)$ induced by an instantiation I is a graph (V_Q, E_q, L_Q, T_q) with the same V_Q, output node u_o and L_Q, and moreover,

- for each node $u \in V_Q$ and each literal $l \in T_Q(u)$ in $Q(u_o)$, there is a literal $l = u.A$ op $I(x_l)$ in $T_q(u)$; and
- there is an edge $e \in E_Q$ if and only if $I(x_e)$ = '1'.

In other words, a query instance (or simply "instance") q has no variables but literals and the edges induced by the value binding I. We denote the set of all the possible instances of $Q(u_o)$ as $\mathcal{I}(Q)$.

Matching. Given an instance $q(u_o)$ and a graph G, a matching from $q(u_o)$ to G is a function $h \subseteq V_q \times V$, where (1) for each node $u \in V_Q$, $L_Q(u) = L(h(u))$, and for each literal $u.A$ op c in L_q, $h(u).A$ op c; (2) for each edge $e = (u, u')$ in $q(u_o)$, $h(e) = (h(u), h(u'))$ is an edge in G, and $L_Q(e) = L(h(e))$.

The *matches* of a query node u of $q(u_o)$ in G, denoted as $q(u, G)$, refers to the set of all the nodes in G that can match node u via a matching $h(u)$ from q to G. The *result* of q in G, denoted as $q(G)$, refers to the match set $q(u_o, G)$. Given a set of instances $\mathcal{Q} \subseteq \mathcal{I}(Q)$, the result of $\mathcal{Q}$ in G, denoted as $\mathcal{Q}(G)$, is defined as $\bigcup_{q \in \mathcal{Q}} q(G)$.

Example 2: A parameterized query $Q(u_o)$ that searches for actors in a knowledge graph G [23] is illustrated in Fig. 2. (1) $Q(u_o)$ has three variables $\bar{X} = \{x_o, x_1, x_{(u_o,u_2)}\}$, with two range variables in literals u_o.age $> x_o$ and u_1.genre $= x_1$, respectively, and an edge variable $x_{(u_o,u_2)}$. (2) A corresponding instantiation $\{35, \text{'}\mathit{Action}\text{'}, \text{'}1\text{'}\}$ (resp. $\{20, \text{'}\mathit{Comedy}\text{'}, \text{'}0\text{'}\}$) of $\bar{X}$ induces an instance q_2 (resp. q_3) of $Q(u_o)$. (3) Given G, $q_2(G) = \{v_4, v_8, v_{10}\}$, and $q_3(G) = \{v_4, v_6\}$. Let $\mathcal{Q} = \{q_2, q_3\}$, then $\mathcal{Q}(G) = \{v_4, v_6, v_8, v_{10}\}$. □

Remarks. The query instances are well-defined for a "partial" instantiation I in which not all the variables of $Q(u_o)$ are assigned a constant. For such cases, q is induced by removing any remaining variables in $Q(u_o)$ to ensure valid $q(G)$. Note that a given "initial" query (*e.g.,* q_1 in Example 1) can be readily captured by a parameterized query with a partial instantiation.

3 QUERY GENERATION PROBLEM

Given a parameterized query $Q(u_o)$, graph G and m disjoint groups $\mathcal{P}$, where each group $P_i \in \mathcal{P}$ has a cardinality constraint $c_i \in [1, |P_i|]$, the query generation problem aims to compute a set of query instances $\mathcal{Q} \subseteq \mathcal{I}(Q)$ of Q with maximized diversity and required coverage properties.

To quantify the quality of query instances, we consider two classes of functions. For simplicity, (1) we define a constant $C = \sum_1^{|\mathcal{P}|} c_i$, and (2) we use $\bigcup(S)$ to denote $\bigcup_{s \in S} s$ for a set S.

Max-sum Diversification. We consider Max-sum diversification as a natural objective for result diversification. The function measures the weighted sum of a relevance measure and pairwise dissimilarity of the matches [17], and is defined as follows:

$$f(\mathcal{Q}(G)) = (1-\lambda) \sum_{v \in \mathcal{Q}(G)} r(u_o, v) + \frac{2\lambda}{|V_{u_o}| - 1} \sum_{v, v' \in \mathcal{Q}(G)} d(v, v')$$

where (1) $\lambda \in [0, 1]$ is a constant to balance relevance and diversity; (2) the function $r(u_o, v) \in [0, 1]$ (resp. $d(v, v') \in [0, 1]$) computes a relevance score between u_o and a match v (resp. difference between two matches v and v'). In practice, $d(v, v')$ can be the edit distance between tuples $T(v)$ and $T(v')$, and $r(u_o, v)$ can be an entity linkage score or impact of v in social networks [14], among others [20].

Here V_{u_o} is the set $\{v | L(v) = L(u_o), v \in V\}$, *i.e.,* the nodes in G with the same label of u_o. Given G, the pairwise dissimilarity is normalized with a constant $\frac{|V_{u_o}|-1}{2}$, as there are at most $\frac{|V_{u_o}|(|V_{u_o}|-1)}{2}$ pairs but $|V_{u_o}|$ relevance numbers. That is, $f(\mathcal{Q}(G)) \in [0, |V_{u_o}|]$.

Relative Coverage Error. To characterize the coverage property of $\mathcal{Q}(G)$ over groups $\mathcal{P}$, we introduce a relative error function, that accumulates normalized errors of the coverage cardinality constraints, which is defined as follows:

$$\delta(\mathcal{Q}(G), \mathcal{P}) = \frac{\sum_{P_i \in \mathcal{P}} (|\mathcal{Q}(G) \cap P_i| - c_i)}{|\bigcup(\mathcal{P})|}$$

The function $\delta(\mathcal{Q}, \mathcal{P})$ penalizes the differences between each cardinality c_i posed on $P_i \in \mathcal{P}$ and the size of the fraction of P_i covered by $\mathcal{Q}(G)$. The smaller, the better.

We next state the problem of *diversified query generation with group fairness*, denoted as SQG. We simplify $f(\mathcal{Q}(G))$ (resp. $\delta(\mathcal{Q}, (G)\mathcal{P})$) as $f(\mathcal{Q})$ (resp. $\delta(\mathcal{Q}, \mathcal{P})$) when G is given.

Problem statement. Given G, $Q(u_o)$, and a set of disjoint groups $\mathcal{P}$ with cardinality constraints, the SQG problem is to compute a set of query instances $\mathcal{Q}$ of $Q(u_o)$, such that (1) $|\mathcal{Q}(G) \cap P_i| \geq c_i$ ($P_i \in \mathcal{P}$), and moreover (2):

$$f(\mathcal{Q}) = \underset{\mathcal{Q}' \subseteq \mathcal{I}(Q)}{\arg\max} f(\mathcal{Q}'); \quad \delta(\mathcal{Q}, \mathcal{P}) = \underset{\mathcal{Q}' \subseteq \mathcal{I}(Q)}{\arg\min} \delta(\mathcal{Q}', \mathcal{P})$$

The bi-objective optimization problem SQG aims to discover a set of instances $\mathcal{Q}$ with most diversified answers that can meanwhile properly cover each group with a minimized relative coverage error. Note that $f(\mathcal{Q}) \in [0, |V_{u_o}|]$, and $\delta(\mathcal{Q}, \mathcal{P}) \in [0, 1]$.

Although desirable, SQG remains intractable even for cases when subgraph isomorphism is in PTIME, as verified below.

Theorem 1: *The* SQG *problem is (a) in* Σ_P^2 *and (b)* NP*-complete even when* $Q(u_o)$ *contains a single node with only range variables.* □

Proof sketch: The decision problem of SQG is to decide whether there exists a set of instances $\mathcal{Q}$ of $Q(u_o)$ and two thresholds f_b and δ_b, such that $f(\mathcal{Q}) \geq f_b$ and $\delta(\mathcal{Q}) \leq \delta_b$. For (a), SQG is solvable in Σ_P^2. Here Σ_P^2 is the class of problems in $\mathsf{NP}^{\mathsf{NP}}$. Since the verification of subgraph isomorphism is in NP, a Σ_P^2 algorithm first guesses a set of instances $\mathcal{Q}$. For each instance, it consults an NP oracle to compute the coverage over $\mathcal{P}$ and verifies $f(\mathcal{Q})$ and $\delta(\mathcal{Q})$.

We next prove (b). When $Q(u_o)$ contains a single node u_o and range variables, an NP algorithm first guesses a set of instances and verifies the answers in PTIME. To see the hardness, we construct a PTIME reduction from the subset sum problem (known to be NP-complete). Given a bag of integers S, subset sum is to decide whether there is a non-empty subset $S' \subseteq S$ that is summed to be a target value c. Given $S = \{s_1, \ldots, s_n\}$ and c, we construct an instance of SQG with a single-node parameterized query $Q(u_o)$ as follows. (1) For each integer $s_i \in S$, there are $|s_i|$ nodes v_{ij} in G ($i \in [1, n], j \in [1, s_i]$) in V and $E = \emptyset$. Each node v_{ij} has a label l, and a tuple $T(v)$ defined on $|S|$ Boolean attributes A_k ($k \in [1, n]$), where $v_{ij}.A_i = 1$, and $v_{ij}.A_k = 0$ for $k \neq i$. (2) $Q(u_o)$ has a single output node u_o with label l, and n literals $u_o.A_k = x_k$, with n range variables $\bar{X}_R$ $\{x_1, \ldots, x_n\}$. (3) $\mathcal{P}$ contains a single group V with cardinality constraint c. We set threshold $f_b = 0$ and $\delta_b = 0$. We can verify that there is a set of instances $\mathcal{Q}$ with results that precisely cover c nodes in V ($f(\mathcal{Q}) \geq b$=1) if and only if there is a subset S' with integers that are summed to be c. This completes the proof. □

4 APPROXIMATING QUERY GENERATION

An "optimal" solution of SQG $\mathcal{Q}^*$ should ensure

$$f(\mathcal{Q}^*) = \underset{\delta(\mathcal{Q},\mathcal{P})=0}{\arg\max}\, f(\mathcal{Q})$$

That is, among all instances that cover $\mathcal{P}$ with exactly the required cardinality, $\mathcal{Q}^*$ has the maximized diversity (*i.e.,* a Pareto optimal solution). While desirable, such an "ideal" solution may not always exist as diversity and fairness can be in conflict. On the other hand, it is still desirable to use $f(\mathcal{Q}*)$ and $\delta(\mathcal{Q}*, \mathcal{P})$ as "reference points" to measure the quality of solutions [13, 18].

(α, β)-approximation. Following the reference approach of bi-objective optimization [13, 18], we characterize a desirable quality guarantee. A solution $\mathcal{Q}_r$ is a *reference solution* if (1) $f(\mathcal{Q}_r) \geq \alpha \cdot f(\mathcal{Q}^*)$ ($\alpha \in [0, 1]$), and (2) $\delta(\mathcal{Q}_r, \mathcal{P}) = \arg\min_{f(\mathcal{Q}) \geq \alpha \cdot f(\mathcal{Q}^*)} \delta(\mathcal{Q}, \mathcal{P})$. That is, $\mathcal{Q}_r$ ensures a bounded gap between diversity and an ideal counterpart with the smallest coverage error.

We say an algorithm is an (α, β)-*approximation* for SQG ($\alpha \in [0, 1]$, $\beta > 1$), if for *any* non-empty solutions $\mathcal{Q}$ it generates, (1) $f(\mathcal{Q}) \geq \alpha \cdot f(\mathcal{Q}^*)$, and (2) $\delta(\mathcal{Q}) \leq \beta \cdot \delta(\mathcal{Q}_r)$. In other words, it generates $\mathcal{Q}$ with guaranteed diversity as any reference solutions, and ensures a bounded gap in the minimum coverage error they can achieve. This guarantee is weaker yet remains desirable, especially when a single Pareto optimal solution $\mathcal{Q}^*$ may not exist [13, 18].

We next show that there exists feasible (α, β)-approximations for SQG. We introduce a general strategy (Section 4.1), and investigate two practical approximable cases (Section 4.2).

4.1 A General Approximation Scheme

We present a general approximation scheme, denoted as APXQGen. In a nutshell, APXQGen exploits a *"pivot-and-cover"* strategy to compute a set of instances $\mathcal{Q}$ as follows.

(1) Pivoting. The pivoting step finds an optimal set $S^* \subseteq V_{u_o}$ of size C, such that S^* covers each $P_i \in \mathcal{P}$ exactly with c_i nodes, and has the maximized diversity. We refer to the output of pivoting step, denoted as S_p, as a *pivot set* (ideally, $S_p = S^*$).

(2) Covering. The covering step aims to compute $\mathcal{Q}$ that ensures (a) $S_p \subseteq \mathcal{Q}(G)$, and (b) $|\mathcal{Q}(G) \cap (V_{u_o} \setminus S_p)|$ is minimized.

```
Algorithm APXQGen
Input: graph $G$, parameterized query $Q(u_o)$,
       groups $\mathcal{P}$ with cardinality constraints.
Output: a set of instances $\mathcal{Q}$.
1.  set $S_p := \emptyset$; set $\mathcal{Q} := \emptyset$; queue $\mathcal{S}_Q := \emptyset$;
2.  set $V_{u_o} := \{v | L(v) = L_Q(u_o); v \in V\}$;
3.  $S_p :=$ Pivot($\mathcal{P}$); $S_n := V_{u_o} \setminus S_p$;
4.  $\mathcal{S}_Q :=$ MatchGen($G, Q, S_p, S_n$);
5.  $\mathcal{Q} :=$ Cover($S_p, S_n, \mathcal{S}_Q$);
6.  return $\mathcal{Q}$;
```

Figure 3: Algorithm APXQGen

Intuitively, the pivoting step computes a most diversified subset S_p of size C, which ensures the exactly required coverage of $\mathcal{P}$ as a hard constraint (thus ensures $\delta(S_p, \mathcal{P}) = 0$). The pivot set S_p hence serves as a desired diversified answer to be approached. The covering step then aims to "recovers" a reference solution $\mathcal{Q}$ that approaches S_p. By ensuring the containment of S_p as hard constraint, $\mathcal{Q}$ preserves the diversity (as $f(\cdot)$ is non-decreasing) and ideally with a bounded coverage error.

Algorithm. The algorithm APXQGen, outlined in Fig. 3, maintains a queue $\mathcal{S}_Q$ to store a set of query instances to be verified (line 1). (1) In the pivoting step, it invokes a procedure Pivot to compute a pivot set S_p (line 3). (2) For the covering step, APXQGen first initializes a set $S_n = V_{u_o} \setminus S_P$ to track the nodes that are not to be covered by the desired instances $\mathcal{Q}$ (line 3). It then invokes a procedure MatchGen (line 4) to generate a set of instances $\mathcal{S}_Q$, where each individual instance may have matches to cover a fraction of S_p (to be discussed). It then invokes procedure Cover (line 5) to further refine $\mathcal{S}_Q$ to $\mathcal{Q}$ towards a complete coverage of S_p with minimized nodes from S_n.

We next introduce procedures Pivot, MatchGen and Cover.

Procedure Pivot. The procedure Pivot is "plug-able" to specify APXQGen to (α, β)-approximations. Given $\mathcal{P}$, it returns a pivot set S_p of size C with maximized diversity, and covers each group $P_i \in \mathcal{P}$ with exactly c_i nodes. To this end, it solves a class of *diversified subset selection* problems, with variants depending on the specific cardinality constraints [1, 6, 26]. This partly ensures the general performance guarantees of APXQGen as long as Pivot approximates optimal S_p (see "Analysis"). We defer the discussion by providing two such approximations in Section 4.2.

Example 3: Recall G and $Q(u_o)$ in Fig. 2, and consider the gender groups $\mathcal{P}$ with $P_1 = \{v_4, v_6\}$ (female actors) and $P_2 = \{v_2, v_8, v_{10}\}$ (male actors). Given a function $f(\cdot)$ that aims to diversify the age differences and a coverage measure that enforces $c_1 = c_2 = 2$ ("equal opportunity"), a proper pivot set S_p is $\{v_4, v_6, v_8, v_{10}\}$, which satisfies $\delta(S_p, \mathcal{P}) = 0$ with maximized age diversity. □

Procedure MatchGen. Given pivot set S_p, S_n, $Q(u_o)$ and G, the procedure MatchGen (not shown) generates a small set of "promising" instances that may likely to output S_p. It maintains a priority queue. $\mathcal{S}_Q$, where each entry of $\mathcal{S}_Q$ records an instance q, match set $q(G)$ (initialized as $\emptyset$), and an integer $q.r$ to track $|q(G) \cap S_n|$). The goal is to retain all instances of Q_{u_o} that contribute to cover S_p (which in turn properly covers $\mathcal{P}$) and avoids the generation of

```
Procedure Cover
Input: pivot set $S_p$, set $S_n$, queue $\mathcal{S}_Q$.
Output:a set of instances $\mathcal{Q}$.
1.  set $\mathcal{Q} := \emptyset$; set $S^* := \emptyset$; integer $B$:= $\max(q.r | q \in \mathcal{S}_Q.\mathcal{Q})$;
2.  for integer $i = 1$ to $B$ do
3.      $\overline{S_i} := \{\overline{q(G)} | q \in \mathcal{S}_Q.\mathcal{Q}, q.r \leq i\}$;
4.      if $S_p \not\subseteq \bigcup(\overline{S_i})$ then continue ;
5.      set $S_n^H := \{v \in S_n | v$ occurs in at least $\sqrt{|\mathcal{S}_Q|/\log C}$ sets in $\overline{S_i}\}$;
6.      set $S_i :=$ Refine$(\overline{S_i})$; update $\mathcal{S}_Q.\mathcal{S}$;
7.      $S_i^c :=$ GreedySC$(S_i, S_p, S_n^H)$;
8.      restore $S_i^c$ to original match sets in $\mathcal{S}_Q.\mathcal{S}$;
9.      if $\delta(S^*, \mathcal{P}) > \delta(\bigcup(S_i^c), \mathcal{P})$ then $S^* := S_i^c$;
10. Induce $\mathcal{Q}$ from $S^*$; return $S^*$;
```

Figure 4: Procedure Cover

an excessive number of instances. To this end, it exploits the data locality of subgraph isomorphism without expensive verification.

Constraining Q_{u_o}. MatchGen first rewrites $Q(u_o)$ and its literals by "narrowing down" the values the variables may take towards targeted answer S_p. It induces a set of subgraphs $N(S_p)$ of G, where each subgraph G_v contains a node $v \in S_p$ and its n-hop neighbors in G (n is the diameter of $Q(u_o)$).

(1) For each range variable x_l in the literal $u.A$ op x_l, MatchGen constrains a set of possible values of x_l as $\{T(v).A\}$, where v ranges over the nodes in $N(S_p)$ with $L(v) = L(u)$. If x_l has no proper value, it removes the literal $u.A$ op x_l from $Q(u_o)$.

(2) For each edge variable x_e defined on an edge $e = (u, u')$, it "presets" $x_e = 0$, if there exists no subgraph of $N(S_p)$ that contain a path from $v_o \in V_p$ with an edge (v, v') such that $L_Q(e) = L((v, v'))$. If $x_e = 0$ and e is a bridge of $Q(u_o)$, *i.e.,* removing e leads to two connected components in $Q(u_o)$, MatchGen removes e and the entire connected component that does not contain u_o.

Instance generation. Given the constrained Q_{u_o} (denoted as $Q'_(u_o)$), MatchGen then instantiates $Q'_(u_o)$ with constrained values. For each instance q, MatchGen *overestimates* $q(G)$ as $\overline{q(G)} = \bigcap q_i(G)$, where each q_i is a path query from u_o to a distinct node in q. If $\overline{q(G)} \cap S_p \neq \emptyset$, it inserts q, $\overline{q(G)}$ and $q.r = |\overline{q(G)} \cap S_n|$ into $\mathcal{S}_Q$.

It suffices to instantiate the constrained Q'_{u_o}, as verified below.

Lemma 2: *For any instance $q \in \mathcal{I}(Q_{u_o})$ where $q(G) \cap S_p \neq \emptyset$,* MatchGen *guarantees that $q \in \mathcal{S}_Q \subseteq \mathcal{I}(Q'_{u_o})$.* □

Procedure Cover. Given $\mathcal{S}_Q$ generated by MatchGen, we refer to the set of instances (resp. corresponding match sets) as $\mathcal{S}_Q.\mathcal{Q}$ (resp. $\mathcal{S}_Q.\mathcal{S}$). The procedure (given in Fig. 4) solves a *red-blue set cover* problem [8, 28]. Given a collection of sets with "blue" and "red" elements, the red-blue set cover problem computes a sub-collection of sets that covers all "blue" elements and the minimum number of "red" elements. We map S_p to "blue" elements, S_n to "red" elements, and the match sets $\mathcal{S}_Q.\mathcal{S}$ to the sets.

Procedure Cover follows a greedy approximation [28]. It optimizes the process with a "late verification" strategy, which exploits the overestimated match sets and defers unnecessary verification to reduce cost. It conducts B rounds of computation, where B is the maximum number of nodes in S_n contained in a single match set in $\mathcal{S}_Q.\mathcal{S}$. At the i-th round, it performs the following (lines 3-9).

(1) It first verifies if $\overline{S_i} \subseteq \mathcal{S}_Q.\mathcal{S}$ can cover S_p (lines 3-4). Here $\overline{S_i}$ is induced by the overestimated match sets $\overline{q(G)} \in \mathcal{S}_Q.\mathcal{S}$, where $q.r = |\overline{q(G)} \cap S_n| \leq i$, *i.e.,* each $\overline{q(G)}$ contains no more than i nodes from S_n. It early terminates if no cover can be found for i (line 4).

(2) Cover then refines $\overline{S_i}$ to a minimum weighted set cover S_i^c of S_p (lines 5-6). Intuitively, it favors the match sets that cover S_p and also repeatedly cover the same set of nodes in S_n. (controlled by a threshold). The weighted set cover leads to a bounded coverage error for S_p, as the repeatedly covered nodes are counted once in the induced answer $\mathcal{Q}(G)$ (see "Analysis").

(a) It updates each $\overline{q(G)} \in \overline{S_i}$ to $q(G)$ by verifying if each node $v \in \overline{q(G)}$ remains a match of q, by invoking established subgraph isomorphism algorithm (*e.g.,* VF2 [10]) over q and a subgraph $N_n(v)$ induced by the n-hop neighbors of v (n is the diameter of q). This refines $\overline{S_r}$ to S_i. $\mathcal{S}_Q.\mathcal{S}$ is updated accordingly (line 6).

(b) Cover then constructs an instance of weighted set cover. It "trims" $q(G)$ to $q(G) \setminus S_n$ and assigns a weight $|q(G) \cap S_n^H|$), where S_n^H refers to a set of nodes in S_n such that each node occurs in more than $\sqrt{|\mathcal{S}_Q|/logC}$ match sets in S_i. It invokes a greedy approximation GreedySC [32] to compute the weighted set cover $S_i^c \subseteq S_i$ of S_p (line 7). It then "restores" S_i^c to be the set of original match sets $q(G)$ by augmenting their trimmed counterparts in S_c, and computes $\delta(\bigcup_{q(G) \in S_i^c} q(G), \mathcal{P})$ (lines 8-9).

(3) Cover keeps track of the best S_i^c as S^* with the smallest $\delta(\bigcup_{q(G) \in S_i^c} q(G), \mathcal{P})$ (line 9). It finally sets $\mathcal{Q}$ as the instances in $\mathcal{S}_Q.\mathcal{Q}$ whose match set is in S^*, and returns $\mathcal{Q}$ (line 10).

Example 4: We continue with Example 3. Given the pivot set $S_p = \{v_4, v_6, v_8, v_{10}\}$, MatchGen yields a set of instances including q_2 and q_3. Cover then verifies $q_2(G)$=$\{v_4, v_8, v_{10}\}$ and $q_3(G) = \{v_4, v_6\}$ as a minimum weighted set cover for S_p (with no additional nodes), and returns $\mathcal{Q} = \{q_2, q_3\}$ as a solution. This suggests a proper gender coverage with equal opportunity by considering both "Action" and "Comedy" movie actors. □

Analysis. We show that APXQGen ensures the following guarantee for SQG with the condition below.

Theorem 3: APXQGen *is an $(\alpha, 2\sqrt{|\mathcal{S}_Q| \log C}$-approximation for* SQG*, for any* Pivot *that approximates the optimal pivot set with approximation ratio α.* □

Proof sketch: Let the ideal solution be $\mathcal{Q}^*$. A reference solution $\mathcal{Q}_r$ ensures $f(\mathcal{Q}_r) \geq \alpha f(\mathcal{Q}^*)$ with minimized relative coverage error $\delta(\mathcal{Q}_r, \mathcal{P})$. Consider $\mathcal{Q}$ ($\mathcal{Q} \neq \emptyset$) computed by APXQGen.

(1) $f(\mathcal{Q}) \geq f(\mathcal{Q}_r) \geq \alpha \cdot f(\mathcal{Q}^*)$. As Pivot is an α-approximation of the subset selection problem with output S_p from V_{u_o}, we have $f(S_p) \geq \alpha f(S^*) \geq \alpha f(\mathcal{Q}^*)$. Given that $S_p \subseteq \mathcal{Q}(G)$, we have $f(\mathcal{Q}) \geq f(S_p)$. Thus $f(\mathcal{Q}) \geq \alpha f(\mathcal{Q}^*)$.

(2) $\delta(\mathcal{Q}, \mathcal{P}) \leq 2\sqrt{|\mathcal{S}_Q| \log |C}\delta(\mathcal{Q}_r, \mathcal{P})$. For any set $S \subseteq V_{u_o}$, $\delta(S, \mathcal{P}) \cdot |\bigcup(\mathcal{P})| = \sum_{P_i \in \mathcal{P}}(|S \cap P_i| - c_i) = \sum_{P_i \in \mathcal{P}}(S \cap P_i) - C = |S \cap \bigcup \mathcal{P}| - C$. Assume $S_p \subseteq S$, we have $\delta(S, \mathcal{P}) \cdot |\bigcup(\mathcal{P})| = |S_p \cap \bigcup(\mathcal{P})| + |(S \setminus S_p) \cap \mathcal{P}| - C = |(S \setminus S_p) \cap \mathcal{P}| = |S \cap S_n|$. Denote the optimal weighted set cover of S_p as S_c^*. As Cover simulates the greedy approximation in [28] without missing instances (Lemma 2), we

```
Procedure PivotEq
Input: groups $\mathcal{P}$ with equal cardinality constraint $c$.
Output:a pivot set $S_p$;
1.  set $S_p := \emptyset$;
2.  for each $P_i \in P$ do
3.      initializes $S_p^i$ with $c$ random nodes in $P_i$;
4.      while $S_p^i$ changes do
5.        for each pair of nodes $(v, v') \in S_p^i \times (P_i \setminus S_p^i)$ do
6.        if $f((S_p^i \setminus \{v\}) \cup \{v'\}) > (1 + \frac{\epsilon}{|\bigcup(\mathcal{P})|}) f(S_p^i)$;
7.            $S_p^i := S_p^i \setminus \{v\} \cup \{v'\}$;
8.      $S_p := S_p \cup S_p^i$;
9.  return $S_p$;
```

Figure 5: Procedure PivotEq

have $|Q(G) \cap S_n| \leq 2\sqrt{|\mathcal{S}_Q| \log C} |\bigcup(S_c^*) \cap S_n|$. Hence, $\delta(Q, \mathcal{P}) \cdot |\bigcup(\mathcal{P})| \leq 2\sqrt{|\mathcal{S}_Q| \log C} \delta(S_c^*, \mathcal{P}) \cdot |\bigcup(\mathcal{P})| \leq 2\sqrt{|\mathcal{S}_Q| \log C} \delta(Q_r, \mathcal{P}) \cdot |\bigcup(\mathcal{P})|$. This completes the proof of Theorem 3. □

Time cost. There are at most $|Q(u_o)|$ variables. Let md be $\max(|\text{adom}(A)|)$ where adom(A) is the active domain of a node attribute. Thus procedure MatchGen takes in total $O(|N_n(S_p)| + md \cdot |Q(u_o)| + |\mathcal{S}_Q|)$ time to generate instances $\mathcal{S}_Q$ (n is the diameter of $Q(u_o)$). Procedure Cover takes at most B rounds, and each round takes $O(|C||\mathcal{S}_Q|)$ time to compute the set cover. It takes in total $|\mathcal{S}_Q| \cdot T$ time to verify the exact matches, where T is the time cost of verifying the matches of a single instance. Let $T(S_p)$ be the time cost of the procedure Pivot on computing S_p. The total time cost is thus in $O((B \cdot C + T)|\mathcal{S}_Q| + T_{S_p})$. We found that $|\mathcal{S}_Q|$ is not large in practice (a few hundreds after pruning), and APXQGen is feasible for large graphs due to late verification (see Section 6).

We next investigate two practical scenarios for diversified query generation, and provide approximations ensured by specific Pivot.

4.2 Diversified Query Generation

Query Generation with Equal opportunity. SQG with equal opportunity computes Q with answers that (1) maximize a max-sum diversity in terms of pairwise distances ($\lambda = 1$), and (2) retrieves equally c nodes from each group (*e.g.,* gender, topics), as seen in social [5, 16, 26] or Web search[1]. We show the following result.

Theorem 4: *The* SQG *problem with equal opportunity has an* $(\frac{1}{2} - \frac{\epsilon}{|\bigcup(\mathcal{P})|}, 2\sqrt{|\mathcal{S}_Q| \log c|\mathcal{P}|})$*-approximation in time* $O((B \cdot C + T)|\mathcal{S}_Q| + |\mathcal{P}| \frac{||\bigcup(\mathcal{P})|}{\epsilon} \log(|\mathcal{P}|))$. □

Given Theorem 3, it suffices to show that SQG permits an $\frac{1}{2} - \frac{\epsilon}{|\bigcup(\mathcal{P})|}$-approximation for pivot sets with equal opportunity. We present such a procedure., denoted as PivotEq.

Procedure PivotEq. The procedure (illustrated in Fig. 5) computes S_p as the union of $|\mathcal{P}|$ diversified sets. Each set $S_p^i \subseteq P_i \in \mathcal{P}$ contains c nodes that are picked with a local search strategy as follows. (1) PivotEq initializes S_p^i with c randomly selected nodes (line 3). (2) It verifies if "swapping" of a pair (v, v') where $v \in S_p^i$ and v' in $P_i \setminus S_p^i$ improves the diversity, and if so, updates S_p^i (lines 5-7). This repeats until $f(S_p^i)$ can no longer be locally improved (line 4). It finally returns S_p as the union of all S_p^i (line 9).

```
Algorithm HeuQGen
Input: graph $G$, parameterized query $Q(u_o)$, integer $k$,
       groups $\mathcal{P}$ with cardinality constraints.
Output:a set of instances $Q$.
1.  set $Q := \emptyset$; queue $\mathcal{S}_Q := \emptyset$; set $\mathcal{P}' := \mathcal{P}$;
2.  while $|Q| < k$ do
3.      $Q := Q \cup \{$ getNext $(\mathcal{P}'Q, \mathcal{S}_Q)\}$;
4.      set $\mathcal{P}' := \{P_i' | P_i' := P_i \setminus Q(G); P_i \in \mathcal{P})\}$;
5.      $c_i' := |P_i'|$ for each $P_i' \in \mathcal{P}'$;
6.  while $hasNext(Q)$ do
7.      $q := getNext(\mathcal{P}', Q, \mathcal{S}_Q)$;
8.      update $\mathcal{P}'$ and each $c_i$ of $P_i \in \mathcal{P}$ with $\overline{q(G)}$;
9.      if $(\min(f(q') | q' \in Q) > f(\overline{q(G)})$
        or no $P_i' \in \mathcal{P}'$ has $c_i' > 0$ then return $Q$;
10.     if there is no change in $\mathcal{P}'$ then continue ;
11.     find $q' \in Q$ with maximum $f(Q \setminus \{q'\} \cup q) - f(Q)$
        or maximum $\delta(Q, \mathcal{P}) - \delta(Q \setminus \{q'\} \cup q, \mathcal{P})$);
12.     $Q := Q \setminus \{q'\} \cup q)$;
13. return $Q$;
```

Figure 6: Algorithm HeuQGen

Correctness & Complexity. PivotEq returns a pivot set S_p with size $c|\mathcal{P}|$, and ensures a coverage of each group $\mathcal{P}_i$ with c nodes. We show that it approximates an optimal pivot set S^* with ratio $\frac{1}{2} - \frac{\epsilon}{|\bigcup(\mathcal{P})|}$ for a small constant ϵ. To see this, we perform an approximation preserving reduction to clustered diversity maximization with matroid constraints (CDM) [1]. Given a set D and a set of subsets $\mathcal{D} = (D_1 \ldots, D_k)$ of D, a matroid is a pair $(\bigcup(\mathcal{D}), \mathcal{M})$, where $\mathcal{M}$ is a set of independent sets with certain matroid constraints. An instance of CDM $(\mathcal{D}\mathcal{M}, p)$ is to select a set of subsets $\mathcal{R} = \{R_1, \ldots, R_k\}$, such that $R_i \subseteq D_i$, $R_i \in \mathcal{M}$, and $|R_i| = p$, with maximized diversity $f'(\bigcup(\mathcal{R}))$ (f' is a special case of $f(\cdot)$ with λ = 1). By setting $\mathcal{P} = \mathcal{D}$, $c = p$, and $\mathcal{M}$ as the subsets of $\bigcup(\mathcal{P})$ with size up to c, we can verify that PivotEq simulates a local search approximation [1] with approximation ratio $\frac{1}{2}$ - $\frac{\epsilon}{|\bigcup(\mathcal{P})|}$.

For time cost, PivotEq takes at most $|\mathcal{P}|$ rounds. In each round, it takes in total $O(\frac{|\bigcup(\mathcal{P})|}{\epsilon} \log(|\mathcal{P}|))$ time (lines 4-8) for a small constant ϵ, as PivotEq only triggers a bounded number of swapping upon a factor of $(1+\frac{\epsilon}{|\bigcup(\mathcal{P})|})$ incremental improvement from any random selection to optimal diversity. The total cost of PivotEq is thus in $O(|\mathcal{P}||\bigcup(\mathcal{P})|\epsilon \log(|\mathcal{P}|))$. Theorem 4 thus follows.

Generate Queries with Cardinality Constraints. We next investigate SQG with cardinality constraints, which sets $\mathcal{P} = \{V_{u_o}\}$ (a single group). This allows us to generate Q with diversified answers that also has a desired output size c as needed in query benchmarking and database tests [3–5]. We show the following result.

Theorem 5: *The* SQG *problem with cardinality constraints has a* $(\frac{1}{2}, 2\sqrt{|\mathcal{S}_Q| \log c})$*-approximation in time* $O((B \cdot C + T)|\mathcal{S}_Q| + c|\bigcup(\mathcal{P})|)$. □

Procedure PivotC. In this case, Pivot solves a max-sum diversification problem. Given a set V_{u_o}, it aims to select a subset S_p with size c and maximizes $f(S_p)$. Procedure PivotC adopts a greedy strategy to iteratively enlarge S_p (initialized as $\emptyset$) by selecting a node v' from $V_{u_o} \setminus S_p$, with the maximal marginal gain, *i.e.,* $f(S_p \cup \{v'\})$ - $f(S_p)$ is maximized. This step repeats until $|S_p| = c$. It is known that the above greedy selection strategy yields a $\frac{1}{2}$-approximation for

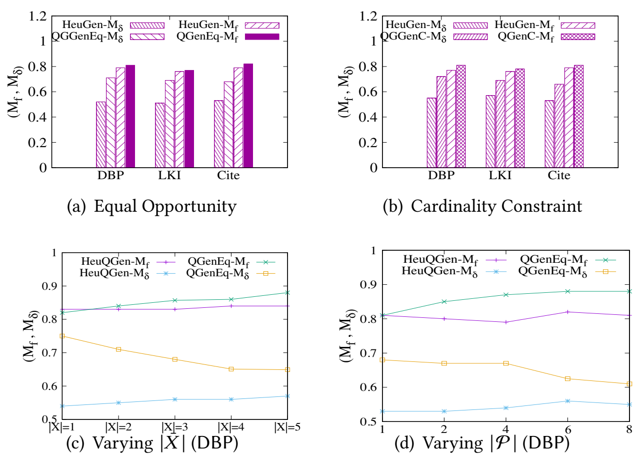

Figure 7: Effectiveness of Query Generation

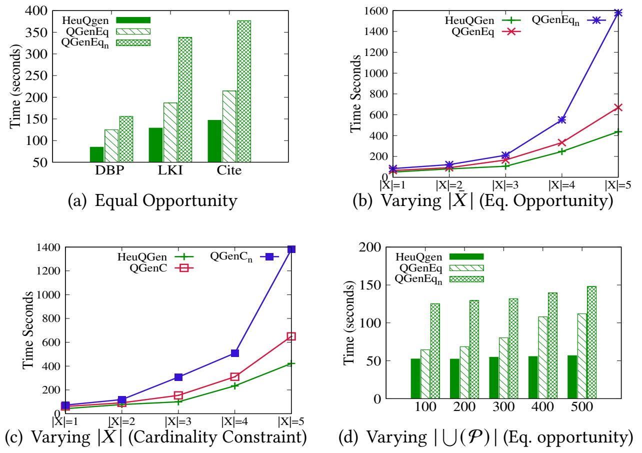

Figure 8: Efficiency of Query Generation

Max-Sum Diversification [17]. The procedure Cover then generates Q with answers that covers S_p and minimizes additional nodes from $V_{u_o} \setminus S_p$. The time cost of PivotC is in $O(c|\bigcup(\mathcal{P})|)$. Putting these and Theorem 3, Theorem 5 follows.

5 EARLY TERMINATION HEURISTICS

Algorithm APXQGen requires a "batch" generation of promising instances and may return an excessive number of instances. It is also desirable to find Q with k high-quality instances as soon as possible [22]. We present a heuristic algorithm for SQG, denoted as HeuQGen, that *early terminates* without enumerating instances.

Algorithm. Similar to APXQGen, HeuQGen uses a priority queue $\mathcal{S}_Q$ to maintain Q up to k instances, with an additional pair $(f(q(G)), \delta(q(G)))$ per instance q. It populates Q by invoking a procedure getNext (lines 2-5), and greedily update Q by replacing an instance q' with a next new instance from getNext that can maximally improve $f(Q)$ or $\delta(Q)$ (lines 11-12). In this process, it (1) ensure that q covers new nodes in $\mathcal{P}$ by retaining uncovered nodes in groups $\mathcal{P}'$ (initialized as $\mathcal{P}$; lines 4-5; line 8); and (2) skips unnecessary verification if q has no contribution to $f(Q)$ or $\delta(Q, \mathcal{P})$ (line 10). It *early terminates* when (a) no instance can improve $f(Q)$, or (b) all nodes in $\mathcal{P}$ are covered (thus $\delta(Q, \mathcal{P})$ converges).

Procedure hasNext. The procedure (1) generates and puts the first instance to $\mathcal{S}_Q.Q$ with "most constrained" predicates (*e.g.,* setting all edge variables to 1, and range variables to minimum or maximum values); and (2) ensures to generate a next instance by *relaxing* the latest one in $\mathcal{S}_Q.Q$, one variable and one "step" at a time (*e.g.,* changing a range variable value to the next smaller or larger counterpart). It also skips instances that cannot cover any nodes in the input $\mathcal{P}$, similar to MatchGen (details omitted due to space limit).

Analysis. Algorithm HeuQGen correctly identifies Q with k instances. The early termination property is ensured by hasNext, which produces a sequence of instances by keep "relaxing" the search predicates. This ensures an invariant that Q has monotonically increasing $\min(f(q)|q \in Q)$, and a monotonically decreasing $\delta(Q, \mathcal{P})$ before termination. As each iteration covers at least one node in $\mathcal{P}$, it takes up to $|\bigcup(\mathcal{P})|$ rounds of replacement with verification (line 11). The time cost is thus in $O(|\bigcup(\mathcal{P})|(k + T))$.

6 EXPERIMENTS

Based on real-world attributed graphs, we experimentally verify the effectiveness and efficiency of our algorithms for SQG.

Experiment Setting. We used the following setting.

Datasets. We use three real-life data graphs. (1) DBP is a movie knowledge graph induced from DBpedia [23] with $1M$ entities (*e.g.,* movies, directors, actors) and $3.18M$ relations (*e.g.,* directed, collaboration). Each node has attributes such as title, genre. We induce movie groups based on their genres or countries. (2) LKI [34] contains $3M$ nodes (users, organizations) with labels (professions, skills) and $26M$ edges (*e.g.,* co-review). We induce gender groups $\mathcal{P}$ with synthetic genders generated by gender inference tools [11]. (3)Cite [29] contains 4.9 nodes (*e.g.,* papers, authors) with labels (*e.g.,* topics) and $46M$ edges (*e.g.,* citations, authorship). We induce groups $\mathcal{P}$ of papers having different topics.

Queries. We developed a generator to produce parameterized queries with practical search conditions, controlled by the number of variables, query size $|Q(u_o)|$ and topologies.

Algorithms. We implemented the following algorithms in Java.: (1) algorithms QGenEq for SQG with equal opportunity, and QGenC for SQG with cardinality constraints, which specifies APXQGen with procedures PivotEq and PivotC, respectively; (2) algorithms QGenEq$_n$ (resp. QGenC$_n$), a counterpart of QGenEq (resp. QGen) without the late verification strategies; (3) the early terminating algorithm HeuQGen, and (4) an exact algorithm EnumGen that enumerates all instances and their sets, and terminates once the reference solution Q_f is found.

Reference solutions. For a fair comparison, we use EnumGen to ensure the existence of a "yardstick" reference solution Q_f, by setting a long running time (2 hours) unless it terminates with all possible enumerations. exhausted. We track the solution Q_f with the smallest coverage error, and ensure a local optimal diversity by "perturbing" the instances in Q_f with other domain values.

Experimental results. We next present our findings.

Exp-1: Effectiveness. We first evaluated the effectiveness of our algorithms. QGenEq$_n$ (resp. QGenC$_n$) report the same results as

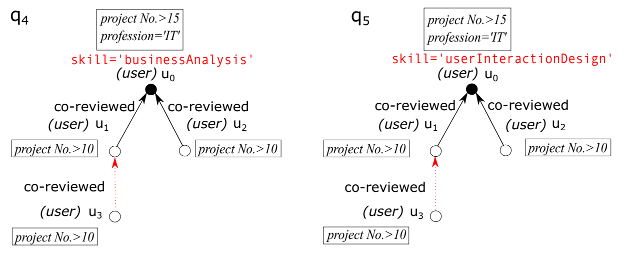

Figure 9: Case study: Query Suggestion for Talent Search

QGenEq (resp. QGenC), thus are omitted. We quantify the effectiveness with a normalized M_1 score [35] between the returned Q and reference solution Q_f. Specifically, M_1 is a pair (M_f, M_δ), where $M_f = \frac{f(Q)}{f(Q_f)}$, and $M_\delta = \frac{\delta(Q_f,\mathcal{P})}{\delta(Q,\mathcal{P})}$ (if $\delta(Q,\mathcal{P}) = 0$ then M_δ = 1), both in $[0, 1]$. This measure also indicates the ratios in (α, β)-approximations: the closer to 1, the better.

Equal Opportunity. We compare the performance of EnumGen, QGenEq and HeuQGen for SQG with equal opportunity over the three real life datasets (Fig. 7(a)). We set $|Q| = 3$ with 3 variables (1 edge variables, and 2 range variables), and $|\mathcal{P}| = 2$. We set k = 50 for HeuQGen, which is calibrated with the output size from QGenEq. (1) QGenEq achieves high scores, at least 0.69 and 0.8 in both diversity and coverage, respectively, over all the datasets. (2) HeuQGen has comparable quality. While it terminates early, we found that it achieves $M_f = 0.52$ and $M_\delta = 0.79$ and inspects 20% less instances compared with QGenEq.

Cardinality Constraints. Using the same setting as in Fig. 7(a) while fixing $|\mathcal{P}| = 1$ ($\mathcal{P} = \{V_{u_o}\}$), and k = 50 for HeuQGen, we next report the performance of EnumGen, QGenEq, QGenC and HeuQGen for SQG with cardinality constraints in Fig. 7(b). To apply QGenEq, we set $\lambda = 1$. While QGenC and QGenEq have comparable scores in both diversity and coverage, QGenC achieves better ratios, and in general inspects fewer instances due to a simpler greedy strategy. HeuQGen can achieve 0.54 and 0.78 in diversity and coverage by further inspecting less instances compared with QGenC, due to the early termination strategy.

Varying $|\bar{X}|$. Fixing $|Q| = 4$, $|\mathcal{P}| = 200$, k= 50 for HeuQGen, we varied the number of range variables from 1 to 3 and enlarge to five by allowing two additional edge variables, and evaluated the impact to the performance of QGenEq and HeuQGen for equal opportunity. Fig. 7(c) shows that the M_δ score of QGenEq is more sensitive to the number of variables. As $|\bar{X}|$ increases, more instances need to be inspected in $\mathcal{S}_Q$. This makes it more difficult for the greedy set cover strategy (Cover) to find Q with good coverage, which is consistent with our analysis in Theorem 3. HeuQGen reports lower scores, but is less sensitive, as it incurs less verification cost.

Varying $|\mathcal{P}|$. Fixing $|Q| = 3$, $|\bar{X}| = 3$ (with 1 range variables and 2 edge variables), and k= 50 for HeuQGen, we varied $|\mathcal{P}|$ from 1 to 6 by fixing $\bigcup(\mathcal{P})$ and cardinality constraints, but only change its partitions). As shown in Fig. 7(d), QGenEq achieves a higher ratio than HeuQGen, and reports M_f that is less sensitive than M_δ. Indeed, it is harder to generate Q that can properly cover all the groups as the number of groups increases. On the other hand, the performance of HeuQGen is less sensitive to $|\mathcal{P}|$.

Exp-2: Efficiency. We next evaluate the efficiency of the algorithms QGenEq, QGenEq$_n$, QGenC, QGenC$_n$, HeuQGen, and EnumGen.

Efficiency over real-life graphs (Equal opportunity). Fig. 8(a) reports the performance of QGenEq, QGenEq$_n$, and HeuQGen over the real datasets. For all the datasets, HeuQGen achieves the best performance. On average, it outperforms QGenEq by 2.4 times due to the early termination property. The late verification strategy of QGenEq improves the efficiency of QGenEq$_n$ by 1.65 times. In general, QGenEq and HeuQGen are feasible. For example, while EnumGen takes more than 2 hours over DBP, it takes QGenEq (resp. HeuQGen) 187s (resp. 129s) over LKI with $3M$ nodes and $26M$ edges to generate queries with comparable quality.

Varying $|\bar{X}|$ (Equal opportunity). Using the same setting as in Fig. 7(c), we reported the performance of QGenEq, QGenEq$_n$, HeuQGen and EnumGen over DBP in Fig. 8(b). (1) All the algorithms take longer time for larger $|\bar{X}|$ due to that more instances need to be inspected, as expected. (2) QGenEq outperforms QGenEq$_n$ by 1.88 times, and improves the latter better for larger $|\bar{X}|$ due to the late verification strategy. It inspects 10% less instances than QGenEq$_n$. (3) HeuQGen outperforms all the algorithms due to the early termination strategy, and remains least sensitive.

Varying $|\bar{X}|$ (Cardinality constraint). Using the same setting as in Fig. 7(b) over DBP, while varying $|\bar{X}|$ from 1 to 5 we verify the performance of all the algorithms in Fig. 8(c). (1) HeuQGen achieves the best performance among all the algorithms, and is less sensitive compared with others. QGenC (resp. QGenEq) outperforms QGenC$_n$ (resp. QGenEq$_n$) by 1.9 times (resp. 2) on average. On the other hand, QGenC is less sensitive compared with QGenEq with a simpler greedy strategy.

Varying $|\bigcup(\mathcal{P})|$ (Equal opportunity). Fig. 8(d) reports the efficiency in the same setting as in Fig. 7(d), while keeping $|\mathcal{P}|$ unchanged. All the algorithms take more time to compute Q when $|\bigcup(\mathcal{P})|$ is larger. QGenEq outperforms QGenEq$_n$ by 1.5 times on average.

Exp-3: Case Study. We also manually verified k=2 instances found by HeuQGen for talent search with fairness constraints, and report a case in Fig. 9. The parameterized query Q (not shown) contains a range variable on skills and an edge variable between two users (u_3, u_1). An initial query with an instantiation $\{\emptyset, `1'\}$ returns a set of 62 male and 12 female candidates. Posing a gender equality requirement, q_4 and q_5 are suggested, resulting a balanced set of 30 female and 30 male. The queries suggest to relax long "co-review" chains while specifying two different skillsets.

7 CONCLUSIONS

We have introduced and studied the diversified query generation with group fairness problem. We verified the hardness of the problem. We have provided both feasible approximation algorithms (for equal opportunity and cardinality constraint on output sizes) and fast heuristics (for the general problem) with optimization strategies such as late verification and properties of *early termination*. As verified analytically and experimentally, our methods are feasible for large graphs, and can achieve desirable diversity and coverage properties over targeted groups.

Acknowledgments. This work is supported by NSF under CNS-1932574, OIA-1937143, ECCS-1933279, CNS-2028748 and DoE under DE-IA0000025.

REFERENCES

[1] Z. Abbassi, V. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *KDD*, 2013.

[2] A. Asudeh and H. Jagadish. Fairly evaluating and scoring items in a data set. *Proceedings of the VLDB Endowment*, 13(12):3445–3448, 2020.

[3] G. Bagan, A. Bonifati, R. Ciucanu, G. H. Fletcher, A. Lemay, and N. Advokaat. Controlling diversity in benchmarking graph databases. *arXiv preprint arXiv:1511.08386*, 11, 2015.

[4] G. Bagan, A. Bonifati, R. Ciucanu, G. H. Fletcher, A. Lemay, and N. Advokaat. gmark: Schema-driven generation of graphs and queries. *IEEE Transactions on Knowledge and Data Engineering*, 29(4):856–869, 2016.

[5] A. Bonifati, I. Holubová, A. Prat-Pérez, and S. Sakr. Graph generators: State of the art and open challenges. *ACM Computing Surveys (CSUR)*, 53(2):1–30, 2020.

[6] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. PODS, 2012.

[7] S. Bouhenni, S. Yahiaoui, N. Nouali-Taboudjemat, and H. Kheddouci. A survey on distributed graph pattern matching in massive graphs. *CSUR*, 54(2):1–35, 2021.

[8] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *SODA*, 2000.

[9] S. Chaudhuri, H. Lee, and V. R. Narasayya. Variance aware optimization of parameterized queries. In *SIGMOD*, 2010.

[10] 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.

[11] Y. Dong, Y. Yang, J. Tang, Y. Yang, and N. V. Chawla. Inferring user demographics and social strategies in mobile social networks. In *KDD*, 2014.

[12] T. Draws, N. Tintarev, and U. Gadiraju. Assessing viewpoint diversity in search results using ranking fairness metrics. *ACM SIGKDD Explorations Newsletter*, 23(1):50–58, 2021.

[13] M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[14] W. Fan, X. Wang, and Y. Wu. Diversified top-k graph pattern matching. *VLDB*, 2013.

[15] Y. Ge, S. Zhao, H. Zhou, C. Pei, F. Sun, W. Ou, and Y. Zhang. Understanding echo chambers in e-commerce recommender systems. In *SIGIR*, pages 2261–2270, 2020.

[16] S. C. Geyik, S. Ambler, and K. Kenthapadi. Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. In *KDD*, 2019.

[17] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.

[18] T. F. Gonzalez. *Handbook of approximation algorithms and metaheuristics*. CRC Press, 2007.

[19] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri. Querying knowledge graphs by example entity tuples. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2797–2811, 2015.

[20] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, 2006.

[21] W. Le, A. Kementsietsidis, S. Duan, and F. Li. Scalable multi-query optimization for sparql. In *ICDE*, pages 666–677, 2012.

[22] M. Lissandrini, D. Mottin, T. Palpanas, and Y. Velegrakis. Graph-query suggestions for knowledge graph exploration. In *The Web Conference*, 2020.

[23] J. Lu, J. Chen, and C. Zhang. Helsinki Multi-Model Data Repository. https://www2.helsinki.fi/en/researchgroups/unified-database-management-systems-udbms/, 2018.

[24] T. Ma, S. Yu, J. Cao, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan. A comparative study of subgraph matching isomorphic methods in social networks. *IEEE Access*, 6:66621–66631, 2018.

[25] D. Mottin, F. Bonchi, and F. Gullo. Graph query reformulation with diversity. In *KDD*, 2015.

[26] Z. Moumoulidou, A. McGregor, and A. Meliou. Diverse data selection under fairness constraints. In *ICDT*, 2021.

[27] M. H. Namaki, Q. Song, Y. Wu, and S. Yang. Answering why-questions by exemplars in attributed graphs. In *SIGMOD*, 2019.

[28] D. Peleg. Approximation algorithms for the label-covermax and red-blue set cover problems. *Journal of Discrete Algorithms*, 5(1):55–64, 2007.

[29] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. P. Hsu, and K. Wang. An overview of microsoft academic service (mas) and applications. WWW, 2015.

[30] Q. Song, M. H. Namaki, and Y. Wu. Answering why-questions for subgraph queries in multi-attributed graphs. In *ICDE*, 2019.

[31] J. Stoyanovich, K. Yang, and H. Jagadish. Online set selection with fairness and diversity constraints. In *Proceedings of the EDBT Conference*, 2018.

[32] V. V. Vazirani. *Approximation algorithms*. Springer, 2013.

[33] Y. Wang, Y. Li, J. Fan, C. Ye, and M. Chai. A survey of typical attributed graph queries. *World Wide Web*, 24(1):297–346, 2021.

[34] Y. Zhang, J. Tang, Z. Yang, J. Pei, and P. S. Yu. Cosnet: Connecting heterogeneous social networks with local and global consistency. In *KDD*, 2015.

[35] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.