Expressive Languages for Selecting Groups from Graph-Structured Data

Vitaliy Liptchinsky, Benjamin Satzger, Rostyslav Zabolotnyi, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
lastname@dsg.tuwien.ac.at

ABSTRACT

Many query languages for graph-structured data are based on regular path expressions, which describe relations among pairs of nodes. We propose an extension that allows to retrieve groups of nodes based on group structural characteristics and relations to other nodes or groups. It allows to express group selection queries in a concise and natural style, and can be integrated into any query language based on regular path queries. We present an efficient algorithm for evaluating group queries in polynomial time from an input data graph. Evaluations using real-world social networks demonstrate the practical feasibility of our approach.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Query Languages

Keywords

Graph Query Languages; Semantic Web; Social Selection

1. INTRODUCTION

The World Wide Web has rapidly evolved from a network providing access to static webpages to a highly interactive experience. Web-based social networking services connect millions of people and have deeply changed the way we communicate with each other. Graphs are the natural data structure to represent social networks. The swift growth of social networking services fueled with further developments in the Web (e.g., Semantic Web) has led to a rising interest in graph databases and related query languages.

Today, SPARQL [33], the W3C[1] query language for RDF [27], is probably the most widely known query language for graph-structured data. SPARQL and many other graph query languages allow users to retrieve nodes based on conjunctive queries (CQs). A simple CQ for finding persons who are friends with Adam and Eve could be expressed as $x\ friendof\ Adam \wedge x\ friendof\ Eve$. An extension called regular path queries (CRPQs) allows querying for nodes that are connected by a path satisfying a regular expression rather than relying solely on static paths. CRPQs have been thoroughly studied and form the basis for many query languages [15, 13, 2, 20, 10, 26, 37] including SPARQL 1.1 [1].

[1]http://www.w3.org/

With CRPQs, however, it is not possible to query sets of nodes. For instance, Adam and Eve might not just want to find their common friends, but a group of common friends that are connected between them. Finding sets of nodes that satisfy certain characteristics is of general interest for graph-structured data, and particularly important for social networks. Let us consider a graph representing the skills and relationships of developers of open source projects. With CRPQ-based query languages it is, for instance, not possible to find socially coherent teams capable of conducting a certain project. While groups are a fundamental concept in social networks they are not well supported by current graph query languages.

In this paper we propose an elegant and simple, but at the same time expressive, extension - conjunctive set regular path queries (CSRPQs), which extends CRPQs with the notion of sets of nodes. Using the example of SPARQL, we show how CSRPQs can be integrated into existing graph query languages. We show that data complexity of CSRPQ queries is in *PTIME*, and propose a novel algorithm for efficient query evaluation, which leverages structure of the db-graph and the query itself for search space minimization. Experiments show that query evaluation is feasible even for real-world social networks of large scale.

In the remainder of this paper, we first give a motivating scenario in Section 2 and discuss related work in Section 3. After recapturing CRPQs in Section 4, we present CSRPQs in Section 5, and demonstrate their expressivity in Section 6. Section 7 presents our algorithms for CSRPQ evaluation and reasons about its complexity. An evaluation based on real-world datasets is presented in Section 8. Section 9 concludes the paper and gives a future work outlook.

2. MOTIVATION

The importance of social ties in software engineering is often emphasized in literature [29, 17]. Conway's law suggests that "organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" [14]. We can rephrase it as: organization's social network structure influences its systems architecture. Also, Peopleware [32] claims the main reason for project failures to be the human factor, i.e., lack of communication or bad social environment, and not the technology factor. Taking into account these two postulates, a proper process of project team selection can be defined as a selection of a subgraph, which exhibits certain structural characteristics, from social and organizational overlay networks. For example, when assembling a project team it is preferable to maximize social coherence within the team as well as to maintain good social connections with other project teams. In social network analysis a number of characteristics and patterns have been defined to characterize social coherence of a group, such as geodesics, n-clique, n-clan, n-club, k-core, k-plex, and so on [36]. Also, efficient coordination and integration of project teams requires selection of special actors with certain structural properties, such as closeness, degree, or betweenness centralities.

Let us consider now a motivating scenario with a few examples. An organization wants to start a new open source software (OSS) project, consisting of a number of sub projects. For this purpose it uses a social network comprised of independent software engineering experts as well as its own employees. In the social network two software engineering experts are considered to be connected if they either worked on the same software engineering project in the past or they are connected in a social networking site. Now, let us consider few examples of queries the organization may want to issue in order to select project teams:

1. Closeness centrality and connectedness. In case of a core component, it may be needed to select a project team that has good social connections to all the other teams, i.e. located in an intersection of their neighborhoods. Moreover, the project team can be either implicitly or explicitly connected. A team is implicitly connected if it forms a connected graph. Otherwise, if team members are connected via external nodes, the team is considered to be implicitly connected. Closeness to other teams improves inter-team communication, while explicit connectedness improves communication within the team itself.

2. Cluster. To select a project team for implementation of a monolithic component (i.e., a component with a lot of interdependencies between its parts) one might want to select a clique (complete graph). This is necessary to maximize, again, communication within the team and ensure that everyone is aware of all the changes introduced by other team members. However, in practice cliques of required size and comprised of necessary experts can rarely be found. Therefore, one may want to relax the requirement by selecting a k-plex, where k-plex is defined as a graph G where every vertex has a degree $|G| - k$.

3. Independent teams. In case of highly reliable software systems a valid practice to minimize probability of failures is to execute several mutually replaceable components in parallel. In order to mitigate failures introduced by a human factor (software defects), it is important that such mutually replaceable components are developed by teams isolated from each other. This is necessary to avoid propagation of erroneous approaches.

4. Liaison. When integrating two different components being developed by project teams that have no direct social relations it may be helpful to search for a liaison/broker between two teams. Such a liaison would facilitate communication and increase chances for successful integration.

5. Structural equivalence. In order to replace an expert, that recently left a project, it is necessary to find a structurally equivalent expert, i.e. an expert that has almost the same social neighborhood as the former expert with respect to dependent project teams. This would decrease onboarding time and restore structural characteristics of the team. Also, if it is not possible to find such a replacement, we may want to find a group of people that is structurally equivalent.

Figure 1 shows an exemplary snippet from a graph representing relations among developers of OSS. Developers are represented as nodes, and two of them are connected by an edge if they are socially connected. One possible solution to each of the five aforementioned queries is highlighted in the figure.

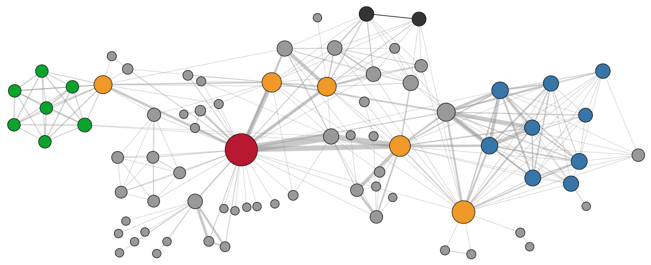


Figure 1: Simplified graph representing social relationship among open source software developers. Possible answers for OSS example queries are highlighted: 1) Closeness Centrality: yellow nodes, 2) Cluster: green nodes (and blue nodes in case of 1-plex), 3) Independent teams: green nodes for first team and blue nodes for second team, 4) Liaison: red node, 5) Structural equivalence: the two black nodes connected with bold edge can replace each other

In the next section we discuss existing graph query languages and show their shortcomings with respect to the query examples enlisted above.

3. RELATED WORK

In this section we discuss existing graph query languages and consider their capabilities of expressing the queries outlined in the previous section. A comprehensive overview of existing graph query languages can be found in [38] and [4]. Functionality of canonical *Conjunctive Regular Path Queries* (CRPQs) has been employed in many graph query languages, such as **G** [15], GraphLog [13], Lorel [2], StruQL [20], UnQL [10], NAGA [26], Cypher [37], and SPARQL 1.1 [1].

Query 1 in the motivating scenario defined in CRPQs would search for a disconnected set of nodes satisfying the neighborhood condition, thus being incapable of finding a tightly connected subgraph in a neighborhood. Queries expressed in CRPQ-based (and not only) languages typically return node tuples, and are not able to return groups of varying size. One exception here is NAGA, whose queries return graphs. Extended Conjunctive Regular Path Queries [5] can return paths along with nodes, which is somewhat similar to returning groups. Another interesting exception is SQL-based PQL [28] language, which can return nodes with their neighborhoods. *Query 2* is flexible in the sense that it does specify neither exact number of nodes in a group to be found, nor its exact topology, but rather general characteristics. In literature there were a number of approaches employed for definition of "flexible" topologies in query languages. Even canonical CRPQs allow number of nodes to vary in paths. GraphQL [22] with repetition of graph motifs allows to define not only paths, but also, for example, cycles and trees being thus more expressive in this matter than regular path expressions. Approach studied in [19] relaxes constraints on topology of returned graphs in favor of performance. This approach is the most extreme with respect to allowed relaxations on topologies, and can lead to completely irrelevant results. Queries specifying "flexible" topologies are quite the opposite approach to approximate graph matching [25, 21, 23], where queries define precise graph structure, and returned results may not match but rather be "close" to the query graph by means of edit operations (insert/delete/substitute).

In the domain of social networks there are several dedicated graph query languages, such as SoQL [35], BiQL [16], SocialScope [3], SNQL [30], QGraph [7], as well as extensions of SPARQL for social network analysis [18]. SoSQL is the only one of them that can specify flexible selection of groups of nodes with SELECT FROM GROUP queries. Its expressivity, however, is limited with respect to graph patterns used in social network analysis, as it is not capable, for example, to select groups exhibiting characteristics of k-plex, n-clique, n-club, etc. Also, it does not have groups as first-class citizens, i.e., it is not possible to specify paths or connections between two or more groups. For instance, in *Query 3*: select two groups that are mutually isolated. Selection of actors, as in *Query 4* and *Query 5*, is also not supported. Finally, SoSQL provides no algorithm for query evaluation, which seems to be a complex computational problem.

BiQL [16] suggests integration of external tools for finding graphs with certain characteristics, e.g., quasi-cliques or clusters. Its distinguishing feature is unification of nodes and edges, which makes it possible, though with integration of external tools and algorithms, to find clusters of edges and not only clusters of nodes. Integration of external tools for querying certain graph patterns, however, reduces flexibility of a query language with respect to definition of possible graph patterns. SocialScope [3] defines an algebraic language with node and link selection operators, union, disjunction, subtraction and composition of graphs, as well as set and numerical aggregations. Node and link selection can utilize an optional scoring function. SNQL [30] is a query language of similar functionality, as it has been claimed to cover SocialScope [34]. SNQL is intended for data management in social networks. It extends GraphLog with Skolem functions to create new nodes as part of the output. Both languages do not allow flexible selection of groups and actors exemplified in the motivating scenario. Extension of SPARQL [18] for social network analysis can examine global metrics of a graph, such as density and diameter. However, it is not capable to search for groups exhibiting specific metrics. QGraph [7] is a visual query language employed in a tool called Proximity [24], which is used for data mining in social networks. QGraph queries graph patterns can have numeric annotations, e.g., 'Find all directors that had at least 2 movies each of them winning at least 3 awards'. Such annotations resemble basic quantitative conditions needed for k-plex, e.g., each node has at least N neighbors in the group. However, QGraph does not have notion of groups to succeed in such selection.

Many efficient algorithms (e.g., [9, 12]) were proposed for selection of social formations exhibiting certain structural characteristics (e.g., regular equivalence, n-clique, k-plex, n-club), and implemented in such popular social network analysis tools as Pajek [6] and Ucinet [8]. Being capable of handling some examples in the motivating scenario, these algorithms, however, are limited to specific problems they address and are not as flexible and expressive as a query language might be. While offering far greater expressivity, a query language requires a complete and generalized query interpretation algorithm capable of solving mixed and combined problems.

The overview of related work shows that no existing language is able to fully cope with the queries enumerated in Section 2. In the next section we provide definition of CRPQs, and then, in Section 5, we present our extension.

4. PRELIMINARIES

A database is defined as a directed graph $K = (V, E)$ labeled over the finite alphabet Σ. If there is a path between node a and node b labeled with $p_1, p_2, ..., p_n$ we write $a \xrightarrow{p_1 p_2 \cdots p_n} b$. In the remainder of this section we give definitions of (conjunctive) regular path queries, similar to other works, like [11].

Definition 1 (Regular Path Queries)**.** A regular path query (RPQ) $Q^R \leftarrow R$ is defined by a regular expression R over Σ. The answer $ans(Q^R, K)$ is the set connected by a path that conforms to the regular language $L(R)$ defined by R:

$$ans(Q^R, K) = \{(a, b) \in V \times V \mid a \xrightarrow{p} b \text{ for } p \in L(R)\}.$$

Conjunctive regular path queries allow to create queries consisting of a conjunction of RPQs, augmented with variables.

Definition 2 (Conjunctive Regular Path Queries)**.** A conjunctive regular path query (CRPQ) has the form

$$Q^C(x_1, ..., x_n) \leftarrow y_1 R_1 y_2 \wedge ... \wedge y_{2m-1} R_m y_{2m},$$

where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are node variables. The variables x_i are a subset of y_i (i.e., $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$), and they are called distinguished variables. The answer $ans(Q^C, K)$ for a CRPQ is the set of tuples $(v_1, ..., v_n)$ of nodes in K such that there is a total mapping σ to nodes, with $\sigma(x_i) = v_i$ for every distinguished variable, and $(\sigma(y_i), \sigma(y_{i+1})) \in ans(Q^R, K)$ for every RPQ Q^R defined by the term $y_i R_i y_{i+1}$.

5. CONJUNCTIVE SET REGULAR PATH QUERIES

In this section we describe how CRPQs can be extended to overcome their shortcoming for finding sets of nodes. Our extensions allow to express all queries presented in the motivating scenario. The proposed extensions can be used to augment any graph query language that employs CRPQs, like SPARQL.

Before we introduce set regular path queries (SRPQs), which extend RPQs, we introduce a set of generalized quantifiers. SRPQs allow to make statements about which fraction of a set is affected by a path query. For giving SRPQs the expressiveness necessary to handle sets we allow quantifiers beyond the standard quantifiers $\forall$ and $\exists$, similar as proposed in [31]. All extended quantifiers refer to a certain set. In the following we define the quantifiers we use by showing the mapping they signify with relation to some arbitrary set M.

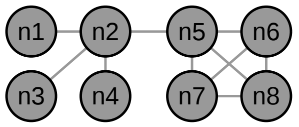


Figure 2: Exemplary graph K, serving as knowledge base for some simple queries. Nodes represent persons and edges represent the friendship relation. Since we assume that this is a symmetric relation, arrows have been omitted.

- Universal quantification $\forall_M = \{M\}$
- Existential quantification $\exists_M = \{A \subseteq M : A \neq \varnothing\}$
- Counting quantification: $\exists_M(\odot n) = \{A \subseteq M : |A| \odot n\}$, where $\odot \in \{>, \geqslant, =, \leqslant, <\}$ and $n \in \mathbb{N}$
- Fractional quantification: $\exists_M(\odot p) = \{A \subseteq M : |A| \odot p|M|\}$, where $\odot \in \{>, <\}$ and $p \in [0, 1]$

We use capital Greek letters Ξ and Ψ as placeholders for one of the above defined quantifiers. SRPQs are similar to RPQs, but extend them with the notion of sets. They come in different flavors since we distinguish between paths from a single node to a set of nodes, paths from a set to a single node, and paths from a set of node to another set of nodes. Like RPQs, all SRPQs are defined by a regular expression R over Σ.

Definition 3 (SRPQ: Node – Set)**.** A set regular path query $Q^{\bullet\Xi} \leftarrow R$ describes a relation between a single node and a set, based on a regular expression R together with a quantifier Ξ. The quantifier defines to how many nodes from the set the single node must be connected by a path conforming to the regular language $L(R)$. The respective answer set $ans(Q^{\bullet\Xi}, K)$ is defined as

$$\{(a, B) \in V \times 2^V : a \xrightarrow{p} b \text{ for } b \in \Xi_B\ ,\ p\ \in L(R)\}.$$

Example. The following example queries refer to graph K, as defined in Figure 2. SRPQ $Q^{\bullet\forall} \leftarrow friendof$ requires that a node is connected to all nodes within a set. Thus, a partial answer set is $\{(n1, \{n2\}), (n2, \{n1, n3, n4, n5\})\} \subset ans(Q^{\bullet\forall}, K)$. The query $Q^{\bullet\exists<3} \leftarrow friendoffriendof?$ requires that the node is connected to less than three nodes within the set. Nodes are connected when there is a "friendof" path of length one or two. A partial answer set of this query is $\{(n1, \{n2, n5, n6, n8\}), (n2, \{n1, n8\})\} \subset ans(Q^{\bullet\exists<3}, K)$.

Definition 4 (SRPQ: Set – Node)**.** A set regular path query $Q^{\Xi\bullet} \leftarrow R$ describes a relation between a set and a single node, based on a regular expression R together with an quantifier Ξ. The quantifier defines how many nodes within the set must be connected to the single node by a path conforming to the regular language $L(R)$. The respective answer set $ans(Q^{\Xi\bullet}, K)$ is defined as

$$\{(A, b) \in 2^V \times V : a \xrightarrow{p} b \text{ for } a \in \Xi_A,\ p\ \in L(R)\}.$$

Example. The following example queries refer to graph K, as defined in Figure 2. Query $Q^{\exists\bullet} \leftarrow friendoffriendof$ defines that there must be at least one node in the set that is connected to the single node by a "friendof" path of length two. A partial answer set is $\{(\{n1, n2\}, n5), (\{n4\}, n1)\} \subset ans(Q^{\exists\bullet}, K)$. The second example query defined as $Q^{\exists>50\%\bullet} \leftarrow friendof$ requires that more than half of the nodes within the set are directly connected to the single node. A partial answer would be the set $\{(\{n1, n4, n7\}, n2), (\{n1\}, n2)\} \subset ans(Q^{\exists>50\%\bullet}, K)$.

Definition 5 (SRPQ: Set – Set)**.** A set regular path query $Q^{\Xi\Psi} \leftarrow R$ describes a relation between two sets, based on a regular expression R together with two quantifiers Ξ and Ψ. The quantifiers define how many nodes from within the "left" set must be connected to how many nodes from the "right" set by a path conforming to the regular language $L(R)$. The respective answer set $ans(Q^{\Xi\Psi}, K)$ is defined as

$$\{(A, B) \in 2^V \times 2^V : a \xrightarrow{p} b,\ a \in \Xi_A,\ b \in \Psi_B,\ p \in L(R)\}$$

Example. As before, the following example queries refer to graph K, as defined in Figure 2. Query $Q^{\forall\forall} \leftarrow friendof$ requires that all nodes within the first set are friends with all nodes in the second set, for which a partial query answer is $\{(\{n5, n6\}, \{n7, n8\}), (\{n1\}, \{n2\})\} \subset ans(Q^{\forall\forall}, K)$. Query $Q^{\exists,\exists>2} \leftarrow friendoffriendof?$ determines that there must be at least one element in the first set that is connected to more than two elements in the second set by a path of length one or two. Thus, a partial query answer would be the set $\{(\{n1, n5\}, \{n6, n7, n8\})\} \subset ans(Q^{\exists,\exists>2}, K)$.

Through introduction of set variables SRPQs extend RPQs in a similar way as *Monadic Second-Order Logic (MSOL)* extends *First-Order Logic (FO)*. Along with set variables *MSOL* introduces an atomic formula $t \in S$, where t is a first-order term and S is a set variable. Next, we show how this atomic formula can be expressed in SRPQs. Empty string $\perp$ is a valid regular expression. However, it is never used in RPQs, as $\perp$ path does not convey any functional load. In SRPQs it does: the query $a \xrightarrow{\perp} B$ means that node a is an element of B, and $A \xrightarrow{\perp} B$ defines the subset relation.

In RPQs paths between nodes are specified over the input graph G. Specification of more advanced structural properties, like explicitly connected groups, requires greater flexibility on this matter. Therefore, we define the following query type.

Definition 6 (SRPQ Closure)**.** For the above defined query flavors $Q^{\Xi\Psi}$ and $Q^{\Xi\bullet}$ the closures $\bar{Q}^{\Xi\Psi}$ and $\bar{Q}^{\Xi\bullet}$ further restrict the answer set by requiring that paths connecting A and B/b stay within A, i.e.,

$$ans(\bar{Q}^{\Xi\bullet}, K) = \{(A, b) \in 2^V \times V :$$
$$a \xrightarrow{p} b \text{ for } a \in \Xi_A,\ p \in L(R) \text{ and}$$
$$\forall i \in \{1, \ldots, n-1\}\ a \xrightarrow{p_1 \cdots p_i} c \implies c \in A, \text{ for } p = p_1 \ldots p_n\}$$

and

$$ans(\bar{Q}^{\Xi\Psi}, K) = \{(A, B) \in 2^V \times 2^V :$$
$$a \xrightarrow{p} b \text{ for } a \in \Xi_A,\ b \in \Psi_B,\ p \in L(R) \text{ and}$$
$$\forall i \in \{1, \ldots, n-1\}\ a \xrightarrow{p_1 \cdots p_i} c \implies c \in A, \text{ for } p = p_1 \ldots p_n\}.$$

Example. The following example queries again refer to graph K, as defined in Figure 2. Query $\bar{Q}^{\exists\bullet} \leftarrow friendoffriendof$ defines that there must be at least one node in the set that is connected to the single node by a "friendof" path of length two. A partial answer set is $\{(\{n1, n2\}, n5)\} \subset ans(\bar{Q}^{\exists\bullet}, K)$, but $(\{n1\}, n5) \notin ans(\bar{Q}^{\exists\bullet}, K)$.

Definition 7 (Set Size Query). A set size query $Q^{|\cdot|} \leftarrow (from, to)$ describes an unary relation. The variables $from$, $to \in \mathbb{N}$, with $from \leqslant to$, define minimum and maximum allowed set sizes. The respective answer set is defined as the sets of subsets of sizes from $from$ to to.

$$ans(Q^{|\cdot|}, K) = \{A \in 2^V : |A| \in \{from, \ldots, to\}\}$$

Example. The following example queries refer to graph K, as defined in Figure 2. Query $Q^{|\cdot|} \leftarrow \{2, 3\}$ requires that all sets have size 2 or 3. A partial query answer is $\{\{n1, n2\}, \{n1, n3\}, \{n1, n2, n3\}\} \subset ans(Q^{|\cdot|}, K)$.

Definition 8 (Conjunctive Regular Set Path Query). A conjunctive regular set path query (CSRPQ) has the form

$$Q^S(x_1, ..., x_n) \leftarrow$$
$$\tilde{y}_1[(R_1)^{\Psi_1^1 \Psi_2^1}]y_2 \wedge ... \wedge \tilde{y}_{2m-1}[(R_m)^{\Psi_1^m \Psi_2^m}]y_{2m}$$
$$\wedge \; Z_1[f_1, t_1] \wedge ... \wedge Z_l[f_l, t_l]$$

where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are either node or set variables. $Z = \{Z_1, ..., Z_l\}$ represents all set variables among y_i, i.e., there is no set variable y_k such that $y_k \notin Z_i$. The variables x_i are among y_i. The $\sim$ symbol may be either empty or $-$; the latter case is only possible if y_i is a set and defines a SRPQ closure. Each of the Ψ is either a quantifier, or $\bullet$. Each R_i is a regular expression. The answer set $ans(Q^S, K)$ for a CSRPQ is the set of tuples $(v_1, ..., v_n)$ of nodes and sets of nodes in K such that there is a total mapping σ to nodes and sets of nodes with $\sigma(x_i) = d_i$ for every distinguished variable, and $(\sigma(y_i), \sigma(y_{i+1})$ in the set of the answer set of the respective query type; i.e., if y_{2i-1} and y_{2i} represent both nodes, then $y_{2i-1}R_iy_{2i}$ represents Q^R. If y_{2i-1} is node and y_{2i} is set, then $Q^{\bullet\Xi}$. If y_{2i-1} is set and y_{2i} is node, then $Q^{\Xi\bullet}$. If y_{2i-1} is set and y_{2i} is set, then $Q^{\Xi\Psi}$. And finally, $Z_i[f_i, t_i]$ denotes set size queries.

6. CSRPQ EXPRESSIVENESS

In this section we demonstrate the expressiveness of CSRPQs via formal specification of the use cases from the motivating scenario. For this purpose we use the formal notation defined in the previous section. Also, we exemplify ease of CSRPQs integration into CRPQ-based languages by showing how the same queries could be implemented in CSRPQ-enhanced version of SPARQL 1.1.

6.1 Closeness Centrality and Connectedness

Goal. Given the three predefined organizational groups `Team1`, `Team2`, and `Team3`, we need to assemble an explicitly connected team of three to five members with a maximum diameter of two. The team should have a connection to every group.

Formal specification. In knowledge graphs organizational groups are usually represented as single nodes, e.g., `Sales Department` or `Human Resources Department`, and affiliation of a person to a group is represented as a connection between the corresponding nodes. We use the same approach and assume there are three predefined nodes: `Team1`, `Team2`, and `Team3`. Affiliation of a person to a team is represented with the *inteam* edge, and social connection is represented with the *knows* edge in the semantic graph. Question mark applied to an atom, e.g., *knows?*, in regular expression specifies that the edge is optional.

$$\begin{aligned} Q^S(A) \;\; \leftarrow \;\; & A\,[(knows \cdot inteam)^{\exists}]\mathbf{team1} \\ \wedge \;\; & A\,[(knows \cdot inteam)^{\exists}]\mathbf{team2} \\ \wedge \;\; & A\,[(knows \cdot inteam)^{\exists}]\mathbf{team3} \\ \wedge \;\; & \bar{A}\,[(knows \cdot knows?)^{\forall\forall}]A \wedge A[3,5] \end{aligned}$$

SPARQL. In the formal notation we used capital letters to define variables representing groups. In SPARQL we use two question marks (??) for this purpose in analogy with single question marks (?) that precede node variables. **ALL** and **SOME** keywords are used to denote universal and existential quantification respectively, and **CLOSURE** keyword denotes a query closure. To conform to SPARQL semantics we put group size operator in the **FILTER** clause.

```
SELECT ??A
WHERE {
 SOME ??A knows/inteam 'Team1'.
 SOME ??A knows/inteam 'Team2'.
 SOME ??A knows/inteam 'Team3'.
 ALL CLOSURE(??A) knows{1,2} ALL ??A .
 FILTER ( ??A{3,5} ) }
```

Expressiveness. This example shows how explicitly connected teams can be defined with CSRPQs. Note, that implicitly connected teams can be defined by removing the query closure.

6.2 Cluster

Goal. Find a socially coherent group of people of size 10, which exhibits property of 2-plex.

Formal specification. In contrast to the previous example, for the sake of simplicity we omit specification of neighborhoods. According to the definition of 2-plex, we need to find a team, where every team member has at least $8 = 10 - 2$ connections to other team members.

SPARQL. In this query parameterized version of **SOME** keyword denotes $FO(COUNT)$ quantification. This goes in line with previous example, where **SOME** keyword without parameters denotes existential quantification.

$$\begin{aligned} & Q^S(A) \\ \leftarrow \;\; & A\,[(friendof)^{\forall,\exists\geqslant 8}]A \\ \wedge \;\; & A[10,10] \end{aligned}$$

```
SELECT ??A
WHERE {
 ALL ??A knows SOME(>=8) ??A.
 FILTER ( ??A{10,10} ) }
```

Expressiveness. This example shows succinctness of CSRPQs with respect to definition of structures with relaxed coherence.

6.3 Independent Teams

Goal. Find two independent teams of experts, i.e., no inter-team social connections exist.

Formal specification. Here we specify neither any search space, nor structural constraints for the teams A and B. The interesting peculiarity is that negation expressed in the query goal (i.e., no inter-team social connections exist) can be expressed with countable quantifiers.

SPARQL. This query does not introduce any additional extensions to SPARQL and simply reuses keywords already defined in the previous examples.

$$\begin{array}{ll} Q^S(A) & \\ \leftarrow & A\ [(knows)^{\forall,\exists=0}]B \\ \wedge & B\ [(knows)^{\forall,\exists=0}]A \\ \wedge & A[3,5] \wedge B[3,5] \end{array}$$

```
SELECT ??A ??B
WHERE {
 ALL ??A knows SOME(=0) ??B
 ALL ??B knows SOME(=0) ??A
 FILTER ( ??A{3,5}, ??B{3,5})
 }
```

Expressiveness. This example shows expressivity of CSRPQs with respect to definition of multiple groups. Also, it shows how non-existence can be expressed with extended quantifiers.

6.4 Liaison

Goal. Find a liaison between `Operational Department` and a team of three external consultants **C** in order to foster adoption of new practices. The liaison should know at least two out of the three consultants, and at least 70% of `Operational Department` members.

Formal specification. In this example, in contrast to the previous ones, it is necessary to find an actor, not a group. The group **T** used in the example should represent the whole department. Therefore, in order to avoid selection of subgroups we fix size of **T** to the known size of `Operational Department`, which is retrieved as an additional query expressed in canonical $CRPQ^{agg}$ (CRPQs extended with aggregation).

SPARQL. This example shows that notion of groups also allows specification of predefined constant groups, i.e., group of four consultants. For this purpose we use syntax reminiscent to many mainstream programming languages. In order to retrieve size of `Operational Department` we use a nested SPARQL query.

$$C := \{\textbf{jill}, \textbf{jade}, \textbf{jack}\}$$

$$\begin{array}{ll} Q^{count}(t) & \\ \leftarrow & t\ [(inteam)]\textbf{optdept} \end{array}$$

$$\begin{array}{ll} Q^S(a) & \\ \leftarrow & a\ [(knows)^{>0.7}]T \\ \wedge & T\ [(inteam)^{\forall}]\textbf{optdept} \\ \wedge & a\ [(knows)^{>2}]C \\ \wedge & T[Q^{count}, Q^{count}] \end{array}$$

```
SELECT ?a
WHERE {
 ?a knows SOME(>70%) ??T.
 ALL ??T inteam 'OptDept'.
 ?a knows SOME(>2)
     {'jill', 'jade', 'jack'}.
 {
  SELECT COUNT(?t) as ?c
  WHERE {?t inteam 'OptDept'}
 }
 FILTER ( ??T{?c,?c} )
}
```

Expressiveness. This example shows how CSRPQs can define selection of single nodes based on relations to groups.

6.5 Structural Equivalence

Goal. In order to replace a coordinator *John* we need to find a structurally equivalent person, i.e., a person that has the same social and organizational connections as *John*. Sometimes, however, it may not be possible to find a single person fulfilling this requirement, and a group of persons may be needed.

Formal specification. This example is somewhat similar to the previous one as we want to consider the whole neighborhood of a predefined node, e.g., group (T) in both examples. Therefore, for the sake of clarity, we omitted the aggregation query as it can be, with a small adjustment, reused from the previous example.

SPARQL. Again, to keep the definition succinct, in SPARQL implementation we also omitted the nested aggregation query.

$$\begin{array}{ll} Q^S(EQ) & \\ \leftarrow & \textbf{john}\ [(workswith)^{\forall}]T \\ \wedge & EQ\ [(workswith)^{\forall}]T \\ \wedge & EQ[1,3] \\ \wedge & T[Q^{count}, Q^{count}] \end{array}$$

```
SELECT ??EQ
WHERE {
 'john' workswith ALL ??T.
 SOME ??EQ workswith ALL ??T.
 FILTER ( ??EQ{1,3},
          ??T{?count,?count})}
```

Expressiveness. This query exemplifies an opposite approach to the one employed in the previous example. In the previous example we allowed to neglect certain connections in favor of the size of the resulting group (single node), while in this example we relax the constraint on the resulting group size in order to preserve all connections.

7. COMPLEXITY OF QUERY EVALUATION

In this section we discuss complexity of CSPRQs evaluation, optimization techniques enabled by the design of CSRPQs, and propose an algorithm for the purpose of query evaluation. Greater expressiveness always comes at a price, and CSRPQ extension is no exception here. The problem of SRPQ evaluation contains a variant of a subset selection problem, e.g., find a subgraph connected to a node a, incurring thus high complexity. In the next two subsections we discuss upper bounds for the problem of CSRPQ evaluation.

7.1 Data Complexity

We fix query Q over a finite alphabet Σ, and K to be the maximum of upper bounds for all the groups specified in the query. Now, to show data complexity of CSRPQs we consider the following decision problem:

PROBLEM:	CSRPQS-EVAL(Q)
INPUT:	A Σ-labeled *db*-graph G of size N, a tuple of nodes $\overline{v}$, and a tuple of groups $\overline{V}$
QUESTION:	Does $(\overline{v}, \overline{V})$ belong to $Q(G)$

Since number of groups is fixed in the query, let us denote it as some constant C. Similarly, let us define c as a number of nodes in the query. In order to answer the problem we need to traverse N^{c+K*C} possible solutions. Since c, C, and K are all fixed, we can see that data complexity of CSRPQ evaluation is in *PTIME*.

7.2 Query Complexity

We turn now to query complexity, i.e., where a query Q is an input parameter, while Σ-labeled graph G is fixed.

PROBLEM:	CSRPQS-EVAL(G)
INPUT:	A CSRPQ query Q over Σ, a tuple of nodes $\overline{v}$, and a tuple of groups $\overline{V}$
QUESTION:	Does $(\overline{v}, \overline{V})$ belong to $Q(G)$

Let us define $k_1, ..., k_l$ as upper bounds of groups specified in Q, where l is number of groups. Let us also define c as number of nodes in the query graph. In order to answer the problem we need to traverse $N^{c+k_1+...+k_l}$ possible solutions. Given that N is fixed we can see that query complexity of CSRPQ evaluation is in *EXPTIME*.

7.3 The CSRPQ Evaluation Algorithm

The key observation behind CSRPQ evaluation is the mandatory presence of constraints on the group size. A candidate for inclusion in a set should not only preserve structural characteristics of the set, but also enable other potential candidates to be added later in order to satisfy the set size constraints. This approach enables for efficient search space pruning techniques. For example, a node with degree 5 cannot be selected as a candidate for a clique of size 10. Without the loss of generality we can consider CSRPQs with any number of predefined node/set constants and only one set variable specified. Such an CSRPQ technically can be represented as a composite filter $F(G, S, K)$ that selects potential candidates from the search space S for inclusion in assembled so far set G with target set size upper bound K. Given queries $Q^{\Xi\bullet}_{const}$, where the left operand is the single set variable and the right operand is a constant node $\mathbf{c}$, and queries $Q^{\Xi\Psi}_{self}$, where both operands are the single set variable itself, let us consider several examples of filters $F(G, S, K)$:

- $Q^{\exists>5\bullet}_{const} \leftarrow R$: $F_Q(G, S, K) := f(s) \rightarrow |G^*| > 5 \vee K > 5 - |G^*| \vee s \overset{p\in L(R)}{\rightarrow} \mathbf{c}$, where $G_* = \{g^* \in G | g^* \overset{p\in L(R)}{\rightarrow} \mathbf{c}\}$.
- $Q^{\forall\forall}_{self} \leftarrow R$. $F_Q(G, S, K) := f(s) \rightarrow \forall g \in G : g \overset{p\in L(R)}{\rightarrow} s \wedge s \overset{p\in L(R)}{\rightarrow} g \wedge d(s) \geqslant K - |G|$, where $d(s)$ denotes an out degree of a node s.
- $Q^{\forall\exists\leqslant m}_{self} \leftarrow R$. $F_Q(G, S, K) := f(s) \rightarrow \forall g \in G : (d_G(g) \geqslant \min\{m, |G| - m + 1\} \vee g \overset{p\in L(R)}{\rightarrow} s) \wedge d_G(s) \geqslant \min\{m, |G| - m + 1\}$, where $d_G(s)$ denotes an out degree of a node s towards elements of a set G.
- $\bar{Q}^{\forall\forall}_{self} \leftarrow R$. $F_Q(G, S, K) := f(s) \rightarrow \exists p \in L(R), |p| = 1, \forall g \in G, \exists u \in L(R) : pu \in L(R) \wedge s \overset{pu}{\rightarrow} g$, where for each u there exists a physical path corresponding to u that resides within G.

Algorithm 1 sketches basic inductive routine for CSRPQ evaluation. Given a set G, we say that it does not violate CSRPQ Q, if it is possible to add nodes to G, such that all SRPQs in Q are satisfied including the set size query. The algorithm takes set G of size L, and tries to build a set G^* of size $L + 1$, $G \subset G^*$. The core of the algorithm, filter function F_Q represents conditions of CSRPQ Q violation. F_Q returns a set of nodes from the search space, such that the set G^*, induced by adding any of them to G, does not violate Q. Then, the algorithm traverses through each of the nodes returned by F_Q outputting all solutions.

Algorithm 1: CSRPQ evaluation $CSRPQ_{EVAL}$

```
Input: Search space $S$, group size $K$;
program stack $G$, list of already processed nodes $P$;
CSRPQ $Q$;
Result: List of graphs
if $K = 0$ then
    output $G$
end
/* initialize the neighborhood function
    */
$f_N(s) \leftarrow F_Q(S, G, K)$ ;
/* filter the neighborhood                        */
$N \leftarrow \{s_i \in S/P | f_N(s_i)\}$;
for $neighbor \in N$ do
    $G^* \leftarrow G \cup \{neighbor\}$;
    $CSRPQ_{EVAL}(S, K - 1, G^*, P \cup \{neighbor\}, Q)$;
    $P \leftarrow P \cup \{neighbor\}$;
end
```

As a basis for $CSRPQ_{EVAL}$ algorithm we took the algorithm proposed by Chiba and Nishizeki [12], which has been designed for search of complete subgraphs (cliques) of predefined size k. They have shown that the algorithm has time complexity $O(ka(G)^{k-2}m)$, where G is an input graph, m is number of edges and $a(G)$ is *arboricity* of the graph. The arboricity of a graph is a measure of how dense the graph is: graphs with many edges have high arboricity, and graphs with high arboricity must have a dense subgraph.

As it can be seen, performance of $CSRPQ_{EVAL}$ heavily depends on the efficient reduction of the search space at each step. Therefore, given a query graph Q, graphs containing high number of subgraphs that satisfy Q, or are close to Q within small edit distance, result in longer execution times.

$CSRPQ_{EVAL}$ algorithm can be easily converted to a greedy algorithm. Indeed, instead of traversing in undefined order all nodes in N, it can first choose the fittest one in order to come up with a first solution faster. Randomization approaches are also possible to balance query evaluation time. Greedy and randomization approaches are crucial when it is enough to return only one result. Also, similarly to the subset selection problem, our problem exhibits natural data parallelism, making thus applicable parallel data processing techniques.

So far we discussed only evaluation of queries defining groups with structural constraints, i.e., constraints that specify relations within the group. Absence of such constraints may introduce high space complexity incurred by large result set. For example, a CSRPQ query specifying selection of a group F without any predefined structure from a finite search space S could yield $|P_{\leqslant|F|}(S)|$ possible results, where $|P_{\leqslant|F|}(S)|$ is a power set of S of cardinality $|F|$. The result set, however, can be represented as a single element describing a commutative monoid corresponding to the power set. This approach would discard time required to enumerate possible answers as well as minimize space needed for the output.

8. EVALUATION

Query evaluation performance is one of the key attributes of database query language success. In Section 7 we have shown that data complexity of CSRPQ evaluation is in *PTIME*. Since data complexity of CRPQ evaluation is in *NLogspace* and $NLOGSPACE \subset PTIME$ it is hard to reason about actual query evaluation times. Therefore, we quantify the evaluation time of CSRPQ queries by conducting experiments. In the remainder of this section we compare the time of CSRPQ evaluation based on our approach (Algorithm 1) with a baseline. Also, we investigate key aspects influencing CSRPQ evaluation time.

8.1 Evaluation Setup

Our evaluation is based on two real-world data sets from the social networks Slashdot[2] and Friendster[3]. Both data set graphs consist of anonymous users represented as nodes and social relationships represented as edges. Friendster is a social gaming web site and Slashdot is a community-enabled news website. Table 8.1 depicts main structural characteristics of the two data set graphs. Due to the large size of the Friendster graph, which exceeded our evaluation computer's physical memory constraints, we extracted a subgraph consisting of around 11 million nodes out of the full graph, which contains almost 120 million nodes.

	Friendster	Slashdot
#Nodes	10,999,986	77,359
#Edges	297,395,506	905,468
Avg Node Degree	27.036	11.705
Node Degree Dev	94.939	36.844
Max Node Degree	4,014	2,508
Avg Clustering	0.169	0.0555
Clustering Dev	0.309	0.382

Table 1: Main characteristics

The queries chosen for evaluation try to cover a large spectrum of CSRPQs by varying structural constraints (e.g., how tightly connected the group should be), the search space size (e.g., in what neighborhood the group should be searched for), and the size of the group defined by the query. Query 1 represents a query with strict structural constraints (complete subgraph), while Query 2 looks for less strictly defined groups (k-plex). For each query the group size is provided as input parameter, while the search space size is varied by both input parameter *rand* and its neighborhood depth.

Query 1 selects a complete graph A of size n in the neighborhood of input node **rand**. The neighborhood includes friends of the predefined node, as well as friends of friends.

$$\begin{aligned} Q_1(A) \quad \leftarrow \quad & \mathbf{rand}[(friendOf friendOf?)^{\forall}]A \\ \wedge \quad & A[(friendOf)^{\forall\forall}]A \\ \wedge \quad & \mathbf{rand}[(friendOf)^{\exists}]A \wedge A[\mathbf{n},\mathbf{n}] \end{aligned}$$

Query 2 selects a 2-plex B from the neighborhood of input node **rand**. Here numeral n not only specifies group size, but also its structural characteristics.

$$\begin{aligned} Q_2(B) \quad \leftarrow \quad & \mathbf{rand}[(\perp)^{\exists}]B \\ \wedge \quad & B[(friendOf)^{\forall\exists(\geqslant \mathbf{n}-2)}]B \wedge B[\mathbf{n},\mathbf{n}] \end{aligned}$$

Strictness of structural constraints influences the number of results: Query 1 above describes more rigid structural constraints than Query 2, e.g., there might be no cliques in a graph, but many 2-plexes. For the input parameter n, which defines the group size, we choose values 5, 10, 15, and 20, as they seem to be reasonable for real life group selection tasks. For the input node *rand* we randomly chose evenly distributed non-isolated nodes, 200 nodes for Friendster and 100 nodes for Slashdot, to get informative, averaged results.

[2]http://snap.stanford.edu/data/soc-Slashdot0811.html
[3]http://archive.org/details/friendster-dataset-201107

All CSRPQ evaluation scenarios are implemented in Java 1.7.0 x64, and time values are measured using the standard API's *System.nanotime()* method. The system configuration used for the tests is: Intel Core i7 2840QM (2GHz, Sandy Bridge), 8GB DDR3 1333MHz RAM, Windows 7 x64. Even though Algorithm 1 is inherently parallel, we execute all experiment runs in a single thread in order to show the real running time.

The baseline algorithm is based on transforming CSRPQ to SPARQL queries and evaluating them using Apache Jena[4]. Only the pure SPARQL evaluation time is measured, without query transformation overhead. The transformation is necessary because SPARQL does not support queries for group patterns. However, it is possible to rewrite a fixed query expressing strict structural constraints (complete graph) as a conjunction of terms, i.e., as a series of CRPQs. Relaxation of structural constraints (k-plex) can be expressed as a disjunction of CRPQs, where each single term represents a possible group pattern. This can be seen as a demonstration of the problem CSRPQ tries to solve, because with growing complexity and size of the queried groups, length and complexity of the SPARQL query increases exponentially. This is the reason the group size in baseline evaluation was set to 5 only, as bigger group sizes resulted in immersive queries impossible to be handled by the SPARQL evaluation engine.

The code we use for evaluation along with baseline query examples is publicly available[5], and can be compiled and executed to verify the published results.

8.2 Comparison to Baseline

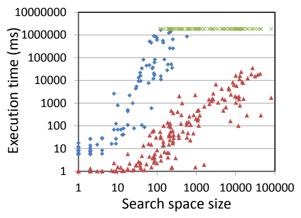
Algorithm 1 is compared to the baseline algorithm by executing Query 1 and Query 2 for different input nodes and data sets. Each single dot in Figure 3 represents the execution on particular query instance, i.e., with fixed input node. The horizontal axis reflects the size of the search space that is relevant for executing the defined queries, i.e., accumulation of node degrees of input nodes' direct neighbors. Query evaluation times for our approach are shown as red triangles, valid query evaluations for the baseline are depicted as blue diamonds, and green '×' stand for instances, for which the baseline failed to deliver the result within 30 minutes. In such timeout cases we discontinued query evaluation.

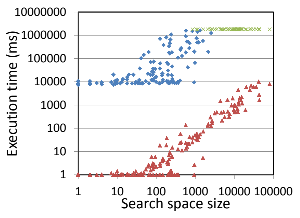
Figure 3 demonstrates that our approach outperforms the baseline in orders of magnitude. There are many timeouts for the baseline algorithm; in contrast, our algorithm is always able to return the query result without timeout violations. Thus, CSRPQ extension proves to be not only more expressive, but obviously also enables efficient search space pruning and traversal that marks already visited nodes, as discussed in the previous section.

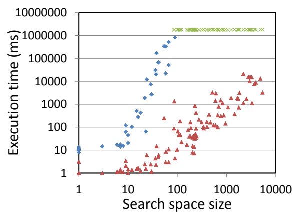
8.3 Influences on Evaluation Time

In this section we illustrate how varying groups sizes influence the evaluation time of Algorithm 1, together with information about the corresponding sizes of result sets. In particular, Figure 4 depicts dependencies between the queried group size and resulting average query evaluation times ((a) for Query 1 and (c) for Query 2), and between the group size and average number of results ((b) for Query 1 and (d) for Query 2).

[4]http://jena.apache.org/
[5]http://www.infosys.tuwien.ac.at/prototypes/csrpq


(a) Friendster Query 1


(b) Friendster Query 2


(c) Slashdot Query 1

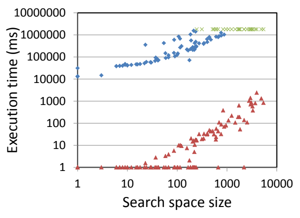

(d) Slashdot Query 2

Figure 3: Performance of CSRPQ evaluation algorithm versus CRPQ SPARQL-based implementation. Each dot denotes a single query evaluation. ◆ and × symbols stand for regular and timed-out CRPQ baseline evaluations respectively, ▲ symbols denote CSRPQ-based evaluations.

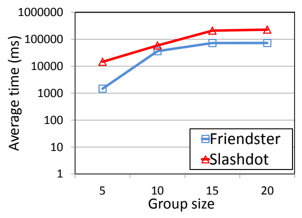
The evaluation results show that the group size exponentially influences query evaluation time only up to some limit. We can see that group sizes greater than 10 do not result in substantial increase of evaluation time. As per design of our CSRPQ evaluation algorithm, further increasing of the group size will eventually lead to zero evaluation time as it will not be possible to find a single node with high enough degree, so that query evaluation can be stopped immediately with empty result set.

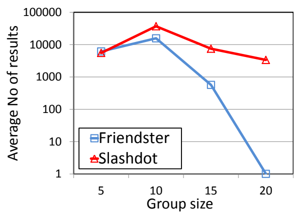
As for the amount of results, we notice that a big number of results guarantees high query evaluation time. But vice versa does not hold, i.e., absence of results does not guarantee fast evaluation. This confirms our observation that the key aspect of query evaluation time is the similarity of the underlying search space graph structure to the query graph structure. A big number of subgraphs structurally similar to the query graph, i.e., with small edit distances, would result in increased evaluation times.

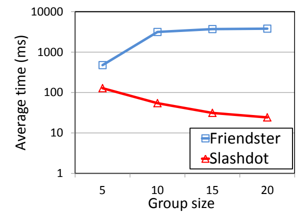
Overall, the evaluation results prove the feasibility of Algorithm 1 for solving CSRPQs, even for large, real world data sets. It is much more efficient compared to solving a corresponding SPARQL query with a state of the art evaluation engine. Evaluation times of our algorithm depends on the query and the dataset, and how effectively pruning can be applied.

9. CONCLUSION AND FUTURE WORK

Being able to query groups based on structural properties and relations to other entities is an important asset for languages working on graph-structured data. However, to the best of our knowledge, there is no query language backed by an evaluation engine able to express such queries. We present CSRPQs, which extend CRPQs with such group selection capabilities. Being more expressive, CSRPQs also enable efficient evaluation techniques, outperforming thus


(a) Average runtime for Query 1.


(b) Average number of results for Query 1.


(c) Average runtime for Query 2.

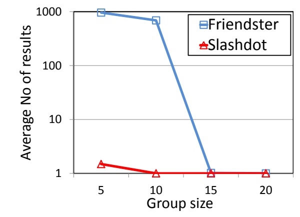

(d) Average number of results for Query 2.

Figure 4: Influences of queried group size and size of result set on the evaluation time.

CRPQ-based implementations. Experiments show that our CSRPQ-based implementation is capable of finding groups in orders of magnitude faster than the state-of-the-art CRPQ-based SPARQL query evaluation engine.

As discussed in Section 3, there exist many algorithms for selecting social formations exhibiting specific structural characteristics (e.g., clique, k-plex). Being more restrictive, these special-purpose algorithms might be more efficient than the general-purpose query evaluation algorithm at hand. By operating with the notion of a set, however, CSRPQs enable the evaluation algorithm to recognize special cases and fall back to the special-purpose algorithms, which is to be considered in the future work. Also, application of approximation approaches to CSRPQs is an interesting topic to investigate.

10. REFERENCES

[1] Sparql 1.1 query language - w3c working draft 05 january 2012. 2012.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[3] S. Amer-Yahia, L. V. S. Lakshmanan, and C. Yu. Socialscope: Enabling information discovery on social content sites. *CoRR*, abs/0909.2058, 2009.

[4] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, Feb. 2008.

[5] P. Barcelo, C. Hurtado, L. Libkin, and P. Wood. Expressive languages for path queries over graph-structured data. In *29th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 3–14, New York, NY, USA, 2010. ACM.

[6] V. Batagelj and A. Mrvar. Pajek - analysis and visualization of large networks. In *Graph Drawing*,

volume 2265 of *Lecture Notes in Computer Science*, pages 8–11. Springer Berlin / Heidelberg, 2002.

[7] H. Blau, N. Immerman, and D. Jensen. A visual language for querying and updating graphs. Technical Report 2002-037, Department of Computer Science, University of Massachusetts, 2002.

[8] S. Borgatti, M. Everett, and L. Freeman. UCINET 6 For Windows: Software for Social Network Analysis, 2002.

[9] S. P. Borgatti and M. G. Everett. Two algorithms for computing regular equivalence. *Social Networks*, 15(4):361 – 376, 1993.

[10] P. Buneman, M. Fernandez, and D. Suciu. Unql: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal*, 9(1):76–110, Mar. 2000.

[11] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185. Morgan Kaufmann, 2000.

[12] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, Feb. 1985.

[13] M. P. Consens and A. O. Mendelzon. Expressing structural hypertext queries in graphlog. In *Proceedings of the second annual ACM conference on Hypertext*, HYPERTEXT '89, pages 269–292, New York, NY, USA, 1989. ACM.

[14] M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[15] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. *SIGMOD Rec.*, 16(3):323–330, Dec. 1987.

[16] A. Dries, S. Nijssen, and L. De Raedt. A query language for analyzing networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 485–494, New York, NY, USA, 2009. ACM.

[17] S. Dustdar and K. Bhattacharya. The social compute unit. *Internet Computing, IEEE*, 15(3):64 –69, may-june 2011.

[18] G. Erétéo, M. Buffa, F. Gandon, and O. Corby. Analysis of a real online social network using semantic web frameworks. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 180–195, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 39 –50, apr 2011.

[20] M. Fernández, D. Florescu, A. Levy, and D. Suciu. Declarative specification of web sites with strudel. *The VLDB Journal*, 9(1):38–55, Mar. 2000.

[21] G. Grahne and A. Thomo. Regular path queries under approximate semantics. *Annals of Mathematics and Artificial Intelligence*, 46(1-2):165–190, Feb. 2006.

[22] H. He and A. K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 405–418, New York, NY, USA, 2008. ACM.

[23] C. A. Hurtado, A. Poulovassilis, and P. T. Wood. Ranking approximate answers to semantic web queries. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC 2009 Heraklion, pages 263–277, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] D. Jensen and J. Neville. Data mining in social networks. In *National Academy of Sciences Symposium on Dynamic Social Network Analysis*, 2002.

[25] Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 40–51, New York, NY, USA, 2001. ACM.

[26] G. Kasneci, F. M. Suchanek, G. Ifrim, S. Elbassuoni, M. Ramanath, and G. Weikum. Naga: harvesting, searching and ranking knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1285–1288, New York, NY, USA, 2008. ACM.

[27] G. Klyne and J. J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium, Feb. 2004.

[28] U. Leser. A query language for biological networks. *Bioinformatics*, 21(suppl 2):ii33–ii39.

[29] V. Liptchinsky, R. Khazankin, H. L. Truong, and S. Dustdar. A novel approach to modeling context-aware and social collaboration processes. In *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 565–580. Springer, 2012.

[30] M. S. Martín, C. Gutierrez, and P. T. Wood. Snql: A social networks query and transformation language. In *AMW*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[31] A. Mostowski. On a generalization of quantifiers. *Fundamenta Mathematicae*, 44:12–36, 1957.

[32] P. D. Oyer. Peopleware: key to success of information systems. *SIGCPR Comput. Pers.*, 5(2):2–6, Dec. 1974.

[33] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. In *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2006.

[34] M. S. M. Ramas, C. Gutierrez, and P. T. Wood. Snql: Social networks query language. Technical Report TR/DCC-2011-05, Departamento de Ciencias de la Computacion, Universidad de Chile, 2011.

[35] R. Ronen and O. Shmueli. Soql: A language for querying and creating data in social networks. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1595–1602, Washington, DC, USA, 2009. IEEE.

[36] J. P. Scott. *Social Network Analysis: A Handbook*. SAGE Publications, 2 edition, Jan. 2000.

[37] N. Team. The neo4j manual. 2012.

[38] P. T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, Apr. 2012.