Gagg: A Graph Aggregation Operator

Fadi Maali(✉), Stéphane Campinas, and Stefan Decker

Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland
{fadi.maali,stephane.campinas,stefan.Decker}@insight-centre.org

Abstract. Graph aggregation is an important operation when studying graphs and has been applied in many fields. The heterogeneity, fine-granularity and semantic richness of RDF graphs introduce unique requirements when aggregating the data. In this work, we propose *Gagg*, an RDF graph aggregation operator that is both expressive and flexible. We provide a formal definition of Gagg on top of SPARQL Algebra, define its operational semantics and describe an algorithm to answer graph aggregation queries. Our evaluation results show significant improvements in performance compared to plain-SPARQL graph aggregation.

1 Introduction

With the increasing adoption of graph data in various domains, the importance of graph measures and algorithms is growing. Graph traversal, centrality measures, and graph aggregation are being used to analyse social [31], transportation [2] and biological networks [23,27]. This paper focuses on graph aggregation.

Graph aggregation condenses a large graph into a structurally similar but smaller graph by collapsing vertices and edges. Graph aggregation was applied to the Web graph to group Web pages by their domains in order to efficiently compute PageRank scores [6]. Similarly, [11] proposed ranking RDF datasets by aggregating their resources. In biological network studies, example usages of graph aggregation include enhancing data visualisation [19] and studying transcriptional regulatory networks [12]. Graph aggregation is also used to provide business intelligence on top of graph data [5,8,28,33]. Furthermore, many measures in social network analysis [14] and in bibliometrics [15,17] are also based on aggregating the underlying graphs.

RDF, the data model underlying the Semantic Web, is a graph data model that is used in bio-informatics[1], social networks [24], bibliography [13,26], etc. Therefore, graph aggregation is one of the tools used to analyse RDF data. In addition to the examples mentioned before, aggregating RDF graphs has been also used to induce schemas [7,20,21], produce descriptive statistics [1,3,22] and build indices [18,25].

While there exists a number of systems to aggregate graph data [8,30,32,33], these systems do not provide the expressivity necessary to handle RDF data. The heterogeneity, fine-granularity and semantic richness of RDF graphs introduce

[1] http://www.ebi.ac.uk/about/news/press-releases/RDF-platform.

© Springer International Publishing Switzerland 2015
F. Gandon et al. (Eds.): ESWC 2015, LNCS 9088, pp. 491–504, 2015.
DOI: 10.1007/978-3-319-18818-8_30

unique requirements when aggregating the data. For instance, Fig. 2 shows an example bibliographic RDF data describing some papers and their authors. One might be interested in studying co-authorship structure between authors or at a more coarse-grained level, between organisations (Fig. 3a). Similarly, one might choose to aggregate the data to study the structure of citation between authors, organisations or conferences (Fig. 3c). Such expressivity cannot be achieved by existing tools, proposed mainly in the field of graph databases, because the dimensions of aggregation and the relationship (e.g., co-authorship) is not explicitly defined in the original data.

Furthermore, all graph aggregation operators have been defined as separate operators. Hence, these operators cannot make use of the expressivity and optimisation techniques already built in existing data models such as relational or SPARQL algebra. Moreover, this necessitates transforming the data and loading it into different systems.

In this paper, we define *Gagg*, an RDF graph aggregation operator for RDF data that is both expressive and flexible. We provide a formal definition of Gagg based on existing SPARQL operators (Sect. 3) and devise an algorithm for efficient evaluation of Gagg expressions (Sect. 4). Furthermore, we demonstrate the expressivity and efficiency of Gagg in our evaluation (Sect. 5).

Aggregation of RDF graphs can be achieved using existing SPARQL 1.1 [16] operators. An aggregated graph requires a complicated single SPARQL query (a combination of sub-queries, CONSTRUCT and GROUP BY operators) or a series of SPARQL queries to aggregate nodes and edges (as done in [22] for instance). Such queries become complicated and verbose and therefore hard to write, debug and optimise. Having graph aggregation as a first-class operator simplifies query writing and optimisation. Moreover, our evaluation results show that Gagg can run up to orders of magnitudes faster than a monolithic SPARQL query and about 3 times faster than a series of fine tuned queries.

2 Related Work

On-Line Analytical Processing [9] (OLAP) has been first proposed as a way for people to analyse multi-dimensional data. Several works aim at analysing graph data using the OLAP paradigm. The Graph Cube [33] paper considers only simple graphs, while we target more complex graphs that are possible with the RDF graph model. The work in [8] proposes the aggregation of graph which changes the topology of the graph. However, we allow a more flexible definition of the aggregation dimensions in our approach. In this work, we introduce the graph aggregation as a graph operator that is anchored into graph algebra, allowing the graph aggregation to be part of a wider graph analysis flow.

The challenge of graph analysis has been studied within the Semantic Web community as well. The RDF Analytics [10] paper introduces an analytical schema over the data. This schema allows then to write analytical queries over the data, abstracting from the actual structure of the data. However, unlike Gagg this approach does not generate a graph as the analytical query output, thus preventing the use of the aggregated graph as the input of other graph operations.

The analysis of graph data under an aggregated form is investigated in several works [5,28] with an application oriented towards business logic.

The process of graph summarisation which represents a graph with a smaller graph that is homomorphic to the original graph is investigated in [7,30,32]. Such works apply a Gagg-like operation over the data. Therefore, the Gagg operator can be used to alleviate the cost of performing such operations.

3 Model

Gagg is defined in two-steps as shown in Fig. 1. Firstly, nodes and edges in the original graph G are grouped together in an intermediate graph that we call the *grouped graph*. Secondly, the grouped graph is reduced into an *aggregated graph*.

The first step is based on a set of dimensions that defines how nodes and edges are grouped together. For example, in Fig. 2 the resources `:a1` and `:a2` are grouped together when aggregating authors by the organisation they are member of. As we detail later, we use SPARQL operators as a flexible and powerful way to define grouping criteria.

The second step condenses the grouped graph into the final aggregated graph. This operation is done by what we call a *graph reduce function*. A template similar to that of SPARQL construct is used by the graph reduce function to structure the final results. In this template, typical aggregate functions such as `sum` and `count`, can be applied to the nodes and edges in the grouped graph. It is worth pointing out that the grouped graph is not an RDF graph as its nodes represent sets of resources, while the aggregated graph is an RDF graph that can be used as input for further processing.

$$\begin{tikzcd} G \arrow[r, "\texttt{Group}"] & \mathcal{G} \arrow[r, "\texttt{Reduce}"] & H \\ \text{Graph} & \text{Grouped Graph} & \text{Aggregated Graph} \end{tikzcd}$$

Fig. 1. Operational flow of the graph aggregation

3.1 Preliminaries

We present in this section the fundamental concepts used throughout the paper.

Definition 1 (Data Graph). *A* data graph G *defined over a set of terms* $\mathcal{T}$ *is a tuple* $G = \langle V, A, l_V \rangle$*, where* V *is a set of nodes,* $A \subseteq V \times \mathcal{T} \times V$ *is the set of labelled edges, and* $l_V : V \mapsto \mathcal{T}$ *is a node labelling function. The* l_V *function is an injection, meaning that each node has a unique label.*

We assume the existence of a set of variables $\mathcal{X}$ which symbols are not part of the set of labels $\mathcal{T}$, i.e., $\mathcal{X} \cap \mathcal{T} = \emptyset$. A query is expressed as a set of patterns that are matched against the graph data. A *triple pattern* is the atomic element for building a graph query, which matches an edge of the graph.

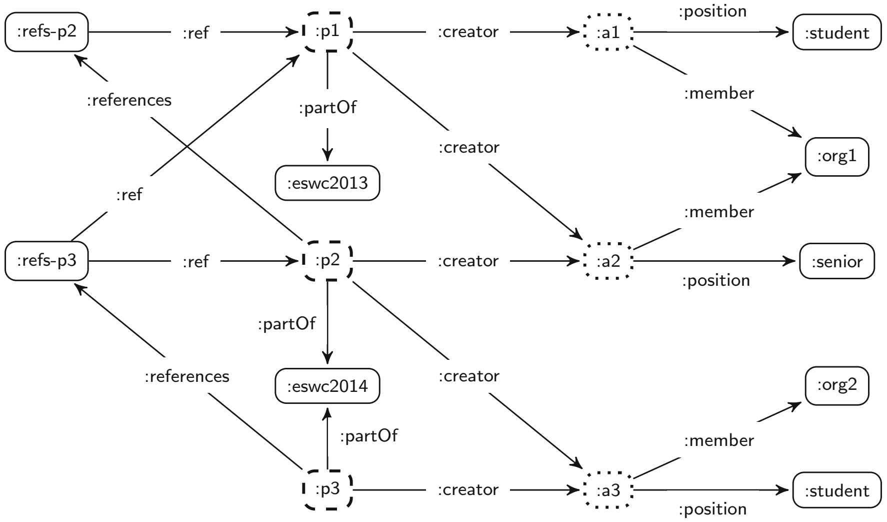


Fig. 2. Bibliographic network inspired from [29]. Dashed nodes represent papers, and dotted nodes the authors.

Definition 2 (Triple Pattern). *A triple pattern* (s, p, o) *is a triple where any of the three components can be either a variable or a term, i.e., the triplet* $(s, p, o) \in (\mathcal{T} \bigcup \mathcal{X}) \times (\mathcal{T} \bigcup \mathcal{X}) \times (\mathcal{T} \bigcup \mathcal{X})$.

Definition 3 (Basic Graph Pattern). *A basic graph pattern (BGP) is a set of triple patterns.*

Basic graph patterns in SPARQL represent conjunctive queries. From henceforward, we use the conjunctive query notation[2] $q(\bar{x}) := t_1, \cdots, t_n$ where $t_1, \cdots, t_n$ are triple patterns and therefore $\{t_1, \cdots, t_n\}$ is a BGP.

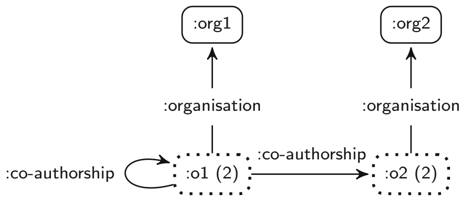
We denote by $Var(q)$ the set of variables occurring in the query q. The *query head* variables $\bar{x}$ are called *distinguished variables*, and are a subset of the variables occurring in $t_1, \cdots, t_n$, i.e., we have $\bar{x} \subseteq Var(q)$.

We denote with $q(G)$ the set of solutions of q on G. For the evaluation of a query q against a graph G, we refer the reader to the W3C Recommendation [16].

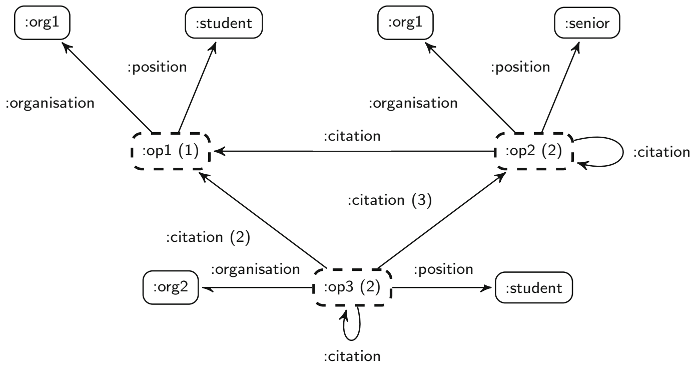
Definition 4 (Join Query). *Let* $q_1, \cdots, q_n$ *be basic graph patterns which non-distinguished variables are pairwise disjoints. We call the query* $q(\bar{x}) := q_1(\bar{x}_1) \wedge \cdots \wedge q_n(\bar{x}_n)$ *a join query of* $q_1, \cdots, q_n$*, where* $\bar{x} \subseteq \bar{x}_1 \cup \cdots \cup \bar{x}_n$.

A join query combines multiple BGPs and joins them based on their shared distinguished variables.

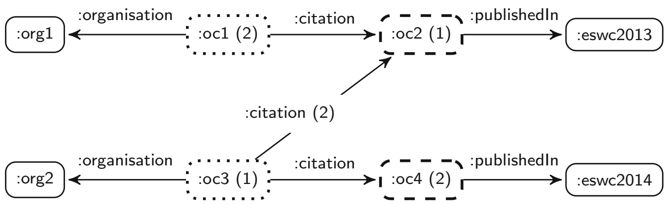
[2] We reuse here some of the notations and definitions from [10].



(a) Aggregation of people by the organisation, counting the number of papers authored by the organisation. The edges *:co-authorship* link organisations by the papers they authored.



(b) Aggregation of people by their organisation and position, counting the number of their papers. The edge *:citation* represents an author citing an other via a paper.

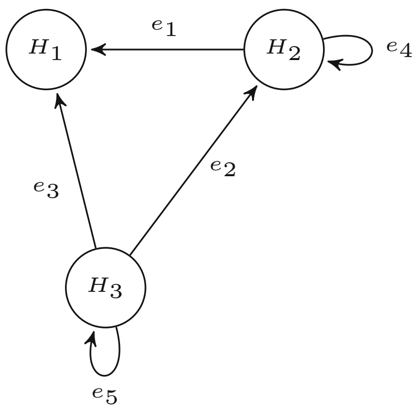


(c) Aggregation of people by their organisation and papers by their conference. We count the number of authors in an organisation, and the number of papers in a conference. The edge *:citation* represents a link from an organisation to a conference, where an author cited a paper from a particular conference.

Fig. 3. Summary graphs of the data in Fig. 2. The number within parenthesis report a count statistics of the aggregated data.

3.2 Graph Aggregation Operator

We build on the definitions introduced before to define a graph aggregation operator. In particular, dimensions used for aggregation, measures that are to be aggregated, and the relations between nodes in the graph are all expressed as queries. A *dimension* is a query with two distinguished variables $q(x, v)$ that defines how resources (bound to x) are grouped based on associated values



Dimensions	
$H_1 \rightarrow$	(`:org1`, `:student`)
$H_2 \rightarrow$	(`:org2`, `:student`)
$H_3 \rightarrow$	(`:org1`, `:senior`)

Measures	
$H_1 \rightarrow$	(:p1)
$H_2 \rightarrow$	(:p2, :p3)
$H_3 \rightarrow$	(:p2, :p3)
$e_1 \rightarrow$	(:p2)
$e_2 \rightarrow$	(:p2, :p3, :p3)
$e_3 \rightarrow$	(:p2, :p3)
$e_4 \rightarrow$	(:p2)
$e_5 \rightarrow$	(:p3)

Fig. 4. The grouped graph that corresponds to the aggregated graph in Fig. 3b. This graph aggregate authors in Fig. 2 by their organisation and position.

(bound to v). Similarly, a measure is also a query with two distinguished variables $q(x, m)$ that associates with each resource (bound to x) a value to be measured and later aggregated (bound to m). In Fig. 3b, $q(x, org) := (x, \texttt{:member}, org)$ is one of the dimensions used to group authors while $q(x, p) := (p, \texttt{:creator}, x)$ is the measure associated with each node as we are counting the papers written by each author.

A relation query is a query with four distinguished variables $q(x, p, y, m)$ that defines related resources (bound to x and y, respectively), labels the relation (via the value bound to p), and determines the measure associated with this relation (bound to m). In Fig. 3b, the relation query used is:

$$\begin{aligned} q(x, \texttt{:cite}, y, p1) := {} & (p1, \texttt{:creator}, x) \wedge \\ & (p2, \texttt{:creator}, y) \wedge \\ & (p1, \texttt{:references}, refs) \wedge \\ & (refs, \texttt{:ref}, p2) \end{aligned}$$

Notice that $p1$ is used as a measure since we are interested in counting the papers. Additionally, the relation is bound to the constant value `:cite` to give it a readable name[3].

Definition 5 (Grouped Graph). *A grouped graph is a graph* $(\mathcal{V}, \mathcal{A}, l_{\mathcal{V}})$ *with two associated functions: dimensions* : $\mathcal{V} \rightarrow 2^{\mathcal{T}}$ *and measures* : $\mathcal{V} \cup \mathcal{A} \rightarrow 2^{\mathcal{T}}$.

Figure 4 shows an example grouped graph with its associated *dimensions* and *measures* functions.

[3] For the sake of simplicity, we slightly violated conjunctive query notation by using a constant in the header.

Definition 6 (Graph Reduce Function). *A graph reduce function f is a function that maps a grouped graph into a data graph.*

We are now ready to provide a definition of the Graph Aggregation Query.

Definition 7 (Graph Aggregation Query). *A graph aggregation query is a tuple $\mathcal{Q} = (D, M, E, N, R, f)$ such that:*

- *R is a relation query with distinguished variables x, p, y, o;*
- *D is a set of dimensions such that for each dimension $d \in D$ the distinguished variables of d are x, v_d for some unique variable v_d;*
- *E is a set of dimensions such that for each dimension $c \in E$ the distinguished variables of c are y, v_c for some unique variable v_c;*
- *two measure queries M and N with distinguished variables x, m and y, n, respectively;*
- *a graph reduce function f.*

In the definition above, R is meant to define related nodes in the typical form of (subject, predicate, object) with an associated measure. D and E are the set of dimensions to group the subjects and objects defined by R. While the dimensions for subjects and objects can be the same, Fig. 3c depicts an example where it is beneficial to group subjects and object by different dimensions. Finally, M and N are the measures for subject and object nodes.

3.3 Operational Semantics

A grouped graph $\mathcal{G} = (\mathcal{V}, \mathcal{A}, l_{\mathcal{V}})$ is the result of grouping a data graph $G = (V, A, l_V)$ according to the graph aggregation query $Q = (D, M, E, N, R, f)$ if the following holds:

1. $w \in \mathcal{V}$ and $dimensions(w) = d$ iff $\exists (r, d) \in D(G)$ or $(r, d) \in E(G)$[4]. In this case we say that the node r in the original graph *maps to* the node w in the grouped graph and denote this as $r \mapsto w$;
2. $(u, p, v) \in \mathcal{A}$ and $o \in measures((u, p, v))$ iff $\exists (r, p, s, o) \in R(G)$ such that $r \mapsto u$ and $s \mapsto v$;
3. For all $u \in \mathcal{V}$; $a \in measures(u)$ iff $(r, a) \in M(G)$ or $(r, a) \in N(G)$ and $r \mapsto u$.

4 Answering Graph Aggregation Queries

The evaluation of a graph aggregation query is performed in three steps: (1) building a *binding table* that is the solution of the queries defined by the dimensions, relation and measures; (2) building a grouped graph from the binding table; and (3) applying the *reduce function* over the grouped graph to achieve the final results.

[4] $q(G)$ is the solution of query q against graph G. We extend the notion here to a set of queries where the result is the join query as defined before.

4.1 Binding Table

Evaluating a query in SPARQL results in a binding table as detailed in [16]. The first step to answer a graph aggregation query $\mathcal{Q}$ is to combine and evaluate the query:

$$\begin{aligned} q(x,p,y,o,v_D,m,v_E,n) := q_R(x,p,y,o)\ \wedge\ q_D(x,v_D) \wedge \\ q_M(x,m)\ \wedge\ q_E(y,v_E)\ \wedge\ q_N(y,n) \end{aligned}$$

The result is a binding table $\mathfrak{B}$ which is a set of rows. If r is a row in $\mathfrak{B}$ and x is a variable, we use $r[x]$ to refer to the value bound with the variable x in r. Delegating the evaluation of the binding table exploits the best practices and optimisation techniques already built in existing SPARQL engines.

4.2 Building the Grouped Graph

Algorithm 1 describes how starting from the binding table resulting from the previous step, a grouped graph compatible with the defined operational semantics can be built. The function `GetGroupNode(`v`)` creates a node corresponding to the dimensions values v, updates the *dimensions* mapping of $\mathcal{G}$ and adds the created node to $\mathcal{V}$. If a node corresponding to the dimension values already exists in $\mathcal{V}$, the function just returns the node. The `GetGroupNode` function needs some hash structure that maps dimension values to nodes and keeps the measures associated with each node and edge. Assuming that this data structure fits in memory, the complexity of `GetGroupNode` is $O(1)$. Consequently, Algorithm 1 has a complexity of $O(|\mathfrak{B}|)$ as it scans the binding table only once.

4.3 Applying the Graph Reduce Function

The graph reduce function applies an aggregate function on the set of measures and restructure the grouped graph. One possible way to express a graph reduce function is to base it on a template similar to the one used in CONSTRUCT queries in SPARQL.

5 Evaluation

In this section, we discuss the expressivity of the Gagg operator and present an experimental performance comparison with plain SPARQL queries.

5.1 Expressivity

We have used Gagg to express type summary as defined in [7] and to reproduce VoID statistics similar to the results provided by previous systems [1,22]. Furthermore, Gagg was applied to bibliographic data to compute a number of

Algorithm 1. Aggregated graph creation from the table of bindings

```
Data: The binding table $\mathfrak{B}$ and the graph aggregation query
      $(D, q_M(x,m), E, q_N(x,n), q_R(x,p,y,o), f)$
Result: The grouped graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{A}, l_{\mathcal{V}}, dimensions, measures \rangle$.
1 for $r \in \mathfrak{B}$ do // For each row in the binding table
      // Retrieve grouped nodes corresponding to the dimensions
2     $u \leftarrow$ GetGroupNode($r[D]$)
3     $v \leftarrow$ GetGroupNode($r[E]$)
      // create an edge
4     $e \leftarrow (u, r[p], v)$
5     $\mathcal{A} \overset{\cup}{\leftarrow} e$
      // Add measures
6     $measures(u) \overset{\cup}{\leftarrow} r[m]$
7     $measures(v) \overset{\cup}{\leftarrow} r[n]$
8     $measures(e) \overset{\cup}{\leftarrow} r[o]$
9 end
```

bibliometrics as shown in the next subsection. Finally, we have used Gagg to aggregate the LOD Cloud[5]. As this data is available in RDF[6] we were able to generate different versions of the LOD diagram by aggregating datasets by their topic, license, publisher, etc. Gagg is expressive enough to aggregate datasets and count them or sum their triples counts[7]. Similarly, it can count the number of linksets and sum the number of interlinking triples. This provided views over the LOD cloud from a variety of perspectives.

5.2 Performance Evaluation

We extended the query algebra in Apache Jena[8] with the Gagg operator. The implementation builds the intermediate graph and then apply aggregate functions whose implementations are re-used from Jena. We compare the performance of the usage of Gagg to that of standard SPARQL. In particular, we report the average running time of four approaches[9]:

reduced provides incomplete results as it aggregates only the relationships and not the subjects and objects. This is included as a *baseline* to quantify the extra time needed by the other approaches to get the full results.

[5] http://lod-cloud.net/.
[6] http://lod-cloud.net/data/void.ttl.
[7] The scripts to generate the aggregated graphs of the LOD Cloud diagram are available at https://github.com/fadmaa/rdf-graph-aggregation.
[8] https://jena.apache.org/ version 2.12.0.
[9] We report the SPARQL queries of the approaches at https://github.com/fadmaa/rdf-graph-aggregation.

fullSparql builds the aggregated graph using one CONSTRUCT SPARQL query. This query contains three sub-queries such that the first one defines the relationship, the second aggregates and counts subjects and the final one aggregates and counts objects.

3Sparqls uses three separate CONSTRUCT SPARQL queries to build the aggregated graph. Similar to the fullSPARQL approach, one query defines the relationship and the other two queries aggregate subjects and objects. Notice that these queries need to rely to some characteristics of the data or some hashing function to assure that the results of the three queries use the same identifiers for the aggregated nodes. Therefore, writing these queries was relatively hard. The final result is the union of the three graphs resulted from the queries, however, to prevent introducing extra penalty on the running time, the values we report do not include the time needed to union the three results.

Gagg uses the Gagg operator.

We used JUnitBenchmarks[10] to run the evaluation. JUnitBenchmarks performs JVM warm-up phases and repeat the execution multiple times to enhance the reliability of the reported times. The evaluation was run on a 4 core machine running Linux (3.18.1-3) at 2.60 GHz with 8 Gb of RAM. The JAVA version is 1.7.0_71.

We experimented with BSBM [4] and SP2B [29] datasets, varying the sizes of data. Two set of queries are used to perform two tasks: building a type summary and calculating some bibliometrics-based summary.

Type Summary. For this graph aggregation query, all resources in the RDF data are grouped by their types (i.e., the values of `rdf:type`) and all relations between resources are grouped. The summary reports the number of instances per type and the number of relations of each type that exists between instances of two classes. This type of statistics are the ones reported in VoID statistics and in the RDF graph summary in [7].

In the corresponding Gagg operators, the set of dimensions of subjects and objects is $(?x,$`rdf:type`$, ?t)$ and the relation ship query is $(?x, ?p, ?y)$.

Bibliometrics. We report on three graph aggregation queries that were evaluated on top of SP^{2}B data:

co-authorship gets the graph structure of co-authorship. Nodes in the summary graph represent authors along with the number of papers they published while edges represent co-authorship between them along with the counts.

citation gets the graph structure of author citation. Nodes in the summary graph represent authors along with the number of papers they authored while edges represent citation across authors along with their counts.

[10] http://labs.carrotsearch.com/junit-benchmarks.html.

conf-citation gets the graph structure of citation among papers grouped by the conference they are published in. Nodes in the summary graph represent conferences along with the number of papers published in each while edges represent citation across conferences along with their counts.

5.3 Discussion

The average running times to compute the Type Summary queries are reported in Table 1 while the average running times of the bibliometrics queries are shown in Table 2. Entries marked with N/A failed to finish as the machine ran out of memory or the execution took too long.

In general fullSparql approach showed the worst performance among the tested approaches. This is not surprising giving that dimensions and measures are evaluated three times in the fullSparql because results need to be aggregated along different dimensions each time, i.e., for subject, object and relation. Results of the three sub-queries of fullSparql also need to be joined together. In particular, the dramatic growth in the TypeSummary queries is due to the need for joining the results of the three sub-queries before aggregation. Because we are interested in aggregating all relations in the graph to compute a type summary, the sub-queries are not selective and produce large intermediary results. This is an extra overhead cost that both Gagg and 3Sparqls approaches avoid. In comparison to fullSparql, Gagg achieved up to two orders of magnitude improvement in response time.

The performance improvement of Gagg of about a 2.5 factor in comparison to 3Sparqls is reasonable. Gagg builds and scans the binding table once instead of three times. Nonetheless, it still performs the three aggregations.

Finally, in comparison to the reduced approach, the overhead that Gagg adds to achieve the full aggregated graphs is small. We remind that this approach gives incomplete results, since it does not aggregate the subject and object resources.

Table 1. Average running times of type summary queries.

dataset-size (# triples)	fullSparql (s)	3Sparqls (s)	reduced (s)	Gagg (s)
bsbm-5K	0.08	0.06	0.01	0.03
bsbm-190K	9.84	1.25	0.42	0.55
bsbm-370K	31.88	2.82	1.00	1.13
bsbm-1.8M	454.07	13.48	4.37	5.61

(a) BSBM

dataset-size (# triples)	fullSparql (s)	3Sparqls (s)	reduced (s)	Gagg (s)
sp2b-50K	0.42	0.22	0.08	0.09
sp2b-100K	0.84	0.44	0.15	0.16
sp2b-500K	4.75	2.19	0.75	0.84
sp2b-1M	10.32	4.84	1.56	1.8

(b) SP^{2}B

Table 2. Average running times of bibliometrics queries on the SP^{2}B data.

dataset-size (# triples)	fullSparql (s)	3Sparqls (s)	reduced (s)	Gagg (s)
sp2b-50K	3.93	0.18	0.053	0.09
sp2b-100K	4.05	0.20	0.05	0.08
sp2b-500K	4.02	0.18	0.05	0.08
sp2b-1M	N/A	6.80	1.04	2.62

(a) Co-authorship

dataset-size (# triples)	fullSparql (s)	3Sparqls (s)	reduced (s)	Gagg (s)
sp2b-50K	1.15	1.02	0.32	0.07
sp2b-100K	4.89	4.29	1.44	0.09
sp2b-500K	94.06	86.39	29.41	0.18

(b) Citation

dataset-size (# triples)	fullSparql (s)	3Sparqls (s)	reduced (s)	Gagg (s)
sp2b-50K	0.14	0.17	0.05	0.06
sp2b-100K	0.67	0.68	0.22	0.23
sp2b-500K	N/A	29.31	9.7814	9.8091

(c) Conf-citation

6 Conclusion

We introduce in this work an operator called *Gagg* for aggregating graph data. We defined the operator as a two-steps processing, where the graph is first grouped based on some dimensions, which is then reduced into an aggregated graph. The aggregated graph exhibits groups of dimensions and relations between such groups, as well as statistics associated to the groups and links. Our definition of the operator is formally anchored in the RDF algebra, making it possible to be composed with other graph operations. The actual aggregation of the graph can be fully customised by a user via the use of BGP queries. We showed in our evaluation that the graph aggregation Gagg improves significantly the performance of plain-SPARQL graph aggregation.

In future work, we plan to refine the formalisation of the Gagg operator to include (a) multi-valued dimensions; and (b) missing data, especially for the definition of dimensions. We plan also to investigate optimisations of graph queries using the Gagg operator. Moreover, providing a distributed implementation of Gagg might be an interesting direction to pursue.

Acknowledgements. Fadi Maali is funded by the Irish Research Council, Embark Postgraduate Scholarship Scheme. This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References

1. Auer, S., Demter, J., Martin, M., Lehmann, J.: LODStats – An extensible framework for high-performance dataset analytics. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) EKAW 2012. LNCS, vol. 7603, pp. 353–362. Springer, Heidelberg (2012)
2. Barrett, C., Jacob, R., Marathe, M.: Formal-language-constrained path problems. SIAM J. Comput. **30**(3), 809–837 (2000)
3. Böhm, C., Lorey, J., Naumann, F.: Creating VoID descriptions for web-scale data. Web Semantics: Science, Services and Agents on the World Wide Web **9**(3), 339–345 (2011)
4. Bizer, C., Schultz, A.: The Berlin Sparql Benchmark. Int. J. Seman. Web Inf. Syst. (IJSWIS) **5**(2), 1–24 (2009)
5. Bleco, D., Kotidis, Y.: Business intelligence on complex graph data. In: Proceedings of the 2012 Joint EDBT/ICDT Workshops, EDBT-ICDT 2012. ACM (2012)
6. Broder, A.Z., Lempel, R., Maghoul, F., Pedersen, J.O.: Efficient pagerank approximation via graph aggregation. Inf. Retrieval **9**(2), 123–138 (2006)
7. Campinas, S., Perry, T., Ceccarelli, D., Delbru, R., Tummarello, G.: Introducing RDF graph summary with application to assisted SPARQL formulation. In: Hameurlain, A., Tjoa, A.M., Wagner, R. (eds.) DEXA Workshops. IEEE Computer Society (2012)
8. Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.S.: Graph OLAP: towards online analytical processing on graphs. In: Eighth International Conference on Data Mining (ICDM 2008). IEEE (2008)
9. Codd, E.F., Codd, S.B., Salley, C.T.: Providing OLAP. On-Line Analytical Processing) to User-Analysts, An IT Mandate. E. F. Codd and Associates (1993)
10. Colazzo, D., Goasdoué, F., Manolescu, I., Roatiş, A.: RDF analytics: lenses over semantic graphs. In: 23rd International Conference on World Wide Web, WWW 2014, International World Wide Web Conferences Steering Committee (2014)
11. Delbru, R., Toupikov, N., Catasta, M., Tummarello, G., Decker, S.: Hierarchical link analysis for ranking web data. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 225–239. Springer, Heidelberg (2010)
12. Dobrin, R., Beg, Q.K., Barabási, A.L., Oltvai, Z.N.: Aggregation of topological motifs in the escherichia coli transcriptional regulatory network. BMC Bioinform. **5**(1), 10 (2004)
13. Fernandez, M., d'Aquin, M., Motta, E.: Linking data across universities: an integrated video lectures dataset. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part II. LNCS, vol. 7032, pp. 49–64. Springer, Heidelberg (2011)
14. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **40**, 35–41 (1977)
15. Georgieva-Trifonova, T.: Warehousing and OLAP analysis of bibliographic data. Intell. Inf. Manage. **3**(5), 190–197 (2011)
16. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language (2013). http://www.w3.org/TR/sparql11-query/ (Accessed 18 December 2014)
17. Jakawat, W., Favre, C., Loudcher, S.: OLAP on information networks: a new framework for dealing with bibliographic data. In: Catania, B., Cerquitelli, T., Chiusano, S., Guerrini, G., Kämpf, M., Kemper, A., Novikov, B., Palpanas, T., Pokorny, J., Vakali, A. (eds.) New Trends in Databases and Information Systems. AISC, vol. 241, pp. 361–370. Springer, Heidelberg (2014)

18. Kaushik, R., Bohannon, P., Naughton, J.F., Korth, H.F.: Covering indexes for branching path queries. In: ACM SIGMOD International Conference on Management of Data. ACM (2002)
19. Kazemzadeh, L., Kamdar, M.R., Beyan, O.D., Decker, S., Barry, F.: LinkedPPI: enabling intuitive, integrative protein-protein interaction discovery. In: 4th Workshop on Linked Science, ISWC 2014 Workshop (2014)
20. Khatchadourian, S., Consens, M.P.: ExpLOD: summary-based exploration of interlinking and rdf usage in the linked open data cloud. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 272–287. Springer, Heidelberg (2010)
21. Li, H.: Data profiling for semantic web data. In: Wang, F.L., Lei, J., Gong, Z., Luo, X. (eds.) WISM 2012. LNCS, vol. 7529, pp. 472–479. Springer, Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-33469-6_59
22. Mäkelä, E.: Aether – generating and viewing extended VoID statistical descriptions of RDF datasets. In: Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., Tordai, A. (eds.) ESWC Satellite Events 2014. LNCS, vol. 8798, pp. 429–433. Springer, Heidelberg (2014)
23. Mason, O., Verwoerd, M.H.A.: Graph theory and networks in biology. Syst. Biol. IET **1**(2), 89–119 (2007)
24. Mendes, P.N., Passant, A., Kapanipathi, P.: Twarql: tapping into the wisdom of the crowd. In: Proceedings of the 6th International Conference on Semantic Systems. ACM (2010)
25. Milo, T.: Index structures for path expressions. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 277–295. Springer, Heidelberg (1998)
26. Möller, K., Heath, T., Handschuh, S., Domingue, J.: Recipes for semantic web dog food — The ESWC and ISWC metadata projects. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 802–815. Springer, Heidelberg (2007)
27. Pavlopoulos, G., Secrier, M., Moschopoulos, C., Soldatos, T., Kossida, S., Aerts, J., Schneider, R., Bagos, P.: Using graph theory to analyze biological networks. BioData Mining 4(1) (2011). http://dx.doi.org/10.1186/1756-0381-4-10
28. Petermann, A., Junghanns, M., Muller, R., Rahm, E.: BIIIG: enabling business intelligence with integrated instance graphs. In: 30th International Conference on Data Engineering Workshops (ICDEW). IEEE (2014)
29. Schmidt, M., Hornung, T., Meier, M., Pinkel, C., Lausen, G.: SP2Bench: A SPARQL performance benchmark. In: Semantic Web Information Management. Springer, Heidelberg (2010)
30. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: ACM SIGMOD International Conference on Management of Data, SIGMOD 2008. ACM (2008)
31. Wasserman, S.: Social Network Analysis: Methods and Applications, vol. 8. Cambridge University Press, Cambridge (1994)
32. Zhang, N., Tian, Y., Patel, J.M.: Discovery-driven graph summarization. In: 26th International Conference on Data Engineering (ICDE). IEEE (2010)
33. Zhao, P., Li, X., Xin, D., Han, J.: Graph Cube: on warehousing and OLAP multidimensional networks. In: ACM SIGMOD International Conference on Management of Data, SIGMOD 2011. ACM (2011)