Speeding up RDF aggregate discovery through sampling

Ioana Manolescu
Inria and LIX (UMR 7161 and Ecole polytechnique), France
ioana.manolescu@inria.fr

Mirjana Mazuran*
Inria and LIX (UMR 7161 and Ecole polytechnique), France
mirjana.mazuran@inria.fr

ABSTRACT

RDF graphs can be large and complex; finding out interesting information within them is challenging. One easy method for users to discover such graphs is to be shown *interesting aggregates* (under the form of two-dimensional graphs, i.e., bar charts), where interestingness is evaluated through statistics criteria. Dagger [5] pioneered this approach, however its is quite inefficient, in particular due to the need to evaluate numerous, expensive aggregation queries. In this work, we describe Dagger$^+$, which builds upon Dagger and leverages *sampling* to speed up the evaluation of potentially interesting aggregates. We show that Dagger$^+$ achieves very significant execution time reductions, while reaching results very close to those of the original, less efficient system.

1 INTRODUCTION

RDF graphs are oftentimes large and complex; first-time users have a hard time to understand them. Exploration methods investigated in the past are based on keyword search, or on RDF summaries, which give users a first idea of a graph's content and structure. A different method of exploring RDF graphs was introduced in Dagger [5], based on *aggregation*. Starting from an RDF graph, a set of agregate queries are automatically identified and evaluated, the most *interesting* ones (in a sense to be outlined shortly below) are chosen, and shown to human users as two-dimensional plots, in particular, bar charts.

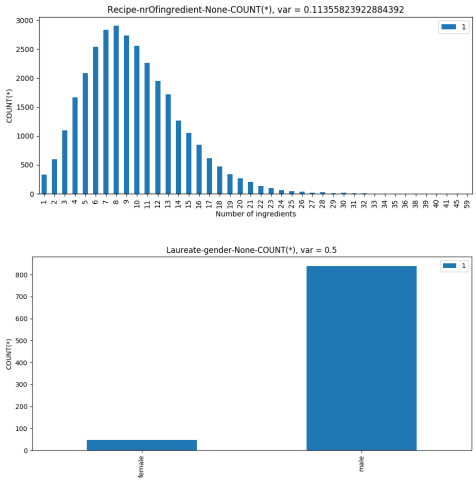

Figure 1: Sample interesting aggregates.

Figure 1 shows two examples. Above, we see the distribution of the number of ingredients appearing in food recipes in an RDF version of the foodista.com Web site, going from 1 to 25 with a peak at 8; we consider this distribution interesting as it has a clearly identified peak and quite a narrow interval. The graph below shows the distribution of Nobel prizes between female (left) and male (right) recipients; we find this interesting as it is very unbalanced.

*M. Mazuran is supported by the H2020 research program under grant nr. 800192.

As these examples illustrate, in Dagger$^+$, we consider an aggregate query *interesting* if its result set has a high variance (second statistical moment). Other criteria can be considered, e.g., we have experimented also with the third statistical moment (skewness); more generally, while an RDF graph may have many interesting features, we seek to find *aggregates over the graph having the highest variance*. The running time of Dagger was quite high: it took hours to identify the 10 most interesting aggregates on a dataset of 20 million triples. In this paper, we show how to speed it up, in particular through novel *sampling* techniques.

The remainder of this article is organized as follows. Section 2 introduces the state of the art. Section 3 describes the main concepts and algorithms of Dagger [5], together with a set of improvements we brought through re-engineering. Then, Section 4 presents our performance-enhancing sampling technique.

2 STATE OF THE ART

The problem of data exploration [8] has received much attention; the automatic extraction of interesting aggregates is just one among many proposed techniques. Multidimensional data is particularly interesting in this respect, however, most works assume a fixed relational schema, which is not available for RDF graphs. More recent works [2], consider graphs, but (unlike Dagger) assume a very regular and simple structure.

In [9] the authors show how to automatically extract the top-k insights from multi-dimensional relational data, with fixed dimensions and measures. An insight is an observation derived from aggregation in multiple steps; it is considered interesting when it is remarkably different from others, or it exhibits a rising or falling trend. SeeDB [10] recommends visualizations in high-dimensional relational data by means of a phased execution framework. The focus is to detect the visualizations with a large deviation with respect to a reference (e.g. another dataset, historical data or the rest of the data) from a database with a snowflake schema. In [6], multi-structural databases are proposed; their schema (i.e. possible dimensions) is known and three analytical operations are defined to: (*i*) determine how data is distributed across a particular set of dimensions, (*ii*) compare two sets of data with respect to given dimensions and (*iii*) separate the data into cohesive groups with respect to the known dimensions.

In contrast, Dagger (and Dagger$^+$) start directly from RDF, and, lacking schema information, automatically derives dimensions and measures that are good candidates to produce insights.

There is no universally accepted definition of interestingness; frequently, something unexpected (which differs from a reference) is considered interesting. The reference might be known a-priori, come from historical data, or be the average behavior. So far we have experimented with variance, skewness and kurtosis; we are also working to use the entropy. Many different RDF visualizations techniques can be used, e.g., [1].

3 DAGGER OVERVIEW

We consider three pairwise disjoint sets: the set of URIs $\mathcal{U}$, the set of literals $\mathcal{L}$, and the set of blank nodes $\mathcal{B}$. An RDF graph G is a finite set of triples of the form (s, p, o), called subject, property

and object, such that $s \in (\mathcal{U} \cup \mathcal{B})$, $p \in \mathcal{U}$ and $o \in (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. The RDF property *rdf:type* is used to attach types (i.e. classes) to an RDF resource, which can have zero, one or several types. G can have an *ontology* stating relationships between its classes and properties, e.g., any *UndegraduateStudent* is a *Student*; its presence may lead to *implicit* triples, which are part of G even if they may not appear explicitly. A graph containing all the triples which may be derived from it is said to be saturated. Without loss of generality, we consider that our input graph is saturated.

Dagger identifies unidimensional aggregate queries to be evaluated in an RDF graph G. However, unlike a traditional relational data warehouse, an RDF graph comes without identified set of *facts*; nor are *dimensions* and *measures* known in advance. Therefore, Dagger must enumerate *candidates* for each of these roles.

A set of *candidate facts* (cf, in short) is a set of G resources which are deemed interesting for the analysis. Dagger considers as cf (*i*) all the resources of a given class C, e.g., all the *Students*; (*ii*) all the resources having a certain set P of properties, e.g., all those having *title* and *author*. A simple way to pick such a property set is to compute the support of the properties in G and select property sets whose support is above a certain threshold.

A *candidate dimension* (denoted d) is used for grouping the candidate facts. Dagger supports as candidate dimensions (*i*) properties present on at least a certain fraction t_{thres} of resources from cf; (*ii*) *derived* properties, computed by Dagger in order to find potentially interesting insights. The derived properties supported in [5] are *count(p)*, where p is a property that some cf resources have. For instance, if resources of type *Student* are stated to *takeCourse*, Dagger derives the property *takeCourse#*. Further, Dagger will consider a candidate dimension only the number of distinct values of this property on the cf resources is smaller than $t_{dist} \times |cf|$ for a certain ratio $0 < t_{dist} < 1$.

A *candidate measure* (denoted m) is something to be evaluated or measured for each candidate fact. Dagger considers candidate measures among the (original or derived) properties of candidate facts whose support is above t_{thresh}. Dimension-measure combinations such that one is a property a and the other is the count a# of this property are excluded.

An *aggregation function* $\oplus$ is chosen among *min, max, avg, sum, count*; the first four are considered only if the measure is numeric. Given that RDF data is often untyped or only partially typed, Dagger implements a type detection mechanism by trying to convert the property values to different types.

A Dagger aggregate *agg* is a tuple $(cf, d, m, \oplus)$. To specify how interesting an aggregate is, let $f(V)$ be a function which inputs a set of numerical values V and returns a number. Given f, the interestingness of *agg* is computed as:

- let $d_1, d_2, \ldots$ be the distinct values that d may take for a resource in cf;
- for each d_i, let cf_i be set of cf resources for which d takes the value d_i; observe that the cf_i sets may overlap, as a resource with more than one value for d belongs to several such sets. For instance, students can be grouped by the courses they take, and each student takes many courses;
- for each cf_i, let M_i be the set of m values of cf_i resources, and $m_i = \oplus(M_i)$;
- the interestingness of *agg* is $f(\{m_1, m_2, \ldots\})$.

The problem considered by Dagger is: given a graph G, a set of value thresholds, function f and an integer k, find the k most interesting aggregates.

Architecture. Dagger leverages the robust query evaluation capabilities of an RDBMS to store the RDF graph, enumerate and evaluate candidate aggregates. SQL queries are used to: determine the resources part of a candidate fact sets; evaluate the suitability of their properties as dimensions, respectively, measures; compute and aggregate the group measures m_i. The remaining operations, e.g., the computation of the interestingness score function, are done in Java.

Aggregate recommendation cost. The most expensive computation steps are: (*i*) finding the candidate dimensions and (*ii*) evaluating the aggregation operations. Indeed, when looking for candidate dimensions, several queries are issued over cf (e.g., find all distinct properties, count the number of subject that have each property, find the values of the derived properties, etc.). Moreover, many candidate aggregates are generated, also leading to a high number of potentially expensive SQL queries.

4 DAGGER$^+$: SPEEDING UP DAGGER

A set of *re-engineering* changes were brought to Dagger since the original demonstration. In particular, it has been ported on top of OntoSQL (https://ontosql.inria.fr), a Java-based platform developed at Inria, providing efficient RDF storage, saturation, and query processing algorithms [3, 4]. OntoSQL encodes space-consuming URIs and literals into compact integers, together with a dictionary table which allows going from one to the other. For a given class c, all triples of the form x *type* c are stored in a single-column table t_c holding the codes of the subjects x; for each property p other than *type*, a table t_p stores (s code, o code) pairs for each (s, p, o) triple in G. This re-engineering has lead to a very significant reduction in Dagger's running time, e.g., on a 20 million triples graph, from several hours to 20 minutes.

Further, we *adapted an optimization* previously proposed in [10]: we evaluate all candidate aggregates that share both dimension and measure by a *single* SQL query. In a relational context, aggregates can be evaluated together as soon as they have the same *dimension* (even if the measures are different) because one relational tuple usually contains all the attributes used in the different measures. In contrast, RDF candidate facts need to be *joined* with the t_d table corresponding to the property d chosen as dimension, and with the t_m table for the property chosen as measure; distinct measures require distinct joins, thus sharing is more limited. This is not due to the storage model, but to RDF heterogeneity: the candidate facts having the measure property m_1 may be different from those having property m_2. This forces evaluating (d, m_1) aggregates separately from (d, m_2) ones.

Below, we focus on two *novel optimization techniques* we applied subsequently: using an *RDF graph summary* to speed up candidate dimension enumeration (Section 4.1), and using *sampling* to accelerate candidate dimension enumeration, aggregate evaluation and ranking (Section 4.2).

4.1 Summary-based dimension enumeration

RDFQuotient [7] is a structural RDF graph summarization tool based on *graph quotients*. Given an equivalence relation over graph nodes, the summary contains one node for each equivalence class in the original graph. Moreover, each edge $n \xrightarrow{a} m$ in the original graph leads to an edge $rep(n) \xrightarrow{a} rep(m)$ in the summary, where $rep(n)$ is the summary representative of node n. A particular equivalence relation groups the nodes by their set of types. Dagger$^+$ uses it to find: (*i*) all the types, (*ii*) for each type, the number of resources of that type, and (*iii*) the set of possible properties these resources might have. These questions can be answered exactly directly from the RDFQuotient reducing dimension enumeration time.

4.2 Sampling-based aggregate selection

We introduced two sampling strategies to trade some accuracy in aggregate selection for running time:

- *CFSampling*: the cf is sampled (draw n_1 samples of size n_2), and candidate dimensions and measures are found for each sample independently. For each of the n_1 samples, candidate aggregates are generated, and their interestingness is evaluated on the sample.
- *ESampling*: candidate dimensions and measures are computed on the whole cf as in Dagger. Then, n_1 samples of size n_2 are drawn from the cf, and the candidate aggregates are evaluated on the samples.

With CFSampling, the aggregates recommended for different sample may be different, e.g., certain properties might be found to be frequent in some sample but not in all of them. After the evaluation, a ranked list of aggregates is obtained for each sample. In ESampling, instead, the set of aggregates is unique as it is derived directly from the original data. However, given that all the aggregates are evaluated on samples, it also yields a ranked list of aggregates for each sample.

To be able to compare the results found without sampling with those found through sampling, we need to reconcile the results found on different samples into a single ranked list. We can do this by taking (*i*) the union or (*ii*) the intersection of the lists obtained from all the samples. Then, we re-rank the aggregates according to a estimation of their global interestingness measure (based on their interestingness on each sample), e.g., through *pooled variance*; at the end of this process, we obtain a unique ranked list of candidate aggregates.

5 EXPERIMENTAL RESULTS

We measure the run time performances and to test the accuracy of the results obtained with sampling techniques. We used a 2,7 GHz Intel Core i7 with 16 GB of RAM. We describe experiments on the articles from the DBLP dataset[1]; we use 20.132.491 triples describing information about 888.183 distinct articles.

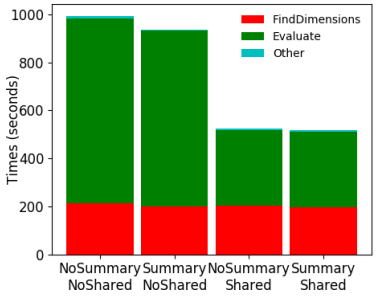

Figure 2: Impact of summary and sharing.

Dagger$^+$ without sampling. First, we evaluate how much the use of the summary, and the shared evaluation, improve performance. Figure 2 shows the two main components (finding dimensions and aggregate evaluation), as well as a third, very small bar representing the other operations. Note that the property analysis which allows to select candidate dimensions also enables to select candidate measures (thus candidate measure selection time is negligible and wrapped under "Other").

Without any optimization ("NoSummary, NoShared" bar in Figure 2), 21% of the time is spent looking for dimensions, while 78% of the time goes to evaluating aggregates. Summary use introduces a modest performance improvement, while shared aggregate evaluation has a much more significant result (see the "NoSummary, Shared" and "Summary, Shared" bars). Here,

[1]http://www.rdfhdt.org/datasets/

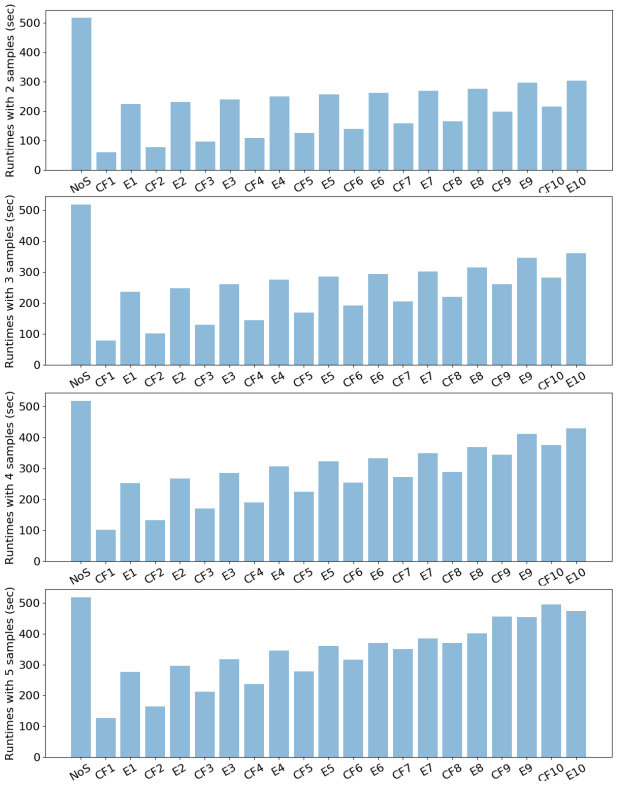

Figure 3: Running times with sampling.

Dagger$^+$ generates a total of 371 candidate aggregates; without shared evaluation, each of them is evaluated as one query, however, with shared evaluation, only 131 queries are executed as this is the number of distinct pairs of dimension and measure.

Sampling. We measure the running time and the accuracy of the results by varying 3 parameters: (*i*) the sampling strategy (CFSampling or ESampling), (*ii*) the number of samples (2, 3, 4 or 5); and (*iii*) the sample size, as a ratio of the candidate fact set size (from 1% to 10% of the number of distinct CF resources). Figure 3 shows the **running times**: one plot for each number of samples (from top to bottom, 2, 3, 4, 5), each of which varies the strategy and the sample size. **Ten runs** are averaged for each parameter combinations (different samples are drawn in each run).

Figure 3 shows that *ESampling is slower than CFSampling*, as the former looks for candidate dimensions on the whole candidate fact set, whereas the latter does this on smaller-size samples. ESampling *enumerates* all the candidate aggregates also found without sampling, while CFSampling might enumerate a different set of aggregates: some properties that are overall frequent (infrequent) might be found infrequent (frequent) in a specific sample, and thus not be considered (considered) as candidate dimensions and measures. However, even though ESampling enumerates all the no-sampling candidate aggregates, it evaluates them on samples, and may end up considering them interesting (uninteresting) differently from the way they are on the complete CF.

To evaluate the **accuracy** of the results obtained through sampling, we compare the Top-5, Top-10 and Top-20 aggregates found without sampling, with those found through sampling. The ranked list of aggregates found with sampling is built by taking the union of all those found across the samples and re-ranking them according to the interestingness measure (in our experiments we use the pooled variance to rank all the results).

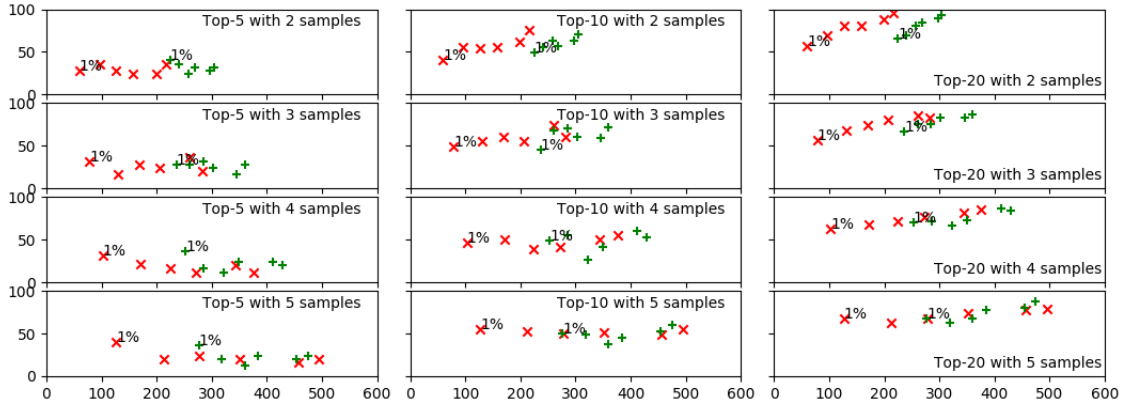

Figure 4: Accuracy of results (%) with respect to running times (sec).

#samples	Top-K	1%	3%	5%	7%	9%	10%
2	Top5	28%	36%	28%	24%	24%	36%
	Top10	41%	55%	54%	55%	62%	75%
	Top20	57%	69%	80%	81%	88%	96%
3	Top5	32%	16%	28%	24%	36%	20%
	Top10	49%	55%	60%	55%	74%	61%
	Top20	57%	68%	74%	80%	85%	83%
4	Top5	32%	22%	16%	12%	20%	12%
	Top10	46%	50%	39%	42%	50%	55%
	Top20	63%	68%	72%	76%	81%	85%
5	Top5	40%	20%	24%	20%	16%	20%
	Top10	55%	53%	50%	51%	49%	55%
	Top20	67%	63%	68%	74%	78%	79%

Table 1: Accuracy of results with CFSampling.

#samples	Top-K	1%	3%	5%	7%	9%	10%
2	Top5	40%	36%	24%	32%	28%	32%
	Top10	49%	55%	63%	57%	63%	71%
	Top20	65%	69%	81%	84%	89%	93%
3	Top5	28%	28%	32%	24%	16%	28%
	Top10	45%	68%	71%	60%	59%	72%
	Top20	67%	76%	75%	83%	83%	87%
4	Top5	36%	16%	12%	24%	24%	20%
	Top10	49%	55%	27%	41%	60%	53%
	Top20	70%	71%	66%	73%	87%	84%
5	Top5	36%	20%	12%	24%	20%	24%
	Top10	50%	49%	38%	45%	52%	60%
	Top20	68%	63%	68%	77%	80%	88%

Table 2: Accuracy of results with ESampling.

Aggregates whose interestingness is zero are not considered. Notice that we do not break ties, that is: if we search e.g. the Top-5 most interesting aggregates, but find that the sixth element in the list has the same interestingness value as the fifth, we also include it in the Top-5, as we did not find a meaningful way to break such ties. When no sampling is used, Top-5 returned 5 aggregates, respectively, Top-10 returned 11 while the Top-20 returned 20. We compute the *accuracy* as the percentage of aggregates in the sampling result, that are also in the result without sampling.

Table 1 shows the precision of CFSampling while Table 2 shows that of ESampling. *In* 57% *of the cases the accuracy obtained using ESampling is higher*; on average, the two accuracy values differ by 6%. The accuracy is quite low when searching for the the Top-5 aggregates; in contrast, both Top-10 and Top-20 can be well approximated (accuracy above 80% for the Top-20 even with few samples). In general, the bigger the samples, the better the accuracy, however, results show situations where the Top-10 and Top-20 have better accuracy with a lower number of samples; there is a 10% difference in accuracy on average, i.e. the top-Ks differ by 2/3 aggregates. This does not mean that such aggregates were not found though but they have been ranked differently.

Figure 4 plots the accuracy with respect to the running times. Each line of graphs represents the Top-5, Top-10 and Top-20 for a different number of samples; red x's indicate CFsampling, while green +'s indicate ESampling, for different sample dimensions. Clearly, the accuracy increases with K (the amount of top aggregates) and with the size of the samples. The increase in the number of samples does not significantly improve accuracy.

6 CONCLUSION

Automatic selection of interesting aggregates in an RDF graph is challenging, as candidate dimensions and measures must be "guessed" from the data, and candidate aggregates must be evaluated to assess their interestingness. After re-engineering Dagger [5] to exploit an existing efficient RDF platform, we have shown that *sharing* work and (to a lesser extent) using a *graph summary* can reduce its running time by a factor of 2. Our main focus has been on *sampling*, which, for Top-10 and Top-20 search, achieves good accuracy (above 70%) while reducing running time by another factor of 2. Overall, CFSampling appears the most interesting strategy. Our current work stretches in several directions: (i) generalizing Dagger to more complex dimension and measure selection, (ii) adopt more interestingness metrics, (iii) introduce new types of derived properties, e.g., extract meaningful keywords from textual properties to be used as potential dimensions and/or measures.

REFERENCES

[1] Fabio Benedetti, Sonia Bergamaschi, and Laura Po. 2015. LODeX: A Tool for Visual Querying Linked Open Data. In *ISWC 2015 Posters & Demonstrations*.

[2] D. Bleco and Y. Kotidis. 2018. Using entropy metrics for pruning very large graph cubes. *Information Systems* (2018). To appear.

[3] D. Bursztyn, F. Goasdoué, and I. Manolescu. 2015. Optimizing Reformulation-based Query Answering in RDF. In *EDBT*.

[4] D. Bursztyn, F. Goasdoué, and I. Manolescu. 2016. Teaching an RDBMS about ontological constraints. *PVLDB* 9, 12 (2016).

[5] Y. Diao, I. Manolescu, and S. Shang. 2017. Dagger: Digging for Interesting Aggregates in RDF Graphs. In *ISWC Posters & Demonstrations Track*.

[6] R. Fagin, R. V. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. 2005. Multi-structural databases. In *PODS*. ACM, 184–195.

[7] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. 2019. Incremental structural summarization of RDF graphs (demo). In *EDBT*.

[8] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. 2015. Overview of Data Exploration Techniques. In *SIGMOD*. 277–281.

[9] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. 2017. Extracting Top-K Insights from Multi-dimensional Data. In *SIGMOD*. ACM, 1509–1524.

[10] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. 2015. SEEDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *PVLDB* 8, 13 (2015), 2182–2193.