GraFa: Scalable Faceted Browsing for RDF Graphs

José Moreno-Vega and Aidan Hogan

IMFD Chile & Department of Computer Science, University of Chile

Abstract. Faceted browsing has become a popular paradigm for user interfaces on the Web and has also been investigated in the context of RDF graphs. However, current faceted browsers for RDF graphs encounter performance issues when faced with two challenges: scale, where large datasets generate many results, and heterogeneity, where large numbers of properties and classes generate many facets. To address these challenges, we propose GraFa: a faceted browsing system for heterogeneous large-scale RDF graphs based on a materialisation strategy that performs an offline analysis of the input graph in order to identify a subset of the exponential number of possible facet combinations that are candidates for indexing. In experiments over Wikidata, we demonstrate that materialisation allows for displaying (exact) faceted views over millions of diverse results in under a second while keeping index sizes relatively small. We also present initial usability studies over GraFa.

1 Introduction

The Semantic Web community has overseen the publication of a rich collection of datasets on the Web according to a variety of proposed standards [12]. However, current interfaces for accessing such datasets are not generally designed nor intended for end users to interact with directly. The Semantic Web community still lacks effective methods by which end users can interact with such datasets; or as Karger [18] phrases it: "*The Semantic Web's potential to deliver tools that help end users capture, communicate, and manage information has yet to be fulfilled, and far too little research is going into doing so.*"

On the other hand, *faceted search* [32][1] has become a familiar mode of interaction for many Web users, popularised in particular by e-Commerce websites like Amazon and eBay. Such interaction is characterised by iteratively refining the active result-set through filter conditions – called *facets* – typically defined to be an attribute (e.g., *type*, *brand*, *country*) and value (e.g., *Toothbrush*, *Samsung*, *India*) that the filtered results should have. Such interaction enables end users to find specific results corresponding to concrete criteria known in advance, or simply to explore and iteratively refine results based on available options.

While the queries that can be formulated through an iterative selection of facets are generally less expressive than those that can be specified through a

[1] Also known as "*faceted browsing*", "*faceted navigation*", etc.

structured query language such as SPARQL, faceted browsing is more accessible to a broader range of users unfamiliar with such query languages; furthermore, the end user need not be as familiar with the content or schema of the dataset in question since the facets offered denote the possible filters that can be applied and the number of results to be expected, helping users to avoid empty results.

Adapting faceted search for a Semantic Web context is then a natural idea, where various authors have explored faceted navigation over RDF graphs [7,28] as a potential way to bridge from Semantic Web to end-users. Such works – discussed in more detail in the following section on related work – have explored core themes relating to faceted navigation, including query expressivity, ranking, usability, indexing, performance, reasoning, complexity, etc. However, despite the breadth of available literature on the topic, we argue that more work is required, in particular for faceted browsing over RDF graphs that are *large-scale* (with many triples) and *diverse* (with many properties and classes).

The work presented in this paper was motivated, in particular, by the idea of providing faceted search for Wikidata [29]: a large, collaboratively-edited knowledge-base where users can directly add and curate structured knowledge relating to the Wikipedia project. Though a variety of interfaces exist for interacting with Wikidata[2], including a SPARQL endpoint, query builders, and so forth, none quite cover the main characteristics of a faceted browser (e.g., only displaying options with non-empty results). On the other hand, despite the breadth of works on faceted browsing, we could not find an available system that could load the full ("truthy") Wikidata graph available at the time of writing.

We thus propose a novel faceted browser for diverse, large-scale RDF graphs called GraFa – GRAph FAcets – that we demonstrate is able to handle the scale and diversity of a dataset such as Wikidata. An initial result set in the system is generated through either keyword search or by selecting an entity type (e.g., *person*, *building*, etc.). Thereafter, a result set can be refined by selecting a particular property–value (*facet*) that all entities in the next result set should have. A combination of auto-completion and ranking features help ensure that the user is presented with relevant facets and results. Furthermore, at each stage of interaction, only options that lead to non-empty results are returned; this aspect in particular proves the most challenging to implement.

Similar to previous faceted systems [31,5], the GraFa system is based on Information Retrieval (IR)-style indexes that combines unstructured (text) and semi-structured (facet) information. However, unlike previous such systems, we propose a novel materialisation technique to enable interactive response times at higher levels of scale. The core hypothesis underlying this technique is that although there is a potentially exponential (in the size of the graph) number of combinations of facets that could be considered, few combinations will lead to large result sets that cause slow response times. Hence we propose a technique to perform an offline analysis of the graph to select facets that are then materialised. Our results show that materialisation can improve worst-case response times by orders of magnitude using a modestly-sized index of precomputed facet views.

[2] https://wikidata.org/wiki/Wikidata:Tools/External_tools; retr. 2018/04/05

To assess the usability of our system, we also present the results of two initial studies. The first user study compares the GraFa system and the Wikidata Query Helper (WQH) interface provided by the Wikidata SPARQL endpoint, asking participants to solve a number of tasks using both systems. Based on the results of this first study, we then made some improvements to the GraFa system, where in the second study, we asked members of the Wikidata community to use the modified GraFa system and to answer a questionnaire to rate the usability, usefulness, responsiveness, novelty etc., of the system.

Outline: Section 2 first discusses related work. Section 3 defines the inputs and interactions considered in our faceted browsing framework. Section 4 describes the base indexing scheme used to support these interactions, and Section 5 describes the materialisation strategies we use to improve worst-case response times. Turning to evaluation, Section 6 focuses on performance, while Section 7 focuses on usability. Finally Section 8 concludes and discusses future work.

2 Related Work

Various faceted browsers have been proposed for RDF over the years [32,28,7]. Some earlier works include mSpace [23], Ontogator [20], BrowseRDF [21], /facet [15], with later proposals including gFacet [14,13], Explorator [2], Rhizomer [6], Facete [25], ReVeaLD [17], Sparklis [8] and Hippalus [27]. These works describe evaluations or use-cases involving domain-specific data of low heterogeneity, such as multimedia [23,20,26], suspect descriptions [21], movies [6], cultural heritage [20,15], tweets [1], places [25], biomedicine [17], fish species [27], etc.; furthermore, many of these works delegate data management and query processing to an underlying triple-store/SPARQL engine, and rather focus on issues such as expressiveness, ranking and usability, etc.

Recently Petzka et al. [22] proposed a benchmark for SPARQL systems to test their ability to support faceted browsing capabilities, but again the dataset (referring to transport) contains in the order of tens of classes and properties and we could not find details on the scale of data used for experiments.

A number of later works have explored faceted navigation over more heterogeneous RDF datasets, such as VisiNav [11] operating on RDF data (19 million triples with 21 thousand classes and properties) crawled and integrated from numerous sources on the Web; however, aside from brief discussion of top-k ordering of facets, performance issues were not discussed in detail. Another more scalable proposal is the Neofonie [10] system, proposed for faceted search over DBpedia; however, only a small selection of target facets are displayed and no performance results are provided. A more recent scalable approach is that of eLinda [33], which allows for real-time browsing of DBpedia; however, navigation is not based on facets but rather on interactive bar-charts.

A number of approaches have proposed to use indexing techniques developed for Information Retrieval (IR) to support faceted browsing for RDF. The Semplore system [31] builds faceted browsing on top of IR-indexes, where facets for the current result set are computed from types, as well as incoming and outgoing relations; a set of top-k facets are constructed by count. Experiments were conducted over DBpedia [19] and LUBM [9] datasets in the order of 100 million triples, showing mean sub-second response times faster than those achievable over selected triple stores. Though this system is along similar lines to what we wish to achieve, the size of the result-sets for which facets are generated in the evaluation is not specified, nor is the value of k for the top-k generation; we could not find materials online to replicate these results, but using a similar implementation later on a more modern version of the same IR engine (Lucene), we find that construction of the full set of facets takes minutes over large result-sets with millions of results. Wagner et al. [30] likewise propose IR-style indexing to support faceted browsing and conduct evaluation over DBpedia, but performance issues are explicitly considered out of scope; however, for their evaluation, we note that the authors mention use of caching to speed-up response times for selected tasks, though no further details are provided.

To the best of our knowledge, the closest published results we found for faceted search over RDF data at the scale of Wikidata was the Broccoli system [5,4], which is also based on IR indexes. Though the system has a slightly different focus to ours (semantic search over Wikipedia text enriched with Freebase relations), an index over relations is defined to enable faceted search. The authors propose caching methods to identify and re-use sub-combinations of facets that are frequently required; unlike our approach, this LRU cache is built online from user-queries, whereas we materialise query results offline.[3]

The SemFacet [3] system addresses a number of issues with respect to faceted browsing for RDF graphs, including reasoning, expressiveness, complexity and efficiency. Though their system can process facets for tens of millions of answers in about 2 seconds, this requires having all data indexed in memory, which limits scale; hence their evaluation is limited to 20% of DBpedia [19] (3.5 million triples), as well as selected slices of YAGO [16] that fit in memory. Though the system is available for download, we failed to load Wikidata with it. Later work by Sherkhonov et al. [24] discusses the addition of other features to faceted navigation, such as aggregation and recursion, but focuses on studying the complexity of query answering and containment.

3 Faceted Browsing

We now outline the faceted browsing interactions that the GraFa system currently supports. Beforehand we provide preliminaries for RDF graphs considered as input to the system, mainly to establish notation and nomenclature.

RDF triples and graphs: An *RDF triple* (s, p, o) is an element of $\mathbf{IB} \times \mathbf{I} \times \mathbf{IBL}$, where $\mathbf{I}$ is a set of IRIs, $\mathbf{L}$ a set of literals, and $\mathbf{B}$ a set of blank nodes; the sets $\mathbf{I}$,

[3] We did not find source code for the system to be able to perform tests for Wikidata, though a Freebase demo is available demonstrating interactive runtimes on large result sets: http://broccoli.informatik.uni-freiburg.de/demos/BroccoliFreebase/; retr. 2018/04/05

L and **B** are considered pairwise disjoint. The positions of the triple are called *subject*, *predicate*, and *object*, respectively. An *RDF graph* G is a set of triples. Letting $\pi_{\text{S}}(G) = \{s \mid \exists p, o : (s, p, o) \in G\}$ project the ("flat") set of all subjects of G, and letting $\pi_{\text{P}}(G)$ and $\pi_{\text{O}}(G)$ likewise project the set of all predicates and objects of G, we call $\pi_{\text{S}}(G) \cup \pi_{\text{O}}(G)$ the *nodes* of G, $\pi_{\text{S}}(G) \cap \mathbf{I}$ the *entities* of G, and $\pi_{\text{P}}(G)$ the set of *properties* of G. Given an entity s and a property p, we call any o such that $(s, p, o) \in G$ the *value* of property p for entity s.

Keyword selection: We assume most entities to have values for a label property (e.g., `rdfs:label`, `skos:prefLabel`, `skos:altLabel`) and/or a description property (e.g., `rdfs:comment`, `schema:description`); we also assume that the system is configured with a list of such properties. To generate an initial result-set, users can specify a keyword search, returning a set of entities whose label/description values match the search. Notation-wise, we will denote keyword search as a function $\kappa : 2^G \times \mathbb{S} \to 2^{\pi_{\text{S}}(G)}$, where $\mathbb{S}$ denotes the set of strings (keyword searches). However, to simplify notation, we will consider the input graph as fixed throughout this paper. Hence we abbreviate the function as $\kappa : \mathbb{S} \to 2^{\pi_{\text{S}}(G)}$, taking a string and returning a set of entities according to a keyword-matching function (we discuss implementation of the function in Section 4).

Type selection: To generate an initial set of results, rather than use the keyword search function, a user may prefer to select entities of a given *type* (e.g., *human*, *movie*, etc.). We define a *type* (aka. *class*) to be any value of a *type property* (e.g., `rdf:type`, `wdt:P31`$_{[instance\ of]}$) for any entity; we assume that a fixed set of type properties P_T are preconfigured in the system. We then denote the set of types in a graph G as $T(G) := \{o \mid \exists s, p : (s, p, o) \in G \text{ and } p \in P_T\}$. We denote type selection as $\tau : T(G) \to 2^{\pi_{\text{S}}(G)}$, where $\tau(t) := \{s \mid \exists p \in P_T \text{ such that } (s, p, t) \in G\}$. In summary, $\tau(t)$ returns the set of all entities with the type $t \in T(G)$. Note that we do not currently consider type/class hierarchies.

Facet selection: Given a current set of results, a user may select a *facet* to further restrict the presented results. Such a facet is here defined to be a property–value pair – e.g., (*director*,*Kurosawa*) – that each entity in the next result set must have. More formally, given a current result set of entities $E \subseteq \pi_{\text{S}}(G)$, we denote by $E(G) := \{(s, p, o) \in G \mid s \in E\}$ the projection from G of all triples with a subject term in E. Now we can define the facet selection function $\zeta : 2^{\pi_{\text{S}}(G)} \times \pi_{\text{P}}(G) \times \pi_{\text{O}}(G) \to 2^{\pi_{\text{S}}(G)}$ where $\zeta(E, p, o) := \{s \mid (s, p, o) \in E(G)\}$.

Faceted navigation: We call a sequence of selections of either of the following forms a *faceted navigation*, initiated by keyword or type selection, respectively:

– $\zeta(\zeta(...(\zeta(\kappa(q), p_1, o_1)..., p_{n-1}, o_{n-1}), p_n, o_n)$
– $\zeta(\zeta(...(\zeta(\tau(t)\,, p_1, o_1)..., p_{n-1}, o_{n-1}), p_n, o_n)$

We remark that the ζ function is *commutative*: we can apply the facet selections in any order and receive the same result. Hence, with some abuse of notation, we can unnest and thus more clearly represent the above navigation sequences as a conjunction of criteria, where we use $[\cdot]$ to represent optional criteria:

– $\kappa(q)\ [\wedge\, \zeta(p_1, o_1) \wedge ... \wedge \zeta(p_n, o_n)]$
– $\tau(t)\ [\wedge\, \zeta(p_1, o_1) \wedge ... \wedge \zeta(p_n, o_n)]$

Type and facet interactions: The type selection and facet selection interactions take as input a type t and a facet (p, o) respectively. However, the users may not know the corresponding identifier, hence GraFa will offer auto-completion search on the labels and aliases of types and the values of facet properties. For example, a user typing al* into the auto-completion box for type selection will receive suggestions such as album, alphabet, military alliance, etc.

Result display: For each result we display its label, description, and an associated image if available (again we assume that image properties are preconfigured). We further assume that entity identifiers are dereferenceable IRIs, which we can use to offer a link to further information about the entity from the source dataset. We also present the available facets for the current results.

Ranking: We combine three forms of ranking: *frequency*, *relevance* and *centrality*. Frequency indicates the number of results generated by a particular selection. Relevance is particular to keyword-search and uses a TF–IDF style measure to indicate how well a given entity's label(s) and description(s) match a keyword. Centrality measures the importance of a node in the graph, where we use PageRank: we consider each triple $(s, p, o) \in G \cap (\mathbf{I} \times \mathbf{I} \times \mathbf{I})$ in the graph to be a directed edge $s \to o$ and then apply a standard PageRank algorithm to derive ranks for all nodes. Thereafter, we use these measures in the following way:

– Entities in result pages generated directly from a keyword selection are ranked according to a combination of TF–IDF and PageRank score.
– Entities in result pages generated directly from a type or facet selection are ranked purely according to PageRank score.
– Types suggested by auto-completion are ranked according to PageRank.[4] The count of entities in each type are also displayed.
– Properties displayed in the list of facets are ordered by frequency: the number of entities in the current results with some value for that property.
– Auto-completed facet values are ordered by PageRank.

Multilingual support: Where language-tagged labels and descriptions are provided for entities in multiple languages (e.g., `"Denmark"@en`, `"Dinamarca"@es`), GraFa can support multiple languages: the user can first select the desired language where search matches text from that language and where labels from that language are used to generate results views. The current online demo of GraFa supports English and Spanish; language can be switched at any time.

4 Indexing Scheme

The GraFa system is implemented on top of standard IR-style inverted indexes. More specifically, we base our indexing scheme on Apache Lucene (Core): a popular open source library offering various IR-style indexes, measures, etc.

[4] We originally implemented type ranks per frequency (number of results generated), but noted that certain popular types featured undesirably low in this ordering; for example, the type *country* has 207 entities, whereas *third-level administrative country subdivision* has 3792 entities. Hence we changed this ranking to consider PageRank.

```
SELECT ?p (COUNT(DISTINCT ?s) AS ?c)
WHERE {
 ?s wdt:P31 wd:Q5.        # human
 ?s wdt:P21 wd:Q6581097. # male
 ?s ?p ?o .
}
GROUP BY ?p
```

```
SELECT ?o (COUNT(?s) AS ?c)
WHERE {
 ?s wdt:P31 wd:Q5.        # human
 ?s wdt:P21 wd:Q6581097. # male
 ?s wdt:P106 ?o .        # occup
}
GROUP BY ?o
```

Fig. 1: Example SPARQL queries to compute facet properties and values over Wikidata; the left query would generate the facet properties and their frequencies for current results representing *male humans*; the right query would generate the facet values and their frequencies if the property *occupation* were then selected

Why not SPARQL? The first reason relates to the features supported, where GraFa requires keyword search, prefix search (for auto-completion), and ranking primitives; though SPARQL vendors often provide keyword search functionality, these are non-standard and cannot be easily configured; additionally ranking measures based on, for example, PageRank would need to be implemented by reordering (not top-k). Furthermore, to generate, rank and display facet properties and values, our index needs to be able to cope with aggregate queries such as shown in Figure 1; on the Wikidata Query Service running BlazeGraph, the left query times out, while the right query takes in the order of 37 seconds. In a locally built index on the same version of Wikidata that we use in our evaluation, Virtuoso requires 4 minutes for the left query and 16 seconds for the right query. Hence we build custom indexes on top of Lucene, offering us the required features such as keyword search, prefix search, ranking, etc.

Indexing schemes: We base our search on two (initial) inverted indexes:

– The *entity index* stores an entry (doc.) for each entity. Each entry stores fields to search entities by IRI, labels, description, type IRIs, property IRIs, and property–value pairs. The PageRank value of each entity is also stored.
– The *type index* stores an entry for each type. Each entry stores fields to search types by IRI and labels. The PageRank value of each type is also stored along with its frequency.

Note that types are also entities, and thus types will be included in both indexes. We use a separate types index to quickly find types according to an auto-complete prefix string; furthermore, the types index additionally contains the frequency of (number of entities associated with) a type. We highlight that properties are described by the entity index and are associated with labels, descriptions and defining properties (e.g., *sub-property-of*), etc.

Query processing: For each type of interaction, we perform the following:

– Keyword selection ($\kappa(q)$): we perform a keyword search on the labels and descriptions fields of the entity index.
– Type selection ($\tau(t)$):

  - Given a user-specified prefix (e.g., "`al*`") generated by an auto-complete request, we perform a prefix search on label field of the type index and return a list of labels, frequencies and IRIs for matching types.
  - Given a type IRI t selected by the user from the previous auto-complete options, we perform a lookup on the type field of the entity index.

– Facet generation/selection ($\phi \wedge \zeta(p, o)$, where ϕ generates current results E):

  - For the current result set E, we must generate all possible facet properties: their IRIs, labels and frequency *with respect to* E. We thus iterate over E and generate the required information from the property field.
  - Once a p is selected, we must generate all possible facet values: their IRIs, labels, frequency and PageRanks. Let $\epsilon(p)$ denote a query to find all entities with some value for property p executed over the *property* field of the entity index. We thus generate and execute the conjunctive query $\phi \wedge \epsilon(p)$ to find all entities in E with property p, and from these results we generate the list of all pertinent values.
  - Once a (p, o) is selected, we execute the conjunctive query $\phi \wedge \zeta(p, o)$.

To generate the results for any page (for keyword, type or facet selection), the first step of facet generation must be applied to generate the next possible steps.

Performance: Lucene implements efficient intersection algorithms to apply conjunctions. Hence performance issues rather occur when large sets of results are present and the facet selection must find (only) the properties present in E and their frequency with respect to E. For example, given a query $\tau(\mathit{human})$ in Wikidata, the above process would require scanning 3.6 million results and computing the frequencies of 358 properties. Next, when a property is selected to restrict E with, we may still have to scan many results to compute the available values for p in the set E (and their frequencies). For example, when we execute $\tau(\mathit{human}) \wedge \epsilon(\mathit{occupation})$, we would now need to scan 3.3 million results to find the values of occupation. Hence the challenge for performance is not due to the difficulty of query processing, but rather the amount of results generated. Under initial experiments with the above indexing scheme, generating the facet properties for type *human* took 135 seconds; furthermore, such queries are very common as an entry point onto the data. Hence we require optimisations.

5 Materialisation Strategy

To address the aforementioned performance issues, we propose a selective materialisation strategy. This strategy enumerates, off-line, all queries of the form $\tau(t)[\wedge \zeta(p_1, o_1) \wedge ... \wedge \zeta(p_n, o_n)]$ that generate greater than or equal to a given threshold α of results. More specifically, the goal is to identify all queries generating a high number ($\geq \alpha$) of results, such as $\tau(\mathit{human})$, or $\tau(\mathit{human}) \wedge \zeta(\mathit{gender}, \mathit{male})$, or $\tau(\mathit{human}) \wedge \zeta(\mathit{gender}, \mathit{male}) \wedge \zeta(\mathit{country}, \mathit{U.S.})$, etc.; the facet properties and values for these queries can then be materialised and indexed.

Choice of threshold: When selecting α, we are faced with a classical time–space trade-off: we should select a value for α such that queries generating fewer than α results can be processed efficiently using the base indexes, while there are as few as possible queries generating α results to avoid exploding the index. The underlying hypothesis here is that such a value of α exists, which is non-trivial and requires empirical validation (as we will provide in Section 6). We say that this is non-trivial since a relatively low value of α can generate a huge number of queries: let $\pi_{\text{PO}}(G) = \{(p, o) \mid \exists s : (s, p, o) \in G\}$ project the property–value facet pairs from G and let $\pi^*_{\text{PO}}(G)$ denote $\pi_{\text{PO}}(G)$ but removing pairs (p, o) where p is a type property. Recall that we denote by $T(G)$ the types of G. For $\alpha = 0$, we would have $|T(G)| \times 2^{|\pi^*_{\text{PO}}(G)|}$ possible queries to contend with containing every combination of type with the powerset of $\pi^*_{\text{PO}}(G)$. For $\alpha = 1$, we could still have the same number (if, e.g., G contains a single subject). More generally:

Lemma 1. *Let $\alpha \geq 1$. Given an RDF graph G with m triples, the total number of queries of the form $\tau(t)[\wedge \zeta(p_1, o_1) \wedge \ldots \wedge \zeta(p_n, o_n)]$ generating more than α results is bounded by the interval $[0, 2^{\lfloor \frac{m}{\alpha} \rfloor} - 1]$.*

Proof. If $|\pi_{\text{S}}(G)| < \alpha$, then no query can generate more than α results, giving the lower bound. Towards the upper-bound, let $\pi^{\alpha}_{\text{PO}}(G)$ denote the property–value pairs with more than α subjects and let $\Pi^{\alpha}_{\text{PO}}(G) \subseteq 2^{\pi^{\alpha}_{\text{PO}}(G)}$ denote all sets of such pairs that cooccur on more than α subjects; these are the queries we need to materialise. We now construct a worst-case G that maximises the value $|\Pi^{\alpha}_{\text{PO}}(G)|$ with a budget of m triples. To do this, for each subject in G, we will assign the same set of (pairwise distinct) property–value pairs $\{(\mathsf{p}_1, \mathsf{o}_1), \ldots, (\mathsf{p}_k, \mathsf{o}_k)\}$. In this case, $|\Pi^{\alpha}_{\text{PO}}(G)| = 2^k$, representing the powerset of the k property–value pairs. We then need to maximise k; given the inequality $k|\pi_{\text{S}}(G)| \leq m$ for m the budget of triples, we thus need to minimise the number of subjects $|\pi_{\text{S}}(G)|$. But we know that $|\pi_{\text{S}}(G)| \geq \alpha$, otherwise no queries return more than α results; hence we should set $|\pi_{\text{S}}(G)| = \alpha$, which gives us $k = \lfloor \frac{m}{\alpha} \rfloor$ and $|\Pi^{\alpha}_{\text{PO}}(G)| = 2^{\lfloor \frac{m}{\alpha} \rfloor}$. With respect to types, note that we can consider this as any other facet by, e.g., setting p_1 to a type property; the only modification required is to not consider the empty set in $\Pi^{\alpha}_{\text{PO}}(G)$, which leads us to the upper bound $2^{\lfloor \frac{m}{\alpha} \rfloor} - 1$. □

Algorithm: We outline the algorithm to compute the queries generating more than α results. Note that for brevity, we will consider type as a facet. Let $\sigma_{\text{S}=x}(G) := \{(s, p, o) \in G \mid x = s\}$ select the triples in G whose subject is x. In order to compute $\Pi^{\alpha}_{\text{PO}}(G)$ representing the set of all queries with at least α results, a naive algorithm would be to compute from each subject x the powerset of all its property–value pairs $2^{\pi_{\text{PO}}(\sigma_{\text{S}=x}(G))}$ containing at least one type property and then count these sets over all subjects, outputting those with a count of at least α. However, in a dataset such as Wikidata, some subjects have hundreds of property–value pairs, where the powerset for such a subject would be clearly unfeasible to materialise. Instead, we optimise for the fact that a property–value pair with fewer than α subjects can never appear in a conjunctive query with more than α subjects: we compute a restricted powerset $2^{\pi_{\text{PO}}(\sigma_{\text{S}=x}(G)) \cap \pi^{\alpha}_{\text{PO}}(G)}$ that only considers individual (p, o) pairs on each subject x with at least α subjects in

G. Thereafter, we can then count the number of subjects for each query and add those with more than α subjects to $\Pi^{\alpha}_{\text{PO}}(G)$. The number of queries generated is still, of course, potentially exponential, and hence it will be important to select a relative high value of α to minimise the set $\pi^{\alpha}_{\text{PO}}(G)$, and thus the exponent.

Indexing: For each query in $\Pi^{\alpha}_{\text{PO}}(G)$ computed in the previous stage, we compute its result set offline, and from that set, we compute the set of facet properties, their frequencies, and the sets of their values. Thus we have precomputed the information needed to generate the results page of each such query (with an index lookup), and to facilitate explorations of the facets on that page.

Keyword selections: Note that for $\kappa(q)$, given that the number of possible keyword queries q is not bounded, our materialisation approach is not applicable, where we rather simply restrict $\kappa(q)$ to return the top-α results.

6 Performance Evaluation

We now discuss the performance of indexing, materialisation and querying.

Data and Machine: We take the "truthy" dump of Wikidata from 2017/09/13, containing 1.77 billion triples and 74.1 million entities. However, given that we do not consider datatype values, nor labels and descriptions in other languages, the number of Wikidata triples used by GraFa is 195 million (120 million (p, o) pairs; 75 million labels and descriptions in English and Spanish). The machine used for all experiments has 2× Intel Xeon 4-Core E5-2609 V3 CPUs (@1.9GHz), 32GB of RAM, and 2× 2TB Seagate 7200 RPM 32MB Cache SATA hard-disks (RAID-1). The code used is available online: https://github.com/joseignm/GraFa/.

Threshold selection: The selection of the threshold α must find a balance: too high and queries just under the threshold will take too long to run; too low and the number of queries to materialise will explode exponentially. We choose three seconds as a reasonable worst-case response time, which from initial experiments suggested a value of $\alpha = 50,000$. To verify that this would not require materialising too many queries, we counted the subjects associated with each $(p, o) \in \pi_{\text{PO}}(G)$ and found that $\frac{149}{10,348,199} \approx 0.001\%$ of (p, o) pairs were associated with more than 50,000 subjects. Ultimately we materialise 141 queries.

Indexing times: In Table 1, we provide the details of all indexing times. The initial PageRank computation takes 04:30 (hh:mm) and creating the base indexes requires 06:52. Computing the set $\Pi^{\alpha}_{\text{PO}}(G)$ for α =50,000 took 04:16, while building an index of the properties and their frequency for each such query took 01:13. The most expensive step in the process is materialising the values of such properties, which took 107:18 (4.5 days), where, for each query, we need to build a list of all values for each property. This index of values contains 16,048 query–property keys in total (one for each facet property of a materialised query). An important question is then: is an index on values necessary or could it be optimised? Without indexing values, if a user selects a property on a materialised query with lots of results, where the majority of results have some value

Table 1: Times of all index-creation steps

Process	**Time (s)**	**Time (hh:mm)**
Computing PageRank	15,595	004:20
Creating base indexes	24,756	006:52
Identifying queries to materialise	15,382	004:16
Indexing properties	4,356	001:13
Indexing values	386,304	107:18

for that property, we may still require scanning all the results to generate the value list. For example, if the query is τ(`:Human`) (3.6 million results) and the user selects ϵ(`:occupation`) (3.3 million results), without an index for values, all people with some occupation must be scanned to generate all possible values for that property, which would again take minutes. However, some compromise may be possible to reduce this indexing time; one idea is to not materialise values for properties with a low frequency, where of the 358 properties associated with `:Human` for example, only 31 have more than α results; another idea is to index values for properties independent of the current query, thus potentially suggesting values that may lead to empty results (e.g., on a query for human males, suggesting *first lady* for *occupation*). For now, we simply accept the longer indexing time. On disk, the base index takes up 6GB of space, the properties index requires 5MB, while the values index requires 1GB.

Query performance testing: To test online query performance over these indexes, we created sequential queries simulating user sessions. Each session starts with $\tau(person)$, which offers a lot of results and facet properties; from this initial interaction, the index returns the top 50 ranked results and facet properties for all results. The session then randomly selects a property from the top 20 ordered by frequency ($\epsilon(p)$);[5] the system must then respond with the list of values for that property on the full result set. The session continues by selecting a random value ($\zeta(p, o)$); the system then generates the next results set and list of facet properties for that result set. This process is iterated until there is only one result or there is no further interaction possible, at which point the session terminates.

Query performance results: One thousand such sessions were executed. Figure 2 presents the response times for generating results pages with facet properties (τ and ζ queries), while Figure 3 presents the response times for selecting the values for a property (ϵ queries). These figures show times in milliseconds plotted against the number of results generated (entities or values, resp.); note that the x-axis of Figure 2 is presented in log-scale and the dashed vertical line indicates the selected value for the α-threshold. In the worst case, a query interaction takes approximately 3 seconds (for queries just below α), while value selection is possible in all cases under 500 ms. To the right of the α line, we see that

[5] We thus avoid the majority of facet properties with few results, selecting from those with the most results (and thus those more prominently displayed to users).

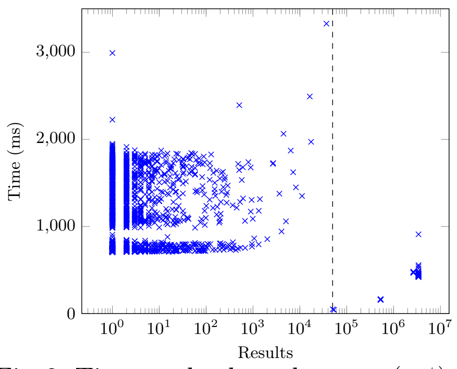

Fig. 2: Times to load result pages (τ,ζ)

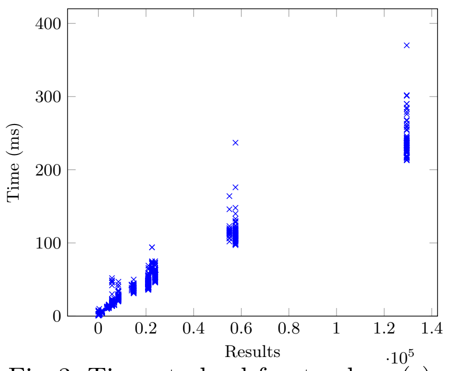

Fig. 3: Times to load facet values (ϵ)

materialised queries can be executed in under a second despite large result sizes; without materialisation, these queries took upwards of 2 minutes to process.

Data: Please note that we make evaluation data, queries, etc., available at https://github.com/joseignm/GraFa/tree/master/misc.

7 User Evaluation

While the previous section establishes performance results for indexing, materialisation and querying, we now present an initial usability study of the GraFa system. For this, we implemented a prototype of a user interface as a Java servlet with Javascript enabling interactive client-side features, such as auto-completion. A demo (for Wikidata) is available at http://grafa.dcc.uchile.cl.

User study design: We chose a task-driven user study where we give participants ten questions in natural language; for this, we selected the questions and question text from the example queries provided for the Wikidata Query Service (selecting examples answerable as faceted navigations).[6] We list the question text provided to the user and the expected queries they should generate in Table 2; these reflect the SPARQL query and its description in the source.

User study baseline: In order to establish a baseline for the tasks, we selected the Wikidata Query Helper (WQH) provided on the official Wikidata SPARQL Endpoint[7]; this interface first provides auto-completion on the labels of values and automatically proposes an associated property. For example, a user typing `"mal"` may be suggested *male organism*, *male*, etc.; upon selecting the latter, the property *sex or gender* is automatically selected, though it can be changed through another auto-completion dialogue. The user can add several property–value pairs in this manner. Suggestions generated through auto-completion are not restricted in a manner that assures non-empty results.

[6] https://wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples; retr. 2018/04/05.
[7] https://query.wikidata.org/; retr. 2018/04/05.

Table 2: User study tasks, with question text and expected query to be generated

Question Text	**Expected Query**
"Plays"	$\tau(\textit{plays})$
"Lakes in Cameroon"	$\tau(\textit{lake}) \wedge \zeta(\textit{country, Cameroon})$
"Lighthouses in Norway"	$\tau(\textit{lighthouse}) \wedge \zeta(\textit{country, Norway})$
"Popes"	$\tau(\textit{human}) \wedge \zeta(\textit{position held, Pope})$
"Women born in Wales"	$\tau(\textit{human}) \wedge \zeta(\textit{gender, female}) \wedge \zeta(\textit{place-of-birth, Wales})$
"Papers about Wikidata"	$\tau(\textit{scientific article}) \wedge \zeta(\textit{main subject, Wikidata})$
"Law & Order episodes"	$\tau(\textit{TV series episode}) \wedge \zeta(\textit{series, Law \& Order})$
"Fictional characters from Marvel Universe"	$\tau(\textit{fictional character}) \wedge \zeta(\textit{from fictional universe, Marvel Universe})$
"People dying by burning"	$\tau(\textit{human}) \wedge \zeta(\textit{manner of death, death by burning})$
"Mosquito species"	$\tau(\textit{taxon}) \wedge \zeta(\textit{parent-taxon, Culicidae}) \wedge \zeta(\textit{taxon-rank, species})$

Participants and instructions: We attained 11 volunteers (students of a Semantic Web course) for a study. Given the question text, we asked the volunteers to use either GraFa or WQH (switching on every second question) to find the results and submit the URL, or click skip if they felt unable to find the results; the next task would then be loaded. Half of the participants began with GraFa and the other half with WQH. They were not instructed on how to use either of the two systems. Afterwards they responded to a brief questionnaire.

User task results: We collected results for 55 tasks per system ($\frac{10\times11}{2}$). Of these, $\frac{23}{55} \approx 42\%$ were solved correctly in GraFa, while $\frac{37}{55} \approx 67\%$ were solved correctly in WQH. This was unambiguously a negative result for GraFa. Investigating the errors further, for GraFa (32 errors), 10 involved users typing questions directly into the keyword-query text field rather than using type selection as intended; 3 involved selecting incorrect types/facets/values; 19 responses were skipped/left blank/invalid. On the other hand, for WQH (18 errors), 11 responses selected incorrect types/facets/values, while 7 were left blank. Through this study we found a variety of interface issues that we subsequently fixed. We additionally realised that users had a difficult time starting with a type selection; an example is "Popes" where users typed "pope" into the GraFa type selection (rather than "human" or "person"); on the other hand, in WQH, typing "pope" in the value selection suggested the value *Pope* and, upon selection, the correct property *position held*. On the other hand, in WQH, users sometimes selected the incorrect property, where for a query such as *Women born in Wales*, neither the value *woman* nor *Wales*, when selected, suggests the correct property.

User questionnaire: After the task, we asked users to answer a brief questionnaire rating the responsiveness and usability of both systems on a Likert 1–7 scale; users rated GraFa with a mean of 4.5/7 for usability and 4.7/7 for responsiveness; WQH had an analogous mean rating of 5.5/7 for usability and 6.0/7 for responsiveness. Again in this case WQH scored considerably higher than GraFa. Regarding responsiveness, with subsequent investigation we found that the Javascript libraries for auto-completion were creating lag in the client browser, where we implemented smaller thresholds for suggestions.

Community questionnaire: Based on the results of this user study, we fixed a number of interface issues in the system, blocking on the selection of a type or

Table 3: Responses to Wikidata community questionnaire

Statement	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Mean**
The system is useful	0	0	0	2	2	4	1	5.4/7
The system offers a novel way to query Wikidata	0	0	0	1	3	4	1	5.6/7
The system is usable	0	1	1	2	2	1	2	4.8/7
Knowledge of Wikidata is not required	0	1	1	2	3	1	1	4.6/7
Load-times did not affect interactivity	0	0	2	2	2	1	2	4.9/7
Ranking of results is intuitive	0	0	1	2	2	4	0	5.0/7
Ranking of facets is intuitive	0	0	3	1	2	2	1	4.7/7

value suggestion, separating type/keyword selection in the interface and so forth. We created a questionnaire that we sent to the Wikidata mailing list asking to try the GraFa system and then answer a set of 12 questions, where we received nine responses. The results of the questionnaire are presented in Table 3, where most responses were moderately positive about the system. We further asked if they would use the system in future (YES|MAYBE|NO): 4 said YES, while 5 said MAYBE. We made some further improvements based on text comments received, such as to add placeholder examples in the text fields for auto-suggestions.

8 Conclusion

Motivated by the goal of providing users with a faceted interface over Wikidata – and the lack of current techniques and tools by which this could be achieved – in this paper, we have presented methods to enable faceted browsing over large-scale, diverse RDF graphs. A key contribution is our proposed materialisation strategy, which identifies facet queries that are good candidates for indexing. With this technique, worst-case response times drop from minutes to seconds at the cost of increased indexing time. To the best of our knowledge, GraFa is the only faceted browsing system demonstrated at this level of scale while filtering suggestions that would lead to empty results. With the current system, the faceted browser could be updated for Wikidata on a weekly basis.

On the other hand, the results of our usability experiments were mixed: GraFa was outperformed by the legacy WQH system in our task-driven user study. Some superficial issues were then fixed, such as blocking auto-complete fields until a selection is made. Though the results were more negative than hoped, we also drew more general conclusions, key amongst which is that, in a diverse graph like Wikidata, users unfamiliar with the dataset may struggle to select types, properties and values corresponding to their intent (e.g., is a *pope* a type or a value?; is *fictional character* a property or a type?). After some improvements to the system, a questionnaire issued to the Wikidata community generated moderately positive results regarding usefulness, novelty, usability, etc.

There are various directions in which this work could be continued. An important aspect for improvement is usability, where based on the aforementioned user study, we conclude that the system should offer more flexible selections; e.g., to automatically detect that *pope* is a value, not a type. The system could also be extended to support more expressive queries, such as ranges on datatype values, value selections, inverses, nested facets, and so forth. Other features – such as reasoning – would yield further challenges at the proposed scale. Furthermore, indexing time is currently prohibitive: investigating incremental indexing schemes would be an important practical contribution. Another important next step would be performing evaluations for other RDF datasets.

In conclusion, although there are various avenues for future work in terms of performance, expressiveness and usability, we hope that by enabling faceted browsing over RDF graphs at new levels of scale, GraFa already makes a significant step towards making the Semantic Web more accessible to end users.

Acknowledgements The work was supported by the Millennium Institute for Foundational Research on Data (IMFD) and by Fondecyt Grant No. 1181896.

References

1. Abel, F., Celik, I., Houben, G., Siehndel, P.: Leveraging the Semantics of Tweets for Adaptive Faceted Search on Twitter. In: International Semantic Web Conference (ISWC). pp. 1–17 (2011)
2. Araújo, S., Schwabe, D.: Explorator: A tool for exploring RDF data through direct manipulation. In: Workshop on Linked Data on the Web (LDOW) (2009)
3. Arenas, M., Grau, B.C., Kharlamov, E., Marciuska, S., Zheleznyakov, D.: Faceted search over RDF-based knowledge graphs. J. Web Sem. 37-38, 55–74 (2016)
4. Bast, H., Bäurle, F., Buchhold, B., Haußmann, E.: Easy access to the Freebase dataset. In: International World Wide Web Conference (WWW). pp. 95–98 (2014)
5. Bast, H., Buchhold, B.: An index for efficient semantic full-text search. In: Conference on Information and Knowledge Management (CIKM). pp. 369–378 (2013)
6. Brunetti, J.M., González, R.G., Auer, S.: From overview to facets and pivoting for interactive exploration of Semantic Web data. IJSWIS 9(1), 1–20 (2013)
7. Dadzie, A., Rowe, M.: Approaches to visualising Linked Data: A survey. Semantic Web 2(2), 89–124 (2011)
8. Ferré, S.: Expressive and scalable query-based faceted search over SPARQL endpoints. In: International Semantic Web Conference (ISWC). pp. 438–453 (2014)
9. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem. 3(2-3), 158–182 (2005)
10. Hahn, R., Bizer, C., Sahnwaldt, C., Herta, C., Robinson, S., Bürgle, M., Düwiger, H., Scheel, U.: Faceted Wikipedia Search. In: Business Information Systems (BIS). pp. 1–11 (2010)
11. Harth, A.: Visinav: A system for visual search and navigation on web data. J. Web Sem. 8(4), 348–354 (2010)
12. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers (2011)
13. Heim, P., Ertl, T., Ziegler, J.: Facet graphs: Complex semantic querying made easy. In: The Semantic Web (ESWC). pp. 288–302 (2010)
14. Heim, P., Ziegler, J., Lohmann, S.: gfacet: A browser for the web of data. In: International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW) (2008)

15. Hildebrand, M., van Ossenbruggen, J., Hardman, L.: /facet: A browser for heterogeneous Semantic Web repositories. In: International Semantic Web Conference (ISWC). pp. 272–285 (2006)
16. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. Artif. Intell. 194, 28–61 (2013)
17. Kamdar, M.R., Zeginis, D., Hasnain, A., Decker, S., Deus, H.F.: Reveald: A user-driven domain-specific interactive search platform for biomedical research. Journal of Biomedical Informatics 47, 112–130 (2014)
18. Karger, D.R.: The Semantic Web and End Users: What's Wrong and How to Fix It. IEEE Internet Computing 18(6), 64–70 (2014)
19. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web 6(2), 167–195 (2015)
20. Mäkelä, E., Hyvönen, E., Saarela, S.: Ontogator - A semantic view-based search engine service for web applications. In: International Semantic Web Conference (ISWC). pp. 847–860 (2006)
21. Oren, E., Delbru, R., Decker, S.: Extending faceted navigation for RDF data. In: International Semantic Web Conference (ISWC). pp. 559–572 (2006)
22. Petzka, H., Stadler, C., Katsimpras, G., Haarmann, B., Lehmann, J.: Benchmarking faceted browsing capabilities of triplestores. In: International Conference on Semantic Systems (SEMANTICS). pp. 128–135 (2017)
23. schraefel, m., Wilson, M., Russell, A., Smith, D.A.: mSpace: improving information access to multimedia domains with multimodal exploratory search. Commun. ACM 49(4), 47–49 (2006)
24. Sherkhonov, E., Grau, B.C., Kharlamov, E., Kostylev, E.V.: Semantic faceted search with aggregation and recursion. In: International Semantic Web Conference (ISWC). pp. 594–610 (2017)
25. Stadler, C., Martin, M., Auer, S.: Exploring the web of spatial data with facete. In: International World Wide Web Conference (WWW). pp. 175–178 (2014)
26. Tvarožek, M., Bieliková, M.: Generating exploratory search interfaces for the Semantic Web. In: Human-Computer Interaction Symposium (HCIS). pp. 175–186 (2010)
27. Tzitzikas, Y., Bailly, N., Papadakos, P., Minadakis, N., Nikitakis, G.: Using preference-enriched faceted search for species identification. IJMSO 11(3), 165–179 (2016)
28. Tzitzikas, Y., Manolis, N., Papadakos, P.: Faceted exploration of RDF/S datasets: a survey. J. Intell. Inf. Syst. 48(2), 329–364 (2017)
29. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM 57(10), 78–85 (2014)
30. Wagner, A., Ladwig, G., Tran, T.: Browsing-oriented semantic faceted search. In: Database and Expert Systems Applications (DEXA). pp. 303–319 (2011)
31. Wang, H., Liu, Q., Penin, T., Fu, L., Zhang, L., Tran, T., Yu, Y., Pan, Y.: Semplore: A scalable IR approach to search the web of data. J. Web Sem. 7(3), 177–188 (2009)
32. Wei, B., Liu, J., Zheng, Q., Zhang, W., Fu, X., Feng, B.: A survey of faceted search. J. Web Eng. 12(1&2), 41–64 (2013)
33. Yahav, T., Kalinsky, O., Mishali, O., Kimelfeld, B.: eLinda: Explorer for Linked Data. In: International Conference on Extending Database Technology (EDBT). pp. 658–661 (2018)