ARTICLE IN PRESS

Web Semantics: Science, Services and Agents on the World Wide Web ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

# On the formulation of performant SPARQL queries

Antonis Loizou [a,*], Renzo Angles [b], Paul Groth [a]

[a] Department of Computer Science, VU University of Amsterdam, The Netherlands
[b] Department of Computer Science, Universidad de Talca, Chile

ARTICLE INFO

*Article history:*
Received 25 February 2014
Received in revised form
20 June 2014
Accepted 7 November 2014
Available online xxxx

*Keywords:*
SPARQL
Heuristics
Optimisation
RDF store
Data integration
Biomedical data

ABSTRACT

The combination of the flexibility of RDF and the expressiveness of SPARQL provides a powerful mechanism to model, integrate and query data. However, these properties also mean that it is nontrivial to write performant SPARQL queries. Indeed, it is quite easy to create queries that tax even the most optimised triple stores. Currently, application developers have little concrete guidance on how to write "good" queries. The goal of this paper is to begin to bridge this gap. It describes 5 heuristics that can be applied to create optimised queries. The heuristics are informed by formal results in the literature on the semantics and complexity of evaluating SPARQL queries, which ensures that queries following these rules can be optimised effectively by an underlying RDF store. Moreover, we empirically verify the efficacy of the heuristics using a set of openly available datasets and corresponding SPARQL queries developed by a large pharmacology data integration project. The experimental results show improvements in performance across six state-of-the-art RDF stores.



## 1. Introduction

Since the release of the Resource Description Framework (RDF) as a W3C Recommendation in 1999 [1,2], the amount of data published in various RDF serialisations has been rapidly increasing. Sindice[1] currently indexes 15+ billion triples [3,4]. The Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch[2] provides a striking visualisation of the diversity of domains that this data covers. The query language SPARQL [5] and SPARQL 1.1 update [6] is the W3C Recommendation for querying RDF data.

The flexibility in terms of both data structures and vocabularies make RDF and Linked Open Data attractive from a data provider perspective, but poses significant challenges in formulating correct, complex and performant SPARQL queries [7].[3,4] Application developers need to be familiar with various data schemas, cardinalities, and query evaluation characteristics in order to write effective SPARQL queries [8].

The contribution of this paper is a set of heuristics that can be used to formulate complex, but performant SPARQL queries to be evaluated against a number of RDF datasets. The heuristics are grounded in our experience in developing the OpenPHACTS[5] Platform [9]—a platform to facilitate the integration of large pharmaceutical datasets. The efficiency of the SPARQL query templates obtained by applying these heuristics is evaluated on a number of widely used RDF stores and contrasted to that of baseline queries.

The rest of the paper is organised as follows. Section 2 gives the context and motivation for this work, while Section 3 introduces the SPARQL syntax and semantics. Section 4 presents the five heuristics. An empirical comparison of the performance of SPARQL queries optimised using the defined heuristics is provided in Section 5. Section 6 discusses the inherent difficulties in providing paginated RDF views and how these can be addressed through some of the heuristics defined in this paper. A brief overview of related work is provided in Section 7. Finally, we provide concluding remarks in Section 8.

## 2. Motivation and context

The work presented in this paper was carried out in the context of the OpenPHACTS project [10], a collaboration of research

* Corresponding author.
*E-mail addresses:* a.loizou@vu.nl (A. Loizou), rangles@utalca.cl (R. Angles), p.t.groth@vu.nl (P. Groth).

[1] http://sindice.com.

[2] http://lod-cloud.net/.

[3] 'YarcData: Tuning SPARQL Queries for Performance', see: http://yarcdata.com/blog/?p=312.

[4] 'YarcData: Dont use a hammer to screw in a nail: Alternatives to REGEX in SPARQL', see: http://yarcdata.com/blog/?p=811.

[5] http://www.openphacts.org.

http://dx.doi.org/10.1016/j.websem.2014.11.003

institutions and major pharmaceutical companies. The key goal of the project is to support a variety of common tasks in drug discovery through a technology platform that integrates pharmacological and other biomedical research data using Semantic Web technologies. In order to achieve this goal, the platform must tackle the problem of public domain data integration in the pharmacology space and provide efficient access to the resulting integrated data. The development of the OpenPHACTS platform is driven by a set of concrete research questions presented in [11]. Its architecture is described in [9].

In the context of OpenPHACTS, the decision was made to avoid pushing the burden of performant query formulation to developers, but instead to provide them with an API driven by parameterised SPARQL queries [12]. This in turn created a need to formulate a set of performant query templates that are instantiated through API requests. The information that should be returned by such queries is first defined by domain experts, and subsequently the API developers are tasked with formulating a performant query template to satisfy the requirements given. The work presented in this paper stems largely from our experiences in hand-crafting such query templates. The heuristics rely on established SPARQL algebra equivalences, but in some cases also require extensive data statistics. We argue that the large overhead associated with the latter is justified by the associated increase in query performance and the need to repeatedly evaluate queries obtained from the same template.

A large body of work has been carried out on defining formal semantics for RDF and SPARQL in order to analyse query complexity and provide upper and lower bounds for generic SPARQL constructs [13–21]. These approaches are mainly focused on exploiting the formal semantics of SPARQL in order to prove generic rewrite rules for SPARQL patterns that are used in order to evaluate equivalence or subsumption between (sets of) queries. While a more detailed overview of the various SPARQL formalisation and optimisation techniques is provided in the next section, we note here that while the findings of these studies are invaluable to better understand the complexity of evaluating SPARQL queries and provide solid foundations for designing RDF store query planners and optimisers, *the issue of query formulation is not addressed*.

In contrast, the work presented here provides a set of heuristics to be used in formulating performant SPARQL queries based on concrete application requirements and known dataset schemata. The goal is to identify patterns that can be used to formulate queries that can be effectively optimised by a wide range of RDF stores. To that end, we provide a comparison on the performance of six state-of-the-art RDF storage systems with respect to the various query formulation techniques in order to study their effectiveness and applicability. As the implementation details of each system, the indices that are available and the manner in which the indices are used vary greatly across the different systems, investigating why one may outperform another is considered outside the scope of this work. Instead, we report on the efficacy of the heuristics across the different systems, considering each RDF store as a black box.

In summary, the paper has four main contributions:

1. A mapping between formal results published in the literature and SPARQL syntax.
2. A set of heuristics through which performant SPARQL queries can be formulated based on application requirements.
3. Guidance for RDF store selection based on the formulated SPARQL queries.
4. A reference set of queries and openly available datasets.

## 3. Preliminaries

In this section, we introduce the SPARQL query language by following the syntax used in [14,13,22], which is better suited to do formal analysis than the syntax presented by the W3C specification. The terminology given here is adopted for the remainder of this paper.

### 3.1. RDF datasets

Assume there are pairwise disjoint infinite sets **I** (IRIs), **B** (Blank nodes), and **L** (Literals). An *RDF term* is an element in the set $\mathbf{T} = \mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$. Given any structure $\alpha$, we denote by $\text{iri}(\alpha)$, $\text{blank}(\alpha)$, $\text{literal}(\alpha)$ and $\text{term}(\alpha)$ the set of IRIs, blank nodes, literals and terms occurring in $\alpha$ respectively.

A tuple $(v_1, v_2, v_3) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times \mathbf{T}$ is called an *RDF triple*, where $v_1$ is the *subject*, $v_2$ the *predicate*, and $v_3$ the *object*. An *RDF Graph* (just graph from now on) is a set of RDF triples. The *union* of graphs, $G_1 \cup G_2$, is the set theoretical union of their sets of triples.

An *RDF dataset* $D$ is a set $\{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$ where each $G_i$ is a graph and each $u_j$ is an IRI. $G_0$ is called the *default graph* of $D$. Each pair $\langle u_i, G_i\rangle$ is called a *named graph*; define $\text{name}(G_i)_D = u_i$ and $\text{graph}(u_i)_D = G_i$. The set of IRIs $\{u_1, \ldots, u_n\}$ is denoted $\text{names}(D)$. Every dataset satisfies that: (i) it always contains one default graph (which could be empty); (ii) there may be no named graphs; (iii) each $u_j \in \text{names}(D)$ is distinct; and (iv) $\text{blank}(G_i) \cap \text{blank}(G_j) = \emptyset$ for $i \neq j$. Finally, the *active graph* of $D$ is the graph $G_i$ used for querying $D$.

### 3.2. The SPARQL query language

**SPARQL syntax**

Assume the existence of an infinite set **V** of variables disjoint from **T**. We denote by $\text{var}(\alpha)$ the set of variables occurring in the structure $\alpha$.

A *SPARQL select query*[6] (or just *query* from now on) is a tuple $Q = (W, F, P)$ where $W$ is a set of variables (the symbol $*$ can be used to express "all variables"), $F$ is a set-possibly empty-of dataset clauses, and $P$ is a graph pattern. Next we define each component.

A *dataset clause* is either an expression FROM $u$ or FROM NAMED $u$ where $u \in \mathbf{I}$. The set of dataset clauses is used to define the RDF dataset used by the query.

A *graph pattern* is defined recursively as follows:

- The expression () is a graph pattern called the empty graph pattern.
- A tuple from $(\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$ is a graph pattern called a *triple pattern*.
- If $P_1$ and $P_2$ are graph patterns then ($P_1$ AND $P_2$), ($P_1$ UNION $P_2$), and ($P_1$ OPT $P_2$) are graph patterns.
- If $P_1$ is a graph pattern and $u \in \mathbf{I} \cup \mathbf{V}$ then ($u$ GRAPH $P_1$) is a graph pattern.
- If $P_1$ is a graph pattern then ($P_1$ FILTER $C$) is a graph pattern, where $C$ is a *filter constraint* which is defined recursively as follows: (i) If $?X, ?Y \in \mathbf{V}$ and $u \in \mathbf{I} \cup \mathbf{L}$, then $?X = u$ and $?X = ?Y$ are *atomic filter constraints*.[7] (ii) If $C_1$ and $C_2$ are filter constraints then $(\neg C_1)$, $(C_1 \wedge C_2)$ and $(C_1 \vee C_2)$ are filter constraints.

In this paper, we will assume that every query $Q = (W, F, P)$ satisfies the following conditions:

- If $?X \in \text{var}(W)$ then $?X \in \text{var}(P)$. (*Safe result condition.*)

[6] In this paper, we restrict our study to Select queries and we do not consider solution modifiers.

[7] For a complete list of atomic filter constraints see [6].

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

- For every graph pattern of the form $(P_1$ FILTER $C)$ in $P$, if $?X \in \text{var}(C)$ then $?X \in \text{var}(P_1)$. (*Safe filter condition.*)
- For every graph pattern of the form $P' = (P_1$ OPT $P_2)$ in $P$, all the variables that occur both inside $P_2$ and outside $P'$ also occur in $P_1$. (*Well-designed graph patterns* [13].)

The terms *inside* and *outside* a graph pattern refer to a syntactic check of whether an element of $(\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$ appears within the specified graph pattern throughout this paper.

**SPARQL semantics**

A *mapping* $\mu$ is a partial function $\mu : \mathbf{V} \to \mathbf{T}$. The domain of $\mu$, $\text{dom}(\mu)$, is the subset of $\mathbf{V}$ where $\mu$ is defined. The *empty mapping* $\mu_\emptyset$ is a mapping such that $\text{dom}(\mu_\emptyset) = \emptyset$. Two mappings $\mu_1, \mu_2$ are *compatible*, denoted $\mu_1 \sim \mu_2$, when for all $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ it satisfies that $\mu_1(?X) = \mu_2(?X)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping. Note that two mappings with disjoint domains are always compatible and that the empty mapping $\mu_\emptyset$ is compatible with every other mapping.

Given a finite set of variables $W \subset \mathbf{V}$, the restriction of a mapping $\mu$ to $W$, denoted $\mu_{|W}$, is a mapping $\mu'$ satisfying that $\text{dom}(\mu') = \text{dom}(\mu) \cap W$ and $\mu'(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu) \cap W$. The expression $\mu_{?X \to a}$ denotes a mapping where $\text{dom}(\mu) = \{?X\}$ and $\mu(?X) = a$.

A *multiset of solution mappings* is denoted by $\Omega$. We use $\Omega_\emptyset$ to denote the set consisting of exactly the empty mapping $\mu_\emptyset$. $\Omega_\emptyset$ is the join identity.

Given a mapping $\mu$ and a filter constraint $C$, we say that $\mu$ satisfies $C$, denoted $\mu \models C$, if:

- $C$ is $?X = u$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = u$;
- $C$ is $?X = ?Y$, $?X \in \text{dom}(\mu)$, $?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- $C$ is $(\neg C_1)$ and it is not the case that $\mu \models C_1$;
- $C$ is $(C_1 \wedge C_2)$, $\mu \models C_1$ and $\mu \models C_2$;
- $C$ is $(C_1 \vee C_2)$, and $\mu \models C_1$ or $\mu \models C_2$.

Let $\Omega_1, \Omega_2$ be sets of mappings, $C$ be a filter constraint, and $W$ be a set of variables. The SPARQL algebra is defined by the following operations over solution mappings:

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \mu_1 \sim \mu_2\}$$

$$\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}$$

$$\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall \mu_2 \in \Omega_2, \mu_1 \nsim \mu_2\}$$

$$\Omega_1 \mathbin{⟕} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$$

$$\sigma_C(\Omega_1) = \{\mu \in \Omega_1 \mid \mu \models C\}$$

$$\pi_W(\Omega_1) = \{\mu_{|W} \mid \mu \in \Omega_1\}.$$

The evaluation of a graph pattern $P$ over an RDF dataset $D$ having active graph $G$, denoted $[\![P]\!]^D_G$ (or $[\![P]\!]$ where $D$ and $G$ are clear from the context), is defined recursively as follows:

- $[\![()]\!]^D_G = \Omega_\emptyset$
- if $t$ is a triple pattern then $[\![t]\!]^D_G = \{\mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in G\}$
- $[\![(P_1 \text{ AND } P_2)]\!]^D_G = [\![P_1]\!]^D_G \bowtie [\![P_2]\!]^D_G$
- $[\![(P_1 \text{ UNION } P_2)]\!]^D_G = [\![P_1]\!]^D_G \cup [\![P_2]\!]^D_G$
- $[\![(P_1 \text{ OPT } P_2)]\!]^D_G = [\![P_1]\!]^D_G \mathbin{⟕} [\![P_2]\!]^D_G$
- $[\![(P_1 \text{ FILTER } C)]\!]^D_G = \sigma_C([\![P_1]\!]^D_G)$
- $[\![(u \text{ GRAPH } P_1)]\!]^D_G = [\![P_1]\!]^D_{\text{graph}(u)_D}$
- $[\![(?X \text{ GRAPH } P_1)]\!]^D_G = \bigcup_{v \in \text{names}(D)} ([\![P_1]\!]^D_{\text{graph}(v)_D} \bowtie \{\mu_{?X \to v}\})$.

Given a set of dataset clauses $F$, the evaluation of $F$ returns an RDF dataset which contains: (i) a default graph consisting of the merge of the graphs referred to in the FROM $u$ clauses, or the empty graph where there are no FROM $u$ clauses; and (ii) a named graph $\langle \text{graph}(u), u \rangle$ for each clause FROM NAMED $u$. We denote by $DS(F)$ the RDF dataset obtained from $F$.

Let $Q = (W, F, P)$ be a SPARQL select query, $D = DS(F)$ be the dataset of $Q$, and $G_0$ be the default graph of $D$. The evaluation of $Q$, denoted $\text{eval}(Q)$, is defined as

$$\text{eval}(Q) = \pi_{|W}([\![P]\!]^D_{G_0})$$

### 3.3. SPARQL view templates

For ease of presentation, the heuristics described below assume a particular style of SPARQL queries. A resource oriented approach is used, whereby each SPARQL query must return information related to a single *resource*; i.e. an element of the set **I**. In turn, different sets of information may be required for the same type of resource; we refer each to of these sets as a view template of the resource.

**Definition 3.1.** A *view template* is a pair $T = (Q, S)$ where $Q = (W, F, P)$ is a select query and $S$ is a set of variables, called *substitution variables*, satisfying that $S \subseteq \text{var}(P)$.

Let $T = (Q, S)$ be a view template and $\mu$ be a mapping such that $\text{dom}(\mu) \subseteq S$. The *instantiation* of the template $T$ according to the mapping $\mu$, denoted $\text{inst}(T, \mu)$, returns a query $Q'$ such that $Q'$ is a copy of $Q = (W, P, F)$ where, for each triple pattern $t$ in $P$, if $t$ contains a variable $?X \in S \cap P$ then $t$ is replaced by $(t \text{FILTER}(?X = \mu(?X)))$.

## 4. Heuristics—description and formal definition

We summarise the heuristics as follows and then explain them in more detail in the following sections.

- *H1: Minimise optional graph patterns*: Reduce the number of optional triple patterns by identifying those triple patterns for a given query that will always be bound using dataset statistics.
- *H2: Use named graphs to localise SPARQL sub-graph patterns*: Use named graphs to specify the subset of triples in a dataset that portions of a query should be evaluated against.
- *H3: Reduce intermediate results*: Use sequence paths to replace connected triple patterns where the object of one triple pattern is the subject of another.
- *H4: Reduce the effects of cartesian products*: Use aggregates to reduce the size of solution sequences.
- *H5: Specifying alternative URIs*: Consider different ways of specifying alternative URIs beyond `UNION`.

Note that these heuristics can be applied to multiple types of SPARQL queries, however the assumption that each query return information about a single resource according to a given view template allows us to guarantee the termination of the algorithms proposed and to provide succinct definitions. We now look at each heuristic in detail.

### 4.1. H1: Minimise optional graph patterns

Since real world datasets will often contain missing values, view templates must also allow for optional elements. As shown in the literature, SPARQL graph pattern expressions given by the conjunction of triple patterns and built-in conditions can be evaluated in $O(|P| \cdot |D|)$ time, where $|P|$ is number of triple patterns in the query and $|D|$ is the number of RDF tuples in a dataset $D$. However, by adding the OPT operator, evaluation complexity becomes coNP-complete for well-designed graph pattern expressions. It is thus desirable to minimise the number of optional elements in a view template.

The heuristic for minimising optional triple patterns is based on the following equivalence.

**Theorem 4.1.** *Let $P_1, P_2$ be graph patterns and $D$ be an RDF dataset with active graph $G$. The graph patterns $(P_1$ OPT $P_2)$ and $(P_1$ AND $P_2)$ are equivalent when $[\![P_1]\!]^D_G \setminus [\![P_2]\!]^D_G = \emptyset$.*

**Proof.** Let $\Omega_1 = [\![P_1]\!]^D_G$ and $\Omega_2 = [\![P_2]\!]^D_G$. We have that $[\![(P_1 \text{ OPT } P_2)]\!]^D_G = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$ and $[\![(P_1 \text{ AND } P_2)]\!]^D_G = \Omega_1 \bowtie \Omega_2$. If we have that $[\![P_1]\!]^D_G \setminus [\![P_2]\!]^D_G = \emptyset$, that is $\Omega_1 \setminus \Omega_2 = \emptyset$, then it applies that $[\![(P_1 \text{ OPT } P_2)]\!]^D_G = (\Omega_1 \bowtie \Omega_2) \cup \emptyset$. Hence, $[\![(P_1 \text{ OPT } P_2)]\!]^D_G$ and $[\![(P_1 \text{AND} P_2)]\!]^D_G$ are equivalent. □

Given a query $Q = (W, F, P)$ and the dataset $D = DS(F)$. The function $\mathscr{F}_1(P, D)$, defined by Algorithm 1, evaluates recursively the graph pattern $P$ and, for each graph pattern $P'$ of the form $(P_1 \text{ OPT } P_2)$ satisfying that $[\![P_1]\!]^D_G \setminus [\![P_2]\!]^D_G = \emptyset$, the function replaces $P'$ by the graph pattern $(P_1 \text{ AND } P_2)$. Therefore, $Q$ can be transformed into the query $Q' = (W, F, \mathscr{F}_1(P, D))$ such that $Q'$ contains a minimal number of optional graph patterns.

**Algorithm 1** $\mathscr{F}_1(P, D)$

**Require:** Let $P$ be a graph pattern and $D$ be a dataset.
**Ensure:** $\mathscr{F}_1$ returns a graph pattern.
1: **if** $P$ is a triple pattern **then return** $P$
2: **if** $P = (P_1 \text{ AND } P_2)$ **then**
3: **return** $(\mathscr{F}_1(P_1, D) \text{ AND } \mathscr{F}_1(P_2, D))$
4: **if** $P = (P_1 \text{ UNION } P_2)$ **then**
5: **return** $(\mathscr{F}_1(P_1, D) \text{ UNION } \mathscr{F}_1(P_2, D))$
6: **if** $P = (P_1 \text{ OPT } P_2)$ **then**
7: $P'_1 \leftarrow \mathscr{F}_1(P_1, D)$
8: $P'_2 \leftarrow \mathscr{F}_1(P_2, D)$
9: **if** $[\![P'_1]\!]^D_G \setminus [\![P'_2]\!]^D_G = \emptyset$ **then**
10: **return** $(P'_1 \text{ AND } P'_2)$
11: **else**
12: **return** $(P'_1 \text{ OPT } P'_2)$
13: **if** $P = (u \text{ GRAPH } P_1)$ **then**
14: **return** $(u \text{ GRAPH } \mathscr{F}_1(P_1, D))$

The algorithm assesses whether an optional graph pattern should be eliminated from a query by evaluating the set difference between the results from the remaining patterns and the results from the optional pattern in consideration. As such, it poses a considerable overhead. Considering the dramatic increase in query performance achieved through the application of this heuristic (see Section 5.3) however, we argue that this overhead is justified.

### 4.2. H2: use named graphs to localise SPARQL sub-graph patterns

The run-time performance of any SPARQL query has a positive correlation to the number of RDF triples it is evaluated against. For example, consider the query:

```
SELECT *
FROM g1
FROM g2
WHERE { ?X p ?Y . ?X q ?Z }
```

Note that the cost of evaluating each triple pattern in the above query is given by the size of the default graph (formed by the merge of the graphs g1 and g2). Now, assume that the predicate p occurs in both graphs g1 and g2, and predicate q only occurs in g2. Then, the above query could be rewritten as:

```
SELECT *
FROM g1
FROM g2
FROM NAMED g2
WHERE { ?X p ?Y . { g2 GRAPH { ?X q ?Z } } }
```

In this case, we use the GRAPH operator to reduce the cost of evaluating the triple pattern `{ ?X q ?Z }` to the size of the named graph g2. Note that, the FROM clauses (whose merge is the default graph) remain unchanged in order to preserve the results for the triple pattern `?X p ?Y`. Therefore, the use of named graphs and the GRAPH operator provides an effective way to restrict the data against the triple pattern is matched. This approach is very useful when the dataset of the query collate together information from different RDF graphs, and specific graph patterns match against specific RDF graphs.

Embedding graph patterns inside GRAPH clauses can allow RDF store optimisers to consider a smaller set of triples in evaluating individual sub-graph patterns. Thus, the localisation of SPARQL sub-graph patterns in this manner is expected to reduce the complexity of the evaluation problem, and result in performance improvements.

In this section, we provide a method to the approach described above. Our method holds for queries satisfying that the target dataset just contains the default graph (i.e. it does not contains FROM NAMED clauses), and consequently it does not contain ($u$ GRAPH $P$) graph patterns.

**Algorithm 2** $\mathscr{F}_2(Q)$

**Require:** Let $Q = (W, F, P)$ be a query.
**Ensure:** $\mathscr{F}_2$ returns a query $Q'$.
1: Let $F' = \emptyset$
2: **for** each clause FROM $u$ in $F$ **do**
3: Add a clause FROM $u$ to $F'$
4: Add a clause FROM NAMED $u$ to $F'$
5: Let $D$ be the dataset obtained from $F'$
6: $P' = \mathscr{F}_4(\mathscr{F}_3(P, D))$
7: **return** $Q' = (W, F', P')$

Let $Q = (W, F, P)$ be a query such that $F$ only contains FROM $u$ clauses and $P$ does not contain clauses of the form ($v$ GRAPH $P'$). The function $\mathscr{F}_2(Q)$, defined by Algorithm 2, allows to transform $Q$ into a query $Q' = (W, F', P')$ such that: (i) $F'$ is a copy of $F$ extended with a clause FROM NAMED $u$ for each clause FROM $u$; and (ii) $P'$ is the graph pattern $\mathscr{F}_4(\mathscr{F}_3(P, D))$ where the function $\mathscr{F}_3$ modifies $P$ by introducing GRAPH graph patterns that relate triple patterns with named graphs, and $\mathscr{F}_4$ merges the GRAPH graph patterns pointing to the same named graph.

The recursive function $\mathscr{F}_3(P, D)$, defined by Algorithm 3, returns a copy of $P$ where each triple pattern $t$ has been replaced by an expression

$$((u_1 \text{ GRAPH } t) \text{ UNION } \cdots \text{ UNION } (u_n \text{GRAPH } t))$$

such that, the evaluation of $t$ over each named graph $u_i$ returns at least one solution. If $t$ matches against all the named graphs (i.e. it matches the default graph), then it remains unchanged. Note that the evaluation of each triple pattern over each named graph (Algorithm 3, line 4), which can be easily implemented with an ASK query, is expected to be a fast procedure supported by appropriate indexes [23]. In addition, this overhead is lifted when the various schemata of the datasets represented in each named graph are known at the outset, thus enabling the query author to know a priori which named graph each triple pattern should be routed to.

The function $\mathscr{F}_4$, defined by Algorithm 4, takes the graph pattern returned by $\mathscr{F}_3$, and recursively group graph patterns of the form $((u \text{ GRAPH } P_1) \text{ AND } (u \text{ GRAPH} P_2))$ and replace them by $(u \text{ GRAPH } (P_1 \text{ AND } P_2))$. The same procedure is applied for the UNION operator. This rewriting is based on the associativity property of the AND and UNION operators.

*Note.* The transformation defined above is valid under set semantics. Under bag semantics (i.e. duplicate solutions), we must consider the following condition. Let $P$ be the triple pattern $(?X\ p\ ?Y)$ and $D = \{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$ be a dataset where $G_0 = G_1 \cup \cdots \cup G_n$. Under bag semantics, the equivalence $[\![P]\!]^D_{G_0} \equiv [\![P]\!]^D_{G_1} \cup \cdots \cup [\![P]\!]^D_{G_n}$ holds when $G_1 \cap \cdots \cap G_n = \emptyset$.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

**Algorithm 3** $\mathscr{F}_3(P, D)$

**Require:** Let $P$ be a graph pattern and $D$ be a dataset.
**Ensure:** $\mathscr{F}_3$ returns a graph pattern $P'$.

```
if P is a triple pattern t then
    Let P_t = () and flag = true
    for each named graph ⟨u_i, G_i⟩ in D do
        if [[t]]^D_{G_i} ≠ ∅ then
            if P_t = () then
                P_t ← (u_i GRAPH t)
            else
                P_t ← ((u_i GRAPH t) UNION P_t)
        else
            flag = false
    if flag = true then
        return t
    else
        return P_t
if P is (P_1 AND P_2) then
    return (F_3(P_1, D) AND F_3(P_2, D))
if P is (P_1 UNION P_2) then
    return (F_3(P_1, D) UNION F_3(P_2, D))
if P is (P_1 OPT P_2) then
    return (F_3(P_1, D) OPT F_3(P_2, D))
if P is (P_1 FILTER C) then
    return (F_3(P_1, D) FILTER C)
```

**Algorithm 4** $\mathscr{F}_4(P)$

**Require:** Let $P$ be a graph pattern.
**Ensure:** $\mathscr{F}_4$ returns a graph pattern.

```
if P is a triple pattern then
    return P
if P is (P_1 AND P_2) then
    P' ← (F_4(P_1) AND F_4(P_2))
    bag[] ← F_5(P', AND)
    return F_6(bag[], AND)
if P is (P_1 UNION P_2) then
    P' ← (F_4(P_1) UNION F_4(P_2))
    bag[] ← F_5(P', UNION)
    return F_6(bag[], UNION)
if P is (P_1 OPT P_2) then
    return (F_3(P_1) OPT F_3(P_2))
if P is (P_1 FILTER C) then
    return (F_3(P_1) FILTER C)
if P is (u GRAPH P_1) then
    return (u GRAPH F_3(P_1))
```

**Algorithm 5** $\mathscr{F}_5(P, \text{OP})$

**Require:** Let $P$ be a graph pattern and OP be an operator in the set {AND, UNION}.
**Ensure:** $\mathscr{F}_5$ returns a multiset of graph patterns.

```
Let bag[] be an empty multiset
if P is (P_1 OP P_2) then
    bag[] ← F_5(P_1, OP) ∪ F_5(P_2, OP)
else
    bag[] ← P
return bag[]
```

**Algorithm 6** $\mathscr{F}_6(bag[], \text{OP})$

**Require:** Let $bag[]$ be a multiset of graph patterns and OP be an operator in the set {AND, UNION}.
**Ensure:** $\mathscr{F}_6$ returns a graph pattern.

```
P ← ()
for each P_1 in bag[] do
    bag[] = bag[] \ P_1
    if P_1 is (u GRAPH P_3) then
        P' ← P_3
        for each P_2 in bag[] do
            if P_2 is (u GRAPH P_4) then
                bag[] = bag[] \ P_2
                P' ← (P_4 OP P')
        P'' ← (u GRAPH P')
    else
        P'' ← P_1
    if P = () then
        P ← P''
    else
        P ← (P'' OP P)
return P
```

### 4.3. H3: Reduce intermediate results

Property paths are a feature introduced in SPARQL 1.1 which allows to specify paths of arbitrary length between two graph nodes, including facilities for expressing recursive queries [24]. A *property path* is defined recursively as follows: (i) if $a \in \mathbf{I}$, then $a$ is a property path; and (ii) if $p_1$ and $p_2$ are property paths, then $[p_1|p_2]$, $[p_1/p_2]$, $[p_1]*$, $[p_1]?$ and $[p_1]+$ are property paths. A *property path triple* is a tuple of the from $(u, p, v)$ where $u, v \in (\mathbf{I} \cup \mathbf{V})$ and $p$ is a property path. Here, we concentrate on property paths of the form $[p_1/p_2]$, called *sequence paths*.

In order to define the semantics of sequence paths, we extend the definition of the function $[\![.]\!]^D_G$. The evaluation of a property path triple $t = (u, p, v)$ is defined as follows:

- If $p \in \mathbf{I}$ then $[\![t]\!]^D_G = [\![(u, p, v)]\!]$.
- If $p$ is the sequence path $[p_1/p_2]$ then $[\![t]\!]^D_G = \pi_{|W}\{(\mu_1 \cup \mu_2) \mid \mu_1 \in [\![(u, p_1, ?X)]\!],\ \mu_2 \in [\![(?X, p_2, v)]\!], \mu_1 \sim \mu_2\}$ where $W = \{u, v\} \cap \mathbf{V}$ and $?X$ is a free variable.

For example, the evaluation of the property path triple $(?B, [q/r], ?D)$ will be given by the algebra expression $\pi_{?B,?D}([\![(?B, q, ?X_1)]\!] \bowtie [\![(?X_1, r, ?D)]\!])$, i.e. it returns pairs of resources $?B$, $?D$ satisfying that $?B$ is connected with $?D$ by a sequence of predicates $q$ and $r$.

In this section, we propose a technique to improve the performance of a query based on the reduction of intermediate results (i.e. solution mappings obtained by the evaluation of sub-graph patterns) by using property path expressions. In order to show the idea, consider the following query:

```
SELECT ?A ?B ?D
WHERE
{ ?A p ?B . OPTIONAL { ?B q ?C . ?C r ?D } }
```

By evaluating the scope of the variables in this query, we can see that variable $?C$ is not required outside the sub-graph pattern `{ ?B q ?C. ?C r ?D }`. Hence, the idea for rewriting is simple: to replace the sub-graph pattern by the sequence path $(?B, [q/r], ?D)$ as a way to filter the variable $?C$. It allows to reduce the amount of intermediate results as well as the cost of subsequent operations, and therefore improve the evaluation of the query.

The use of property paths to project intermediate variables is formalised in the following proposition.

**Proposition 4.2.** *Let $P$ be a graph pattern of the form $((u, p, ?X)$ AND $(?X, q, v))$ where $u, v \in (\mathbf{I} \cup \mathbf{V})$, $p, q \in \mathbf{I}$, and $?X \in \mathbf{V}$. If $P$ occurs inside a graph pattern $P'$ and $?X$ does not occur outside $P$, then $P$ can be replaced by the sequence path $(u, [p/q], v)$.*

**Proof.** The proof of the equivalence between $((u, p, ?X)$ AND $(?X, q, v))$ and $(u, [p/q], v)$ under the conditions defined above, is easy to see from the definition of the semantics for sequence paths. □

The proposed rewriting can be extended to a complex graph pattern containing a sequence of triple patterns of the form

$(\cdots((u, p_1, ?X_1)$ AND $(?X_1, p_2, ?X_2))\cdots$ AND $(?X_{n-1}, p_n, v))$

where $u, v \in \mathbf{I} \cup \mathbf{V}$, $p_i \in \mathbf{I}$ and $?X_j \in \mathbf{V}$. In this case, the intermediate variables $?X_j$ can be projected by using the property path triple:

$(u, [\cdots[p_1/p_2]\cdots/p_n], v)$.

The implementation of this transformation is presented in Algorithm 7. Given a graph pattern $P$, the function $\mathscr{F}_7(P, \emptyset)$ returns a copy of $P$ where specific triple patterns have been replaced by equivalent sequence paths, as defined by Proposition 4.2. For a query $Q = (W, F, P)$, the transformation returns a query $Q' = (W, F, P')$ such that $P' = \mathscr{F}_7(P, W)$. The algorithm evaluates recursively the graph pattern $P$, looking for AND graph patterns that can be replaced by property path triples. Note that the array *var*[] is used to register and verify which variables are useful outside a given sub-graph pattern.

**Algorithm 7** $\mathscr{F}_7(P, vars[])$

**Require:** Let $P$ be a graph pattern and *vars*[] be a set of variables.
**Ensure:** $\mathscr{F}_7$ returns a graph pattern.
1: Let $u, v \in (\mathbf{I} \cup \mathbf{V})$
2: Let $p, q$ be property paths
3: **if** $P$ is a triple pattern **then**
4: **return** $P$
5: **if** $P$ is ($P_1$ AND $P_2$) **then**
6: $P'_1 \leftarrow \mathscr{F}_7(P_1, vars[] \cup \text{var}(P_2))$
7: $P'_2 \leftarrow \mathscr{F}_7(P_2, vars[] \cup \text{var}(P_1))$
8: **if** $P'_1$ is $(u, p, ?X)$ and $P'_2$ is $(?X, q, v)$ **then**
9: **if** $?X \in \mathbf{V}$ and $?X \notin vars[]$ **then**
10: **return** $(u\ p/q\ v)$
11: **if** $P$ is ($P_1$ UNION $P_2$) **then**
12: $P'_1 \leftarrow \mathscr{F}_7(P_1, vars[])$
13: $P'_2 \leftarrow \mathscr{F}_7(P_2, vars[])$
14: **return** ($P'_1$ UNION $P'_2$)
15: **if** $P$ is ($P_1$ OPT $P_2$) **then**
16: $P'_1 \leftarrow \mathscr{F}_7(P_1, vars[] \cup \text{var}(P_2))$
17: $P'_2 \leftarrow \mathscr{F}_7(P_2, vars[] \cup \text{var}(P_1))$
18: **return** ($P'_1$ OPT $P'_2$)
19: **if** $P$ is ($P_1$ FILTER $C$) **then**
20: $P'_1 \leftarrow \mathscr{F}_7(P_1, vars[] \cup \text{var}(C))$
21: **return** ($P'_1$ FILTER $C$)
22: **if** $P$ is ($u$ GRAPH $P_1$) **then**
23: **if** $u \in \mathbf{V}$ **then**
24: $P'_1 \leftarrow \mathscr{F}_7(P_1, vars[] \cup u)$
25: **else**
26: $P'_1 \leftarrow \mathscr{F}_7(P_1, vars[])$
27: **return** ($u$ GRAPH $P'_1$)

Section 5.3.3 provides empirical evidence to support the claim that replacing connected triple patterns with sequence paths can provide performance improvements.

### 4.4. H4: reduce the effects of cartesian products

In this heuristic, we propose a method to reduce redundant values occurring in a solution sequence. Each individual solution in the sequence is a set that contains at most one mapping per variable which appears in the query, and individual mappings may appear in more than one solution. Therefore, the number of solutions in a sequence is given by the product of the number of mappings obtained for each variable in the query.

For example, consider the following solution sequence consisting of two mappings $\mu_1$ and $\mu_2$:

$\mu_1 : \{\mu_1(?s) = a, \mu_1(?p) = b, \mu_1(?o) = c\}$
$\mu_2 : \{\mu_2(?s) = a, \mu_2(?p) = b, \mu_2(?o) = d\}$.

We can see that the values for the variables $?s$ and $?p$ are the same in both mappings, and just the values for the variable $?o$ are distinct. This situation is often perceived by end users as duplicating information in the results and in turn introduces an expensive post-processing step for applications that consume and present the results to users.

Note that this heuristic differs from the one presented in the previous section in that here we are interested in minimising the size and improving the presentation of a result sequence rather than eliminating redundant intermediate variables in the query to improve its evaluation performance.

SPARQL 1.1 [6] introduces a set of 7 aggregates that combine groups of mappings for the same variable: `SUM`, `MIN`, `MAX`, `AVG`, `COUNT`, `SAMPLE`, and `GROUP_CONCAT`. According to the idea presented above, the instruction `GROUP_CONCAT` is of particular interest such that it can be used to reduce a group of mappings into a single mapping containing the string concatenation of duplicate values in the group. Considering the initial example, we can apply the `GROUP_CONCAT` aggregate to variable $?o$ to obtain the singleton solution sequence:

$\{\mu(?s) = a, \mu(?p) = b, \mu(?o) = \text{“}c, d\text{”}\}$.

While the above only considers `GROUP_CONCAT`, similar examples can be given for the other 6 aggregates. For instance consider a query that should retrieve a single name for a resource, while several synonyms exist in the dataset. In this case `SAMPLE` can be used to retrieve a single mapping, `MIN` will retrieve the first synonym sorted in lexicographical order, while `MAX` will retrieve the last. Similarly, `AVG` and `COUNT` can effectively be used to summarise numerical measurements, where such summarisation is permitted by the use case.

Aggregates can thus be used to eliminate perceived duplication in result sequences, and obtain a succinct result format. However, as the introduction of aggregates creates a disconnect between the information presented in the results to the data stored in the RDF store, this heuristic is not well-suited to automatic application, unless coupled with a mechanism to make the user aware of the query transformation that has taken place.

### 4.5. H5: Specifying alternative URIs

The SPARQL specification [5] recommends the use of the `UNION` keyword as a means of matching one or more alternative graph patterns. This is showed by the following query:

```
SELECT ?label
WHERE {
 { <uri1> label ?label }
 UNION
 { <uri2> label ?label }
}
```

The evaluation of this query[8] will then contain mappings for the variable `?label` matching one of the two triple patterns, i.e. the mappings will include labels associated with either `<uri1>` or `<uri2>`.

The same effect can be achieved by using two other features: filter conditions in SPARQL 1.0, and the VALUES keyword introduced in SPARQL 1.1.

A filter constraint can be used to restrict a graph pattern solution by specifying conditions over its variables. Hence, the example query above can be rewritten as follows:

[8] From the point of view of data provenance, the query brings an important problem: the fact that two alternative resource URIs have been used in the query cannot be inferred from the results alone (i.e. one must also have access to the original query). Moreover, it is not possible to discern which of the results apply to each of the resources. In order to circumvent these issues, the introduction of a variable containing data provenance is required.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

```
SELECT ?label
WHERE {
 { ?s label ?label . FILTER (?s = <uri1> )}
 UNION
 { ?s label ?label . FILTER (?s = <uri2> )}
}
```

Note that, the UNION operator can be replaced by a complex filter expression including the logical-or operator ( | | ). Hence, the above query can be reduced to:

```
SELECT ?label
WHERE {
 ?s label ?label .
 FILTER (?s = <uri1> || ?s = <uri2> )
}
```

A VALUES expression allows to insert inline data as a solution sequence, hence it can be combined with other graph pattern results. In this case, the initial query can be expressed as:

```
SELECT ?label
WHERE {
 { ?s label ?label . VALUES ?s { <uri1> } }
 UNION
 { ?s label ?label . VALUES ?s { <uri2> } }
}
```

Considering that the VALUES keyword allows to include multiple values for a variable, the above graph pattern can be simplified as follows:

```
SELECT ?label
WHERE {
  ?s :label ?label . VALUES ?s { <uri1> <uri2> }
}
```

Both keywords, FILTER and VALUES, allow to generate mappings for the variable ?*s* which is used to identify which resource each mapping for `?label` refers to. However, we must also consider efficiency. In comparison with the original query, all the above rewritings present a characteristic that can affect their evaluation: the evaluation of the triple pattern (?s, label, ?label), can result in a very large solution sequence, which indeed need to be examined to either verify a condition in case of using FILTER, or to calculate the join in case of using VALUES. This issue can be solved, for the case of VALUES, by a simple rewriting showed in the following query:

```
SELECT ?label
WHERE {
 { <uri1> label ?label . VALUES ?s { <uri1> } }
 UNION
 { <uri2> label ?label . VALUES ?s { <uri2> } }
}
```

Note that, this query avoids a possible large result for the triple graph pattern by replacing the variable `?s` by a constant, and the VALUES keyword is strictly used to preserve data provenance.

The specification of alternative URIs in a SPARQL query does not necessarily introduce cartesian products to the associated result sequence, thus the merit of this heuristic is assessed independently. However, the heuristic described in the previous section can naturally be used to reduce the effects of such cartesian product, where they do exist.

In Section 5.3.4, we provide empirical evidence that shows that performance varies across RDF stores for the different rewriting techniques presented above. It is therefore important to consider their performance in the context of the system being developed.

## 5. Evaluation

An empirical evaluation was carried out to measure improvements obtained through the application of the heuristics presented in the previous section across a number of state-of-the-art RDF stores. Section 5.1 provides details on the experimental setup used, while Section 5.2 describes the queries used in the evaluation. Section 5.3 presents the results obtained, providing a discussion on the utility of the heuristics. The entire experimental setup including the datasets [25], the sample resources [26], the queries [27] and associated scripts [28], and the results [29], is available online.

### 5.1. Experimental setup

#### 5.1.1. Hardware

To ensure that the experiments could be completed in reasonable time and that RDF stores were able to deliver their best performance, the experiments were run using fairly powerful hardware:

- CPU: 4× Intel 8 Core Xeon E5-2650L 1.8 GHz
- RAM: 384 GB RAM 1333 MHz
- Hard drive: 1.4 TB RAID 6 (12 × 128 MB SSD).

#### 5.1.2. Datasets

Some of the main datasets considered by the OpenPHACTS platform were used to carry out the evaluation of this work, since the work of gathering application requirements and mapping them to SPARQL graph patterns had already been carried out. They are:

- ChEMBL v13[9] RDF conversion.[10]
- ChemSpider[11] and ACD Labs[12] Predicted Properties RDF conversion.
- Drugbank RDF conversion provided by the Bio2Rdf[13] project.
- Conceptwiki[14] RDF conversion provided on request.

The above datasets mainly describe two types of resource: chemical compounds and targets (e.g. proteins). Additionally, a third type of resource is the interaction between a compound and a target. In total, the data contains 168 783 592 triples, 290 predicates and are loaded in four separate named graphs (one per dataset).

While it would be interesting to assess the effectiveness of our heuristics against established benchmarks, such as LUBM [30] or BSBM [31], the regularity exhibited by synthetic datasets generated by such benchmarks as well as the expertly formulated sets queries that are provided give little opportunity for optimisation using our heuristics. We also considered using query logs from well-established endpoints such as DBPedia [32], however this is also problematic; the queries are often invalid [7], and do not address well defined use-cases. As such we decided to carry out our evaluation using the datasets listed above.

#### 5.1.3. RDF stores

Table 1 lists the RDF stores used in the evaluation, along with the maximum memory usage measured during the experiments.

### 5.2. SPARQL queries

This section provides details on each SPARQL query template used in the evaluation of the five heuristics. The templates orig-

[9] https://www.ebi.ac.uk/chembl/.
[10] https://github.com/egonw/chembl.rdf.
[11] http://www.chemspider.com/.
[12] http://www.acdlabs.com.
[13] http://bio2rdf.org/.
[14] http://ops.conceptwiki.org/.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

**Table 1**
RDF stores used, corresponding version number and maximum memory use.

| RDF store | Version | Maximum memory use (GB) |
|---|---|---|
| Virtuoso enterprise[a] | 07.00.3204 | 8.0 |
| Virtuoso open source[a] | 07.00.3203 | 8.1 |
| Bigdata[b] | 1.2.2 | 34 |
| OWLIM-Lite[c] | 5.3.5777 | 13 |
| Sesame native Java store[d] | 2.7.9 | 16 |
| Sesame in-memory store[d] | 2.7.9 | 82 |
| 4store[e] | 1.1.5–51 | 17 |

[a] http://virtuoso.openlinksw.com/.
[b] http://www.systap.com/bigdata.htm.
[c] http://www.ontotext.com/owlim.
[d] http://www.openrdf.org/.
[e] http://4store.org/.

inate from four frequently used OpenPHACTS API methods: Compound Information,[15] Compound Pharmacology,[16] Target Information,[17] and Target Pharmacology.[18]

The average monthly usage for each method during the six month period between Sep. 2013 and Feb. 2014 is as follows: Compound Information, 125 460 hits; Compound Pharmacology, 22 369 hits; Target Information, 69 337 hits; Target Pharmacology, 26 752 hits. Queries Q21–Q22 were constructed specifically to illustrate the utility of the heuristics, while queries Q23–Q31 all correspond to the Target Information view template.

**Q1: Compound information (initial)**

*Description:* Return information about a concept of type 'Compound' from four datasets.
*Parameters:* A compound URI.
*Projection variables:* 23
*Graph patterns:*27 (0 `OPTIONAL`)

**Q2: Compound pharmacology (initial)**

*Description:* Return information about the interactions of a 'Compound' concept with 'Target' concepts. Four datasets are used.
*Parameters:* A compound URI.
*Projection variables:* 30
*Graph patterns:* 33 (0 `OPTIONAL`)

**Q3: Target information (initial)**

*Description:* Return information about a concept of type 'Target' from three datasets.
*Parameters:* A target URI.
*Projection variables:* 12
*Graph patterns:* 12 (0 `OPTIONAL`)

**Q4: Target pharmacology (initial)**

*Description:* Return information about the interactions of a 'Target' concept with 'Compound' concepts. Four datasets are used.
*Parameters:* A target URI.
*Projection variables:* 28
*Graph patterns:* 30 (0 `OPTIONAL`)

**Q5–Q8: Named graph queries**

*Description:* Derived by applying the 'Use named graphs to localise SPARQL sub-graph patterns' (Section 4.2) heuristic to queries Q1–Q4 respectively.

**Q9–Q12: Optional queries**

*Description:* Contain the same graph patterns as Q1–Q4, with patterns that do not retrieve instances of the core information types defined in the view template appearing inside an `OPTIONAL` clause. Thus, the results of Q9–Q12 and all subsequent queries (derived from them) differ to those of 'Initial' and 'Named Graph' queries.
*Optional patterns:* Q9: 22, Q10: 21, Q11: 11, and Q12: 16.

**Q13–Q16: Min. optional queries**

*Description:* Derived by applying the 'Minimise optional graph patterns' (Section 4.1) heuristic to queries Q9–Q12.
*Optional patterns:* Q13: 11, Q14: 14, Q15: 8, and Q16: 12.

**Q17–Q20: Graph Optional Queries**

*Description:* Derived by applying the 'Use named graphs to localise SPARQL sub-graph patterns' (Section 4.2) heuristic to queries Q13–Q16 respectively.

**Q21: Redundant intermediate variables**

*Description:* Query constructed to evaluate the 'Reduce intermediate results' (Section 4.3) heuristic.
*Parameters:* A target URI (ConceptWiki)

```
SELECT ?synonym ?cellularLocation {
  [RESOURCE] skos:exactMatch ?chembl_uri.
  ?chembl_uri rdfs:label ?synonym.
  OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri.
  ?db_uri db:cellularLocation
?cellularLocation.}
}
```

**Q22: Sequence path query**

*Description:* Query derived by applying the 'Reduce intermediate results' (Section 4.3) heuristic to Q21.

**Q23–Q25: Alternative URIs:** `UNION`

*Description:* Each query contains the same patterns as Q3, however specifies a number of alternative values for the parameter, which are encoded using `UNION` clauses, as in Section 4.5. The query performance is evaluated using sets of 10, 20 and 50 URIs, in queries Q23, Q24, and Q25 respectively.

**Q26–Q28: Alternative URIs:** `FILTER`

*Description:* Each query contains the same patterns as Q3, however specifies a number of alternative values for the parameter, which are encoded using `FILTER` clauses, as in Section 4.5. The query performance is evaluated using sets of 10, 20 and 50 URIs, in queries Q26, Q27, and Q28 respectively.

**Q28–Q31: Alternative URIs:** `VALUES`

*Description:* Each query contains the same patterns as Q3, however specifies a number of alternative values for the parameter, which are encoded using `VALUES` clauses, as in Section 4.5. The query performance is evaluated using sets of 10, 20 and 50 URIs, in queries Q29, Q30, and Q31 respectively.

We note that the 'Initial' queries, Q1–Q4, were obtained by having domain experts specify their information needs for each use-case. These queries were found to be too restrictive in that only resources that matched all graph patterns were returned. The domain experts then specified the patterns which they considered optional, so that resources without these values could be returned. This selection is reflected in the 'Optional' queries, Q9–Q12, which were found to be the slowest queries considered in our evaluation.

[15] https://beta.openphacts.org/1.3/compound.
[16] https://beta.openphacts.org/1.3/compound/pharmacology/pages.
[17] https://beta.openphacts.org/1.3/target.
[18] https://beta.openphacts.org/1.3/compound/pharmacology/pages.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

**Table 2**
Queries used in the evaluation of heuristics H1: Minimise optional graph patterns and H2: Use named graphs to localise SPARQL sub-graph patterns. G is the initial set of queries. G′ contains the queries of G, extended with `OPTIONAL` graph patterns. H2(G) denotes the application of heuristic H2 to each of the queries in G (a similar interpretation applies to H1(G′)) and H2(H1(G′)).

| View template | Queries | | | | |
|---|---|---|---|---|---|
| | G | H2(G) | G′ | H1(G′) | H2(H1(G′)) |
| Compound information | Q1 | Q5 | Q9 | Q13 | Q17 |
| Compound pharmacology | Q2 | Q6 | Q10 | Q14 | Q18 |
| Target information | Q3 | Q7 | Q11 | Q15 | Q19 |
| Target pharmacology | Q4 | Q8 | Q12 | Q16 | Q20 |

**Table 3**
Queries used in the evaluation of heuristic H5: Specifying alternative URIs. All these queries are copies of Q3, extended with specific clauses (`UNION`, `FILTER`, `VALUES`) to introduce alternative URIs as defined in Section 4.5. The number of alternative URIs used in each query is given by the 3 rightmost columns (10, 20 and 50).

| Method | Number of URIs | | |
|---|---|---|---|
| | 10 | 20 | 50 |
| `UNION` | Q23 | Q24 | Q25 |
| `FILTER` | Q26 | Q27 | Q28 |
| `VALUES` | Q29 | Q30 | Q31 |

**Summary**

In order to facilitate the interpretation of the results, we provide the following summary of the experiments performed:

- The queries used to evaluate heuristics H1: Minimise optional graph patterns and H2: Use named graphs to localise SPARQL sub-graph patterns are described in Table 2.
- Q22 is the query obtained by applying H3: Reduce intermediate results to Q21.
- Heuristic H5: Specifying alternative URIs is evaluated using queries derived from Q3 as described in Table 3.

### 5.3. Results

The only changes made to the default configuration of the RDF stores was to set a maximum memory limit to 90 GB and disable any inferencing. Each store was restarted prior to running an experiment, and the following query issued as a warm-up:

```
SELECT (COUNT (DISTINCT * ) AS ?count )
WHERE { ?s ?p ?o }
```

Random samples of 500 compound URIs and 500 target URIs were used to instantiate the SPARQL queries. We reiterate that targets and compounds are the main concepts described by these datasets. The results are presented in Table 4, while the figures in Appendix B visualise the performance differences observed for queries generated from the same view template. All mentions of significance assume a normal distribution at the 1% level (one sided).

The goal for this evaluation is to assess whether queries obtained through the application of the heuristics have better performance characteristics across the systems considered. Thus, the preprocessing steps required in order to apply a heuristic to a query are not included in the measurements presented, since it would introduce a bias against the application of any heuristic. We reiterate that such pre-processing does not necessarily have to take place prior to evaluating each query and is better performed in bulk, with a set of view templates in mind.

#### 5.3.1. H1: minimise optional graph patterns

The performance improvement observed by minimising optional triple patterns is dramatic for all RDF stores. All but the Virtuoso RDF stores failed to evaluate Q9 within the 1000 s timeout, while the corresponding 'Min. Optional' query, Q13, has at maximum a 3 s response time for 4store and is as low as 0.014 for the Sesame in-memory store. Similar improvements are obtained for almost all combinations of query and RDF store where this heuristic is applied.

These results provide empirical evidence that formal results published in the literature regarding the use of OPTIONAL graph patterns do carry over to practical applications. Specifically, the elimination of redundant OPTIONAL query patterns can yield dramatic improvements on query performance for all of the RDF stores considered.

It is noted that in some cases the upper bound in execution time is increased through the application of this heuristic: for example Q16 with either of the Sesame stores. However, this can be reduced by applying the 'Use named graphs to localise SPARQL sub-graph patterns' heuristic as discussed below.

#### 5.3.2. H2: use named graphs to localise SPARQL sub-graph patterns

Comparing the 'Initial'(Q1–Q4) query response times to the corresponding 'Named Graph' queries, the results appear mostly unaffected; however Q3 is significantly faster across all RDF stores with the exception of 4store. The results vary across RDF stores for the remaining queries, while Q8 failed to return results before the timeout in OWLIM-Lite.

The average evaluation performance of 'Min. Optional' and 'N. Graph Optional' queries is closely matched for all templates, with the exception of the most expensive one: Target Pharmacology (Q16, Q20). When the heuristic is applied a significantly lower execution time on average is observed across all RDF stores except OWLIM-Lite.

As before the application of the heuristic in most cases results in significantly reducing the upper bound in query execution time.

Thus, we argue that the introduction of named graphs is an effective optimisation to lower the variance between execution performance for the same query template, significantly reducing the maximum execution times for complex queries.

#### 5.3.3. H3: Reduce intermediate results

The schemas of the datasets did not allow the creation of a meaningful sequence path query for compounds. Thus, only the sample of 500 targets was used for this experiment. At the time of writing sequence paths are not supported by 4store, as they are not implemented in Rasqal,[19] its SPARQL processing library.

Slight improvements are observed with respect to the mean response time for Bigdata and Virtuoso EE, while the maximum response time is significantly reduced for the same RDF stores. The performance in the remaining cases is almost identical. Considering the simplicity of the queries involved, we intend to explore whether the utility of this heuristic increases with the complexity of queries in the future.

[19] http://librdf.org/rasqal/.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

**Table 4**
Evaluation results. All values are measured in seconds. '–' indicates the timeout of 1000 s was reached, while 'N/A' indicates that a particular feature had not been implemented. The columns $\bar{x}$ and $\sigma$ provide (respectively) the mean and standard deviation for the response times obtained using all resources in the sample. min and max give the minimum and maximum response time obtained for each query. The view template corresponding to each query is given by the superscript symbol: ○: Compound Information, ●: Compound Pharmacology, □: Target information, ■: Target pharmacology, and †: Redundant intermediate variables. For queries Q23–Q31, the subscripts indicate the number of alternative URIs used. Response times obtained from queries which correspond to the same template (and number of alternative URIs) are comparable.

| | | Sesame in-memory | | | | Sesame native Java | | | | OWLIM-Lite | | | | 4store | | | | Bigdata | | | | Virtuoso OS | | | | Virtuoso EE | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\bar{x}$ | $\sigma$ | max | min | $\bar{x}$ | $\sigma$ | max | min | $\bar{x}$ | $\sigma$ | max | min | $\bar{x}$ | $\sigma$ | max | min | $\bar{x}$ | $\sigma$ | max | min | $\bar{x}$ | $\sigma$ | max | min | $\bar{x}$ | $\sigma$ | max | min |
| Initial | Q1○ | 0.011 | 0.001 | 0.014 | 0.009 | 0.017 | 0.004 | 0.049 | 0.009 | 0.013 | 0.001 | 0.014 | 0.011 | 1.37 | 0.303 | 3.12 | 1.01 | 0.041 | 0.008 | 0.099 | 0.023 | 0.129 | 0.208 | 4.72 | 0.008 | 0.108 | 0.024 | 0.308 | 0.007 |
| | Q2● | 0.012 | 0 | 0.012 | 0.011 | 0.014 | 0.001 | 0.017 | 0.011 | 0.013 | 0.001 | 0.014 | 0.007 | 1.55 | 0.495 | 2.09 | 1.01 | 0.095 | 1.16 | 25.9 | 0.025 | 0.165 | 0.241 | 5.53 | 0.016 | 0.121 | 0.286 | 6.49 | 0.008 |
| | Q3□ | 0.011 | 0.003 | 0.028 | 0.005 | 0.013 | 0.004 | 0.034 | 0.009 | 0.017 | 0.008 | 0.135 | 0.007 | 1.85 | 0.468 | 2.87 | 1.01 | 0.033 | 0.005 | 0.053 | 0.02 | 0.095 | 0.047 | 0.7 | 0.011 | 0.087 | 0.034 | 0.379 | 0.01 |
| | Q4■ | 0.01 | 0.001 | 0.011 | 0.006 | 0.013 | 0 | 0.014 | 0.01 | 0.081 | 0.246 | 2.74 | 0.008 | 2.84 | 2.46 | 25.6 | 1.01 | 0.04 | 0.005 | 0.046 | 0.023 | 0.128 | 0.022 | 0.291 | 0.017 | 0.13 | 0.025 | 0.164 | 0.006 |
| Named graph | Q5○ | 0.015 | 0.003 | 0.04 | 0.011 | 0.017 | 0.003 | 0.04 | 0.012 | 0.015 | 0.003 | 0.047 | 0.007 | 1.3 | 0.256 | 2.85 | 1.01 | 0.018 | 0.006 | 0.089 | 0.009 | 0.118 | 0.021 | 0.184 | 0.009 | 2.016 | 0.529 | 3.742 | 0.008 |
| | Q6● | 0.013 | 0 | 0.014 | 0.009 | 0.015 | 0 | 0.017 | 0.012 | 0.013 | 0.001 | 0.034 | 0.011 | 1.03 | 0.109 | 2.01 | 1.01 | 0.047 | 0.003 | 0.051 | 0.035 | 0.155 | 0.021 | 0.344 | 0.018 | 0.137 | 0.021 | 0.196 | 0.008 |
| | Q7□ | 0.006 | 0 | 0.007 | 0.004 | 0.008 | 0.001 | 0.008 | 0.005 | 0.008 | 0.001 | 0.017 | 0.007 | 1.64 | 0.487 | 2.1 | 1.01 | 0.007 | 0 | 0.01 | 0.006 | 0.008 | 0 | 0.012 | 0.007 | 0.008 | 0 | 0.012 | 0.007 |
| | Q8■ | 0.012 | 0 | 0.012 | 0.011 | 0.013 | 0 | 0.014 | 0.012 | – | – | – | – | 1.52 | 0.503 | 2.52 | 1.01 | 0.02 | 0.001 | 0.021 | 0.015 | 0.011 | 0.001 | 0.031 | 0.008 | 0.011 | 0.001 | 0.015 | 0.008 |
| Optional | Q9○ | – | – | – | – | – | – | – | – | – | – | – | 700 | – | – | – | 675 | – | – | – | – | 26.5 | 23.3 | 75.4 | 0.056 | 29.8 | 27.4 | 79.4 | 0.033 |
| | Q10● | 0.065 | 0.155 | 2.07 | 0.008 | 0.089 | 0.217 | 2.76 | 0.014 | 0.177 | 0.56 | 8.47 | 0.013 | 4.47 | 7.29 | 81.4 | 1.08 | 0.866 | 4.18 | 75.1 | 0.09 | 1.5 | 4.39 | 53.4 | 0.027 | 1.62 | 5.28 | 75.7 | 0.01 |
| | Q11□ | 0.018 | 0.007 | 0.1 | 0.01 | 0.02 | 0.007 | 0.091 | 0.01 | 0.018 | 0.011 | 0.213 | 0.008 | 1.75 | 0.486 | 2.43 | 1.05 | 0.064 | 0.007 | 0.109 | 0.044 | 0.067 | 0.04 | 0.423 | 0.026 | 0.057 | 0.038 | 0.391 | 0.017 |
| | Q12■ | 0.425 | 1.38 | 16.9 | 0.012 | 0.82 | 2.71 | 33.7 | 0.013 | 1.72 | 5.8 | 7.4 | 0.013 | 34.3 | 34.9 | 127 | 1.07 | 1.78 | 5.45 | 67.8 | 0.078 | 8.67 | 27.5 | 262 | 0.119 | 13.3 | 37.5 | 270. | 0.119 |
| Min. optional | Q13○ | 0.013 | 0.001 | 0.014 | 0.009 | 0.014 | 0.003 | 0.072 | 0.008 | 0.014 | 0.001 | 0.017 | 0.012 | 1.27 | 0.212 | 3.18 | 1.05 | 0.08 | 0.005 | 0.095 | 0.054 | 0.225 | 0.029 | 0.273 | 0.02 | 0.203 | 0.028 | 0.243 | 0.009 |
| | Q14● | 0.022 | 0.02 | 0.257 | 0.012 | 0.031 | 0.04 | 0.494 | 0.012 | 0.032 | 0.059 | 0.981 | 0.013 | 1.11 | 0.111 | 2.1 | 1.06 | 0.177 | 0.392 | 7 | 0.08 | 0.287 | 0.171 | 2.78 | 0.026 | 0.245 | 0.165 | 2.68 | 0.009 |
| | Q15□ | 0.018 | 0.007 | 0.11 | 0.01 | 0.02 | 0.007 | 0.08 | 0.011 | 0.017 | 0.006 | 0.081 | 0.006 | 1.56 | 0.501 | 2.08 | 1.04 | 0.064 | 0.007 | 0.109 | 0.038 | 0.079 | 0.042 | 0.438 | 0.035 | 0.058 | 0.039 | 0.398 | 0.018 |
| | Q16■ | 0.513 | 5.68 | 126 | 0.013 | 0.586 | 1.98 | 24.6 | 0.011 | 0.37 | 1.2 | 14.6 | 0.01 | 2.3 | 3.43 | 35 | 1.07 | 0.451 | 1.13 | 14.4 | 0.077 | 0.926 | 2.7 | 35.6 | 0.022 | 1.05 | 2.83 | 34.9 | 0.01 |
| N. graph optional | Q17○ | 0.013 | 0.001 | 0.014 | 0.011 | 0.015 | 0.001 | 0.017 | 0.013 | 0.015 | 0.001 | 0.017 | 0.012 | 1.1 | 0.197 | 2.09 | 1.05 | 0.08 | 0.006 | 0.115 | 0.057 | 0.062 | 0.009 | 0.172 | 0.02 | 0.054 | 0.008 | 0.068 | 0.008 |
| | Q18● | 0.024 | 0.02 | 0.266 | 0.013 | 0.032 | 0.042 | 0.497 | 0.012 | 0.029 | 0.038 | 0.484 | 0.013 | 1.09 | 0.138 | 4.11 | 1.06 | 0.179 | 0.392 | 7.02 | 0.088 | 1.85 | 1.97 | 31.3 | 0.982 | 1.33 | 0.161 | 3.72 | 1.02 |
| | Q19□ | 0.017 | 0.005 | 0.062 | 0.01 | 0.018 | 0.005 | 0.062 | 0.011 | 0.016 | 0.006 | 0.082 | 0.01 | 1.56 | 0.501 | 2.12 | 1.04 | 0.064 | 0.008 | 0.097 | 0.04 | 0.059 | 0.031 | 0.276 | 0.024 | 0.047 | 0.029 | 0.252 | 0.017 |
| | Q20■ | 0.101 | 0.277 | 3.36 | 0.014 | 0.193 | 0.594 | 7.17 | 0.014 | 0.37 | 1.2 | 14.6 | 0.013 | 1.65 | 0.51 | 2.23 | 1.07 | 0.02 | 0.001 | 0.022 | 0.016 | 0.012 | 0.001 | 0.016 | 0.009 | 0.012 | 0.001 | 0.016 | 0.008 |
| Seq. paths | Q21† | 0.007 | 0 | 0.007 | 0.004 | 0.008 | 0 | 0.009 | 0.007 | 0.007 | 0. | 0.009 | 0.005 | 0.201 | 0.126 | 0.676 | 0.011 | 0.034 | 0.03 | 0.682 | 0.023 | 0.01 | 0.001 | 0.014 | 0.008 | 0.012 | 0.044 | 0.983 | 0.007 |
| | Q22† | 0.007 | 0 | 0.007 | 0.005 | 0.008 | 0 | 0.008 | 0.007 | 0.007 | 0.001 | 0.024 | 0.006 | N/A | N/A | N/A | N/A | 0.025 | 0.002 | 0.064 | 0.015 | 0.01 | 0.001 | 0.032 | 0.008 | 0.01 | 0.001 | 0.014 | 0.007 |
| UNION | $Q23^{\square}_{10}$ | 0.047 | 0.002 | 0.057 | 0.037 | 0.049 | 0.002 | 0.063 | 0.044 | 0.05 | 0.002 | 0.066 | 0.043 | 1.08 | 0.05 | 1.15 | 1 | 0.119 | 0.009 | 0.22 | 0.099 | 0.404 | 0.027 | 0.939 | 0.396 | 0.3 | 0.1 | 2.52 | 0.286 |
| | $Q24^{\square}_{20}$ | 0.096 | 0.018 | 0.504 | 0.088 | 0.101 | 0.002 | 0.108 | 0.095 | 0.116 | 0.002 | 0.135 | 0.111 | 1.27 | 0.043 | 1.35 | 1.18 | 0.248 | 0.011 | 0.29 | 0.218 | 1.07 | 0.048 | 2.13 | 1.06 | 0.875 | 0.095 | 1.88 | 0.816 |
| | $Q25^{\square}_{50}$ | 0.19 | 0.042 | 0.88 | 0.181 | 0.213 | 0.009 | 0.417 | 0.209 | 0.232 | 0.003 | 0.25 | 0.225 | 1.56 | 0.036 | 1.62 | 1.52 | 0.706 | 0.026 | 0.761 | 0.634 | 2.15 | 0.142 | 4.97 | 2.08 | 1.65 | 0.149 | 4.72 | 1.58 |
| FILTER | $Q26^{\square}_{10}$ | 0.038 | 0.003 | 0.076 | 0.032 | 0.043 | 0.001 | 0.065 | 0.038 | 0.008 | 0.001 | 0.017 | 0.007 | 1.02 | 0.008 | 1.03 | 1 | 0.168 | 0.016 | 0.358 | 0.14 | 0.287 | 0.033 | 0.978 | 0.269 | 0.275 | 0.028 | 0.834 | 0.263 |
| | $Q27^{\square}_{20}$ | 0.078 | 0.017 | 0.465 | 0.07 | 0.13 | 0.961 | 21.6 | 0.082 | 0.008 | 0.001 | 0.01 | 0.005 | 1.08 | 0.015 | 1.11 | 1.06 | 0.219 | 0.016 | 0.273 | 0.177 | 0.863 | 0.087 | 1.02 | 0.803 | 0.811 | 0.087 | 0.97 | 0.751 |
| | $Q28^{\square}_{50}$ | 0.15 | 0.015 | 0.483 | 0.143 | 0.176 | 0.002 | 0.18 | 0.17 | 0.009 | 0.001 | 0.027 | 0.007 | 1.19 | 0.057 | 1.28 | 1.12 | 0.368 | 0.012 | 0.423 | 0.336 | – | – | – | – | 1.49 | 0.055 | 1.63 | 1.42 |
| VALUES | $Q29^{\square}_{10}$ | 0.051 | 0.2 | 4.51 | 0.033 | 0.048 | 0.002 | 0.071 | 0.044 | 0.007 | 0 | 0.008 | 0.005 | 1.01 | 0.006 | 1.02 | 1 | 0.007 | 0 | 0.01 | 0.006 | 0.292 | 0.023 | 0.714 | 0.28 | 0.269 | 0.013 | 0.343 | 0.259 |
| | $Q30^{\square}_{20}$ | 0.081 | 0.006 | 0.19 | 0.068 | 0.093 | 0.001 | 0.097 | 0.09 | 0.007 | 0. | 0.008 | 0.006 | 1.01 | 0.003 | 1.02 | 1.01 | 0.007 | 0 | 0.008 | 0.007 | 0.904 | 0.137 | 1.17 | 0.818 | 0.847 | 0.139 | 1.1 | 0.759 |
| | $Q31^{\square}_{50}$ | 0.145 | 0.005 | 0.24 | 0.143 | 0.181 | 0.143 | 3.36 | 0.17 | 0.007 | 0.001 | 0.026 | 0.005 | 1.02 | 0.002 | 1.02 | 1.02 | 0.008 | 0 | 0.009 | 0.006 | 1.61 | 0.116 | 2.05 | 1.49 | 1.57 | 0.121 | 1.86 | 1.43 |

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

*5.3.4. H4: Specifying alternative URIs*

For OWLIM-Lite, specifying alternative URIs with either FILTER or VALUES is significantly faster than with UNION for all three sets of 10, 20 and 50 URIs.

For 4store, VALUES[20] was found to be slightly faster than FILTER while UNION is the slowest option, significantly so if 20 or 50 URIs are used.

Bigdata performs significantly faster using VALUES than either of the other two methods. With 50 alternative URIs using FILTER is also significantly faster than using UNION.

Virtuoso OS on the other hand performs significantly faster using UNION or VALUES instead of FILTER when 50 URIs are specified, despite the length of the UNION query itself.

Virtuoso EE and the two Sesame RDF store provide very similar performance when using any of the 3 methods.

We observe from the above that no single method is consistently best across different RDF stores and thus the 3 options should be evaluated on a case-by-case basis.

## 6. Practical implications for providing paginated RDF views

We now describe how these heuristics can be applied in practice for the common use-case of pagination. In many cases, the number of results obtained through the execution of a query is large and can become overwhelming to users if presented all at once. Client side applications are not well placed to deal with this issue, as the processing and pagination of large result sets poses a significant overhead that can severely impact the usability of the application. Thus, a pagination mechanism for SPARQL result sequences is desirable. In this context, a page is considered equivalent to an ordered list of a predefined number of individual results from the same result sequence.

Intuitively, this issue can be dealt with through the use of SPARQLs LIMIT and OFFSET keywords. In practice, applications typically require the ability to change the sort order, and to apply arbitrary filtering rules to control which results from the sequence are displayed in each page, thus posing additional challenges to a server side implementation.

This section illustrates how the heuristics proposed in this paper can be applied to enable the provision of RDF paginated views.

**Minimise optional triple patterns and localise sub-patterns**

Assuming the graph patterns that retrieve the required information are known, their conjunction provides an 'Initial query'. The heuristics presented in Sections 4.1 and 4.2 can then be applied to the initial query to improve its performance.

**Eliminate cartesian products**

Subsequently, any results that appear due to cartesian products of variable mappings must be eliminated from sequences intended for pagination in order to ensure that items that appear in a page correspond to individual and independent data points. This is of particular importance to scientific applications since cartesian products can artificially increase the number of results returned by a query, resulting in an overestimation of the number of data points recorded in a dataset.

**Specify alternative URIs**

In addition, when each result consists of mappings for a large number of variables it can become difficult to assign meaning to each attribute displayed based solely on the names of variables. Arguably the semantics can be retrieved by considering the graph patterns that have generated the result set. However this approach restricts the result semantics to those of the original schema, and can be inappropriate in a data integration setting. Instead, an RDF projection of the mappings through the use of a CONSTRUCT clause can provide both flexibility on the representation of results and clarity with respect to their semantics.

The provision of RDF paginated view poses further challenges. However, ordered lists are represented in RDF through a collection of pairs consisting of a value, and a pointer to the next pair through the use of the rdf:first and rdf:rest predicates. As the rdf:first predicate has to be iteratively applied to individual mappings for the same variable, SPARQL RDF projections cannot be used to generate such structures. That is, a CONSTRUCT query cannot be use to generate an RDF list from its' solutions. Thus, to achieve pagination of RDF projections a two-step process is required. First, the URIs for items that a page consists of must be identified through a SELECT query so that they can be retrieved in the desired order. These are used to generate the page template through rdf:first and rdf:rest. Subsequently, a CONSTRUCT projection query is issued to obtain the required attributes for each item. To do so, this query must specify alternative URIs, each corresponding to one item retrieved in the previous step. The resulting RDF can then be appended to the page template and forwarded to the client.

**Sorting aggregate result sequences**

The use of aggregates introduces some additional complications. To quote [6] "In aggregate queries and sub-queries, variables that appear in the query pattern, but are not in the GROUP BY clause, can only be projected or used in select expressions if they are aggregated". That is, any variables that are not aggregated must appear inside a GROUP BY clause. In turn, any sorting specified via an ORDER BY clause will only be applied inside the generated groups, while the result sequence as a whole will remain in arbitrary order.

An effective way to sort the entire result sequence is through the use of a sub-query which contains only aggregated variables, and thus does not require grouping. The sort order can then be specified at the outermost query and will be applied to the entire result sequence.

Finally, in order to enable results to be sorted and filtered arbitrarily with respect to the values mapped to each of the variables, the graph patterns that retrieve these values must also appear in the first SELECT query so that the correct URIs are available for use in the CONSTRUCT query.

Here, one can see how these heuristics can be applied in practice.

## 7. Related work

Query optimisation is one of the most important research topics in databases and query languages [33]. Since the publication of the initial W3C working drafts of SPARQL, there have been several works discussing techniques for SPARQL query optimisation. Here, we focus on this area.

In the first formal study of SPARQL [34], the authors presented a formal definition of its syntax and semantics, studied the complexity of evaluating graph patterns, and discussed some optimisation procedures. The bounds presented in this study indicate that the complexity of SPARQL query evaluation does not only depend on the operators used, but also on the syntactic form of the queries. In this sense, a class of graph patterns for which the evaluation problem can be solved more efficiently was identified. Additionally, the authors proposed a simple normal form based on several rewriting rules for well-designed graph patterns, and showed their impact in the evaluation of SPARQL queries.

Hartig and Heese [35] proposed a query graph model to study all phases of query processing in SPARQL. They defined transformation rules for query rewriting and developed heuristics used in the selection of efficient query execution plans.

[20] The obsolete BINDINGS syntax was used for 4store.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

The problem of SPARQL basic graph pattern optimisation was studied in [21]. In this work, the authors defined and analysed the characteristics of heuristics for selectivity-based optimisation of simple and joined triple patterns. Selectivity is defined as the fraction of triples in an RDF dataset that contain a bound subject, predicate or object in a sub-graph pattern. The authors provide empirical results using the Lehigh University Benchmark (LUBM) [30] benchmark to compare a number of models to determine the optimal ordering of graph patterns. It shows significant improvements in the execution times of SPARQL queries when their constituent graph patterns are reordered based on the proposed heuristics.

The fundamental aspects related to SPARQL query optimisation were presented in [17]. This work is based on an in-depth analysis of the complexity of the SPARQL operators, showing that the PSpace-completeness of the language is given by the occurrence of OPTIONAL graph patterns. Then, the authors studied several optimisation techniques including well-known rewriting rules (e.g. filter and projection pushing) and semantic optimisation based on the classical chase algorithm.

In the context of query optimisation in distributed environments, Kaoudi et al. [36] proposed efficient and scalable algorithms for optimising SPARQL queries implemented over distributed hash tables. The proposed techniques were implemented and evaluated over a cluster running the system Atlas. In a similar context, Schwarte et al. [37] studied optimisation techniques for federated query processing on Linked Data. The authors proposed novel join processing and grouping techniques to minimise the number of remote requests, and provided a solution for RDF data source selection. These techniques were described in the context of FedX, a framework for SPARQL query processing on heterogeneous, virtuality integrated Linked Data sources.

In [38], real world queries obtained from the DBPedia SPARQL endpoint were studied. The authors found that UNION and OPTIONAL graph patterns are common in practice and they are hard to evaluate. The syntactical structure of the queries was studied and structural restrictions that imply tractable evaluation were identified. The authors argued that the identified restrictions can be used to develop practical heuristics for query optimisation.

Tsialiamanis et al. [39] proposed a set of useful heuristics for SPARQL query optimisers. These heuristics were used to construct a SPARQL query planner which allows to exploit the syntactic and structural variations of the triple patterns and to choose an optimal execution plan without the need of any cost model. The proposed techniques were implemented and evaluated on top of the MonetDB system, and compared with RDF-3X and a relational database.

Our work differs from the above in that the goal is to provide practical guidance for developers that is applicable independently of the underlying RDF store.

## 8. Conclusions

This paper presented a set of 5 heuristics that can be used to guide the formulation of performant SPARQL queries. These heuristics were inspired by formal results found in the literature as well as hands on experience in developing an end-user focused data integration system. The heuristics are proposed as a first step towards helping developers formulate SPARQL queries that are more in-line with the capabilities of state-of-the-art RDF stores. In addition, we hope that this work can help RDF store developers to further optimise their stores.

The heuristics were first formally defined in Section 4 and subsequently evaluated in Section 5, using openly available real world data and queries used in the OpenPHACTS project. We reiterate that the queries used in our evaluation were originally obtained through collaboration with domain experts, which resulted in defining two versions of the query set, Q1–Q4 and Q9–Q12, which were both unacceptable in terms of performance. The application of the heuristics detailed herein allowed for these templates to be re-written and become well-performing. The large uptake of the OpenPHACTS API, which receives at minimum 1 million hits per month is indicative of this fact.

While the results presented in the previous section show performance improvements obtained through the application of the heuristics in most cases, it is important to note that there is a large degree of variability. With that in mind, the only instances of a heuristic having a negative impact on query performance concern the introduction of named graphs to a query with no `OPTIONAL` clauses which in our experience rarely occurs. Based on these results, we argue that the heuristics presented herein can provide a valuable tool in formulating performant SPARQL queries.

The provision of paginated RDF views was considered as a common place application scenario where the application of the heuristics is beneficial. A number of challenges have been identified and solutions based on the heuristics have been proposed for this common use-case.

The large degree of variability observed both across different RDF stores and individual queries provide strong motivation to iteratively test and measure response times for queries considered to drive application requirements. SPARQL, due to its expressiveness, provides a plethora of different ways to express the same constraints, thus, developers need to be aware of the performance implications of the combination of query formulation and RDF Store. This work provides empirical evidence that can help developers in designing queries for their selected RDF Store. However, this raises questions about the effectives of writing complex *generic* queries that work across open SPARQL endpoints available in the Linked Open Data Cloud. We view the optimisation of queries independent of underlying RDF Store technology as a critical area of research to enable the most effective use of these endpoints.

Of course, it is possible to implement the heuristics proposed inside RDF store optimisers. The overhead of applying the proposed algorithms and calculating statistics in an exhaustive manner would be prohibitive, but necessary in a bulk loading scenario.

We believe that questions concerning the performance of particular systems, their implications on the indices stored and the manner in which they are used are best answered by the RDF store vendors and developers themselves. Instead our focus is placed on optimising SPARQL queries outside of the RDF store, thus making the heuristics applicable immediately for any developer in this space, independently from the RDF store used.

Future work includes the identification of further heuristics for the formulation of performant SPARQL queries and the study of properties that these queries share. Further, we plan to study to what extent the efficacy of the heuristics depends on the query processor implementation and in particular on whether the query processor applies well established algebraic rewritings. Moreover, we intend to use the OpenPHACTS datasets and queries in the creation of Linked Data benchmarks in the context of the Linked Data Benchmarking Council project.

## Acknowledgements

The research leading to these results has received support from the Innovative Medicines Initiative Joint Undertaking under grant agreement number IMI 115191, resources of which are composed of financial contribution from the European Union's Seventh Framework Programme (FP7/2007–2013) and EFPIA companies' in kind contribution.

This work was partially supported by EU project LDBC (FP7-317548).[21] Renzo Angles was funded by Fondecyt Chile grant 11100364.

## Appendix A. SPARQL queries

The queries used in the evaluation of the heuristics proposed in this paper are given here. Placeholders for the input parameters are denoted by [RESOURCE].

### Prefixes

```
PREFIX rdf: <http://www.w3.org/1999/02/
  22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01
  /rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX skos:
<http://www.w3.org/2004/02/skos/core#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX cito: <http://purl.org/spar/cito/>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX ss:
<http://semanticscience.org/resource/>
PREFIX cheminf: <http://semanticscience.org/
  ontology/cheminf.owl/>
PREFIX cs: <http://rdf.chemspider.com/#>
PREFIX db: <http://www4.wiwiss.fu-berlin.de/
  drugbank/resource/drugbank/>
PREFIX chembl: <http://rdf.farmbio.uu.se
  /chembl/onto/#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX qudt:
<http://qudt.org/1.1/schema/qudt#>
```

### Q1: Compound information (initial)

```
SELECT ?compound_name ?smiles ?inchi ?cs_uri
?inchiKey ?equiv_compound ?chembl_uri ?bNode1
?molformula ?bNode2 ?molweight ?bNode3
?mw_freebase ?bNode4 ?rtb ?db_uri
?affectedOrganism ?biotrans ?description
?indication ?proteinBinding ?toxicity
?meltingPoint
WHERE {
  [RESOURCE] skos:prefLabel ?compound_name .
  [RESOURCE] skos:exactMatch ?cs_uri .
  ?cs_uri cs:smiles ?smiles .
  ?cs_uri cs:inchi ?inchi .
  ?cs_uri cs:inchikey ?inchiKey .
  ?equiv_compound skos:exactMatch ?cs_uri .
  ?chembl_uri owl:equivalentClass
?equiv_compound .
  ?chembl_uri ss:CHEMINF_000200 ?bNode1 .
  ?bNode1 rdf:type ss:CHEMINF_000042 .
  ?bNode1 ss:SIO_000300 ?molformula .
  ?chembl_uri ss:CHEMINF_000200 ?bNode2 .
  ?bNode2 rdf:type ss:CHEMINF_000198 .
  ?bNode2 ss:SIO_000300 ?molweight .
  ?chembl_uri ss:CHEMINF_000200 ?bNode3 .
  ?bNode3 rdf:type ss:CHEMINF_000350 .
  ?bNode3 ss:SIO_000300 ?mw_freebase .
  ?chembl_uri ss:CHEMINF_000200 ?bNode4 .
  ?bNode4 rdf:type ss:CHEMINF_000311 .
  ?bNode4 ss:SIO_000300 ?rtb .
  [RESOURCE] skos:exactMatch ?db_uri .
  ?db_uri db:affectedOrganism
?affectedOrganism .
  ?db_uri db:biotransformation ?biotrans .
  ?db_uri db:description ?description .
  ?db_uri db:indication ?indication .
  ?db_uri db:proteinBinding ?proteinBinding .
  ?db_uri db:toxicity ?toxicity .
  ?db_uri db:meltingPoint ?meltingPoint .
}
```

### Q2: Compound pharmacology (initial)

```
SELECT ?compound_name ?cs_uri ?smiles ?inchi
?inchiKey ?num_ro5_violations ?chembl_uri
?activity_uri ?assay_uri ?equiv_compound
?equiv_assay ?target_uri ?equiv_target
?target_name ?target_organism ?assay_desc
?assay_organism ?std_type ?relation ?std_value
?std_unit ?doi_int ?doi ?pmid ?bNode1
?molweight ?db_uri ?drug_name ?drugType_uri
?drugType
WHERE {
  [RESOURCE] skos:prefLabel ?compound_name .
  [RESOURCE] skos:exactMatch ?cs_uri .
  ?cs_uri cs:smiles ?smiles .
  ?cs_uri cs:inchi ?inchi .
  ?cs_uri cs:inchikey ?inchiKey .
  _:1 obo:IAO_0000136 ?cs_uri .
  _:1 rdf:type cheminf:CHEMINF_000367 .
  _:1 numericValue ?num_ro5_violations .
  ?equiv_compound skos:exactMatch ?cs_uri .
  ?chembl_uri owl:equivalentClass
?equiv_compound .
  ?activity_uri chembl:forMolecule
?chembl_uri.
  ?activity_uri chembl:onAssay ?assay_uri .
  ?assay_uri owl:equivalentClass
?equiv_assay .
  ?assay_uri chembl:hasTarget ?target_uri .
  ?target_uri owl:equivalentClass
?equiv_target .
  ?target_uri dc:title ?target_name .
  ?target_uri chembl:organism
?target_organism .
  ?assay_uri chembl:hasDescription
?assay_desc .
  ?assay_uri chembl:organism ?assay_organism .
  ?activity_uri chembl:type ?std_type .
  ?activity_uri chembl:relation ?relation .
  ?activity_uri chembl:standardValue
?std_value .
  ?activity_uri chembl:standardUnits
?std_unit .
  ?activity_uri cito:citesAsDataSource
?doi_int .
  ?doi_int owl:sameAs ?doi .
  ?doi_int bibo:pmid ?pmid .
  ?chembl_uri ss:CHEMINF_000200 ?bNode1 .
  ?bNode1 rdf:type ss:CHEMINF_000198 .
  ?bNode1 ss:SIO_000300 ?molweight .
  [RESOURCE] skos:exactMatch ?db_uri .
  ?db_uri db:genericName ?drug_name .
```

[21] http://ldbc.eu.

```
  ?db_uri db:drugType ?drugType_uri .
  ?drugType_uri rdfs:label ?drugType .
}
```

**Q3: Target information (initial)**

```
SELECT ?target_name ?alt_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
?numberOfResidues ?theoreticalPi
WHERE{
  [RESOURCE] skos:prefLabel ?target_name.
  [RESOURCE] skos:exactMatch ?chembl_uri .
  ?chembl_uri owl:equivalentClass
?equiv_target .
  ?chembl_uri chembl:hasKeyword ?keyword .
  ?chembl_uri chembl:hasDescription
?description .
  ?chembl_uri rdfs:subClassOf ?target_type .
  ?chembl_uri rdfs:label ?synonym .
  [RESOURCE] skos:exactMatch ?db_uri .
  ?db_uri db:cellularLocation
?cellularLocation .
  ?db_uri db:molecularWeight
?molecularWeight .
  ?db_uri db:numberOfResidues
?numberOfResidues .
  ?db_uri db:theoreticalPi ?theoreticalPi .
}
```

**Q4: Target pharmacology (initial)**

```
SELECT ?target_name ?assay_uri ?chembl_uri
?equiv_assay ?equiv_target ?activity_uri
?compound_chembl ?equiv_compound ?compound_cs
?target_organism ?assay_organism ?assay_desc
?std_type ?relation ?std_value ?std_unit
?doi_int ?doi ?pmid ?node1 ?molweight ?inchi
?inchi_key ?smiles ?node2 ?num_ro5_violations
?compound_cw ?compound_name
WHERE{
  [RESOURCE] skos:prefLabel ?target_name .
  [RESOURCE] skos:exactMatch ?chembl_uri .
  ?assay_uri chembl:hasTarget ?chembl_uri .
  ?assay_uri owl:equivalentClass
?equiv_assay .
  ?chembl_uri owl:equivalentClass
?equiv_target .
  ?activity_uri chembl:onAssay ?assay_uri .
  ?activity_uri chembl:forMolecule
?compound_chembl .
  ?compound_chembl owl:equivalentClass
?equiv_compound .
  ?equiv_compound skos:exactMatch
?compound_cs .
  ?chembl_uri chembl:organism
?target_organism .
  ?assay_uri chembl:organism ?assay_organism .
  ?assay_uri chembl:hasDescription
?assay_desc .
  ?activity_uri chembl:type ?std_type .
  ?activity_uri chembl:relation ?relation .
  ?activity_uri chembl:standardValue
?std_value .
  ?activity_uri chembl:standardUnits
?std_unit .
  ?activity_uri cito:citesAsDataSource
?doi_int .
  ?doi_int owl:sameAs ?doi .
  ?doi_int bibo:pmid ?pmid .
  ?compound_chembl ss:CHEMINF_000200 ?node1 .
  ?node1 rdf:type ss:CHEMINF_000198 .
  ?node1 ss:SIO_000300 ?molweight .
  ?compound_cs cs:inchi ?inchi .
  ?compound_cs cs:inchikey ?inchi_key .
  ?compound_cs cs:smiles ?smiles .
  ?node2 obo:IAO_0000136 ?compound_cs .
  ?node2 rdf:type cheminf:CHEMINF_000367 .
  ?node2 numericValue ?num_ro5_violations .
  ?compound_cw skos:exactMatch ?compound_cs .
  ?compound_cw skos:prefLabel ?compound_name .
}
```

**Q5: Compound information—graph**

```
SELECT ?compound_name ?smiles ?inchi ?cs_uri
?inchiKey ?equiv_compound ?chembl_uri ?bNode1
?molformula ?bNode2 ?molweight ?bNode3
?mw_freebase ?bNode4 ?rtb ?db_uri
?affectedOrganism ?biotrans ?description
?indication ?proteinBinding ?toxicity
?meltingPoint
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://www.chemspider.com>
FROM NAMED <http://data.kasabi.com/
  dataset/chembl-rdf>
FROM NAMED <http://linkedlifedata.com/
  resource/drugbank>
WHERE {
  GRAPH <http://www.conceptwiki.org> {
    [RESOURCE] skos:prefLabel ?compound_name .
    [RESOURCE] skos:exactMatch ?cs_uri .
  }
  GRAPH <http://www.chemspider.com> {
    ?cs_uri cs:smiles ?smiles .
    ?cs_uri cs:inchi ?inchi .
    ?cs_uri cs:inchikey ?inchiKey .
  }
  GRAPH <http://data.kasabi.com/
    dataset/chembl-rdf> {
    ?equiv_compound skos:exactMatch ?cs_uri .
    ?chembl_uri owl:equivalentClass
     ?equiv_compound .
    ?chembl_uri ss:CHEMINF_000200 ?bNode1 .
    ?bNode1 rdf:type ss:CHEMINF_000042 .
    ?bNode1 ss:SIO_000300 ?molformula .
    ?chembl_uri ss:CHEMINF_000200 ?bNode2 .
    ?bNode2 rdf:type ss:CHEMINF_000198 .
    ?bNode2 ss:SIO_000300 ?molweight .
    ?chembl_uri ss:CHEMINF_000200 ?bNode3 .
    ?bNode3 rdf:type ss:CHEMINF_000350 .
    ?bNode3 ss:SIO_000300 ?mw_freebase .
    ?chembl_uri ss:CHEMINF_000200 ?bNode4 .
    ?bNode4 rdf:type ss:CHEMINF_000311 .
    ?bNode4 ss:SIO_000300 ?rtb .
  }
  GRAPH <http://linkedlifedata.com/resource/
    drugbank> {
    [RESOURCE] skos:exactMatch ?db_uri .
```

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

```
    ?db_uri db:affectedOrganism
?affectedOrganism .
    ?db_uri db:biotransformation ?biotrans .
    ?db_uri db:description ?description .
    ?db_uri db:indication ?indication .
    ?db_uri db:proteinBinding
?proteinBinding .
    ?db_uri db:toxicity ?toxicity .
    ?db_uri db:meltingPoint ?meltingPoint .
  }
}
```

**Q6: Compound pharmacology—graph**

```
SELECT ?compound_name ?cs_uri ?smiles ?inchi
?inchiKey ?num_ro5_violations ?chembl_uri
?activity_uri ?assay_uri ?equiv_compound
?equiv_assay ?target_uri ?equiv_target
?target_name ?target_organism ?assay_desc
?assay_organism ?std_type ?relation
?std_value ?std_unit ?doi_int ?doi ?pmid
?bNode1 ?molweight ?db_uri ?drug_name
?drugType_uri ?drugType
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://www.chemspider.com>
FROM NAMED <http://data.kasabi.com/
  dataset/chembl-rdf>
FROM NAMED <http://linkedlifedata.com/
  resource/drugbank>
WHERE {
  GRAPH <http://www.conceptwiki.org> {
     [RESOURCE] skos:prefLabel ?compound_name .
     [RESOURCE] skos:exactMatch ?cs_uri .
  }
  GRAPH <http://www.chemspider.com> {
     ?cs_uri cs:smiles ?smiles .
     ?cs_uri cs:inchi ?inchi .
     ?cs_uri cs:inchikey ?inchiKey .
     _:1 obo:IAO_0000136 ?cs_uri .
     _:1 rdf:type cheminf:CHEMINF_000367 .
     _:1 numericValue ?num_ro5_violations .
  }
  GRAPH <http://data.kasabi.com/
     dataset/chembl-rdf> {
     ?equiv_compound skos:exactMatch ?cs_uri .
     ?chembl_uri owl:equivalentClass
?equiv_compound .
     ?activity_uri chembl:forMolecule
?chembl_uri .
     ?activity_uri chembl:onAssay ?assay_uri .
     ?assay_uri owl:equivalentClass
?equiv_assay .
     ?assay_uri chembl:hasTarget ?target_uri .
     ?target_uri owl:equivalentClass
?equiv_target .
     ?target_uri dc:title ?target_name .
     ?target_uri chembl:organism
?target_organism .
     ?assay_uri chembl:hasDescription
?assay_desc.
     ?assay_uri chembl:organism
?assay_organism .
     ?activity_uri chembl:type ?std_type .
     ?activity_uri chembl:relation ?relation .
     ?activity_uri chembl:standardValue
     ?std_value .
     ?activity_uri chembl:standardUnits
?std_unit .
     ?activity_uri cito:citesAsDataSource
?doi_int .
     ?doi_int owl:sameAs ?doi .
     ?doi_int bibo:pmid ?pmid .
     ?chembl_uri ss:CHEMINF_000200 ?bNode1 .
     ?bNode1 rdf:type ss:CHEMINF_000198 .
     ?bNode1 ss:SIO_000300 ?molweight .
  }
  GRAPH <http://linkedlifedata.com/
     resource/drugbank> {
     [RESOURCE] skos:exactMatch ?db_uri .
     ?db_uri db:genericName ?drug_name .
     ?db_uri db:drugType ?drugType_uri .
     ?drugType_uri rdfs:label ?drugType .
  }
}
```

**Q7: Target information—graph**

```
SELECT ?target_name ?alt_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
?numberOfResidues ?theoreticalPi
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://data.kasabi.com/
  dataset/chembl-rdf>
FROM NAMED <http://linkedlifedata.com/
  resource/drugbank>
WHERE {
  GRAPH <http://www.conceptwiki.org> {
     [RESOURCE] skos:prefLabel ?target_name.
  }
  GRAPH <http://data.kasabi.com/
     dataset/chembl-rdf> {
     [RESOURCE] skos:exactMatch ?chembl_uri .
     ?chembl_uri owl:equivalentClass
?equiv_target .
     ?chembl_uri chembl:hasKeyword ?keyword .
     ?chembl_uri chembl:hasDescription
?description .
     ?chembl_uri rdfs:subClassOf ?target_type .
     ?chembl_uri rdfs:label ?synonym .
  }
  GRAPH <http://linkedlifedata.com/
     resource/drugbank> {
     [RESOURCE] skos:exactMatch ?db_uri .
     ?db_uri db:cellularLocation
?cellularLocation .
     ?db_uri db:molecularWeight
?molecularWeight .
     ?db_uri db:numberOfResidues
?numberOfResidues .
     ?db_uri db:theoreticalPi ?theoreticalPi .
  }
}
```

**Q8: Target pharmacology—graph**

```
SELECT ?target_name ?assay_uri ?chembl_uri
?equiv_assay ?equiv_target ?activity_uri
?compound_chembl ?equiv_compound ?compound_cs
?target_organism ?assay_organism ?assay_desc
?std_type ?relation ?std_value ?std_unit
```

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

```
?doi_int ?doi ?pmid ?node1 ?molweight ?inchi
?inchi_key ?smiles ?node2 ?num_ro5_violations
?compound_cw ?compound_name
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://data.kasabi.com/
  dataset/chembl-rdf>
FROM NAMED <http://www.chemspider.com>
WHERE {
  GRAPH <http://www.conceptwiki.org> {
     [RESOURCE] skos:prefLabel ?target_name .
     ?compound_cw skos:exactMatch
?compound_cs .
     ?compound_cw skos:prefLabel
?compound_name .
  }
  GRAPH <http://data.kasabi.com/
     dataset/chembl-rdf> {
     [RESOURCE] skos:exactMatch ?chembl_uri .
     ?assay_uri chembl:hasTarget ?chembl_uri .
     ?assay_uri owl:equivalentClass
?equiv_assay .
     ?chembl_uri owl:equivalentClass
?equiv_target .
     ?activity_uri chembl:onAssay ?assay_uri .
     ?activity_uri chembl:forMolecule
      ?compound_chembl .
     ?compound_chembl owl:equivalentClass
     ?equiv_compound.
     ?equiv_compound skos:exactMatch
?compound_cs .
     ?chembl_uri chembl:organism
?target_organism .
     ?assay_uri chembl:organism
?assay_organism .
     ?assay_uri chembl:hasDescription
?assay_desc.
     ?activity_uri chembl:type ?std_type .
     ?activity_uri chembl:relation ?relation .
     ?activity_uri chembl:standardValue
?std_value .
     ?activity_uri chembl:standardUnits
?std_unit .
     ?activity_uri cito:citesAsDataSource
?doi_int .
     ?doi_int owl:sameAs ?doi .
     ?doi_int bibo:pmid ?pmid .
     ?compound_chembl ss:CHEMINF_000200
?node1 .
     ?node1 rdf:type ss:CHEMINF_000198 .
     ?node1 ss:SIO_000300 ?molweight .
  }
  GRAPH <http://www.chemspider.com> {
     ?compound_cs cs:inchi ?inchi .
     ?compound_cs cs:inchikey ?inchi_key .
     ?compound_cs cs:smiles ?smiles .
     ?node2 obo:IAO_0000136 ?compound_cs .
     ?node2 rdf:type cheminf:CHEMINF_000367 .
     ?node2 numericValue ?num_ro5_violations .
  }
}
```

**Q9: Compound information—optional**

```
SELECT ?compound_name ?smiles ?inchi ?cs_uri
?inchiKey ?equiv_compound ?chembl_uri ?bNode1
?molformula ?bNode2 ?molweight ?bNode3
?mw_freebase ?bNode4 ?rtb ?db_uri
?affectedOrganism ?biotrans ?description
?indication ?proteinBinding ?toxicity
?meltingPoint
WHERE {
  [RESOURCE] skos:prefLabel ?compound_name .
  [RESOURCE] skos:exactMatch ?cs_uri .
  ?cs_uri cs:smiles ?smiles .
  ?cs_uri cs:inchi ?inchi .
  ?cs_uri cs:inchikey ?inchiKey .
  OPTIONAL { ?equiv_compound skos:exactMatch
                    ?cs_uri .
    OPTIONAL { ?chembl_uri owl:equivalentClass
                    ?equiv_compound . }
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode1 .
    OPTIONAL { ?bNode1 rdf:type
ss:CHEMINF_000042 .
    OPTIONAL { ?bNode1 ss:SIO_000300
                    ?molformula . } } }
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode2 .
    OPTIONAL { ?bNode2 rdf:type
ss:CHEMINF_000198 .
    OPTIONAL { ?bNode2 ss:SIO_000300
                    ?molweight . } } }
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode3 .
    OPTIONAL { ?bNode3 rdf:type
ss:CHEMINF_000350 .
    OPTIONAL { ?bNode3 ss:SIO_000300
                    ?mw_freebase . } } }
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode4 .
    OPTIONAL { ?bNode4 rdf:type
ss:CHEMINF_000311 .
    OPTIONAL { ?bNode4 ss:SIO_000300 ?rtb . }
} } }
  OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri .
    OPTIONAL { ?db_uri db:affectedOrganism
                    ?affectedOrganism. }
    OPTIONAL { ?db_uri db:biotransformation
                    ?biotrans . }
    OPTIONAL { ?db_uri db:description
                    ?description . }
    OPTIONAL { ?db_uri db:indication
?indication . }
    OPTIONAL { ?db_uri db:proteinBinding
                    ?proteinBinding . }
    OPTIONAL { ?db_uri db:toxicity ?toxicity .
}
    OPTIONAL { ?db_uri db:meltingPoint
                    ?meltingPoint . } }
}
```

**Q10: Compound pharmacology—optional**

```
SELECT ?compound_name ?cs_uri ?smiles ?inchi
?inchiKey ?num_ro5_violations ?chembl_uri
?activity_uri ?assay_uri ?equiv_compound
?equiv_assay ?target_uri ?equiv_target
?target_name ?target_organism ?assay_desc
```

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

```
?assay_organism ?std_type ?relation
?std_value ?std_unit ?doi_int ?doi ?pmid
?bNode1 ?molweight ?db_uri ?drug_name
?drugType_uri ?drugType
WHERE {
  [RESOURCE] skos:prefLabel ?compound_name .
  [RESOURCE] skos:exactMatch ?cs_uri .
  ?cs_uri cs:smiles ?smiles .
  ?cs_uri cs:inchi ?inchi .
  ?cs_uri cs:inchikey ?inchiKey .
  ?equiv_compound skos:exactMatch ?cs_uri .
  ?chembl_uri owl:equivalentClass
?equiv_compound .
  ?activity_uri chembl:forMolecule
?chembl_uri .
  ?activity_uri chembl:onAssay ?assay_uri .
  ?assay_uri owl:equivalentClass
?equiv_assay .
  ?assay_uri chembl:hasTarget ?target_uri .
  ?target_uri owl:equivalentClass
?equiv_target .
  OPTIONAL { ?node obo:IAO_0000136 ?cs_uri .
   OPTIONAL { ?node rdf:type cheminf:
                    CHEMINF_000367 . }
   OPTIONAL { ?node numericValue
                    ?num_ro5_violations . } }
  OPTIONAL { ?target_uri dc:title
?target_name . }
  OPTIONAL { ?target_uri chembl:organism
                    ?target_organism .}
  OPTIONAL { ?assay_uri chembl:hasDescription
                    ?assay_desc .}
  OPTIONAL { ?assay_uri chembl:organism
                    ?assay_organism . }
  OPTIONAL { ?activity_uri chembl:type
?std_type . }
  OPTIONAL { ?activity_uri chembl:relation
                    ?relation . }
  OPTIONAL { ?activity_uri
chembl:standardValue
                    ?std_value.}
  OPTIONAL { ?activity_uri
chembl:standardUnits
                    ?std_unit.}
  OPTIONAL { ?activity_uri
cito:citesAsDataSource
                    ?doi_int .
   OPTIONAL { ?doi_int owl:sameAs ?doi . }
   OPTIONAL { ?doi_int bibo:pmid ?pmid . } }
  OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode1 .
   OPTIONAL { ?bNode1 rdf:type
ss:CHEMINF_000198 . }
   OPTIONAL { ?bNode1 ss:SIO_000300
?molweight . } }
  OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri .
   OPTIONAL { ?db_uri db:genericName
?drug_name . }
   OPTIONAL { ?db_uri db:drugType
?drugType_uri .
    OPTIONAL { ?drugType_uri rdfs:label
?drugType . } } }
}
```

**Q11: Target information—optional**

```
SELECT ?target_name ?alt_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
?numberOfResidues ?theoreticalPi
WHERE{
  [RESOURCE] skos:prefLabel ?target_name.
  OPTIONAL { [RESOURCE] skos:exactMatch
?chembl_uri .
   OPTIONAL { ?chembl_uri owl:equivalentClass
                    ?equiv_target.}
   OPTIONAL { ?chembl_uri chembl:hasKeyword
                    ?keyword . }
   OPTIONAL{?chembl_uri chembl:hasDescription
                    ?description.}
   OPTIONAL { ?chembl_uri rdfs:subClassOf
                    ?target_type . }
   OPTIONAL { ?chembl_uri rdfs:label
?synonym . } }
  OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri .
   OPTIONAL {?db_uri db:cellularLocation
                    ?cellularLocation.}
   OPTIONAL { ?db_uri db:molecularWeight
                    ?molecularWeight.}
   OPTIONAL {?db_uri db:numberOfResidues
                    ?numberOfResidues.}
   OPTIONAL { ?db_uri db:theoreticalPi
                    ?theoreticalPi . } }
}
```

**Q12: Target pharmacology—optional**

```
SELECT ?target_name ?assay_uri ?chembl_uri
?equiv_assay ?equiv_target ?activity_uri
?compound_chembl ?equiv_compound ?compound_cs
?target_organism ?assay_organism ?assay_desc
?std_type ?relation ?std_value ?std_unit
?doi_int ?doi ?pmid ?node1 ?molweight ?inchi
?inchi_key ?smiles ?node2 ?num_ro5_violations
?compound_cw ?compound_name
WHERE {
  [RESOURCE] skos:prefLabel ?target_name .
  [RESOURCE] skos:exactMatch ?chembl_uri .
  ?assay_uri chembl:hasTarget ?chembl_uri .
  ?assay_uri owl:equivalentClass
?equiv_assay .
  ?chembl_uri owl:equivalentClass
?equiv_target .
  ?activity_uri chembl:onAssay ?assay_uri .
  ?activity_uri chembl:forMolecule
?compound_chembl .
  ?compound_chembl owl:equivalentClass
                    ?equiv_compound .
  ?equiv_compound skos:exactMatch
?compound_cs .
  ?compound_cs cs:inchi ?inchi .
  ?compound_cs cs:inchikey ?inchi_key .
  ?compound_cs cs:smiles ?smiles .
  ?compound_cw skos:exactMatch ?compound_cs .
  ?compound_cw skos:prefLabel ?compound_name .
  OPTIONAL { ?chembl_uri chembl:organism
                    ?target_organism. }
  OPTIONAL { ?assay_uri chembl:organism
                    ?assay_organism . }
```

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

```
  OPTIONAL { ?assay_uri chembl:hasDescription
                      ?assay_desc. }
  OPTIONAL { ?activity_uri chembl:type
?std_type . }
  OPTIONAL { ?activity_uri chembl:relation
                      ?relation . }
  OPTIONAL { ?activity_uri
chembl:standardValue
                      ?std_value. }
  OPTIONAL { ?activity_uri
chembl:standardUnits
                      ?std_unit . }
  OPTIONAL { ?activity_uri
cito:citesAsDataSource
                      ?doi_int .
    OPTIONAL { ?doi_int owl:sameAs ?doi . }
    OPTIONAL { ?doi_int bibo:pmid ?pmid . } }
  OPTIONAL { ?compound_chembl
ss:CHEMINF_000200
                      ?node1 .
    OPTIONAL { ?node1 rdf:type
ss:CHEMINF_000198 .
    OPTIONAL { ?node1 ss:SIO_000300
?molweight . } } }
  OPTIONAL { ?node2 obo:IAO_0000136
?compound_cs .
    OPTIONAL { ?node2 rdf:type
cheminf:CHEMINF_000367 .
    OPTIONAL { ?node2 numericValue
?num_ro5_violations. } } }
}
```

**Q13: Compound information—min. optional**

```
SELECT ?compound_name ?smiles ?inchi ?cs_uri
?inchiKey ?equiv_compound ?chembl_uri ?bNode1
?molformula ?bNode2 ?molweight ?bNode3
?mw_freebase ?bNode4 ?rtb ?db_uri
?affectedOrganism ?biotrans ?description
?indication ?proteinBinding ?toxicity
?meltingPoint
WHERE {
  [RESOURCE] skos:prefLabel ?compound_name .
  [RESOURCE] skos:exactMatch ?cs_uri .
  ?cs_uri cs:smiles ?smiles .
  ?cs_uri cs:inchi ?inchi .
  ?cs_uri cs:inchikey ?inchiKey .
  OPTIONAL { ?equiv_compound skos:exactMatch
?cs_uri .
    ?chembl_uri owl:equivalentClass
?equiv_compound .
    ?chembl_uri ss:CHEMINF_000200 ?bNode1 .
    ?bNode1 rdf:type ss:CHEMINF_000042 .
    ?bNode1 ss:SIO_000300 ?molformula .
    ?chembl_uri ss:CHEMINF_000200 ?bNode2 .
    ?bNode2 rdf:type ss:CHEMINF_000198 .
    ?bNode2 ss:SIO_000300 ?molweight .
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode3 .
    ?bNode3 rdf:type ss:CHEMINF_000350 .
    ?bNode3 ss:SIO_000300 ?mw_freebase . }
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode4 .
    ?bNode4 rdf:type ss:CHEMINF_000311 .
    ?bNode4 ss:SIO_000300 ?rtb . } }
  OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri .
    OPTIONAL { ?db_uri db:affectedOrganism
                      ?affectedOrganism. }
    OPTIONAL { ?db_uri db:biotransformation
                      ?biotrans . }
    OPTIONAL { ?db_uri db:description
?description . }
    OPTIONAL { ?db_uri db:indication
?indication . }
    OPTIONAL { ?db_uri db:proteinBinding
                      ?proteinBinding . }
    OPTIONAL { ?db_uri db:toxicity ?toxicity .
}
    OPTIONAL { ?db_uri db:meltingPoint
                      ?meltingPoint . } }
}
```

**Q14: Compound pharmacology—min. optional**

```
SELECT ?compound_name ?cs_uri ?smiles ?inchi
?inchiKey ?num_ro5_violations ?chembl_uri
?activity_uri ?assay_uri ?equiv_compound
?equiv_assay ?target_uri ?equiv_target
?target_name ?target_organism ?assay_desc
?assay_organism ?std_type ?relation
?std_value ?std_unit ?doi_int ?doi ?pmid
?bNode1 ?molweight ?db_uri ?drug_name
?drugType_uri ?drugType
WHERE {
  [RESOURCE] skos:prefLabel ?compound_name .
  [RESOURCE] skos:exactMatch ?cs_uri .
  ?cs_uri cs:smiles ?smiles .
  ?cs_uri cs:inchi ?inchi .
  ?cs_uri cs:inchikey ?inchiKey .
  ?equiv_compound skos:exactMatch ?cs_uri .
  ?chembl_uri owl:equivalentClass
?equiv_compound .
  ?activity_uri chembl:forMolecule
?chembl_uri .
  ?activity_uri chembl:onAssay ?assay_uri .
  ?assay_uri owl:equivalentClass
?equiv_assay .
  ?assay_uri chembl:hasTarget ?target_uri .
  ?target_uri owl:equivalentClass
?equiv_target .
  ?target_uri dc:title ?target_name .
  OPTIONAL { _:1 obo:IAO_0000136 ?cs_uri .
    _:1 rdf:type cheminf:CHEMINF_000367 .
    _:1 numericValue ?num_ro5_violations . }
  OPTIONAL { ?target_uri chembl:organism
                      ?target_organism. }
  OPTIONAL { ?assay_uri chembl:hasDescription
                      ?assay_desc. }
  OPTIONAL { ?assay_uri chembl:organism
                      ?assay_organism . }
  OPTIONAL { ?activity_uri chembl:type
?std_type . }
  OPTIONAL { ?activity_uri chembl:relation
                      ?relation . }
  OPTIONAL { ?activity_uri
chembl:standardValue
                      ?std_value. }
  OPTIONAL { ?activity_uri
chembl:standardUnits
                      ?std_unit. }
```

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

```
  OPTIONAL { ?activity_uri
cito:citesAsDataSource
                    ?doi_int .
    OPTIONAL { ?doi_int owl:sameAs ?doi . }
    OPTIONAL { ?doi_int bibo:pmid ?pmid . } }
  OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode1 .
    ?bNode1 rdf:type ss:CHEMINF_000198 .
    ?bNode1 ss:SIO_000300 ?molweight . }
  OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri .
    ?db_uri db:genericName ?drug_name .
    OPTIONAL { ?db_uri db:drugType
?drugType_uri .
     ?drugType_uri rdfs:label ?drugType . } }
}
```

**Q15: Target information—min. optional**

```
SELECT ?target_name ?alt_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
?numberOfResidues ?theoreticalPi
WHERE {
  [RESOURCE] skos:prefLabel ?target_name.
  OPTIONAL { [RESOURCE] skos:exactMatch
?chembl_uri .
    ?chembl_uri owl:equivalentClass
?equiv_target .
    ?chembl_uri rdfs:subClassOf ?target_type .
    ?chembl_uri rdfs:label ?synonym .
    OPTIONAL { ?chembl_uri chembl:hasKeyword
                    ?keyword . }
    OPTIONAL { ?chembl_uri
chembl:hasDescription
                    ?description . } }
    OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri .
    OPTIONAL { ?db_uri db:cellularLocation
                ?cellularLocation . }
    OPTIONAL { ?db_uri db:molecularWeight
                    ?molecularWeight.}
    OPTIONAL{?db_uri db:numberOfResidues
                    ?numberOfResidues.}
    OPTIONAL { ?db_uri db:theoreticalPi
                    ?theoreticalPi. } }
}
```

**Q16: Target pharmacology—min. optional**

```
SELECT ?target_name ?assay_uri ?chembl_uri
?equiv_assay ?equiv_target ?activity_uri
?compound_chembl ?equiv_compound ?compound_cs
?target_organism ?assay_organism ?assay_desc
?std_type ?relation ?std_value ?std_unit
?doi_int ?doi ?pmid ?node1 ?molweight ?inchi
?inchi_key ?smiles ?node2 ?num_ro5_violations
?compound_cw ?compound_name
WHERE {
  [RESOURCE] skos:prefLabel ?target_name .
  [RESOURCE] skos:exactMatch ?chembl_uri .
  ?assay_uri chembl:hasTarget ?chembl_uri .
  ?assay_uri owl:equivalentClass
?equiv_assay .
  ?chembl_uri owl:equivalentClass
?equiv_target .
  ?activity_uri chembl:onAssay ?assay_uri .
  ?activity_uri chembl:forMolecule
?compound_chembl .
  ?compound_chembl owl:equivalentClass
?equiv_compound .
  ?equiv_compound skos:exactMatch
?compound_cs .
  ?compound_cs cs:inchi ?inchi .
  ?compound_cs cs:inchikey ?inchi_key .
  ?compound_cs cs:smiles ?smiles .
  ?compound_cw skos:exactMatch ?compound_cs .
  ?compound_cw skos:prefLabel ?compound_name .
  OPTIONAL { ?chembl_uri chembl:organism
?target_organism. }
  OPTIONAL { ?assay_uri chembl:organism
?assay_organism . }
  OPTIONAL { ?assay_uri chembl:hasDescription
?assay_desc . }
  OPTIONAL { ?activity_uri chembl:type
?std_type . }
  OPTIONAL { ?activity_uri chembl:relation
?relation . }
  OPTIONAL { ?activity_uri
chembl:standardValue ?std_value. }
  OPTIONAL { ?activity_uri
chembl:standardUnits ?std_unit . }
  OPTIONAL { ?activity_uri
cito:citesAsDataSource ?doi_int .
   OPTIONAL { ?doi_int owl:sameAs ?doi . }
   OPTIONAL { ?doi_int bibo:pmid ?pmid . } }
  OPTIONAL { ?compound_chembl
ss:CHEMINF_000200 ?node1 .
   ?node1 rdf:type ss:CHEMINF_000198 .
   ?node1 ss:SIO_000300 ?molweight . }
  OPTIONAL { ?node2 obo:IAO_0000136
?compound_cs .
   ?node2 rdf:type cheminf:CHEMINF_000367 .
   ?node2 numericValue ?num_ro5_violations . }
}
```

**Q17: Compound information—graph optional**

```
SELECT ?compound_name ?smiles ?inchi ?cs_uri
?inchiKey ?equiv_compound ?chembl_uri
?bNode1 ?molformula ?bNode2 ?molweight
?bNode3 ?mw_freebase ?bNode4 ?rtb ?db_uri
?affectedOrganism ?biotrans ?description
?indication ?proteinBinding ?toxicity
?meltingPoint
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://www.chemspider.com>
FROM NAMED <http://data.kasabi.com/
  dataset/chembl-rdf>
FROM NAMED <http://linkedlifedata.com/
  resource/drugbank>
WHERE {
  GRAPH <http://www.conceptwiki.org> {
    [RESOURCE] skos:prefLabel ?compound_name .
    [RESOURCE] skos:exactMatch ?cs_uri .
  }
  GRAPH <http://www.chemspider.com> {
    ?cs_uri cs:smiles ?smiles .
    ?cs_uri cs:inchi ?inchi .
```

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

```
   ?cs_uri cs:inchikey ?inchiKey .
  }
  OPTIONAL {
   GRAPH <http://data.kasabi.com/
     dataset/chembl-rdf> {
    ?equiv_compound skos:exactMatch ?cs_uri .
    ?chembl_uri owl:equivalentClass
      ?equiv_compound .
    ?chembl_uri ss:CHEMINF_000200 ?bNode1 .
    ?bNode1 rdf:type ss:CHEMINF_000042 .
    ?bNode1 ss:SIO_000300 ?molformula .
    ?chembl_uri ss:CHEMINF_000200 ?bNode2 .
    ?bNode2 rdf:type ss:CHEMINF_000198 .
    ?bNode2 ss:SIO_000300 ?molweight .
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode3 .
     ?bNode3 rdf:type ss:CHEMINF_000350 .
     ?bNode3 ss:SIO_000300 ?mw_freebase . }
    OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode4 .
     ?bNode4 rdf:type ss:CHEMINF_000311 .
     ?bNode4 ss:SIO_000300 ?rtb . }
   }
  }
  OPTIONAL {
   GRAPH <http://linkedlifedata.com/
     resource/drugbank> {
    [RESOURCE] skos:exactMatch ?db_uri .
    OPTIONAL{?db_uri db:affectedOrganism
?affectedOrganism. }
    OPTIONAL { ?db_uri db:biotransformation
?biotrans . }
    OPTIONAL { ?db_uri db:description
?description . }
    OPTIONAL { ?db_uri db:indication
?indication . }
    OPTIONAL { ?db_uri db:proteinBinding
?proteinBinding . }
    OPTIONAL { ?db_uri db:toxicity ?toxicity .
}
    OPTIONAL { ?db_uri db:meltingPoint
?meltingPoint . }
   }
  }
}
```

**Q18: Compound pharmacology—graph optional**

```
SELECT ?compound_name ?cs_uri ?smiles ?inchi
?inchiKey ?num_ro5_violations ?chembl_uri
?activity_uri ?assay_uri ?equiv_compound
?equiv_assay ?target_uri ?equiv_target
?target_name ?target_organism ?assay_desc
?assay_organism ?std_type ?relation
?std_value ?std_unit ?doi_int ?doi ?pmid
?bNode1 ?molweight ?db_uri ?drug_name
?drugType_uri ?drugType
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://www.chemspider.com>
FROM NAMED <http://data.kasabi.com/
  dataset/chembl-rdf>
FROM NAMED <http://linkedlifedata.com/
  resource/drugbank>
WHERE {
  GRAPH <http://www.conceptwiki.org> {
    [RESOURCE] skos:prefLabel ?compound_name .
    [RESOURCE] skos:exactMatch ?cs_uri .
  }
  GRAPH <http://www.chemspider.com> {
   ?cs_uri cs:smiles ?smiles .
   ?cs_uri cs:inchi ?inchi .
   ?cs_uri cs:inchikey ?inchiKey .
  }
  GRAPH <http://data.kasabi.com/
    dataset/chembl-rdf> {
   ?equiv_compound skos:exactMatch ?cs_uri .
   ?chembl_uri owl:equivalentClass
?equiv_compound .
   ?activity_uri chembl:forMolecule
?chembl_uri .
   ?activity_uri chembl:onAssay ?assay_uri .
   ?assay_uri owl:equivalentClass
?equiv_assay .
   ?assay_uri chembl:hasTarget ?target_uri .
   ?target_uri owl:equivalentClass
?equiv_target .
   ?target_uri dc:title ?target_name .
   OPTIONAL { _:1 obo:IAO_0000136 ?cs_uri .
    _:1 rdf:type cheminf:CHEMINF_000367 .
    _:1 numericValue ?num_ro5_violations . }
   OPTIONAL { ?target_uri chembl:organism
?target_organism. }
   OPTIONAL { ?assay_uri chembl:hasDescription
?assay_desc. }
   OPTIONAL { ?assay_uri chembl:organism
?assay_organism . }
   OPTIONAL { ?activity_uri chembl:type
?std_type . }
   OPTIONAL { ?activity_uri chembl:relation
?relation . }
   OPTIONAL { ?activity_uri
chembl:standardValue ?std_value. }
   OPTIONAL { ?activity_uri
chembl:standardUnits ?std_unit. }
   OPTIONAL { ?activity_uri
cito:citesAsDataSource
                    ?doi_int.
    OPTIONAL { ?doi_int owl:sameAs ?doi . }
    OPTIONAL { ?doi_int bibo:pmid ?pmid . } }
   OPTIONAL { ?chembl_uri ss:CHEMINF_000200
?bNode1 .
    ?bNode1 rdf:type ss:CHEMINF_000198 .
    ?bNode1 ss:SIO_000300 ?molweight . }
  }
  OPTIONAL {
   GRAPH <http://linkedlifedata.com/
     resource/drugbank> {
    [RESOURCE] skos:exactMatch ?db_uri .
    ?db_uri db:genericName ?drug_name .
    OPTIONAL { ?db_uri db:drugType
?drugType_uri .
     ?drugType_uri rdfs:label ?drugType . }
  } }
}
```

**Q19: Target information—graph optional**

```
SELECT ?target_name ?alt_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
```

```
?numberOfResidues ?theoreticalPi
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://data.kasabi.com/
   dataset/chembl-rdf>
FROM NAMED <http://linkedlifedata.com/
   resource/drugbank>
WHERE {
  GRAPH <http://www.conceptwiki.org> {
    [RESOURCE] skos:prefLabel ?target_name.
  }
  OPTIONAL {
    GRAPH <http://data.kasabi.com/
      dataset/chembl-rdf {
    [RESOURCE] skos:exactMatch ?chembl_uri .
    ?chembl_uri owl:equivalentClass
?equiv_target .
    ?chembl_uri rdfs:subClassOf ?target_type .
    ?chembl_uri rdfs:label ?synonym .
    OPTIONAL { ?chembl_uri chembl:hasKeyword
                      ?keyword . }
    OPTIONAL { ?chembl_uri
chembl:hasDescription
                       ?description . }
  } }
  OPTIONAL {
    GRAPH <http://linkedlifedata.com/
      resource/drugbank> {
    [RESOURCE] skos:exactMatch ?db_uri .
    OPTIONAL{?db_uri db:cellularLocation
                      ?cellularLocation.}
    OPTIONAL { ?db_uri db:molecularWeight
                      ?molecularWeight.}
    OPTIONAL{?db_uri db:numberOfResidues
                      ?numberOfResidues.}
    OPTIONAL { ?db_uri db:theoreticalPi
                      ?theoreticalPi . }
  } }
}
```

**Q20: Target pharmacology—graph optional**

```
SELECT ?target_name ?assay_uri ?chembl_uri
?equiv_assay ?equiv_target ?activity_uri
?compound_chembl ?equiv_compound ?compound_cs
?target_organism ?assay_organism ?assay_desc
?std_type ?relation ?std_value ?std_unit
?doi_int ?doi ?pmid ?node1 ?molweight ?inchi
?inchi_key ?smiles ?node2 ?num_ro5_violations
?compound_cw ?compound_name
FROM NAMED <http://www.conceptwiki.org>
FROM NAMED <http://data.kasabi.com/
  dataset/chembl-rdf>
FROM NAMED <http://www.chemspider.com>
WHERE {
  GRAPH <http://www.conceptwiki.org>
    [RESOURCE] skos:prefLabel ?target_name .
    ?compound_cw skos:exactMatch ?compound_cs .
    ?compound_cw skos:prefLabel
?compound_name .
  }
  GRAPH <http://data.kasabi.com/
    dataset/chembl-rdf>
    [RESOURCE] skos:exactMatch ?chembl_uri .
    ?assay_uri chembl:hasTarget ?chembl_uri .
    ?assay_uri owl:equivalentClass
?equiv_assay .
    ?chembl_uri owl:equivalentClass
?equiv_target .
    ?activity_uri chembl:onAssay ?assay_uri .
    ?activity_uri chembl:forMolecule
      ?compound_chembl .
    ?compound_chembl owl:equivalentClass
      ?equiv_compound .
    ?equiv_compound skos:exactMatch
?compound_cs .
    OPTIONAL { ?chembl_uri chembl:organism
      ?target_organism. }
    OPTIONAL { ?assay_uri chembl:organism
      ?assay_organism. }
    OPTIONAL{ ?assay_uri chembl:hasDescription
      ?assay_desc. }
    OPTIONAL { ?activity_uri chembl:type
      ?std_type . }
    OPTIONAL { ?activity_uri chembl:relation
      ?relation . }
    OPTIONAL {?activity_uri
chembl:standardValue
      ?std_value.}
    OPTIONAL { ?activity_uri
chembl:standardUnits
      ?std_unit.}
    OPTIONAL { ?activity_uri
cito:citesAsDataSource
      ?doi_int.
    OPTIONAL { ?doi_int owl:sameAs ?doi . }
    OPTIONAL { ?doi_int bibo:pmid ?pmid . } }
    OPTIONAL { ?compound_chembl
ss:CHEMINF_000200
    ?node1 .
    ?node1 rdf:type ss:CHEMINF_000198 .
    ?node1 ss:SIO_000300 ?molweight . }
  }
  GRAPH <http://www.chemspider.com>
    ?compound_cs cs:inchi ?inchi .
    ?compound_cs cs:inchikey ?inchi_key .
    ?compound_cs cs:smiles ?smiles .
    OPTIONAL { ?node2 obo:IAO_0000136
?compound_cs .
    ?node2 rdf:type cheminf:CHEMINF_000367 .
    ?node2 numericValue ?num_ro5_violations . }
  }
}
```

**Q21: Redundant intermediate variables**

```
SELECT ?synonym ?cellularLocation {
    [RESOURCE] skos:exactMatch ?chembl_uri .
    ?chembl_uri rdfs:label ?synonym.
    OPTIONAL { [RESOURCE] skos:exactMatch
?db_uri .
    ?db_uri db:cellularLocation
?cellularLocation .}
}
```

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

**Q22: Sequence path query**

```
SELECT ?synonym ?cellularLocation WHERE {
  [RESOURCE] skos:exactMatch/rdfs:label
?synonym .
  OPTIONAL { [RESOURCE] skos:exactMatch/db:
    cellularLocation
                    ?cellularLocation }
}
```

**Q23–Q25: Alternative URIs:** UNION

```
SELECT ?target_name ?alt_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
?numberOfResidues ?theoreticalPi {
   {
   [RESOURCE]1skos:prefLabel ?target_name .
   [RESOURCE]1skos:exactMatch ?chembl_uri .
   ?chembl_uri owl:equivalentClass
?equiv_target .
   ?chembl_uri chembl:hasKeyword ?keyword .
   ?chembl_uri chembl:hasDescription
?description .
   ?chembl_uri rdfs:subClassOf ?target_type .
   ?chembl_uri rdfs:label ?synonym .
   [RESOURCE]1skos:exactMatch ?db_uri .
   ?db_uri db:cellularLocation
?cellularLocation .
   ?db_uri db:molecularWeight
?molecularWeight .
   ?db_uri db:numberOfResidues
?numberOfResidues .
   ?db_uri db:theoreticalPi ?theoreticalPi .
   } UNION {
              ...
   } UNION {
   [RESOURCE]nskos:prefLabel ?target_name.
   [RESOURCE]nskos:exactMatch ?chembl_uri .
   ?chembl_uri owl:equivalentClass
?equiv_target .
   ?chembl_uri chembl:hasKeyword ?keyword .
   ?chembl_uri chembl:hasDescription
   ?description .
   ?chembl_uri rdfs:subClassOf ?target_type .
   ?chembl_uri rdfs:label ?synonym .
   [RESOURCE]nskos:exactMatch ?db_uri .
   ?db_uri db:cellularLocation
?cellularLocation .
   ?db_uri db:molecularWeight
?molecularWeight .
   ?db_uri db:numberOfResidues
?numberOfResidues .
   ?db_uri db:theoreticalPi ?theoreticalPi .
   }
}
```

**Q26–Q28: Alternative URIs:** FILTER

```
SELECT ?target_name ?alt_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
?numberOfResidues ?theoreticalPi WHERE {
  ?s skos:prefLabel ?target_name.
  ?s skos:exactMatch ?chembl_uri .
  ?chembl_uri owl:equivalentClass
?equiv_target .
  ?chembl_uri chembl:hasKeyword ?keyword .
  ?chembl_uri chembl:hasDescription
?description .
  ?chembl_uri rdfs:subClassOf ?target_type .
  ?chembl_uri rdfs:label ?synonym .
  ?s skos:exactMatch ?db_uri .
  ?db_uri db:cellularLocation
?cellularLocation .
  ?db_uri db:molecularWeight
?molecularWeight .
  ?db_uri db:numberOfResidues
?numberOfResidues .
  ?db_uri db:theoreticalPi ?theoreticalPi .
  FILTER(?s = [RESOURCE]1 || ··· || ?s = [RESOURCE]n)
}
```

**Q29–Q31: Alternative URIs:** VALUES

```
SELECT ?s ?target_name ?equiv_target
?target_type ?description ?keyword ?synonym
?db_uri ?cellularLocation ?molecularWeight
?numberOfResidues ?theoreticalPi WHERE {
VALUES ?s {[RESOURCE]1 ··· [RESOURCE]n}
  ?s skos:prefLabel ?target_name.
  ?s skos:exactMatch ?chembl_uri .
  ?chembl_uri owl:equivalentClass
  ?equiv_target .
  ?chembl_uri chembl:hasKeyword ?keyword .
  ?chembl_uri chembl:hasDescription
?description .
  ?chembl_uri rdfs:subClassOf ?target_type .
  ?chembl_uri rdfs:label ?synonym .
  ?s skos:exactMatch ?db_uri .
  ?db_uri db:cellularLocation
?cellularLocation .
  ?db_uri db:molecularWeight
?molecularWeight .
  ?db_uri db:numberOfResidues
?numberOfResidues .
  ?db_uri db:theoreticalPi ?theoreticalPi .
}
```

## Appendix B. Figures

This appendix provides figures visualising the performance differences between the queries generated based on the same view template. The average execution time for each query is indicated by the × symbol while minimum and maximum execution times are given by the error bars.

**Compound information: Q1, Q5, Q9, Q13 and Q17** (see Fig. B.1)

**Compound pharmacology: Q2, Q6, Q10, Q14 and Q18** (see Fig. B.2)

**Target information: Q3, Q7, Q11, Q15 and Q19** (see Fig. B.3)

**Target pharmacology: Q4, Q8, Q12, Q16 and Q20** (see Fig. B.4)

**Reduce intermediate results: Q21 and Q22** (see Fig. B.5)

**Specifying 10 alternative URIs: Q23, Q26 and Q29** (see Fig. B.6)

**Specifying 20 alternative URIs: Q24, Q27 and Q30** (see Fig. B.7)

**Specifying 50 alternative URIs: Q25, Q28 and Q31** (see Fig. B.8)

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

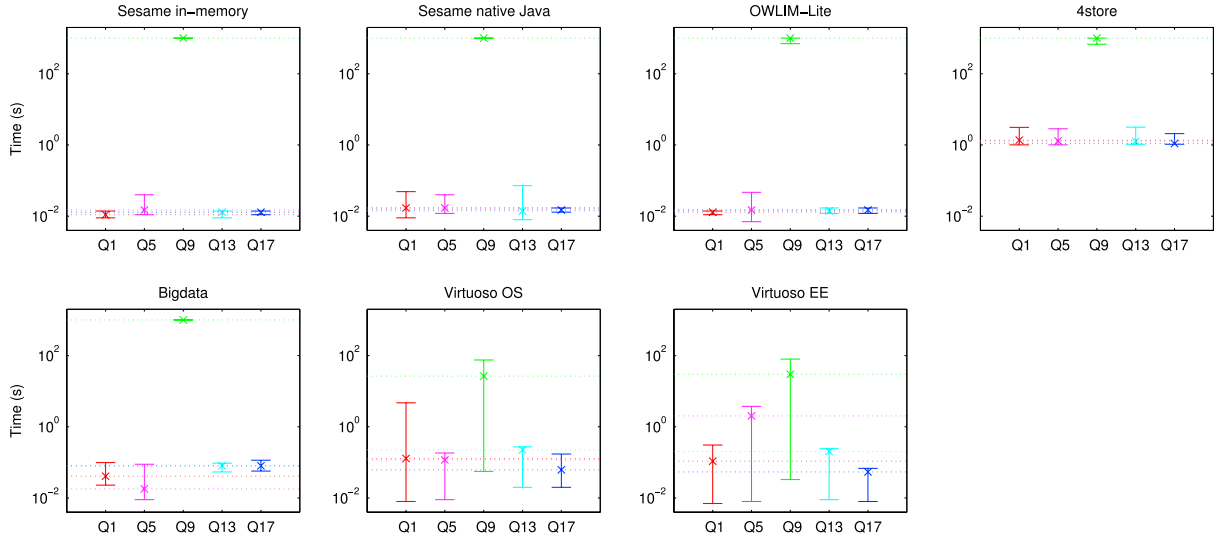


**Fig. B.1.** Comparison of response times for queries based on the 'Compound Information' view template.

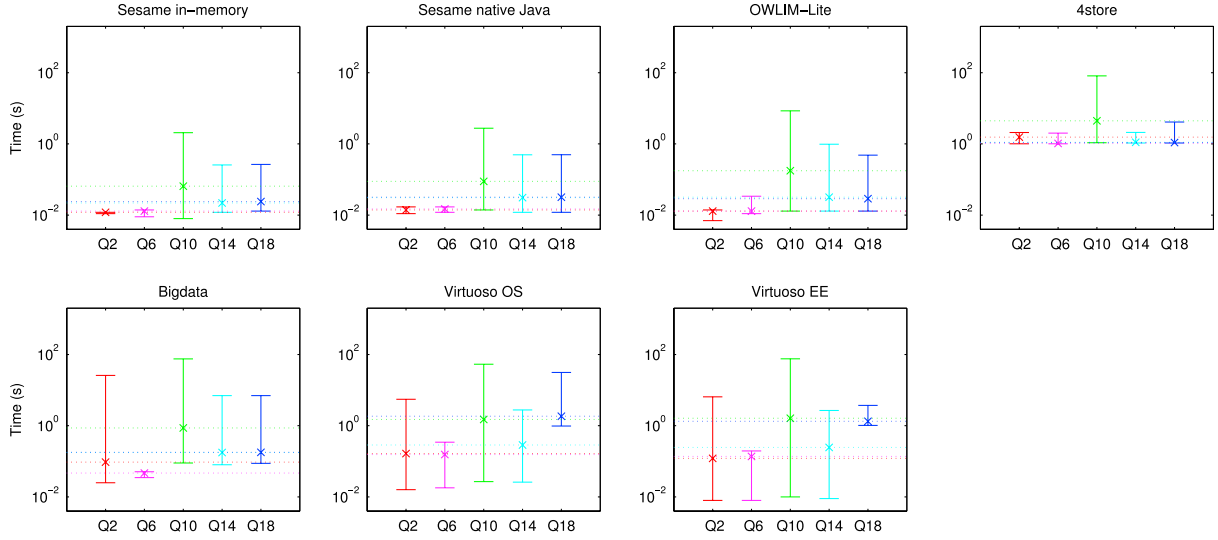


**Fig. B.2.** Comparison of response times for queries based on the 'Compound Pharmacology' view template.

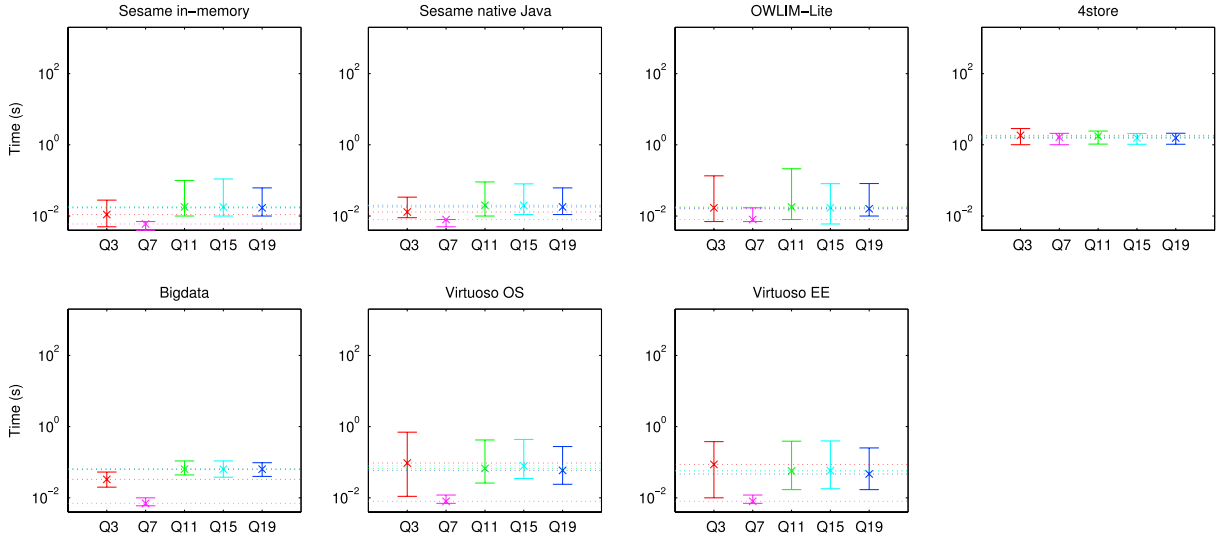


**Fig. B.3.** Comparison of response times for queries based on the 'Target Information' view template.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

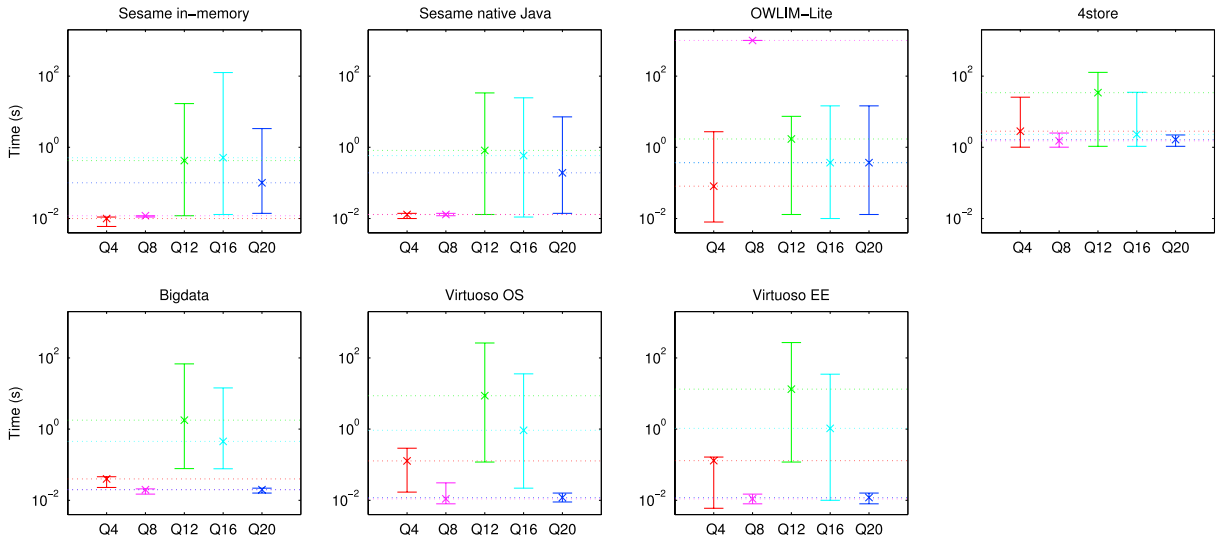


**Fig. B.4.** Comparison of response times for queries based on the 'Target Pharmacology' view template.

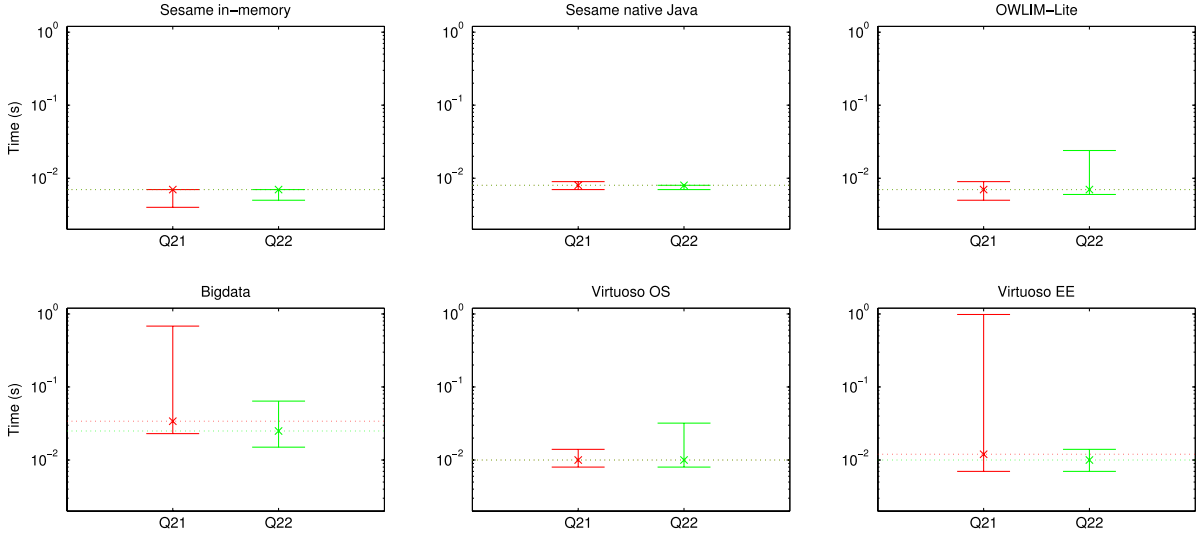


**Fig. B.5.** Comparison of response times for queries constructed to evaluate the 'Reduce intermediate results' heuristic.

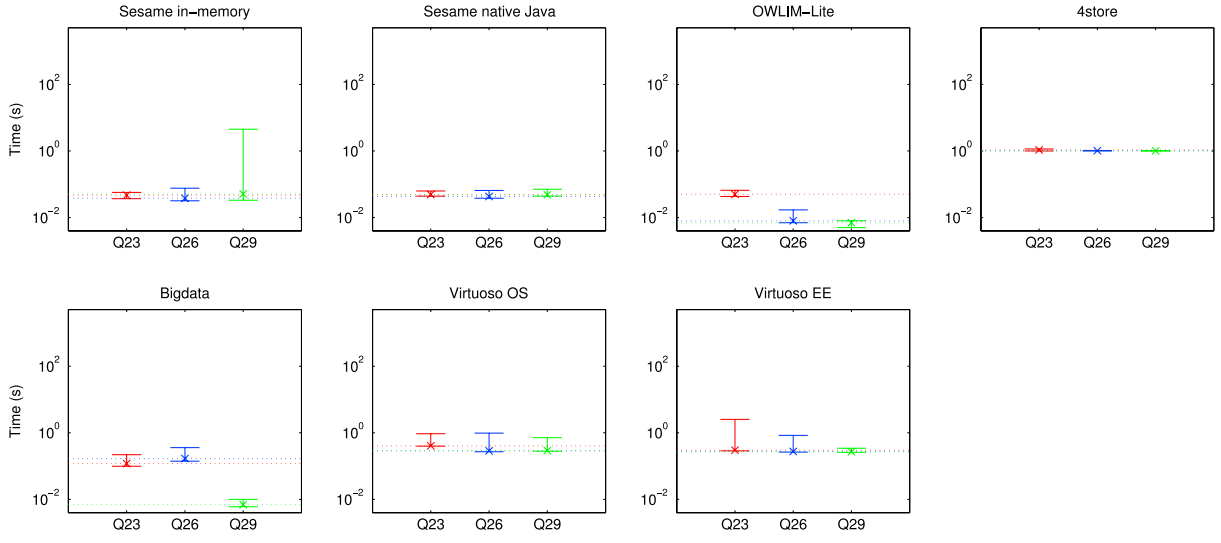


**Fig. B.6.** Comparison of response times for queries specifying 10 alternative URIs, based on the 'Target Information' view template. Q23 uses `UNION`, Q26 uses `FILTER`, while Q29 uses `VALUES`.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

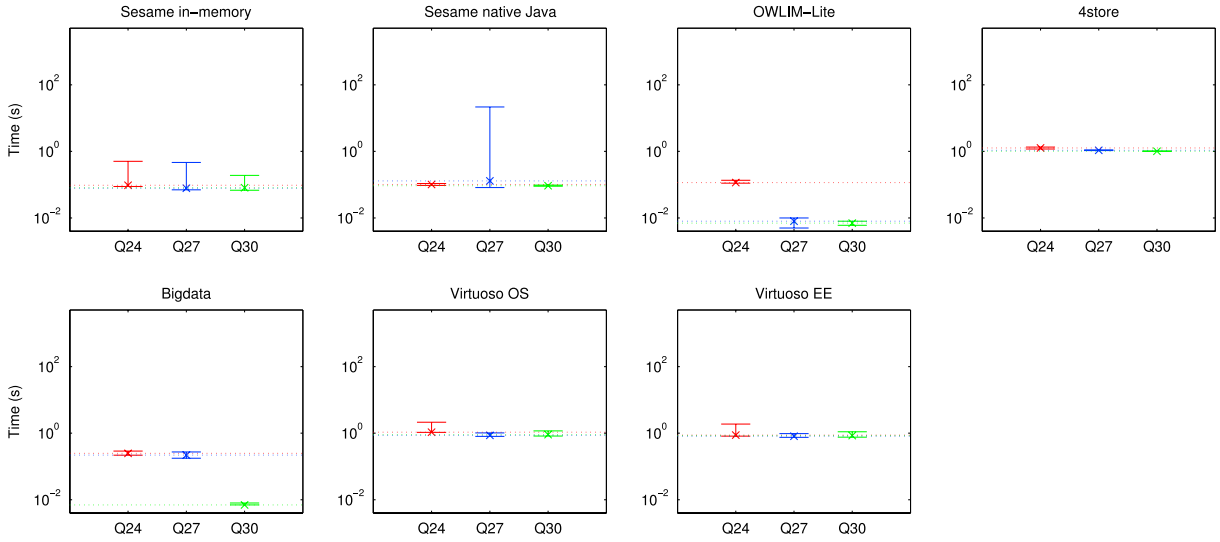


**Fig. B.7.** Comparison of response times for queries specifying 20 alternative URIs, based on the 'Target Information' view template. Q24 uses `UNION`, Q27 uses `FILTER`, while Q30 uses `VALUES`.

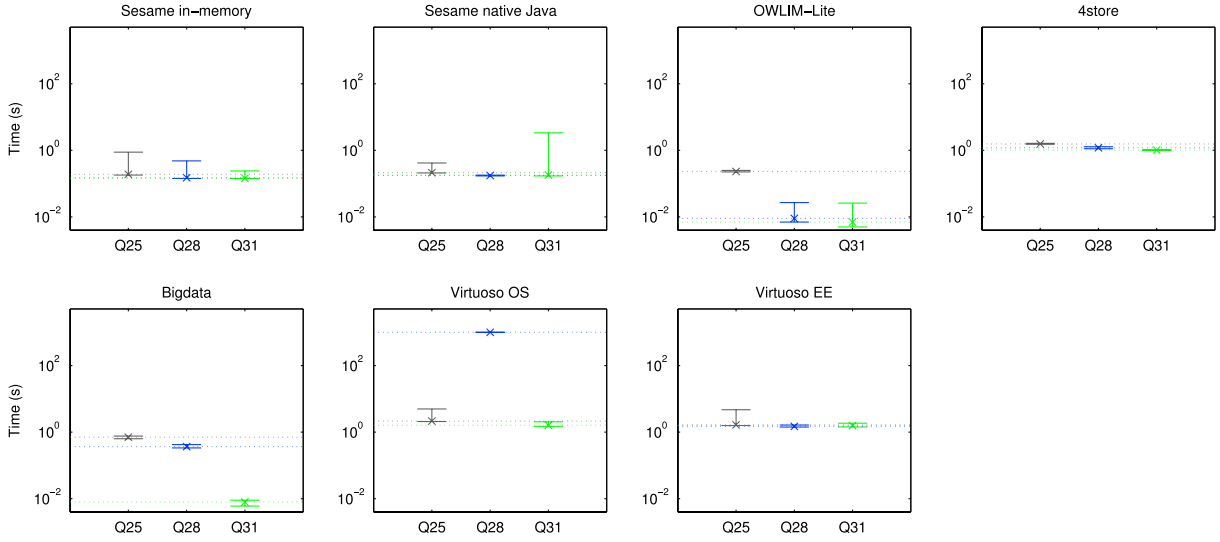


**Fig. B.8.** Comparison of response times for queries specifying 50 alternative URIs, based on the 'Target Information' view template. Q25 uses `UNION`, Q28 uses `FILTER`, while Q31 uses `VALUES`.

## References

[1] R. Swick, Resource description framework (RDF) model and syntax specification. W3C Recommendation, 1999. http://www.w3.org/TR/1999/REC-rdfsyntax-19990222.
[2] G. Klyne, J. Carroll, B. McBride, Resource description framework (RDF): concepts and abstract syntax, W3C Recommendation.
[3] G. Tummarello, R. Delbru, E. Oren, R. Cyganiak, Sindice.com: a semantic web search engine, Presentation, Digital Enterprise Research Institute National University of Ireland, Galway, November 2007.
[4] S. Campinas, T. Perry, D. Ceccarelli, R. Delbru, G. Tummarello, Introducing RDF graph summary with application to assisted SPARQL formulation, in: Proceedings of the 11th International Workshop on Web Semantics and Information Processing, 2012.
[5] E. Prud'hommeaux, A. Seaborne, SPARQL query language for RDF. W3C Recommendation, 2008. http://www.w3.org/TR/rdf-sparql-query/.
[6] S. Harris, A. Seaborne, SPARQL 1.1 query language for RDF. W3C Recommendation, 2013. http://www.w3.org/TR/sparql11-query/.
[7] L. Rietveld, R. Hoekstra, S. Schlobach, Feeling the pulse of linked data, in: ISWC 2014, under submission, 2014.
[8] M. Pham, Self-organizing structured RDF in MonetDB, in: Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on, IEEE, 2013, pp. 310–313.
[9] A.J.G. Gray, P. Groth, A. Loizou, S. Askjaer, C. Brenninkmeijer, K. Burger, C. Chichester, C.T. Evelo, C. Goble, L. Harland, S. Pettifer, M. Thompson, A. Waagmeester, A. Williams, Applying linked data approaches to pharmacology: Architectural decisions and implementation, Semant. Web 5 (2) (2014) 101–113.
[10] A.J. Williams, L. Harland, P. Groth, S. Pettifer, C. Chichester, E.L. Willighagen, C.T. Evelo, N. Blomberg, G. Ecker, C. Goble, B. Mons, Open PHACTS: semantic interoperability for drug discovery, Drug Discov. Today 17 (21–22) (2012) 1188–1198.
[11] E. Jacoby, K. Azzaoui, S. Senger, E.C. Rodríguez, M. Loza, B. Zdrazil, M. Pinto, A.J. Williams, V. de la Torre, J. Mestres, O. Taboureau, M. Rarey, G.F. Ecker, Scientific requirements for the next generation semantic web-based chemogenomics and systems chemical biology molecular information system OPS, in: Computational Chemogenomics, Wiley, 2012, pp. 213–242. chapter 8.
[12] P. Groth, A. Loizou, A.J.G. Gray, C. Goble, L. Harland, S. Pettifer, API-centric linked data integration: the open PHACTS discovery platform case study, Web Semant. Sci. Serv. Agents World Wide Web (2014).
[13] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL, ACM Trans. Database Syst. (TODS) 34 (3) (2009) 16.
[14] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL, in: The Semantic Web-ISWC 2006, 2006, pp. 30–43.
[15] C. Gutierrez, C. Hurtado, A. Mendelzon, J. Pérez, Foundations of semantic web databases, J. Comput. System Sci. 77 (3) (2011) 520–541.
[16] A. Letelier, J. Pérez, R. Pichler, S. Skritek, Static analysis and optimization of semantic web queries, in: Proceedings of the 31st Symposium on Principles of Database Systems, ACM, 2012, pp. 89–100.
[17] M. Schmidt, M. Meier, G. Lausen, Foundations of SPARQL query optimization, in: Proceedings of the 13th International Conference on Database Theory, ACM, 2010.
[18] M. Schmidt, T. Hornung, N. Küchlin, G. Lausen, C. Pinkel, An experimental comparison of RDF data management approaches in a SPARQL benchmark scenario, in: The Semantic Web-ISWC 2008, 2008, pp. 82–97.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003

[19] M. Schmidt, T. Hornung, G. Lausen, C. Pinkel, SP^ 2Bench: a SPARQL performance benchmark, in: Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on, IEEE, 2009, pp. 222–233.
[20] N. Stojanovic, On the query refinement in the ontology-based searching for information, Inf. Syst. 30 (7) (2005) 543–563.
[21] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, D. Reynolds, SPARQL basic graph pattern optimization using selectivity estimation, in: Proceedings of the 17th International Conference on World Wide Web, ACM, 2008, pp. 595–604.
[22] M. Arenas, C. Gutierrez, J. Pérez, On the Semantics of SPARQL, Springer, 2010.
[23] Y. Luo, F. Picalausa, G.H.L. Fletcher, J. Hidders, S. Vansummeren, Storing and indexing massive RDF datasets semantic search over the Web, in: Semantic Search Over the Web, Springer, Berlin, Heidelberg, 2012, pp. 31–60. (Data-centric systems and applications, Chapter 2).
[24] M. Arenas, S. Conca, J. Pérez, Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard, in: Proceedings of the 21st International Conference on World Wide Web, ACM, 2012, pp. 629–638.
[25] A. Loizou, P. Groth, R. Angles, On the formulation of performant SPARQL queries: evaluation datasets. URL: http://dx.doi.org/10.6084/m9.figshare.936655.
[26] A. Loizou, P. Groth, R. Angles, On the formulation of performant SPARQL queries: sample resources used in evaluation. URL: http://dx.doi.org/10.6084/m9.figshare.936890.
[27] A. Loizou, P. Groth, R. Angles, On the formulation of performant SPARQL queries: evaluation queries. URL: http://dx.doi.org/10.6084/m9.figshare.936610.
[28] A. Loizou, P. Groth, R. Angles, On the formulation of performant SPARQL queries: evaluation scripts. URL: http://dx.doi.org/10.6084/m9.figshare.936887.
[29] A. Loizou, P. Groth, R. Angles, On the formulation of performant SPARQL queries: evaluation results. URL: http://dx.doi.org/10.6084/m9.figshare.936689.
[30] Y. Guo, Z. Pan, J. Heflin, LUBM: a benchmark for OWL knowledge base systems, Web Semant. Sci. Serv. Agents World Wide Web 3 (2) (2005) 158–182.
[31] C. Bizer, A. Schultz, The berlin SPARQL benchmark, Int. J. Sem. Web Inf. Syst. (IJSWIS) 5 (2) (2009) 1–24.
[32] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, Dbpedia: A Nucleus for a Web of Open Data, Springer, 2007.
[33] Y.E. Ioannidis, Query optimization, ACM Comput. Surv. 28 (1) (1996) 121–123.
[34] C. Gutierrez, C. Hurtado, A.O. Mendelzon, Foundations of semantic web databases, in: Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS, ACM, New York, NY, USA, 2004, pp. 95–106.
[35] O. Hartig, R. Heese, The SPARQL query graph model for query optimization, in: Proceedings of the 4th European Semantic Web Conference, ESWC, Springer-Verlag, 2007.
[36] Z. Kaoudi, K. Kyzirakos, M. Koubarakis, SPARQL query optimization on top of DHTs, in: Proc. of the International Semantic Web Conference, ISWC, 2010.
[37] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, FedX: optimization techniques for federated query processing on linked data, in: Proc. of the International Semantic Web Conference, ISWC, 2011.
[38] F. Picalausa, S. Vansummeren, What are real SPARQL queries like?, in: Proceedings of the International Workshop on Semantic Web Information Management, SIWM, 2011.
[39] P. Tsialiamanis, L. Sidirourgos, I. Fundulaki, V. Christophides, P. Boncz, Heuristics-based query optimisation for SPARQL, in: Proceedings of the 15th International Conference on Extending Database Technology, EDBT, ACM, 2012.

Please cite this article in press as: A. Loizou, et al., On the formulation of performant SPARQL queries, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.003