Enabling Live SPARQL Queries Over ConceptNet Using Triple Pattern Fragments

Marcelo Machado[1], Guilherme Lima[1], Elton Soares[1], Rosario Uceda-Sosa[2], and Renato Cerqueira[1]

[1] IBM Research Brazil, Rio de Janeiro, Brazil
{mmachado,guilherme.lima,eltons}@ibm.com, rcerq@br.ibm.com
[2] IBM TJ Watson Research Center, Yorktown Heights, NY, USA
rosariou@us.ibm.com

Abstract. We describe how we used a Triple Pattern Fragments (TPF) interface and the Comunica knowledge graph querying framework to enable live SPARQL queries over ConceptNet, one of largest knowledge graphs for commonsense reasoning publicly available on the Web. Despite being a Linked Data resource, the official ConceptNet is not published in RDF and does not support SPARQL. Instead, it provides a RESTful API for live queries, which are restricted to simple triple patterns. This limited API makes it hard for users to search for non-trivial patterns in the graph and hinders the possibility of federated queries offered by SPARQL. There have been attempts to convert ConceptNet to RDF but such proposals tend to quickly become obsolete. In this paper, we take a different route. We use TPF to expose a low-level RDF query interface to ConceptNet. This low-level interface is built on top of the ConceptNet API and can be used by TPF-compatible SPARQL engines such as Comunica. Using this approach, we were able evaluate non-trivial SPARQL queries, including federated queries, over ConceptNet on-the-fly. Our experiments showed that overhead incurred is small and can be further reduced by optimizing ConceptNet's internal edge representation. We argue that such overhead is justified by the gains in expressivity and flexibility. Moreover, the overall approach is general and can be extended to other non-RDF knowledge graphs.

Keywords: ConceptNet · RDF · Linked Data Fragments · Triple Pattern Fragments · Comunica · SPARQL

1 Introduction

ConceptNet [17] is a large public knowledge graph describing commonsense knowledge and its expression in various natural languages. It is a valuable resource for natural language processing applications in general, such as those based on word embeddings [18], and in particular for applications that seek to emulate the kinds of commonsense reasoning performed by humans. These applications include question-answering [9,2], sentiment analysis [21], reading comprehension [1], image understanding [27], etc.

To access ConceptNet, users and applications can either use its live query interface or download one of its data dumps. The latter is the best approach if one is interested in processing the graph offline. However, for applications that want to query ConceptNet without ingesting the whole graph, the simplest approach is to use its live query interface (the ConceptNet API). This interface consists of a RESTful API [6] which accepts queries restricted to a single triple pattern, i.e., a combination of subject, predicate, and object. The triple pattern is matched against the components of edges in the graph and the results are returned in JSON-LD [19] format.

In this paper, we are concerned with overcoming what we think are the two main limitations of ConceptNet's live query interface: (i) its low expressivity and (ii) its lack of support for RDF [4] and SPARQL [26]. The first limitation makes it hard for users and applications to search for non-trivial patterns in the graph, while the second complicates the integration of ConceptNet into the Semantic Web ecosystem. The lack of support for SPARQL, in particular, hinders the possibility of more expressive queries and also of federated queries [15], which would allow users to match external references in ConceptNet against resources in DBpedia [8], Wikidata [25], WordNet [11], etc.

We remark that the absence of RDF support in ConceptNet is by design. It is a consequence of the choice of a relational database (PostgreSQL[3]) as its storage system. Although there have been proposals for converting ConceptNet to RDF [13,3], adopting one of these would require significant changes to its code base, including switching to a different storage system. That said, since ConceptNet 5.5.0, released in 2016, the lack of support for RDF is no longer a big issue. Version 5.5.0 changed the format of query responses to JSON-LD which can be easily converted to RDF on the client side [10].

The lack of support for SPARQL, however, is not so simple to overcome. Here is what the FAQ section of ConceptNet's documentation says about this:[4]

> "*Can ConceptNet be queried using SPARQL?* No. SPARQL is computationally infeasible. Similar projects that use SPARQL have unacceptable latency and go down whenever anyone starts using them in earnest."

Indeed, SPARQL is known to be computationally intractable (query evaluation is PSPACE-complete [14]) but so is SQL (query evaluation in the relational calculus is also PSPACE-complete [22]). The problem is not with SPARQL per se but with exposing on the public Web what essentially is the query interface of the underlying database. Interestingly, the compromise reached by the designers of ConceptNet of exposing only a limited, triple pattern-based query interface is precisely the compromise advocated by the proponents of the Triple Pattern Fragments interface [24]:

> "Between the two extremes of data dumps and SPARQL endpoints lies a whole spectrum of possible (unexplored) Web interfaces. [...] Offering

[3] https://www.postgresql.org/
[4] https://github.com/commonsense/conceptnet5/wiki/FAQ

> *triple-pattern*-based access to RDF knowledge graphs seems an interesting compromise because (i) triple patterns are the most basic building block of SPARQL queries, and (ii) servers can select triples that match a given pattern at low processing cost."

As suggested above, Triple Pattern Fragments (TPF) are a low-cost interface to RDF triples. They were introduced in the context of the Linked Data Fragments [23] framework with the goal of enabling the construction of reliable applications over public knowledge graphs. One crucial advantage of providing a TPF interface instead of a custom query interface, like ConceptNet API, is that the TPF results can be consumed by any TPF-compliant client, and this includes client-side SPARQL engines, like Comunica [20] and TPF client [24]. In this sense, TPF can be seen as a means of obtaining support for SPARQL.

This combination of TPF with a client-side SPARQL engine is exactly the approach we propose here for enabling live SPARQL queries over ConceptNet. To evaluate this proposal, we extended the TPF server Server.js[5] with a new datasource plugin[6] (released under the open-source MIT license) which allows it to communicate with the ConceptNet API. We then used the Comunica SPARQL engine [20] to create a SPARQL endpoint pointing to our modified TPF server. This setup is illustrated in Figure 1.

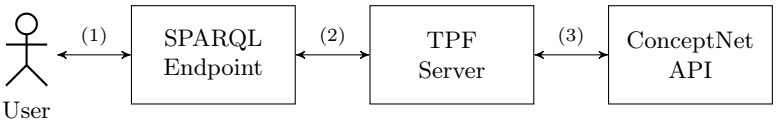

Fig. 1. The proposed approach for enabling SPARQL over ConceptNet.

We tested the viability of this setup through experiments that measured the overhead introduced at (1) and (2) in comparison to (3). The results indicate that the overhead is small and can be further reduced by simplifying the way edges are stored internally by ConceptNet. We argue that these overheads are justified by the gains in expressivity and flexibility obtained. Queries submitted at (1), for example, can contain any SPARQL feature supported by the query engine which, in the case of Comunica, includes property paths, filters, federation, etc. Also, the overall approach is general and can be extended to other non-RDF knowledge graphs.

The rest of the paper is organized as follows. Section 2 presents some background on the ConceptNet API, TPF, and Comunica. Section 3 presents our proposal and implementation. Section 4 presents the experimental evaluation and discusses its results. Section 5 discusses some related work. Finally, Section 6 presents our conclusions and future work.

[5] https://github.com/LinkedDataFragments/Server.js
[6] https://github.com/IBM/tpf-conceptnet-datasource

2 Background

2.1 ConceptNet API

ConceptNet [17] is a knowledge graph for commonsense reasoning that connects words and phrases in various languages using relations like "is a", "used for", "part of", etc. It originated from the MIT Media Lab's Open Mind Common Sense (OMCS) project [16] and has since been expanded with facts from many other resources, including Wikitionary[7], WordNet [11], DBpedia [8], etc.

The easiest way to query ConceptNet is through its public RESTful API. The latest version of the API (5.8.1) accepts simple requests like:

```
https://api.conceptnet.io/query?start=/c/en/cat&end=/c/en/milk
```

This particular request asks for all edges in the graph whose start node is "cat" (`/c/en/cat`) and end node is "milk" (`/c/en/milk`). Its response is a JSON-LD document which, among other things, contains this edge:

```
{ "@id":   "/a/[/r/Desires/,/c/en/cat/,/c/en/milk/]",
  "start": { "@id": "/c/en/cat",  ... },
  "rel":   { "@id": "/r/Desires", ... },
  "end":   { "@id": "/c/en/milk", ... }, ... }
```

This edge asserts that "cat" is related via "desires" to "milk". As illustrated here, every ConceptNet edge is directed and consists of a start node (`start`), a relation (`rel`), and an end node (`end`). The edge also has an id (derived from `start`, `rel`, and `end`) and can contain extra information like its weight (reliability measure), provenance (list of sources), etc.

The ConceptNet API supports essentially two kinds of queries:

1. *Start-rel-end queries.* These are queries in which any of the parameters `start`, `rel`, and `end` are given. The previous query is an example of such a query. It sets the parameter `start` to "cat" and `end` to "milk", which instructs the API to search for any edges leaving "cat" and reaching "milk".
2. *Node-other queries.* These are queries in which any of the parameters `node` or `other` are given. If only `node` or only `other` is given, the API searches for edges where the start or end of the edge matches the parameter. For example, the query `?node=/c/en/cat` asks for edges where the start or end node is "cat". If both `node` and `other` are given, the API searches for edges where the start and end match the parameters regardless of their order. For example, the query `?node=/c/en/cat&other=/c/en/animal` searches for any edges connecting "cat" and "animal" in either direction.

Start-rel-end queries can be seen as equivalent to SPARQL queries that contain a single triple-pattern, while node-other queries correspond to disjunctive SPARQL queries (i.e., queries that use the UNION operator to test for a match in either of the directions).

[7] https://en.wiktionary.org/

By default, the ConceptNet API returns at most 50 edges per request. This number can be increased up to 1000 using the `limit` parameter. When the number of edges exceeds the limit, the extra results are put in separate pages which can be accessed either by setting the `offset` parameter in the request or by using the "next page" link returned at the end the JSON-LD response.

2.2 Triple Pattern Fragments

Triple Pattern Fragments (TPF) [24] are a lightweight interface to RDF graphs. They are part of the Linked Data Fragments (LDF) [23] initiative and were proposed as an intermediate alternative to RDF data dumps and SPARQL endpoints. LDF itself is a framework for the conceptual analysis of Linked Data interfaces. According to LDF, any such interface publishes only parts, or *(linked data) fragments*, of a given knowledge graph. These fragments are considered the "units of response" of the interface and are assumed to consist of three things:

1. *Data*: a subset of the triples of the graph;
2. *Metadata*: triples describing the data; and
3. *Controls*: links and forms that can be used to retrieve other fragments of the same or other knowledge graphs.

For example, a data dump of a knowledge graph can be described as single linked data fragment where the data is the whole content of the dump, the metadata consists of things like version, author, etc., and the controls are empty. Similarly, the response to a SPARQL CONSTRUCT query can be seen as a fragment where the data are the resulting triples, the metadata is empty, and the controls are any parameters used to paginate the result, such as limits and offsets.

A TPF interface provides access to an RDF graph based on a single triple-pattern. A *triple-pattern* is a pattern of the form (s, p, o) where s, p, and o are either fixed values (URIs or literals) or anonymous variables. When given a triple-pattern, the TPF interface responds with a fragment in which (i) the data are triples in the graph that match the pattern; (ii) the metadata are an estimate of the total number such triples; and (iii) the controls are a hypermedia form that allows clients to retrieve other fragments matching the same pattern.

Some popular knowledge graphs, like DBpedia [8] and Wikidata [25], already provide TPF interfaces. (Other public TPF interfaces can be found here[8].) Take the Wikidata TPF interface, for example. We can query it using a request like:

```
https://query.wikidata.org/bigdata/ldf?predicate=wdt:P31&object=wd:Q5
```

This request asks for every triple in Wikidata whose predicate component is `wdt:P31` (instance of) and object component is `wd:Q5` (human). That is, it selects the triples matching the pattern "$(s, \texttt{wdt:P31}, \texttt{wd:Q5})$" for any value of s. In other words, the triples such that subject s is an instance of human. The response to this specific request is a single linked data fragment containing usually 100

[8] https://linkeddatafragments.org/data/

triples matching the pattern. For instance, if you open the above URI in a Web browser, an HTML page with the matched triples will be shown. At the bottom of the page, you will see links (controls) to other pages (fragments) which contain the rest of the result. The actual format of the response depends on the value of the "Accept" header provided to the server in the HTTP request. This can be set to any of the popular RDF serialization formats, such as Turtle and JSON-LD.

Some TPF interfaces also support quad-patterns of the form (s, p, o, g) which take an additional parameter g specifying a named graph of an RDF dataset. If not set, g is assumed to be the default graph.

The advantage of TPF over other Linked Data interfaces is that it offers some query capability while being extremely lightweight. It is more convenient than a data dump and, it can be argued, it provides a more reasonable Web API than SPARQL in the sense that it exposes only a limited interface to the underlying database. A further advantage has to do with caching. Because of their restricted syntax TPF requests are more cache-friendly than SPARQL requests. For instance, just by looking at the requested URI, it is straightforward for an HTTP proxy to determine whether two TPF requests are identical. The same cannot be said about SPARQL where there are many different ways to write essentially the same query.

2.3 Comunica

Comunica [20] is an advanced knowledge graph querying framework written in JavaScript and released under the open-source MIT license.[9] It is not tied to any particular storage system and can even run in the browser. Rather than functioning as a query engine itself, Comunica is a meta query engine that allows the creation of query engines by providing a set of modules that can be wired together in a flexible manner. The biggest differentiator of Comunica, however, is its support for federated queries over heterogeneous interfaces in which one can evaluate a federated SPARQL query over multiple interfaces, including TPF interfaces, SPARQL endpoints, and data dumps.

Comunica has full support for SPARQL 1.0 and implements a large subset of SPARQL 1.1[10]. It also has numerous other features, including support for other query languages (i.e., GraphQL), reasoning, etc. In this paper, however, we use Comunica mainly to evaluate simple (but non-trivial) SPARQL queries over our custom TPF interface which acts as a proxy to the ConceptNet API.

3 Proposal and Implementation

Our proposal to enable SPARQL queries over ConceptNet was summarized in Figure 1 of Section 1. The idea is to build a TPF interface on top of the ConceptNet API. Then use a TPF-compatible SPARQL engine to create a SPARQL endpoint pointing to the TPF interface. A more detailed view of this proposal is given in Figure 2.

[9] https://github.com/comunica/comunica
[10] https://comunica.dev/docs/query/advanced/specifications/

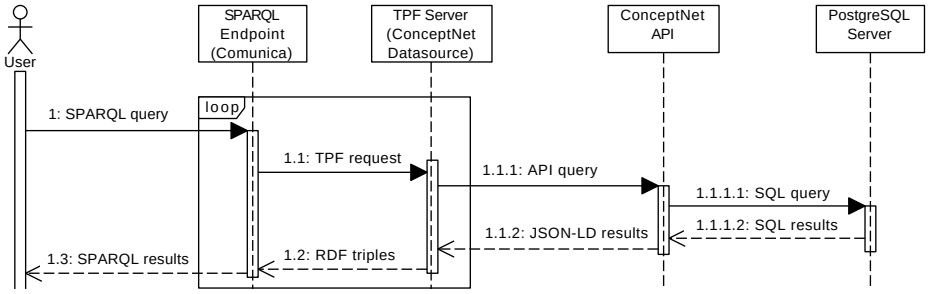

Fig. 2. The proposed approach for enabling SPARQL over ConceptNet. (Detailed)

To evaluate a SPARQL query over ConceptNet, the user sends a request to the SPARQL endpoint (1). This triggers one or more (triple-pattern) requests to our custom TPF server (1.1) each of which is translated into an equivalent (start-rel-end) query and sent to the ConceptNet API (1.1.1). As we mentioned previously, ConceptNet uses PostgreSQL as its storage system. So, each API query gives rise to one or more SQL queries (1.1.1.1) which are resolved by the PostgreSQL server. The other direction is similar: PostgreSQL's results are translated to JSON-LD (1.1.2), then to RDF triples (1.2), and finally to SPARQL results (1.3) which are delivered to the user.

The key element here is the TPF server. It needs to convert TPF queries, results, and controls into equivalent ConceptNet queries, results, and controls. In our case, we chose the TPF server Server.js.[11] We extended it with a new datasource that handles the input-output conversion and communication with the ConceptNet API. As the TPF-enabled SPARQL engine, we chose Comunica [20].

The rest of this section describes the challenges we had to overcome to implement the TPF protocol over the ConceptNet API. We describe two versions of this implementation. The first version, which we call *vanilla*, makes only minor changes to ConceptNet itself. The second version, which we call *simplified*, exposes the same API as the vanilla version but, in an attempt to speed up query evaluation, changes the way ConceptNet edges are represented in PostgreSQL.

3.1 The ConceptNet TPF Datasource

To create a TPF interface for ConceptNet, we extended the TPF server Server.js with a new datasource, called *ConceptNet Datasource* (MIT license).[12] In the Server.js architecture, the datasource is the component responsible for generating a stream of RDF triples from a given triple- or quad-pattern. The Server.js distribution comes with built-in datasources for generating triples from SPARQL endpoints and RDF files (including compressed HDT files [5]).

[11] https://github.com/LinkedDataFragments/Server.js
[12] https://github.com/IBM/tpf-conceptnet-datasource

Every Server.js datasource must implement the method `_executeQuery`(p, s) which takes the triple- or quad-pattern p, evaluates it over the underlying storage interface (SPARQL endpoint, RDF file, etc.), and writes the resulting triples asynchronously to the RDF stream s together with an approximate count of the total number of such triples. In the case of our ConceptNet Datasource, when `_executeQuery` is called, the datasource (i) converts the pattern p into a start-rel-end query; (ii) sends the start-rel-end query to the ConceptNet API; (iii) awaits for the API's JSON-LD response and when it arrives (iv) converts the edges in the response into triples; and finally (v) writes the resulting triples into the stream s. The datasource also writes into s's metadata the total number of triples that will be eventually produced by the query.

We had to solve two issues to implement the behavior we have just described. The first one was the conversion of ConceptNet edges into RDF triples. ConceptNet uses a heavily reified representation for edges which, besides the endpoints (`start` and `end`) and label (`rel`), contain information like weight, license, list of sources (provenance), etc. This means that the standard conversion of a JSON-LD edge to RDF produces a complex result, usually consisting of many triples. Because this would complicate the format of the SPARQL queries, and because in this paper we are mainly interested in the edge data (instead of its metadata), we decided to extract just the ids of the `start`, `rel`, and `end` parts of the edge and, when necessary, make them into valid URIs by prefixing the namespace "`http://conceptnet.io/`". So, for example, the JSON-LD edge listed at the beginning of Section 2.1, which connects "cat" and "milk" via "desires", is translated into the RDF triple:

```
<http://conceptnet.io/c/en/cat>
<http://conceptnet.io/r/Desires>
<http://conceptnet.io/c/en/milk> .
```

The other issue we had to deal with while implementing the ConceptNet Datasource was obtaining the edge count of ConceptNet API queries. This count is returned by the TPF server as metadata and is used by TPF clients, such as Comunica, to optimize their query plan. Currently, the only way to obtain this information is by going through all the pages of the response adding up their edge counts. This of course is impractical. What we did then was to extend the ConceptNet API with a new *count* call which takes a query and returns the total number of matched edges (computed by PostgreSQL). For instance, a call to this count endpoint with the query `start=/c/en/cat` will trigger a SQL query in PostgreSQL that uses the COUNT function to count rows from the table of edges where the column `start` contains the string identifier `/c/en/cat`. The syntax of the count call is the same as that of the regular query call. The only difference is that its target URI ends in `/query/count` instead of `/query`.

The configuration we have just described is what we call *vanilla*. It consists of the ConceptNet TPF Datasource running on top of the original ConceptNet API extended with the `/query/count` API call. The other configuration or version we consider is the *simplified* version, which we describe next.

3.2 The Simplified Version

In the simplified version, the ConceptNet TPF Datasource runs on top of an optimized version of the ConceptNet API.[13] The syntax of this optimized version is the same as that of the vanilla API, i.e., it still consists of `/query` and `/query/count` calls, but the calls themselves are implemented differently. They use an alternative database table (actually, a materialized view) to obtain edge information. Before detailing the contents of this alternative table and the advantages of using it, we need to describe the layout of the tables involved in the process of query evaluation in the original, vanilla ConceptNet API.

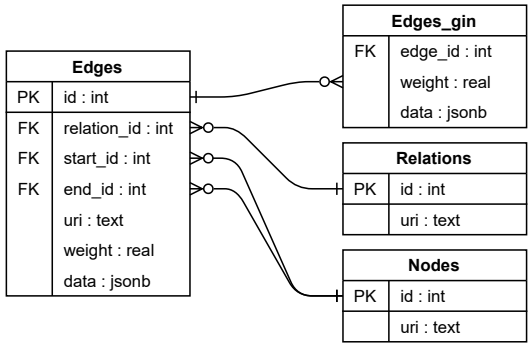

Fig. 3. Database tables involved in query evaluation in vanilla ConceptNet.

The entity-relationship diagram of the vanilla database tables is depicted in Figure 3. The tables *Edges* and *Edges_ gin* store edge data, *Relations* stores relation (predicate) data, and *Nodes* stores node data. The important thing to note is the "data" field present in *Edges* and *Edges_ gin*. This field stores a JSON object similar to the one presented in Section 2.1 and is the field used for matching the query parameters in the vanilla version of `/query` and `/query/count` API calls. More specifically, in the vanilla version, queries are evaluated by first joining *Edges* and *Edges_ gin* and then searching for the requested pattern within the JSON object stored in the data field. These JSON objects are indexed with GIN indices[14] which allows for efficient matching over composite objects using the containment operator "`@>`".

The reason for using this JSON-based matching approach, instead of simply matching the URIs in the *Nodes* and *Relations* tables, is that the vanilla API supports partial matches. For example, using the vanilla API, if we ask for edges matching `?start=/c/en/cat` the API returns not only edges whose start node is `/c/en/cat` but also any edge whose start node id is *prefixed* by

[13] https://github.com/IBM/tpf-conceptnet-datasource/tree/main/simplified-conceptnet5

[14] https://www.postgresql.org/docs/current/datatype-json.html

`/c/en/cat` including, for example, `/c/en/cat/n/wn/animal`. The idea here is that `/c/en/cat/n/wn/animal`, i.e., the term "cat" interpreted as the noun (`/n`) used to name an animal (`/animal`) according to WordNet (`/wn`), is a more specific, disambiguated version of the term `/c/en/cat` ("cat").

In the simplified version, we wanted to avoid this partial matching feature for two reasons. First, because it is not compatible with the exact URI matches performed by SPARQL, and second because it complicates query evaluation. Thus, assuming that the support for partial matches was not desirable, we copied the contents of the data field to the columns of a new table, actually a materialized view, called *Simplified_ edges*. This view is essentially the union of *Edges*, *Nodes* and *Relations* with indexes in the columns "start_uri", "rel_uri", and "end_uri". We then implemented simplified versions of the API calls `/query` and `/query/count` which use the *Simplified_ edges* view instead of the original tables. A further advantage of this approach is that it eliminates the need for join operations during query evaluation.

4 Evaluation

In this section, we describe the experimental evaluation of our proposal. Our main goal was to measure the overhead in query evaluation time introduced by the TPF Server (extended with our ConceptNet TPF datasource) and by the SPARQL endpoint (Comunica) in comparison to the ConceptNet API.

We did two experiments, A and B, which dealt with the evaluation of start-rel-end and node-other queries, respectively. For each experiment, a pool of queries was generated by (i) selecting random triples in ConceptNet, (ii) masking some of their components, and (iii) counting the associated number of matches. We then picked enough queries from the pool to obtain a uniform distribution of queries with match-counts ranging from 1 to an upper limit of 12K matches in the vanilla ConceptNet.

The setup of both experiments was the same. We used an OpenShift cluster to run the following:

1. A private instance of ConceptNet[15] (API plus PostgreSQL server) with the modifications discussed in Section 3, deployed with 20GiB of memory.
2. One instance of the TPF server Server.js[16] extended with our ConceptNet TPF datasource, deployed 24GiB of memory. This TPF instance provided two endpoints: a vanilla TPF pointing to the vanilla ConceptNet API and a simplified TPF pointing to the simplified API.
3. Two instances of Comunica[17], each deployed with 8GiB of memory; one instance (vanilla Comunica) pointing to the vanilla TPF endpoint, and the other (simplified Comunica) pointing to the simplified TPF endpoint.
4. One instance of the script used to run the queries and collect the results.

[15] https://github.com/commonsense/ConceptNet5 (fda1b39, Sep. 7, 2021.)
[16] https://github.com/LinkedDataFragments/Server.js (b8cc6e3, Nov. 11, 2022.)
[17] https://github.com/comunica/comunica (e4b91d5, Nov. 25, 2022.)

To minimize the influence of network conditions, we ran the evaluation script in the same cluster as the servers, and we turned off the caches of the TPF server and Comunica. The numbers below constitute thus a worst-case scenario.

Both experiments used a fixed page size with 100 results per page. As remarked in [24], pages should be kept reasonably sized to not overload clients. We chose the value of 100 because it is default page size used by the TPF server Server.js. Note that the page size also determines the number of requests necessary to consume the full response of a query. For example, if a query produces 1000 results, with a page size of 100 it takes 10 HTTP requests (10 pages) to consume all of its results.

Next, we describe Experiments A and B in detail and discuss their results. At the end of the section, we present examples of more expressive SPARQL queries and discuss informally their evaluation using the same experimental setup.

4.1 Experiments A and B

For Experiment A, we used the method described above to generate 432 random start-rel-end queries with results ranging from 1 to about 10K matched edges in the vanilla ConceptNet. We then translated each of these queries into equivalent TPF and SPARQL queries and evaluated each version of a same query using both the vanilla configuration and the simplified configuration. The aggregated results of 5 runs of each query is shown in Figure 4.

For Experiment B, we generated 382 random node-other queries with results ranging from 1 to about 10K matched edges in the vanilla ConceptNet. Node-other queries cannot be represented directly in TPF but can be emulated by disjunctive SPARQL queries, i.e., using the `UNION` operator to match the pattern in either direction of the edge. So, for Experiment B, we translated each query into an equivalent (disjunctive) SPARQL query and, as before, evaluated the versions of a same query using both the vanilla configuration and the simplified configuration. The aggregated results of 5 runs of the 382 random node-other queries are shown in Figure 5.

Analysis We start by analyzing the overhead of the TPF queries in Experiment A. As shown in Table 1, the average (median) difference between the evaluation time of vanilla TPF vs vanilla ConceptNet in Experiment A is 6.2s, and between simplified TPF vs vanilla ConceptNet is 5.6s. In other words, the simplification discussed in Section 3.2 contributed to an average reduction (delta) of $0.6s$ in the evaluation time of TPF queries.

Consider now the overhead of Comunica queries in Experiments A and B. The average difference between the evaluation time of vanilla Comunica vs vanilla ConceptNet is 9.5s in A and 4.4s in B, while the average difference between simplified Comunica vs vanilla ConceptNet is 3.6s in A and $3.1s$ in B. Hence, the proposed simplification seems to have contributed to an average reduction (delta) of 5.9s in the evaluation time of start-rel-end queries and of 1.3s for node-other queries using Comunica. This is a significant reduction considering that the average time of vanilla Comunica queries is 15.6s in A and 13.6s in B.

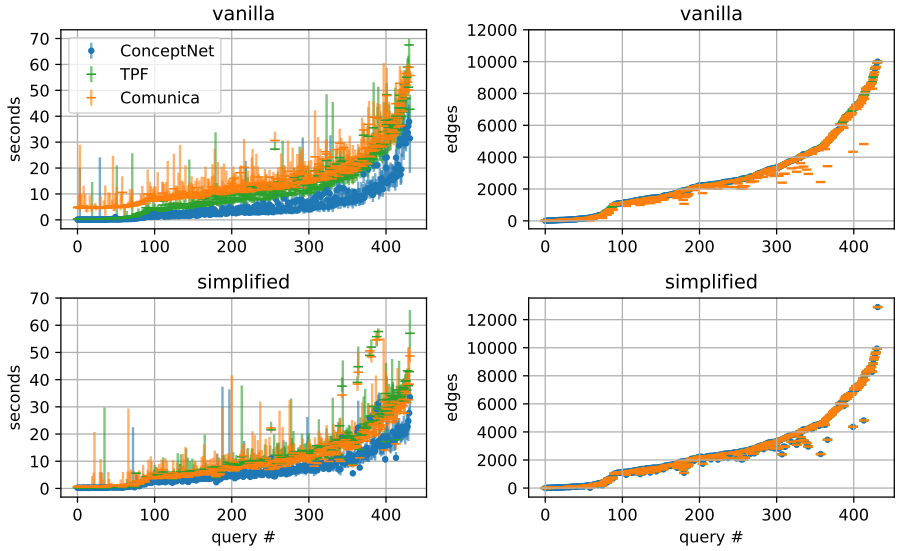

Fig. 4. *Experiment A*: Aggregated results of 5 runs of 432 random start-rel-end queries. The markers indicate the median of the 5 runs. The "query #" axis represents each one of the 432 queries sorted by the number of edges they return. Evaluation time (left) vs edge count (right). Vanilla ConceptNet API, TPF, Comunica (above) vs simplified ConceptNet API, TPF, Comunica (below).

Table 1. Average (median) overhead versus vanilla ConceptNet.

	TPF (A)	Comunica (A)	Comunica (B)
Vanilla	6.2s	9.5s	4.4s
Simplified	5.6s	3.6s	3.1s
delta	0.6s	5.9s	1.3s

Surprisingly, the proposed simplification seems to affect more the first request performed by Comunica than the others. This first request is an empty request which Comunica uses to determine the type of the underlying endpoint. We noticed that this empty request, which is essentially a triple-pattern in which the three components are variables, takes about 3 times longer in the vanilla API than in the simplified API. To make matters worse, Comunica not even uses the results of this request, only its metadata. This suggests further paths for optimization. For example, we could extend the ConceptNet API to handle empty requests in an optimized manner; or we could modify Comunica to avoid such requests altogether by hard-coding in it the service metadata.

We conclude this analysis section by explaining two oddities in Figures 4 and 5. The attentive reader might have noticed that Comunica sometimes seems to beat the ConceptNet API. This of course is impossible, as its requests should

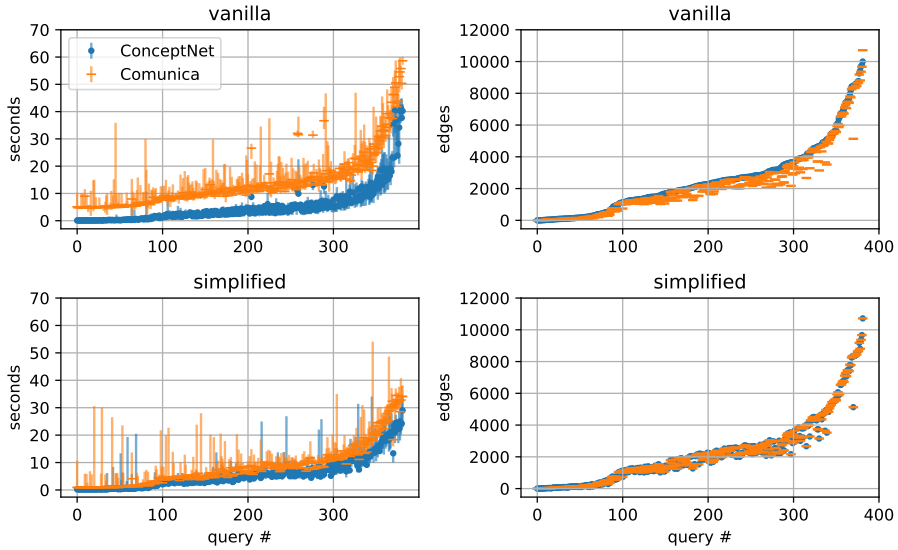

Fig. 5. *Experiment B*: Aggregated results of 5 runs of 382 random node-other queries. The markers indicate the median of the 5 runs. The "query #" axis represents each one of the 382 queries sorted by the number of edges they return. Evaluation time (left) vs edge count (right). Vanilla ConceptNet API, Comunica (above) vs simplified ConceptNet API, Comunica (below).

take at least the same time as those of the underlying API. The explanation is that Comunica uses parallel requests, while the results shown for ConceptNet and TPF assume serial requests. That said, even if the ConceptNet and TPF numbers could have been reduced via parallel requests, the overall improvement induced by our proposed simplification would still apply. Also, the support for parallel requests should count as a feature of the client and might not always be available (e.g., it is not supported by the TPF server).

The second oddity concerns the edge counts for the vanilla Comunica in Figures 4 and 5, which seem to be smaller than those of ConceptNet for some queries. This is due to the partial match feature of vanilla ConceptNet which was explained in Section 3.2. Some vanilla queries return URIs which do not match the URIs in the pattern exactly. Different than the TPF server, Comunica always checks the returned URIs and discard those not matching the pattern exactly. This explains the difference in the number of results for some queries.

4.2 More Expressive SPARQL Queries

We now present some queries that illustrate non-trivial features of SPARQL which are not available in the ConceptNet API but which one gets for free by adopting our approach.

Q1 The first query illustrates the use of multiple triple patterns and regular-expression filters:

```
prefix cnc: <http://conceptnet.io/c/en/>
prefix cnr: <http://conceptnet.io/r/>
select ?x ?y where {
  cnc:cat cnr:Desires ?x.
  ?x cnr:Antonym ?y.
  filter(regex(str(?y), "ing$"))
} limit 10
```

It selects the terms x and y such that "cat" desires x which is an antonym of y and y ends in "ing". This query takes about 5s to execute in our experimental setup using the simplified Comunica endpoint and produces results like "(eat, drinking)", "(sleep, working)", etc.

Q2 The second query illustrates the use of property paths:

```
select ?x ?y where {
  cnc:chair (cnr:MadeOf/cnr:UsedFor) ?x.
  ?x cnr:IsA* ?y.
} limit 10
```

It selects the terms x and y such that "chair" is made of something which is used for x which itself is a type of y. This query takes about 20s to run using the simplified Comunica interface and includes among its results the tuple "(burning, chemical_reaction)".

Q3 The third and last query illustrates the use of federation:

```
prefix wd: <http://www.wikidata.org/entity/>
prefix wdt: <http://www.wikidata.org/prop/direct/>
select ?x where {
  cnc:police cnr:ExternalURL ?x.
  service <http://query.wikidata.org/sparql> {
    ?x wdt:P31 wd:Q5741069.
  }
} limit 10
```

This query selects the entities x in Wikidata such that x is listed as an external reference associated to the term "police" in ConceptNet and, according to Wikidata, is an instance of "rock group" (`wd:Q5741069`). The answer in this case is the Wikidata entity "The Police" (`wd:Q178095`). It takes about 10s to run this query using our simplified Comunica interface. Note that to evaluate it Comunica needs to query the public SPARQL endpoint of Wikidata.

5 Related Work

An early attempt to reconcile ConceptNet and the Semantic Web is [7]. In it the authors discuss the feasibility of an RDF encoding of ConceptNet (then version 3.0) and present a conceptual model of its core relations using OWL (the Web Ontology Language). No implementation or experimental evaluation is provided.

A more recent proposal for converting ConceptNet to RDF is [13]. It is an expansion of ConceptOnto [12], an upper ontology based on ConceptNet. In [13], the authors present an algorithm for extracting edges from ConceptNet 5 data dumps and for converting these edges to RDF. They also discuss use cases involving SPARQL queries, but these are assumed to run over the RDF files resulting from the offline conversion of the data dumps.

Yet another proposal for converting ConceptNet to RDF is [3]. In it the authors present a concise conversion model which attempts simplify the encoding proposed in [13]. They also discuss use cases and present illustrative SPARQL queries which, again, are assumed to be evaluated against the RDF files resulting from the offline conversion of ConceptNet's data dumps.

All of the above proposals have in common the fact that they operate over static data dumps of ConceptNet. The RDF files they produce quickly become obsolete and any attempt to use these files for live queries would require solving the same kind of problems which are already solved by the official ConceptNet API. To the best of our knowledge, our proposal is the first to expose a live RDF interface to ConceptNet which is built on top of the official interface and which supports SPARQL.

6 Conclusion

In this paper, we presented an approach for enabling live SPARQL queries over ConceptNet. Our approach is based on Linked Data Fragments and consists in building a TPF interface on top of the ConceptNet API. As we discussed, this requires only minimal changes to ConceptNet, and with such an interface in place one can then use TPF-compatible engines to evaluate SPARQL queries directly over the ConceptNet server. The experiments we did showed that the overhead incurred is small and can be further reduced by changing the way edges are represented internally by ConceptNet. Also, as discussed in Section 4, there are further opportunities for improvement if one takes into account the peculiarities of the SPARQL engine used.

In future work, we intend to investigate the possibility of making Comunica talk directly to the ConceptNet API, instead of having to go through a TPF interface. This could be done using a Comunica plugin developed for this purpose. Comunica adopts a plugin-based architecture and comes with built-in plugins for reading triples from standard sources like a TPF server. We could develop a custom input plugin that reads triples directly from a given ConceptNet endpoint (using the ConceptNet API) eliminating thus the need of the TPF layer.

References

1. Cai, H., Zhao, F., Jin, H.: Commonsense knowledge construction with concept and pretrained model. In: Proc. 9th Int. Conf. Web Information Systems and Applications (WISA), Dalian, China, September 16–18, 2022. pp. 40–51. Springer (2022)
2. Chen, H., Trouve, A., Murakami, K.J., Fukuda, A.: An introduction to question answering with ConceptRDF. In: Proc. 2nd IEEE Int. Conf. Computational Intelligence and Applications (ICCIA), Beijing, China, September 8–11, 2017. pp. 537–541. IEEE (2017). https://doi.org/10.1109/CIAPP.2017.8167275
3. Chen, H., Trouve, A., Murakami, K.J., Fukuda, A.: A concise conversion model for improving the RDF expression of ConceptNet knowledge base. In: Artificial Intelligence and Robotics, pp. 213–221. Springer (2018)
4. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C (2014), http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
5. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). J. Web Semant. **19**, 22–41 (2013). https://doi.org/10.1016/j.websem.2013.01.002
6. Fielding, R.T.: Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
7. Grassi, M., Piazza, F.: Towards an RDF encoding of ConceptNet. In: Advances in Neural Networks – ISNN 2011. pp. 558–565. Springer (2011)
8. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web **6**, 167–195 (2015). https://doi.org/10.3233/SW-140134
9. Lin, B.Y., Wu, Z., Yang, Y., Lee, D.H., Ren, X.: RiddleSense: Reasoning about riddle questions featuring linguistic creativity and commonsense knowledge. In: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. pp. 1504–1515. ACL (2021). https://doi.org/10.18653/v1/2021.findings-acl.131
10. Longley, D., Kellogg, G., Lanthaler, M., Sporny, M., Lindström, N.: JSON-LD 1.1 processing algorithms and API. W3C recommendation, W3C (2020), https://www.w3.org/TR/json-ld11-api/
11. Miller, G.A.: WordNet: A lexical database for english. Commun. ACM **38**(11), 39–41 (1995). https://doi.org/10.1145/219717.219748
12. Najmi, E., Hashmi, K., Malik, Z., Rezgui, A., Khanz, H.U.: ConceptOnto: An upper ontology based on ConceptNet. In: Proc. IEEE/ACS 11th Int. Conf. Computer Systems and Applications (AICCSA). pp. 366–372. IEEE (2014). https://doi.org/10.1109/AICCSA.2014.7073222
13. Najmi, E., Malik, Z., Hashmi, K., Rezgui, A.: ConceptRDF: An RDF presentation of ConceptNet knowledge base. In: Proc. 7th Int. Conf. Information and Communication Systems (ICICS). pp. 145–150. IEEE (2016)
14. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (2009). https://doi.org/10.1145/1567274.1567278
15. Prud'hommeaux, E., Buil-Aranda, C.: SPARQL 1.1 federated query. W3C recommendation, W3C (2013), https://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/
16. Singh, P.: The public acquisition of commonsense knowledge. In: Proc. AAAI Spring Symposium: Acquiring (and Using) Linguistic (and World) Knowledge for Information Access. AAAI (2002)

17. Speer, R., Chin, J., Havasi, C.: ConceptNet 5.5: An open multilingual graph of general knowledge. In: Proc. 31st AAAI Conf. Artificial Intelligence (AAAI-17), San Francisco, California, USA, February 4–9, 2017. pp. 4444–4451. AAAI (2017)
18. Speer, R., Lowry-Duda, J.: ConceptNet at SemEval-2017 task 2: Extending word embeddings with multilingual relational knowledge. In: Proc. 11th Int. Workshop on Semantic Evaluation (SemEval-2017), Vancouver, Canada, August, 2017. pp. 85–89. ACL (2017). https://doi.org/10.18653/v1/S17-2008
19. Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Champin, P.A., Lindström, N.: JSON-LD 1.1: A JSON-based serialization for linked data. W3C recommendation, W3C (2020), https://www.w3.org/TR/json-ld/
20. Taelman, R., Herwegen, J.V., Sande, M.V., Verborgh, R.: Comunica: A modular SPARQL query engine for the Web. In: The Semantic Web – ISWC 2018. pp. 239–255. Springer (2018)
21. Tamilselvam, S., Nagar, S., Mishra, A., Dey, K.: Graph based sentiment aggregation using ConceptNet ontology. In: Proc. 8th Int. Joint Conf. Natural Language Processing, Taipei, Taiwan, November, 2017 (Volume 1: Long Papers). pp. 525–535. Asian Federation of Natural Language Processing (2017)
22. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: Proc. 14th Annual ACM Symp. Theory of Computing (STOC'82), San Francisco, California, USA, May, 1982. pp. 137–146. ACM (1982). https://doi.org/10.1145/800070.802186
23. Verborgh, R., Sande, M.V., Colpaert, P., Coppens, S., Mannens, E., de Walle, R.V.: Web-scale querying through linked data fragments. In: Proc. Workshop on Linked Data on the Web co-located with the 23rd Int. World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014. CEUR-WS.org (2014)
24. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. J. Web Semantics **37-38**, 184–206 (2016). https://doi.org/10.1016/j.websem.2016.03.003
25. Vrandečić, D., Krötzsch, M.: Wikidata: A free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014). https://doi.org/10.1145/2629489
26. W3C SPARQL Working Group: SPARQL 1.1 overview. W3C recommendation, W3C (2013), http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/
27. Ye, S., Xie, Y., Chen, D., Xu, Y., Yuan, L., Zhu, C., Liao, J.: Improving commonsense in vision-language models via knowledge graph riddles (2022). https://doi.org/10.48550/ARXIV.2211.16504