Fraw: Sampling-Based Approximate Query Processing for Federations of SPARQL Endpoints

Erwan Boisteau-Desdevises[1,*], Thi Hoang Thi Pham[1], Gabriela Montoya[1], Brice Nédelec[1], Hala Skaf-Molli[1] and Pascal Molli[1]

[1]Nantes Université, LS2N, UMR 6004, F-44000 Nantes, France

Abstract

SPARQL federation engines allow users to query multiple SPARQL endpoints as if all RDF data were available through a single virtual endpoint. However, executing complex SPARQL queries over federations while maintaining fast response times remains a major challenge. In this demonstration, we present Fraw, a SPARQL federation engine that supports sampling-based approximate query processing. This approach is particularly useful in scenarios where response time is essential and approximate results are acceptable. We showcase the effectiveness of our engine through an interactive SPARQL query autocompletion use case, where users receive timely suggestions during query authoring, despite the complexity of federated querying.

Keywords

SPARQL, Federation engine, Autocompletion, Sampling, Random Walks

1. Introduction

SPARQL federation engines such as FedX [1], FedUP [2], and CostFed [3] allow users to query multiple SPARQL endpoints as if all the RDF data were available through a single virtual endpoint. A federation engine takes as input a list of SPARQL endpoints and a SPARQL query, decomposes the query into subqueries, sends them to the relevant SPARQL endpoints, and aggregates the results [4].

However, as federations may consist of hundreds of SPARQL endpoints, each containing millions to billions of triples, executing complex SPARQL queries over such federations while maintaining fast response times remains a significant challenge. This limitation currently makes federated querying unsuitable for interactive applications that require real-time responsiveness.

A representative use case of federated querying is SPARQL autocompletions [5, 6, 7], where users receive suggestions for subjects, predicates, or objects as they type their queries, as illustrated in Figure 1a. These autocompletion queries are dynamically generated and executed in real time to provide relevant suggestions for the end user (e.g., query Q_c in Figure 1b). However, current federation engines cannot execute such queries over large federations within the few seconds required for interactivity. For instance, even the fastest current federation engine, FedUP, executes Q_c in 23 seconds.

To address these limitations, we introduce Fraw, a novel SPARQL federation engine capable of producing partial answers within a few seconds (e.g., Q_c is executed in under 3 seconds). Fraw supports Sampling-based Approximate Query Processing (S-AQP) using random walks [8]. S-AQP enables the retrieval of partial or approximate answers rapidly, with the potential to converge towards completeness given sufficient time. In addition to query autocompletion, S-AQP has also been utilized in use cases such as large-scale statistics [9, 10], data summarization [11], approximate aggregations [12], knowledge graph embeddings [13], and join ordering [14].

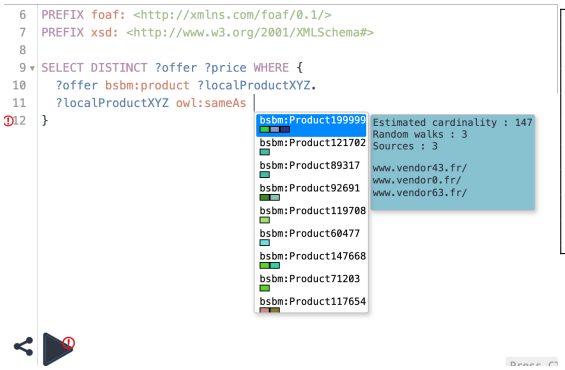
Fraw implements random walks using modified SPARQL operators, supporting the generation of

ISWC 2025 Companion Volume, November 2–6, 2025, Nara, Japan

*Corresponding author.

erwan.boisteau-desdevises@univ-nantes.fr (E. Boisteau-Desdevises); thi-hoang-thi.pham@univ-nantes.fr (T. H. T. Pham); gabriela.montoya@univ-nantes.fr (G. Montoya); brice.nedelec@univ-nantes.fr (B. Nédelec); hala.skaf@univ-nantes.fr (H. Skaf-Molli); pascal.molli@univ-nantes.fr (P. Molli)

0000-0003-0176-2245 (T. H. T. Pham); 0000-0001-5835-0335 (G. Montoya); 0000-0003-4238-5060 (B. Nédelec); 0000-0003-1062-6659 (H. Skaf-Molli); 0000-0001-8048-273X (P. Molli)



(a) Autocompletion suggestions obtained in 2.97s.

```
PREFIX bsbm: <http://www4.wiwiss.../vocabulary/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT ?suggestion_variable (COUNT(*) AS ?count)
WHERE {
  ?offer bsbm:product ?localProductXYZ. #tp1
  ?localProductXYZ owl:sameAs ?suggestion_variable #tp2
}
GROUP BY ?suggestion_variable
```

(b) Q_c Autocompletion query that retrieves suggestions for the second object. Q_c execution time is 23s with FedUP, the current fastest federation engine. Too long for an interactive scenario.

Figure 1: Autocompletion scenario for a SPARQL federation of 200 endpoints.

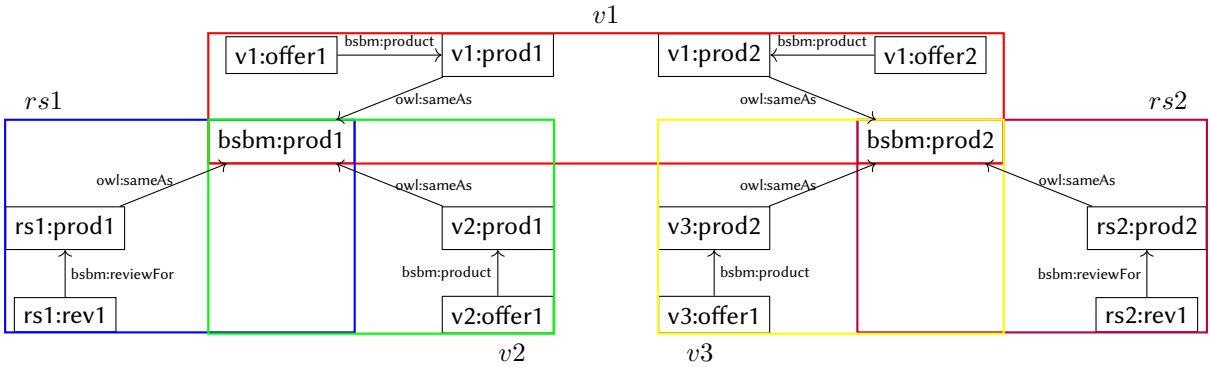


Figure 2: Federation F comprises five SPARQL endpoints $v1$, $v2$, $v3$, $rs1$, $rs2$. $v1$, $v2$, $v3$ are vendor sites, and $rs1$, $rs2$ are reviewing sites.

approximate answers within a predefined budget (i.e., a specified number of random walks). Each successful random walk – one that satisfies the query constraints – constitutes a query result. As the budget increases, more results are obtained, and with a sufficiently large budget, it is possible to generate all results. While sampling techniques have been explored for single SPARQL endpoints [15, 16], they have not yet been investigated in federated query processing settings.

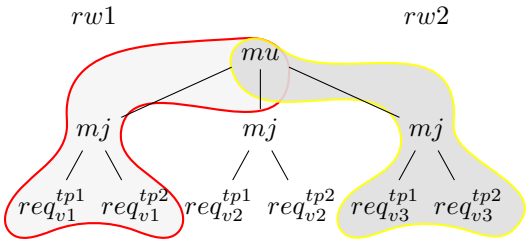
In this demonstration, we illustrate the benefits of S-AQP through a federated SPARQL query autocompletion scenario. As shown in Figure 1a, S-AQP enables an autocompletion query to be executed in just a few seconds over the FedShop federation of 200 endpoints [17]. Colored squares indicate the number of SPARQL endpoints in which a suggested predicate (for this completion) is available. In addition, tooltip boxes provide details about the specific source where the suggestion is available, along with a cardinality estimate.

To the best of our knowledge, Fraw is the first engine to introduce Approximate Query Processing for federated SPARQL querying. Moreover, Fraw is the first system to demonstrate the practical benefits of this approach in the context of federated query authoring through SPARQL autocompletion.

2. Fraw: Sampling-Based Approximate SPARQL Federation Engine

Consider the federation F, simulating an e-commerce scenario where multiple rating sites and vendors each have their own knowledge graphs (Figure 2). F is made of vendors $v1$, $v2$, and $v3$ and rating sites $rs1$, $rs2$. F is based on a simplified version of the FedShop schema [17]:

- Vendors include offer entities (e.g., `v1:offer1`), local product entities (e.g., `v1:prod1`) and their corresponding global product entities (e.g., `bsbm:prod1`). Offers refer to local products



(a) Union-over-join logical plan.

```
PREFIX <http://www4.wiwiss.../vocabulary/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT * WHERE {
  ?offer bsbm:product ?localProductXYZ. #tp1
  ?localProductXYZ owl:sameAs ?suggestion_variable #tp2
}
```

(b) Q_a query that retrieves suggestions for the 2^{nd} object.

Figure 3: FedUP's union-over-join logical plan to process the BGP in Q_a over the federation of five endpoints, F, given in Figure 2. The plan consists of the union of three subqueries, each corresponding to the evaluation of the two triple patterns at one of the three vendor endpoints. Lighter gray (and red) highlights the choice made by the random walk $rw1$ (endpoint $v1$), while darker gray (and yellow) highlights the choice made by $rw2$. The logical plan is represented using FedQPL [18], with multi-union (*mu*), multi-join (*mj*), and request of a triple pattern tpi to a federation member vj (req^{tpi}_{vj}).

via the `bsbm:product` property, and each local product is linked to a global product via the `owl:sameAs` property.

- Rating sites include review entities (e.g., `rs1:rev1`), local product entities (e.g., `rs1:prod1`) and their corresponding global product entities (e.g., `bsbm:prod1`). Reviews are associated with a local product via the `bsbm:reviewFor` property, and each local product is linked to a global product using the `owl:sameAs` property.

Each federation member is running its own SPARQL endpoint that supports sampling as proposed in [15].

We aim to sample the query Q_a from Figure 3b composed of 2 triple patterns tp_1 and tp_2. To do that, Fraw first needs a decomposition and source selection for Q_a. Fraw relies on FedUP [2] for this part. The logical plan produced by FedUP for Q_a on F is given in Figure 3a. mj is a multi-join operator and mu is a multi-union operator. req^{tp}_{src} represents the evaluation of triple pattern tp on source src. During query execution, mj operators from this plan can be executed respectively on v_1, v_2 and v_3 (thanks to exclusive group optimization), but its multi-union operator is executed by the federation engine.

While traditional federation engines must evaluate each triple pattern at all relevant sources to ensure complete results, approximate query processing based on random walks can retrieve sample results along with cardinality estimates, issuing significantly fewer requests to the endpoints and transferring much less data. To do that, Fraw maps logical operators of the plan to physical operators that implement random walks. For example, the multi-union sampled operator randomly chooses one branch of the union for each random walk.

Consider a first random walk, $rw1$, over the federation F, for which Fraw randomly chooses to evaluate both $tp1$ and $tp2$ at endpoint $v1$, as illustrated in Figure 3a. This combination of sources is one of three possible options, giving it a selection probability of $\frac{1}{3}$. Among the two triples in v_1 matching tp_1, one is selected uniformly at random, with probability $\frac{1}{2}$. Suppose this binds the variable `?localProductXYZ` to `v1:prod1`, so that tp_2 becomes: `v1:prod1 owl:sameAs ?suggestion_variable`. Since v_1 contains exactly one triple matching this pattern, `?suggestion_variable` is bound to `bsbm:prod1` with probability 1. Thus, the overall probability of this random walk is: $\frac{1}{3} \times \frac{1}{2} \times 1 = \frac{1}{6}$.

Taking the inverse of this probability yields an estimate of the BGP's cardinality, six in this case. Although only one random walk was performed, it provides an initial estimation of the number of solutions. The actual cardinality is four, indicating that the estimate slightly overstates it.

Performing additional random walks could lead to new possible values for `?suggestion_variable`. For instance, consider a second random walk $rw2$, that selects $v3$ as source for $tp1$ and $tp2$, and binds

```
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?offer ?price WHERE {
    ?offer bsbm:product ?localProductXYZ .
    ?localProductXYZ owl:sameAs ◊$ProductXYZ .
    ?offer bsbm:vendor ?vendor .
    ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#US> .
    ?offer bsbm:deliveryDays ?deliveryDays .
    FILTER (?deliveryDays <= 3)
    ?offer bsbm:price ?price .
    ?offer bsbm:validTo ?date .
    FILTER (?date > $currentDate )
}
ORDER BY xsd:double(str(?price))
LIMIT 10
```

Figure 4: FedShop's template query Q10 [17] that *gets offers for a product that satisfies specific requirements*. The template is instantiated, providing values for variables $ProductXYZ and $currentDate. The autocompletion targets the second object.

variable `?suggestion_variable` to `bsbm:prod2`. In this case, the probability of the random walk is calculated as $\frac{1}{3} \times 1 \times 1 = \frac{1}{3}$, which leads to an estimated cardinality of three.

Multiple random walks may produce duplicate values for the same variable. Moreover, as the number of walks increases, the overall cardinality estimate becomes more accurate, converging toward the actual value. The final estimate is typically computed as the average of the individual estimates [15]. In our demonstration, we leverage these cardinality estimates to rank autocompletion suggestions presented to the user (Figure 1a).

Fraw [1] supports sampling for core SPARQL, i.e., Join, Optional, Union and Filter operators. Fraw combines FedUP's source selection and HeFQUIN's query execution [19]. We implemented sampling in HeFQUIN by overloading the operators with sampled versions.

To the best of our knowledge, federated query autocompletion has not been explored in previous work. However, several approaches exist for query autocompletion in single-endpoint settings [20, 5]. For example, the approach proposed by Parra and Hogan [20] uses graph summaries to provide autocompletion suggestions. While these suggestions can be generated quickly, they are over-approximations, i.e., some answers may be incompatible with the query context and can lead to queries that return no results. In contrast, all suggestions produced by Fraw are context-aware and lead to queries that produce answers. QLever [5] is another system capable of efficiently producing exact answers to SPARQL autocompletion queries. It supports interactive autocompletion in single-endpoint settings through a query engine that leverages specialized indexing, caching, and paging optimizations. Unlike QLever, Fraw can produce suggestions with varying degrees of completeness based on a configurable budget. Fraw does not rely on engine-specific optimizations, but it can take advantage of them if the engine supports approximate query processing.

3. Fraw in Action

In an autocompletion use case, as presented in Figure 1a, users type SPARQL queries over a federation F of 200 SPARQL endpoints available in the FedShop Benchmark [17]. The 200 endpoints are made available as virtual endpoints in a Blazegraph server extended to support sampling [2].

Suppose a user is completing template Q10 (Figure 4). At any time while writing a triple pattern of this template, the user can request some suggestions by pressing CTRL+SPACE. According to the cursor position, an autocompletion query Q_c (Figure 1b) is generated. It is composed of the current state of

[1] Fraw is available at https://github.com/GDD-Nantes/HeFQUIN-FRAW
[2] This sampling extension is available at https://github.com/passage-org/passage/tree/main/raw

(a) n=5 random walks. (b) n=100 random walks.

Figure 5: Suggestions for context `SELECT * WHERE { ?s ◊ }` obtained after executing n random walks against the federation given in Figure 2. With 100 random walks, the estimated cardinality of `rdf:type` (11,769,746) is closer to the real cardinality (9,916,158) than with 5 random walks (16,170,160).

the query, which forms the context, and a generated triple pattern asking for the subject, predicate, or object depending on the cursor position. Next, Q_a (Figure 3b), a subquery of Q_c, is sent to FRAW to be sampled with a budget. The budget is expressed as a number of random walks and can be set in the user interface. When FRAW receives a query to sample, it computes a source selection to generate a plan, then performs random walks until the budget is exhausted. Results are extended with their associated provenance and cardinality estimation. This information can then be displayed on the user interface.

As FRAW relies on S-AQP, the results of autocompletion queries can be empty. This can happen either because a query does not have any results on the federation, or because results do exist but were not found within the imposed budget. By pressing CTRL+SPACE again (repeatedly, if desired), additional information can be retrieved and aggregated with the previously obtained data. New suggestions may appear, potentially from previously unexplored endpoints, and existing cardinality estimates may be refined as more random walks are performed.

As random walks are intrinsically biased by frequent patterns, frequent RDF terms are more likely to appear first as suggestion, which seems helpful in an autocompletion scenario. In other scenarios where the user searches for an infrequent RDF term or wants to be sure that there are no results, then executing the autocompletion query without sampling may be a better solution than waiting for the random walks to cover most of the data.

4. Live Demonstration (https://youtu.be/RYHGjBkvEbM)

During the session, we will deploy a federation comprising 200 SPARQL endpoints generated using the FedShop benchmark [17], including 100 vendor endpoints and 100 review site endpoints. Attendees will be able to formulate their own queries or choose from a set of predefined scenarios, and work with the entire federation or any of the deployed endpoints individually. In the first scenario, the user is unfamiliar with the data exposed by the federation and wishes to explore its structure and contents. The objective is to discover the federation members and gain insights into the properties and classes used by each endpoint.

To support this exploration, we rely on the partial query `SELECT * WHERE { ?s ◊ }`, where ◊ indicates the position to complete, as illustrated in the video (https://youtu.be/RYHGjBkvEbM). We then perform a variable number of random walks to generate suggestion batches. Increasing the number of random walks provides a broader coverage of the federation, improving the diversity of suggestions and the accuracy of the cardinality estimates. Figure 5 presents suggestions obtained using 5 and 100 random walks, respectively.

In the second scenario, the attendee is provided with a natural language description of a query over the FedShop federation and is invited to formulate and execute a corresponding SPARQL query to retrieve relevant results. For instance, the attendee might be asked to answer the following question:

```
SELECT DISTINCT ?sameAsOffer (GROUP_CONCAT(?country) AS ?countries) WHERE {
  ?offer rdf:type bsbm:Offer.
  ?offer owl:sameAs ?sameAsOffer.
  ?offer bsbm:product ?product.
  ?product bsbm:producer ?producer.
  ?producer bsbm:country ?country.
}
GROUP BY ?sameAsOffer
HAVING (!CONTAINS(STR(?countries), "US"))
```

Figure 6: A query that *gets all the offers whose products are exclusively non-US producers*, similar to FedShop's template query Q10.

"What are the offers whose products are exclusively produced by entities that are not from the United States?"

Using the autocompletion feature as needed, attendees are encouraged to build queries while gradually discovering the structure of the federation. The resulting query may resemble the one shown in Figure 6, although it may differ in form. For example, the first triple pattern could be omitted, as any entity with a `bsbm:product` property is, by definition, an offer in the FedShop schema.

5. Conclusion

We presented Fraw, the first SPARQL federation engine to support Sampling-based Approximate Query Processing (S-AQP). We demonstrated its effectiveness for federated query authoring through an autocompletion use case. Beyond this scenario, Fraw opens promising directions for federated query processing, including summarization [21], approximate aggregate queries [22], and cardinality estimation [23]. While our current demo leverages source selection plans from FedUP, an interesting perspective is to investigate sampling strategies at the source selection stage itself.

Declaration on Generative AI

During the preparation of this work, the authors used https://chatgpt.com in order to: check grammar and spelling. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Acknowledgments

This work is supported by the French ANR project MeKaNo (ANR-22-CE23-0021) and the French Labex CominLabs project MiKroloG (Microdata Knowledge Graph).

References

[1] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, FedX: Optimization Techniques for Federated Query Processing on Linked Data, in: International Semantic Web Conference (ISWC), Springer, Bonn, Germany, 2011, pp. 601–616.

[2] J. Aimonier-Davat, B. Nédelec, M. H. Dang, P. Molli, H. Skaf-Molli, FedUP: Querying Large-Scale Federations of SPARQL Endpoints, in: WWW, ACM, 2024, pp. 2315–2324.

[3] M. Saleem, A. Potocki, T. Soru, O. Hartig, A. N. Ngomo, CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation, in: SEMANTiCS, volume 137 of *Procedia Computer Science*, Elsevier, 2018, pp. 163–174.

[4] M. Acosta, O. Hartig, J. Sequeda, Federated RDF Query Processing, Springer, Cham, 2018, pp. 1–8.

[5] H. Bast, J. Kalmbach, T. Klumpp, F. Kramer, N. Schnelle, Efficient and Effective SPARQL Autocompletion on Very Large Knowledge Graphs, in: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, CIKM '22, ACM, 2022, p. 2893–2902.
[6] V. Emonet, A. C. Sima, T. M. de Farias, A User-Friendly SPARQL Query Editor Powered by Lightweight Metadata, in: ESWC (Demos), 2025, p. 5.
[7] A. Vercruysse, J. A. R. Meléndez, P. Colpaert, The Semantic Web Language Server: Enhancing the Developer Experience for Semantic Web Practitioners, in: ESWC (2), volume 15719 of *Lecture Notes in Computer Science*, Springer, 2025, pp. 210–225.
[8] F. Li, B. Wu, K. Yi, Z. Zhao, Wander Join: Online Aggregation via Random Walks, in: Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, ACM, New York, NY, USA, 2016, p. 615–629.
[9] A. Soulet, F. M. Suchanek, Anytime Large-Scale Analytics of Linked Open Data, in: The Semantic Web – ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, pp. 576–592.
[10] J. Debattista, S. Londoño, C. Lange, S. Auer, Quality Assessment of Linked Datasets Using Probabilistic Approximation, in: European semantic web conference, Springer, 2015, pp. 221–236.
[11] L. Heling, M. Acosta, Estimating Characteristic Sets for RDF Dataset Profiles Based on Sampling, in: The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2020, p. 157–175.
[12] Y. Wang, A. Khan, X. Xu, J. Jin, Q. Hong, T. Fu, Aggregate Queries on Knowledge Graphs: Fast Approximation with Semantic-Aware Sampling, in: 2022 IEEE 38th International Conference on Data Engineering (ICDE), IEEE, 2022, pp. 2914–2927.
[13] P. Ristoski, H. Paulheim, RDF2Vec: RDF Graph Embeddings for Data Mining, in: The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I, Springer-Verlag, Berlin, Heidelberg, 2016, p. 498–514. doi:10.1007/978-3-319-46523-4_30.
[14] V. Leis, B. Radke, A. Gubichev, A. Kemper, T. Neumann, Cardinality Estimation Done Right: Index-Based Join Sampling., in: Cidr, 2017, p. 8.
[15] J. Aimonier-Davat, M. H. Dang, B. Nédelec, H. Skaf-Molli, P. Molli, RAW-JENA: Approximate Query Processing for SPARQL Endpoints, in: ISWC (Posters/Demos/Industry), volume 3632 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, p. 5.
[16] T. H. T. Pham, P. Molli, B. Nédelec, H. Skaf-Molli, J. Aimonier-Davat, CRAWD: Sampling-Based Estimation of Count-Distinct SPARQL Queries, in: International Semantic Web Conference, Springer, 2024, pp. 98–115.
[17] M.-H. Dang, J. Aimonier-Davat, P. Molli, O. Hartig, H. Skaf-Molli, Y. L. Crom, FedShop: A Benchmark for Testing the Scalability of SPARQL Federation Engines, in: International Semantic Web Conference (ISWC), Springer, Athens, Greece, 2023, pp. 285–301.
[18] S. Cheng, O. Hartig, FedQPL: A Language for Logical Query Plans over Heterogeneous Federations of RDF Data Sources, in: the 22nd International Conference on Information Integration and Web-Based Applications & Services, ACM, New York, NY, USA, 2021, p. 436–445.
[19] S. Cheng, O. Hartig, A Cost Model to Optimize Queries over Heterogeneous Federations of RDF Data Sources, in: ESWC Workshops, volume 3443 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, p. 14.
[20] G. Parra, A. Hogan, Fast Approximate Autocompletion for SPARQL Query Builders, in: VOILA@ISWC, 2021, pp. 41–55. URL: https://api.semanticscholar.org/CorpusID:247862910.
[21] T. H. T. Pham, H. Skaf-Molli, P. Molli, B. Nédelec, Online Sampling of Summaries from Public SPARQL Endpoints, in: WWW (Companion Volume), ACM, 2024, pp. 617–620.
[22] J. Aimonier-Davat, H. Skaf-Molli, P. Molli, A. Grall, T. Minier, Online Approximative SPARQL Query Processing for COUNT-DISTINCT Queries with Web Preemption, Semantic Web – Interoperability, Usability, Applicability (2022).
[23] J. Aimonier-Davat, H. Skaf-Molli, P. Molli, M.-H. Dang, B. Nédelec, Join Ordering of SPARQL Property Path Queries, in: European Semantic Web Conference, Springer, 2023, pp. 38–54.