Integration and Optimisation of CoSMo with Wikidata for Efficient Content Retrieval in Abstract Wikipedia: A Literature Review

Jordy Kafwe
University of Cape Town
Cape Town, South Africa
KFWJOR001@myuct.ac.za

ABSTRACT

This literature review explores the integration and optimisation of CoSMo with Wikidata for efficient content retrieval in Abstract Wikipedia. By delving into the critical role of structured data and natural language generation techniques, the review highlights the challenges in achieving uniform content coverage across different language editions of Wikipedia. The introduction of Abstract Wikipedia by the Wikimedia Foundation is examined as a step towards automating content creation and ensuring consistency and accuracy across articles. Strategies such as optimised SPARQL queries, caching mechanisms, and heuristics are discussed for improving query response times and system performance within the Abstract Wikipedia ecosystem. The significance of Wikidata as a collaborative knowledge graph and its integration with Wikipedia for data enrichment and cross-language knowledge sharing are emphasised. Overall, this review provides insights into enhancing content retrieval efficiency and addressing scalability challenges in the context of Abstract Wikipedia.

CCS CONCEPTS

• **Information systems** → **Query languages**; **Information retrieval**.

KEYWORDS

Wikidata, Abstract Wikipedia, SPARQL, Query optimisation,

1 INTRODUCTION

Wikipedia stands as an achievement in global knowledge sharing, striving to democratise access to comprehensive information across languages and disciplines [14]. Despite its success, disparities in knowledge representation across different language editions persist, highlighting challenges in achieving uniform content coverage. The introduction of Abstract Wikipedia by the Wikimedia Foundation marks a step towards automating content creation using structured data and natural language generation techniques [15]. This structured approach not only aims to enhance the efficiency of content creation but also ensure consistency and accuracy across articles. It promises enhanced multilingual coverage, facilitating article generation in any natural language based on structured data inputs from Wikidata. [13] This literature review delves into the critical role of Wikidata within the Abstract Wikipedia pipeline and explores methods for optimising content retrieval from Wikidata for efficient article generation.

2 BACKGROUND

2.1 Wikidata and its Significance

Wikidata, established in 2012, serves as a collaborative knowledge graph within the Wikimedia Foundation ecosystem, housing a vast repository of structured data [15]. Wikidata currently consists of about 400 million statements and 5 billion triples. [14] Its integration with Wikipedia provides support for data enrichment and cross-language knowledge sharing. However, the scale and complexity of Wikidata's knowledge graph pose challenges in efficient data retrieval and utilisation.

2.2 Wikidata's RDF Triple Structure and SPARQL

The structured nature of Wikidata's RDF triples facilitates data interoperability and query capabilities through the SPARQL (SPARQL Protocol and RDF Query Language) query language. [14] Wikidata's Resource Description Framework (RDF) triples consist of subject-predicate-object statements that represent relationships between entities in a structured manner. Additionally, Wikidata allows for the inclusion of qualifiers in its triples, providing additional contextual information and enhancing the richness of the data model. This structured format enables efficient storage and retrieval of data, making complex queries across Wikidata's knowledge graph feasible. SPARQL plays a role in this ecosystem by allowing users to formulate queries that target specific properties, qualifiers, and relationships within the RDF triples. This level of granularity provides users with fine-grained control over data retrieval.

2.3 Public SPARQL Endpoint

Wikidata exposes a public SPARQL endpoint, enabling researchers, developers, and enthusiasts to interact programmatically with its rich dataset [15]. This public endpoint allows for real-time querying and exploration of Wikidata's structured data. The SPARQL endpoint enforces a query answering timeout of 60 seconds, after which queries exceeding this duration are terminated. [8]In addition, each client is restricted to a maximum of five concurrent queries and is subject to constraints on processing time and query errors. To manage this, a token bucket protocol is used, granting clients an initial allowance of 120 seconds of processing time and a quota of 60 allowable query errors. These allowances are replenished at a rate of 60 seconds/30 errors per minute, ensuring clients can maintain a consistent query workload within the specified limits. [8]

2.4 Abstract Wikipedia Pipeline and Content Determination

The Abstract Wikipedia initiative introduces a novel approach to content creation by using structured data from Wikidata and employing natural language generation (NLG) techniques [15]. A stage in this NLG pipeline is content determination [12], where structured data from Wikidata guide the assembly of article components. [2] However, the interconnectivity of Wikidata's RDF triples can pose performance challenges during content retrieval. This is primarily due to RDF stores prioritising flexibility in information structuring over performance, compounded by Wikidata's vast scale, comprising 400 million statements and 5 billion triples. [8]

2.5 Need for Optimisation in Content Retrieval from Wikidata

Optimising content retrieval from Wikidata is imperative due to scalability challenges inherent in querying large RDF datasets. Complex queries and data traversal operations can result in increased query times.[8]. Coupled with the aforementioned 60-second query answering timeout, optimising strategies become imperative to ensure efficient data fetching, timely content assembly, and improved user experiences within the Abstract Wikipedia ecosystem.

3 PERFOMANT SPARQL QUERIES

3.1 Heuristics

Due to the flexibility of RDF combined with the expressiveness of SPARQL, it is easy to create queries that strain even the most optimised RDF stores. The paper "On the Formulation of Performant SPARQL Queries" by Loizou and Groth provides an exploration of heuristics for optimising SPARQL queries. [7] The authors address the challenges faced by application developers in formulating correct, complex, and performant SPARQL queries due to the flexibility of RDF. The paper introduces 5 heuristics that can be applied to create optimised SPARQL queries. The heuristics proposed in the paper are inspired by formal results found in the literature and hands-on experience in developing an end-user focused data integration system, the Open-PHACTS Platform. They are intended to help developers formulate SPARQL queries that can be effectively optimised by RDF stores. The heuristics propose minimising optional triple patterns, localising SPARQL subpatterns, using dataset statistics, including necessary graph patterns, and enabling paginated views. [7] The paper also provides an empirical comparison of the performance of six state-of-the-art RDF storage systems with respect to the various heuristics, using large real-world pharmaceutical datasets and queries. The experimental results demonstrate improvements in performance across 6 state-of-the-art RDF stores. The average query response time showed from 1 to 5 orders of magnitude improvement across all heuristics. [7] This being observed using state-of-art RDF stores on large real-world dataset showcases the practical impact of applying the heuristics.

3.2 Wikidata Optimisations Documentation

The Wikidata documentation provides strategies for optimising SPARQL query performance when using their SPARQL endpoint. This advice is speicifically tailored to the Wikidata RDF structure and their RDF engine, Blazegraph. While their approach may not be as empirical as the paper by Loizou and Groth, it provides practical evidence and theoretical explanations for the effectiveness of their strategies. The improvement is highlighted in the documentation by queries that get terminated before, running to completion with a rewrite using their strategies. For example, by leveraging fixed values and ranges within queries, users can target specific data subsets, minimising the amount of data the system needs to traverse. Property paths offer an efficient navigation method, guiding queries directly to relevant sections. However, maintaining conciseness and avoiding overly complex inverse paths is crucial for optimal performance. Experimentation with the order of query clauses allows for prioritising critical tasks for more efficient execution. [16] Additionally, decomposing complex queries into smaller, manageable subqueries enhances both readability and potentially improves query performance. Lastly, when retrieving counts, employing optimised aggregate functions like "fast range counts" instead of generic functions like ‘COUNT(*)‘ can yield superior results. [16] The documentation states that applying the strategies leads to faster data retrieval, improved query execution plans, and quicker calculations, ultimately leading to a significant enhancement in overall query performance within the Wikidata knowledge base.

3.3 Motivation

The motivation for writing optimised SPARQL queries for Abstract Wikipedia's reliance on Wikidata stems from the tangible impact on system performance and scalability. By leveraging heuristics from research literature and platform-specific optimisation strategies from Wikidata's documentation, significant improvements in query response times can be achieved. What makes query optimisation highly feasible is that it uses the current hardware and infrastructure; it primarily involves rewriting queries sent to the existing infrastructure. [7] This approach minimises additional costs and complexities while directly enhancing the efficiency of data retrieval processes. This would help ensure a faster and less timeout prone data fetching experience for users within the Abstract Wikipedia ecosystem.

4 CACHING

Caching refers to storing the results of a query or parts of it in a temporary storage location. This helps speed up future executions of the same or similar queries by retrieving the results from the cache instead of re-executing the query against the SPARQL endpoint. The public Wikidata public SPARQL endpoint already implements some caching, but additional methods are reviewed below. [9]

4.1 Content Aware Caching

Akhtar et al. propose an Adaptive Cache Replacement (ACR) approach to improve SPARQL query processing. The ACR algorithm parallelises the task to calculate the access frequencies and uses the edit distance to identify similar querying patterns. [1]It places frequently accessed queries in the cache to reduce the burden on SPARQL endpoints. The approach achieves better query response time, less space overhead, and a cache hit rate of 80.66 percent, accelerating the querying speed by 6.34 percent. ACR outperforms

existing cache replacement approaches such as Least Recently Used, Least Frequently Used, and SPARQL Query Caching in terms of space efficiency and hit rates. [1] The approach is content-aware because it considers the content of the queries in addition to their structural similarity. This is important because two queries may have the same structure but differ in their content, and treating them as similar can lead to incorrect caching decisions. This means it can. The approach is content-aware because it considers the content of the queries in addition to their structural similarity when checking the cache. This is important because two queries may have the same structure but differ in their content, and treating them as similar can lead to incorrect caching decisions. Content aware caching (CAC) is used by systems such as DBproxy and MTCache. [11] When a query is run, the CAC system verifies if it matches any cached data; if so, it generates the result based on that data or sends the query to the database server. This increases performance

4.2 Content Blind Caching

Martin et al. implement a content blind cache. [9] That is the cache is not aware of its contents. They achieve this by computing the md5 hash of every query. This is approach is very quick and computationally inexpensive, but could result in many false cache misses due to slight syntatic differences in the queries that would give the same result. [9]

4.3 Client-side and Server-side Caching

Caching on the client-side, like Akhtar et al., involves storing cached data on the client's device or browser. [1] This can reduce network calls, improving performance and reducing server load. However, it may require additional resources and storage capacity on the client's end, impacting device performance or storage limits. Server-side caching, like implemented by Martin et al., stores cached data on the server or an intermediary caching server. [9] This can benefit all clients accessing the server, reducing overall response times and server load. Server-side caching can be more easily managed and controlled but may not be as effective for personalised or dynamic content as client-side caching. Both client-side and server-side caching strategies can be used together to optimise performance based on specific application requirements and resource constraints.

5 QUERY EXPANSION

Query expansion refers to the process of augmenting a user's initial search query with additional terms or concepts to improve information retrieval accuracy and relevance. This expansion aims to capture a broader range of relevant documents or data points that may not have been explicitly captured by the original query, thereby enhancing the comprehensiveness and precision of search results. [3]

5.1 Nearest Neighbour

Guisado-Gámez et al. explore the use of a graph-based approach to represent the intricate relationships between Wikipedia articles. [5] Here, each article is conceptualised as a node within the graph, interconnected by edges that signify links between these concepts. This structural representation not only aids in identifying interconnected concepts but also facilitates query expansion based on these interrelations, enhancing the search process's contextual depth. [5] This is akin to a rudimentary Wikidata. Central to this approach is the construction of the graph using knowledge bases such as Wikipedia, Yago, or DBPedia, which inherently provide a structured graph of concepts along with their relations. This incorporation of semantic knowledge into the query expansion process significantly enhances the system's ability to suggest relevant and semantically connected phrases, thereby enriching the search experience for users. Furthermore, the graph-based methodology contributes to query disambiguation by analysing the topological properties of the graph. It can discern related articles, redirects, and even terms that may not have been explicitly introduced by the user but hold relevance to the query context, it does by exploring how closely the nodes are linked. [5] This comprehensive analysis enables the system to not only expand queries effectively but also to provide more accurate and contextually relevant search results, ultimately improving the overall search quality and user satisfaction. In addition to discussing the graph-based techniques, the paper also highlights the potential synergies with linked data techniques like anchor analysis, offering complementary strategies for query expansion and information retrieval. By leveraging semantic knowledge and exploiting the inherent relationships within knowledge graphs, the graph-based approach emerges as a powerful tool for enhancing search relevance and quality in information retrieval systems in Wikidata as it is already a knowledge graph with many relationships.

5.2 Synonym Predicate

The paper "Enhancing Query Answer Completeness with Query Expansion based on Synonym Predicates" delves into strategies aimed at improving query answer completeness by employing query expansion techniques based on synonym predicates. [10] One notable methodology proposed in the paper involves leveraging community-based knowledge graphs to identify synonym predicates, thereby expanding queries to enhance answer completeness. As a result, an assumption that community-based knowledge graphs consist of synonym predicates that complement knowledge graph triples required to raise query answering completeness has been made. Central to this approach is the use of knowledge graph embedding techniques such as TransD, TransH, and RDF2vec, which extract embeddings. Knowledge graph embedding is a technique that aims to represent entities and relations of knowledge graphs into low-dimensional semantic spaces. [10] The experimental validation of this methodology showcases its superiority over baseline techniques, including frequent item set mining and alternative embedding methods like RDF2vec, TransD, and TransH, particularly in terms of query completeness. Using their proposed method, the recall increases from 0.3 to 0.8 and using the reformulated query increases the results found from 6 to 45 in an example provided in the paper. [10] Although this approach has been shown to improve results, it remains to be seen whether expanding queries on synonyms can work on Wikidata. In addition, extracting embeddings adds more work in fetching from the SPARQL endpoint. That is, the completeness of information retrieved may increase, but at the cost of speed.

6 MIDDLEWARE

A middleware is a software layer or component that acts as an intermediary between different applications, systems, or services, facilitating communication and data exchange between them. [4] The BASIL middleware framework bridges the gap between Web APIs and SPARQL endpoints, leveraging the advantages of both paradigms while addressing their limitations. Web APIs are widely adopted among web developers for their stable interfaces, promoting loose coupling and interoperability. They serve as the foundation for BASIL's API development approach. Although the paper highlights a critical drawback of traditional Web APIs: their limited ability to harness SPARQL's robust data integration capabilities. Unlike Web APIs, SPARQL offers unmatched querying and RDF data integration abilities. However, integrating SPARQL into traditional API architectures can lead to challenges such as violating loose coupling principles and creating strong dependencies between data models and client software. BASIL's middleware architecture mediates interactions between SPARQL endpoints and consumer applications, ensuring loose coupling and facilitating seamless API operations while preserving SPARQL's data integration benefits. This approach also improves caching solutions for users, a crucial aspect given the syntactic variations in queries with identical semantics in SPARQL, as they can just based on the HTTP route. [4] Standardised interfaces through Swagger specifications and customisable API outputs empower developers to leverage SPARQL efficiently within their API ecosystems. Insights from the paper regarding embedding SPARQL queries directly into applications emphasise the importance of BASIL's middleware in maintaining loose coupling and mitigating dependencies. This approach enables agility in response to schema evolutions and allows data providers to optimise query performance, enhancing efficiency, reliability, and sustainability in data-driven applications using RDF and SPARQL technologies. [4] BASIL's middleware ultimately aims to integrate Web APIs and SPARQL to data access and integration in modern web clients. This would work well in Abstract Wikipedia because the constructor management and templating frontend clients and engineers (would never have to interact directly interact with SPARQL and would only have to worry about their main tasks.

7 SUBSETTING

Subsetting refers to the process of extracting specific and relevant subsets of data from a larger dataset or knowledge graph, such as Wikidata, to meet particular research, operational, or analytical needs. [6] Beghaeiraveri et al. undertake a comprehensive literature survey of emerging tools designed for Wikidata subsetting, including WDSub, KGTK, WDumper, and WDF. [6] These tools offer functionalities aimed at extracting precise subsets of data from Wikidata, addressing the challenges posed by its massive scale. The evaluation framework employed by the paper encompasses key metrics such as execution performance, extraction accuracy, and flexibility in defining subsets, revealing insights into the efficacy of these tools. The evaluation results demonstrate that all four tools exhibit high accuracy in extracting defined items and statements from Wikidata subsets. This accuracy is crucial for ensuring the reliability and validity of extracted data, especially in research and analytical contexts. Moreover, the paper discusses the advantages and disadvantages associated with these subsetting approaches, offering valuable insights for researchers and users alike. It also emphasises the potential of subsetting tools in creating topic-oriented knowledge graphs by populating new knowledge graphs based on extracted subsets, catering to specific thematic or domain-focused research requirements. One notable advantage of this is the ability of subsetting to help circumvent challenges related to size and computational power inherent in working with massive knowledge graphs like Wikidata, thereby enhancing overall efficiency (e.g. average response time) and reducing resource overheads. [6] Despite these advantages, the paper acknowledges certain challenges and limitations associated with Wikidata subsetting. These include difficulties in ensuring uniform coverage of references across all Wikidata subsets and identifying variations between contributor communities, which can impact data quality and reliability. Moreover, securing necessary funds for infrastructure to host a complete copy of Wikidata may pose challenges, making subsetting a more viable option in resource-constrained environments. Furthermore, existing subsetting approaches exhibit limitations such as the need to define the ontological structure and data model of Wikidata, along with the inability to extract contextual metadata like references, which are crucial for comprehensive data analysis and interpretation. [6] Although subsetting reduces the query overhead inherent to large RDF stores, it is not not feasible for Abstract Wikipedia for 2 reasons. First, Abstract Wikipedia requires the entire sum of knowledge stored in Wikidata as contributors write a wide variety of topics. Second, Data on Wikidata is consistently being updated and with the need of Abstract Wikipedia to be accurate reconcilling these updates with the subset would introduce overheads.

8 CONCLUSION

The literature review highlights the importance of structured data and natural language generation techniques in automating content creation in Abstract Wikipedia through CoSMo integration with Wikidata. Challenges persist in achieving consistent content coverage across Wikipedia's different language editions, underscoring the necessity for innovative solutions like Abstract Wikipedia. Utilising heuristics, caching strategies, and optimised SPARQL queries can significantly improve query response times and system performance, enhancing user experiences. Many of these performance improvement strategies can also be combined to achieve even greater efficiency. Strategies such as leveraging heuristics for optimised SPARQL queries and implementing various caching techniques can work synergistically within the same middleware. Combining these approaches allows for a comprehensive optimisation framework to enhance query response times, reduce server load, and ultimately improve user experiences within the Abstract Wikipedia ecosystem.

REFERENCES

[1] Akhtar, U., Sant'Anna, A., Jihn, C.-H., Razzaq, M., Bang, J., and Lee, S. A cache-based method to improve query performance of linked open data cloud. *Computing 102* (07 2020).

[2] Arrieta, K., Fillottrani, P. R., and Keet, C. M. Cosmo: A constructor specification language for abstract wikipedia's content selection process, 2023.

[3] Azad, H. K., and Deepak, A. Query expansion techniques for information retrieval: A survey. *Information Processing amp; Management 56*, 5 (Sept. 2019), 1698–1735.

[4] Daga, E., Panziera, L., and Pedrinaci, C. A basilar approach for building web apis on top of sparql endpoints. In *Proceedings of the Third Workshop on Services and Applications over Linked APIs and Data* (2015), M. Maleshkova, R. Verborgh, and S. Stadtmüller, Eds., vol. 1359, pp. 22–32. co-located with the 12th Extended Semantic Web Conference (ESWC 2015).

[5] Guisado-Gámez, J., Dominguez-Sal, D., and Larriba-Pey, J.-L. Massive query expansion by exploiting graph knowledge bases, 2013.

[6] Hosseini Beghaeiraveri, S. A., Labra Gayo, J., Waagmeester, A., Ammar, A., Gonzalez, C., Slenter, D., Ul-Hasan, S., Willighagen, E., Mcneill, F., and Gray, A. Wikidata subsetting: Approaches, tools, and evaluation. *Semantic Web* (12 2023), 1–27.

[7] Loizou, A., and Groth, P. On the formulation of performant SPARQL queries. *CoRR abs/1304.0567* (2013).

[8] Malyshev, S., Krötzsch, M., González, L., Gonsior, J., and Bielefeldt, A. Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In *The Semantic Web – ISWC 2018* (Cham, 2018), D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl, Eds., Springer International Publishing, pp. 376–394.

[9] Martin, M., Unbehauen, J., and Auer, S. Improving the performance of semantic web applications with sparql query caching. In *The Semantic Web: Research and Applications* (Berlin, Heidelberg, 2010), L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, Eds., Springer Berlin Heidelberg, pp. 304–318.

[10] Niazmand, E. Enhancing query answer completeness with query expansion based on synonym predicates. In *Companion Proceedings of the Web Conference 2022* (New York, NY, USA, 2022), WWW '22, Association for Computing Machinery, p. 354–358.

[11] Papailiou, N., Tsoumakos, D., Karras, P., and Koziris, N. Graph-aware, workload-adaptive sparql query caching. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, Association for Computing Machinery, p. 1777–1792.

[12] REITER, E., and DALE, R. Building applied natural language generation systems. *Natural Language Engineering 3*, 1 (1997), 57–87.

[13] Vrandecic, D. Capturing meaning: Toward an abstract wikipedia. In *International Semantic Web Conference 2018 - Outrageous Ideas Track* (Monterey, CA, 2018).

[14] Vrandečić, D. Collaborating on the sum of all knowledge across languages. *Wikipedia @ 20* (may 22 2019). https://wikipedia20.mitpress.mit.edu/pub/vyf7ksah.

[15] Vrandečić, D. Architecture for a multilingual wikipedia, 2020.

[16] Wikidata. Wikidata SPARQL Query Service - Query Optimization, Accessed 2024. Accessed on: 2024-03-28.