Using Pregel to Create Knowledge Graphs Subsets Described by Non-recursive Shape Expressions

Ángel Iglesias Préstamo(✉) and Jose Emilio Labra Gayo

WESO Lab, University of Oviedo, Oviedo, Spain
angel.iglesias.prestamo@gmail.com

Abstract. Knowledge Graphs have been successfully adopted in recent years, existing general-purpose ones, like Wikidata, as well as domain-specific ones, like UniProt. Their increasing size poses new challenges to their practical usage. As an example, Wikidata has been growing the size of its contents and their data since its inception making it difficult to download and process its data. Although the structure of Wikidata items is flexible, it tends to be heterogeneous: the shape of an entity representing a human is distinct from that of a mountain. Recently, Wikidata adopted Entity Schemas to facilitate the definition of different schemas using Shape Expressions, a language that can be used to describe and validate RDF data. In this paper, we present an approach to obtain subsets of knowledge graphs based on Shape Expressions that use an implementation of the Pregel algorithm implemented in Rust. We have applied our approach to obtain subsets of Wikidata and UniProt and present some of these experiments' results.

Keywords: Knowledge Graphs · Graph algorithms · RDF · Linked Data · RDF Validation · Shape Expressions · Subsets · Pregel

1 Introduction

Knowledge graphs have emerged as powerful tools for representing and organizing vast amounts of information in a structured manner. As their applications continue to expand across various domains, the need for an efficient and scalable processing of these graphs becomes increasingly critical.

Creating subsets of knowledge graphs is a common approach for tackling the challenges posed by their size and complexity. Such subsets are essential not only to reduce computational overhead but also to focus on specific aspects of the data.

In this paper, we explore the synergy between two essential concepts in the field of graph processing: Shape Expressions (ShEx) [12] and the Pregel model [10]. Shape Expressions allow to describe and validate knowledge graphs based on the Resource Description Framework (RDF). These expressions have gained significant adoption in prominent projects like Wikidata. On the other

F. Ortiz-Rodriguez et al. (Eds.): KGSWC 2023, LNCS 14382, pp. 120–134, 2023.
https://doi.org/10.1007/978-3-031-47745-4_10

hand, Pregel is a distributed graph processing model designed for efficiently handling large-scale graphs across multiple machines.

Motivated by the need for handling massive graphs in a scalable manner, we propose the concept of creating subsets of knowledge graphs using Shape Expressions. By selecting relevant portions of the graph, we can focus computational efforts on specific areas of interest, leading to enhanced efficiency and reduced processing times.

Furthermore, we delve into the capabilities of the Pregel algorithm and its potential for distributed graph processing. We emphasize that the scalability of graph computation can be achieved not only by increasing the number of machines but also by optimizing the use of multi-threading solutions to leverage a single machine's capabilities. Hence, our solution aims to distribute the problem across multiple threads of a single-node machine. This is, a multi-threaded Pregel. The idea is not only to provide a solution that can run on any hardware efficiently but also to explore the capabilities of Rust for enabling some performance gains regarding single-node computation. The main contributions of this paper are the following:

1. We present an approach for subset generation of Knowledge Graphs based on Shape Expressions using the Pregel Framework.
2. We have implemented it in Rust.
3. We have applied it to generate subsets of Wikidata and UniProt and presented some optimizations and results.

Section 2 establishes the alternatives and work related to what is presented in the document. Section 3 presents the key concepts required for describing the foundations of the problem to be solved. Section 4 explains the most important algorithms for creating Knowledge Graph subsets. Section 5, the novel approach introduced by this paper is described. Section 6 depicts the experiment for analyzing how the Pregel-based Schema validating algorithm behaves. Section 7 contains the conclusions and future work.

2 Related Work

2.1 Knowledge Graph Descriptions

Several Knowledge Graph descriptions have been proposed, with many outlined in [4,6]. Notably, this paper focuses on Property, RDF, and Wikibase graphs.

Shape Expressions, which are used to create the subsets, were first introduced in 2014. While SHACL (Shapes Constraint Language) is the W3C recommendation[1], the Wikidata community has been using Shape Expressions [14] since 2017. The preference for Shape Expressions arises from their superior adaptability in describing data models when compared to SHACL. A comparison between both can be found in the book [9].

[1] https://www.w3.org/TR/2017/REC-shacl-20170720/.

2.2 Knowledge Graph Subsets

Although it is possible to create subsets of the RDF Knowledge Graph through SPARQL construct queries, there are limitations to this approach. Notably, the lack of support for recursion. While proposals to extend SPARQL with recursion have been made [13], such extensions are not widely supported by existing processors. In light of these limitations, a new method using Shape Expressions for creating Knowledge Graph subsets is described in [4]. `PSchema` follows a similar approach to that presented in [16]. However, `SP-Tree` uses a SPARQL to query the Knowledge Graph, while `PSchema` uses Shape Expressions and Rust. As such, the `PSchema` algorithm is more flexible, leaving room for optimizations.

The creation of Knowledge graph subsets has gained attention, starting from the 12th International SWAT4HCLS Conference[2]. It has since been selected as a topic of interest in the Elixir Europe Biohackathon 2020[3] and the SWAT4HCLS 2021 Hackathon, which resulted in a preprint collecting various approaches [7].

It has been discussed that the Wikidata Knowledge Graph is not feasible to be processed in a single domestic computer using the existing techniques. To address this issue, a novel method to split the Wikidata graph into smaller subsets using Shape Expressions was introduced in [4].

A comparison between several approaches and tools for creating Wikidata Knowledge graph subsets has been discussed [2], where they evaluated the performance of different approaches and tools. Their methodology for measuring performance and conducting experiments served as the primary inspiration for designing the experiments in Sect. 6.

3 Background

3.1 Knowledge Graphs

Definition 1 (Knowledge Graph [4,6]). *A Knowledge Graph is a graph-structured data model that captures knowledge in a specific domain, having nodes that represent entities and edges modeling relationships between those.*

Definition 2 is a general and open description of a Knowledge Graph. There are several data models for representing Knowledge Graphs, including Directed edge-labeled and Property Graphs [6], to name a few. In this paper, we will focus on RDF-based Knowledge Graphs, a standardized data model based on Directed edge-labeled graphs [6].

RDF-Based Knowledge Graphs. The Resource Description Framework (RDF) is a standard model for data interchange on the Web. It is a W3C Recommendation for representing information based on a directed edge-labeled graph,

[2] https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas/Subsetting.
[3] https://github.com/elixir-europe/BioHackathon-projects-2020/tree/master/projects/35.

where labels are the resource identifiers. The idea behind the RDF model is to make statements about things in the form of subject-predicate-object triples. The subject denotes the resource itself, while the predicate expresses traits or aspects of it and expresses a relationship between the subject and the object, another resource. This linking system forms a graph data structure, which is the core of the RDF model. If the dataset represents Knowledge of a specific domain, the Graph will be an RDF-based Knowledge Graph. There are several serialization formats for RDF-based Knowledge Graphs, including Turtle, N-Triples, and JSON-LD. Its formal definition is as follows:

Definition 2 (RDF-based Knowledge Graph [4]). *Given a set of IRIs $\mathcal{I}$, a set of blank nodes $\mathcal{B}$, and a set of literals l. An RDF-based Knowledge Graph is defined as a triple-based graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ where $\mathcal{S} \subseteq \mathcal{I} \cup \mathcal{B}$, $\mathcal{P} \subseteq \mathcal{I}$, $\mathcal{O} \subseteq \mathcal{I} \cup \mathcal{B} \cup l$, and $\rho \subseteq \mathcal{S} \times \mathcal{P} \times \mathcal{O}$.*

Example 1 (RDF-based Knowledge Graph of Alan Turing).[4] Alan Turing (23 June 1912 – 7 June 1954) was employed by the government of the United Kingdom in the course of WWII. During that time he developed the computer for deciphering Enigma-machine-encrypted secret messages, namely, the Bombe machine. Additional information about relevant places where he lived is also annotated, including his birthplace, and the place where he died.

$\mathcal{I}$ = { *alanTuring*, *wilmslow*, *GCHQ*, *unitedKingdom*, *warringtonLodge*, *bombe*
town, *computer*, *dateOfBirth*, *placeOfBirth*, *employer*, *placeOfDeath*,
country, *manufacturer*, *instanceOf* }
$\mathcal{B}$ = { $\emptyset$ }
l = { 23 June 1912 }
ρ = { (*alanTuring*, *instanceOf*, *Human*),
(*alanTuring*, *dateOfBirth*, 23 June 1912),
(*alanTuring*, *placeOfBirth*, *warringtonLodge*),
(*alanTuring*, *placeOfDeath*, *wilmslow*),
(*alanTuring*, *employer*, *GCHQ*),
(*bombe*, *discoverer*, *alanTuring*),
(*bombe*, *manufacturer*, *GCHQ*),
(*bombe*, *instanceOf*, *computer*),
(*wilmslow*, *country*, *unitedKingdom*)
(*wilmslow*, *instanceOf*, *town*)
(*warringtonLodge*, *country*, *unitedKingdom*) }

URIs in Wikidata follow a linked-data pattern called *opaque URIs* representing them as unique sequences of characters that are language-independent. As an example, Alan Turing's identifier is serialized as `Q7251`. Furthermore, within Wikidata, there is a designated property known as `instanceOf` that serves to describe the type of entity it is associated with, which resembles the `rdf:type` constraint. This can be employed to perform an early evaluation of the nodes.

[4] https://rdfshape.weso.es/link/16902825958.

3.2 ShEx

Shape Expressions (ShEx) were designed as a high-level, domain-specific language for describing RDF graph structures. The syntax of ShEx is inspired by Turtle and SPARQL, while the semantics are motivated by RelaxNG and XML Schema. In this manner, Shape Expressions specify the requirements that RDF data graphs must fulfill to be considered conformant, they allow systems to establish contracts for sharing information; through a common schema, systems agree that a certain resource should be part of the graph. This pattern behaves similarly to interfaces in the object-oriented paradigm. Shapes can be specified using a JSON-LD syntax or a human-friendly concise one called ShExC.

Example 2. The schema below describes the `Person` Shape Expression, which is used to validate the RDF-based Knowledge Graph of Alan Turing (see Example 1). Recall, `Person` is just the label of the resource and does not relate to the Human entity. The ShExC-serialized schema can be found in RDFShape.

$$
\begin{aligned}
\mathcal{L} &= \{\ \mathit{Person}, \mathit{Place}, \mathit{Country}, \mathit{Organization}, \mathit{Date}\ \} \\
\delta(\mathit{Person}) &= \{\ \_ \xrightarrow{\mathit{placeOfBirth}} \text{@Place}, \\
&\qquad \_ \xrightarrow{\mathit{dateOfBirth}} \text{@Date}, \\
&\qquad \_ \xrightarrow{\mathit{employer}} \text{@Organization}\ \} \\
\delta(\mathit{Place}) &= \{\ \_ \xrightarrow{\mathit{country}} \text{@Country}\ \} \\
\delta(\mathit{Country}) &= \{\ \} \\
\delta(\mathit{Organization}) &= \{\ \} \\
\delta(\mathit{Date}) &\in \ \texttt{xsd:date}
\end{aligned}
$$

3.3 Pregel

Pregel (*Parallel, Graph, and Google*) is a data flow paradigm created by Google to handle large-scale graphs. Even if the original instance remains proprietary at Google, it was adopted by many graph-processing systems, including Apache Spark. For a better understanding of Pregel, the idea is to *think like a vertex* [11]; this is, the state of a given node will only depend on that of its neighbors, the nodes linked to it by an outgoing edge (see Definition 4). Hence, by *thinking like a vertex*, the problem is divided into smaller ones. Instead of dealing with huge graphs, smaller ones are processed: a vertex and its neighbors.

The series of steps that Pregel follows to process a graph are depicted in Fig. 1. The execution starts by sending the initial messages to the vertices at iteration 0. Then, the first – actual – *superstep* begins. In our current implementation, this loop will last until the current iteration is greater than the threshold set at the creation of the Pregel instance. At each iteration, the vertices will send messages to their neighbors, provided the given direction, and subsequently, they may receive messages sent from other nodes. Moving forward, an aggregation function is applied, and the vertices are updated accordingly. Finally, the iteration counter is incremented and the next iteration starts.

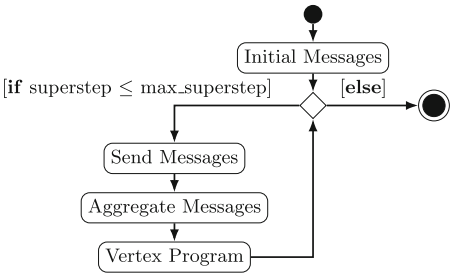

Fig. 1. Pregel model as implemented in pregel-rs

4 Subsetting Approaches

4.1 Knowledge Graph Subsets, a Formal Definition

Definition 3 (RDF-based Knowledge Graph subset [4]). *Given a Knowledge Graph* $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$*, as defined in Definition 2, a RDF sub-graph is defined as* $\mathcal{G}' = \langle \mathcal{S}', \mathcal{P}', \mathcal{O}', \rho' \rangle$ *such that:* $\mathcal{S}' \subseteq \mathcal{S}$*,* $\mathcal{P}' \subseteq \mathcal{P}$*,* $\mathcal{O}' \subseteq \mathcal{O}$ *and* $\rho' \subseteq \rho$*.*

Example 3 (Example of an RDF-based Knowledge Graph subset). Given the RDF-based Knowledge Graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ from Example 1, the subset $\mathcal{G}'$ that only contains information about Alan's birthplace is as follows:

$$
\begin{aligned}
\mathcal{I}' &= \{\ \mathit{alanTuring}, \mathit{warringtonLodge}, \mathit{dateOfBirth}, \mathit{placeOfBirth}\ \} \\
\mathcal{B}' &= \{\ \emptyset\ \} \\
l' &= \{\ 23\ \text{June}\ 1912\ \} \\
\rho' &= \{\ (\mathit{alanTuring}, \mathit{dateOfBirth}, 23\ \text{June}\ 1912), \\
&\qquad (\mathit{alanTuring}, \mathit{placeOfBirth}, \mathit{warringtonLodge})\ \}
\end{aligned}
$$

4.2 ShEx-Based Matching Generated Subsets

ShEx-based matching comprises using a ShEx schema *se* as input, including any nodes whose neighborhood matches any of the shapes from *se* in the produced subset [4]. This approach is used by the `PSchema` algorithm.

Definition 4 (Neighborhood of a node in a Knowledge graph). *The neighbors of an item* $s \in \mathcal{S}$ *in a RDF-based Knowledge graph* $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ *are defined as* $neighbors(s) = \{(s, p, o) : \exists v \in \mathcal{S}$ *such that* $\rho(v) = (p, o)\}$*.*

Example 4 (Neighborhood of Alan Turing (Q7251)). Given the RDF-based Knowledge graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ from Example 1, the neighborhood of Alan Turing (Q7251) $\in \mathcal{S}$ is defined as follows:

$$
\begin{aligned}
neighbors(\mathit{alanTuring}) = \{\ & (\mathit{alanTuring}, \mathit{instanceOf}, \mathit{Human}), \\
& (\mathit{alanTuring}, \mathit{dateOfBirth}, 23\ \text{June}\ 1912), \\
& (\mathit{alanTuring}, \mathit{placeOfBirth}, \mathit{warringtonLodge}), \\
& (\mathit{alanTuring}, \mathit{placeOfDeath}, \mathit{wilmslow}), \\
& (\mathit{alanTuring}, \mathit{employer}, \mathit{GCHQ})\ \}
\end{aligned}
$$

Example 5 (Example of a ShEx-based matching subgraph). Given the RDF-based Knowledge Graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ from Example 1 and the ShEx schema *se* defined in Example 2, the *ShEx-based matching subgraph* of $\mathcal{G}$ from *se* is the RDF-based Knowledge graph $\mathcal{G}'$, which defined as follows:

$$
\begin{aligned}
\mathcal{T}' &= \{\ alanTuring,\ wilmslow,\ GCHQ,\ unitedKingdom,\ warringtonLodge,\\
&\quad\ dateOfBirth,\ placeOfBirth,\ employer,\ country\ \}\\
\mathcal{B}' &= \{\ \emptyset\ \}\\
\mathcal{l}' &= \{\ 23\ \text{June}\ 1912\ \}\\
\rho &= \{\ (alanTuring,\ dateOfBirth,\ 23\ \text{June}\ 1912),\\
&\quad\ (alanTuring,\ placeOfBirth,\ warringtonLodge),\\
&\quad\ (alanTuring,\ employer,\ GCHQ),\\
&\quad\ (wilmslow,\ country,\ unitedKingdom)\\
&\quad\ (warringtonLodge,\ country,\ unitedKingdom)\ \}
\end{aligned}
$$

5 Pregel-Based Schema Validating Algorithm

In this section, both the support data structure and the subsetting algorithm are described, including the different steps followed in the Pregel implementation.

5.1 Shape Expression Tree

The Shape Expression tree is a hierarchical data structure representing Shapes in a tree format. Each node in the tree corresponds to a Shape Expression, with the root node being the one subject of study. Nodes can reference other Shape Expressions, which become its children in the tree.

Definition 5 (Shape Expression tree). *Given a Shape Expression se, the Shape Expression tree $\mathcal{T}$ is defined as follows:*

- *If se does not reference any other* `Shape`*, then $\mathcal{T}$ is a leaf node.*
- *If se references other* `Shapes`*, then $\mathcal{T}$ is an internal node, and its children are the* `Shapes` *referenced by se. Which will be the root nodes of their respective Shape Expression trees.*

Example 6. Given the Shape Expression *se* defined in Example 2, the Shape Expression tree $\mathcal{T}$ obtained from *se* was created using the RDFShape and is depicted in Fig. 2. `Person` is the root node of $\mathcal{T}$, a non-terminal `Shape` that references `Organization`, `Date`, and `Place`. Thus, the children of the root are the `Shapes` referenced by `Person`, which are the root nodes of their respective Shape Expression trees. In the case of the first child, `Organization` is a terminal `Shape`, and thus, it is a leaf node. The same applies to `Date`. However, `Place` is a non-terminal `Shape`, and thus, it is an internal node. Its children are the `Shapes` referenced by it. This representation is recursive, and thus, the `Shapes` referenced by `Place` are the root nodes of their respective ShEx trees.

The currently supported ShEx language does not support recursion; however, it is planned to implement a solution based on the idea of *unfolding* the recursive schema.

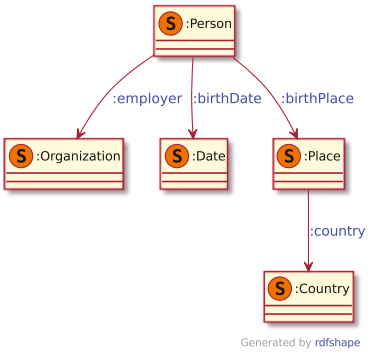

Fig. 2. Example of a Shape Expression tree for the `Person` Shape Expression

5.2 Subsetting Algorithm Using Pregel and ShEx

Algorithm 1: The PSchema algorithm as implemented in Rust

Input parameters:
g: Graph[$\mathcal{V}$, $\mathcal{E}$]
l: $\mathcal{L}$

Output:
sub: Graph[$\mathcal{V}$, $\mathcal{E}$]

`maxIters` = see Lemma 1
`initialMsgs` = `None`
return *Pregel(g,maxIters,initialMsgs,sendMsg,aggMsgs,vProg)*

def `sendMsg`(l: $\mathcal{L}$, g: Graph[$\mathcal{V}$, $\mathcal{E}$]) = *msgs* where **foreach** $l \in \mathcal{L}$

$$msgs.insert\left(\begin{cases} \text{validate}(l,\ g) \text{ if } l = \textit{TripleConstraint} & \textit{see Algorithm 2} \\ \text{validate}(l,\ g) \text{ if } l = \textit{ShapeReference} & \textit{see Algorithm 3} \\ \text{validate}(l,\ g) \text{ if } l = \textit{ShapeAnd} & \textit{see Algorithm 4} \\ \text{validate}(l,\ g) \text{ if } l = \textit{ShapeOr} & \textit{see Algorithm 5} \\ \text{validate}(l,\ g) \text{ if } l = \textit{Cardinality} & \textit{see Algorithm 6} \\ \texttt{None} \quad \text{otherwise} & \end{cases}\right)$$

def `aggMsgs`($msgs$: $\mathcal{M}$) = *msgs* where

$$msgs.insert\left(\begin{cases} msg & \text{if } msg \neq \texttt{None} \\ \emptyset & \text{otherwise} \end{cases}\right)$$

def `vProg`(l: $\mathcal{L}$, g: Graph[$\mathcal{V}$, $\mathcal{E}$], $msgs$: $\mathcal{M}$) = *labels*.concatenate($msgs$)

`PSchema` is a Pregel-based algorithm that creates subsets of RDF-based Knowledge Graphs using Shape Expressions. The algorithm's core idea is to validate the nodes of the Shape Expression tree $\mathcal{T}$ in a bottom-up manner, proceeding from the leaves to the root. The validation is performed in a *reverse level-order traversal* of the tree. The algorithm comprises two main phases: initialization and validation. During the initialization phase, the initial messages are generated and sent to the vertices, while also setting up the *superstep* counter and threshold. This phase establishes the baseline for subsequent steps. In the validation phase, referred to as the *local computation*, the `Shapes` of the tree $\mathcal{T}$ are validated. The

vertices are updated based on the messages they receive from their neighbors. The aforementioned Pregel fork is publicly accessible on Github[5]. For a formal description of the procedure, refer to Algorithm 1.

Algorithm 2: Validate method for the `TripleConstraint` Shape

Input parameters:
- l: $\mathcal{L}$
- (_, p, o): (_, $p \in \mathcal{P}$, $o \in \mathcal{O}$)

Output:
- msg: $\mathcal{M}$

match $l.object$
- **case** *Value(v)*
  - **if** $p == l.predicate \wedge o == v$ **then**
    - **return** l
- **case** *Any*
  - **if** $p == l.predicate$ **then**
    - **return** l

A formal description of the *validating* algorithms for each of the currently implemented Shapes is provided. Note that the input parameters are simplified, as in the actual implementation of the algorithm it can access the whole Graph. Having said that, the first method that is described is the validating algorithm for the `TripleConstraint` Shape, as seen in Algorithm 2. This Shape corresponds to the most basic representation of a Node Constraint. In that manner, it is verified if a node satisfies the predicate and object constraints. In other words, if it exists an outgoing edge from a certain node to another determined by the Shape that is being validated currently. Note that the Object may be an actual value or any.

Algorithm 3: Validate method for the `ShapeReference` Shape

Input parameters:
- l: $\mathcal{L}$
- (_, p, o): (_, $p \in \mathcal{P}$, $o \in \mathcal{O}$)

Output:
- msg: $\mathcal{M}$

if $p == l.predicate \wedge o == l.reference.object$ **then**
- **return** l

[5] https://github.com/weso/pregel-rs.

The `ShapeReference` is in charge of evaluating the cases in which the object of a Triple Constraint is an IRI; that is, a reference to another Shape. Even if the algorithm behaves similarly to the description before, the implementation details vary as the value referenced has to be retrieved. Refer to Algorithm 3.

Algorithm 4: Validate method for the `ShapeAnd` Shape

Input parameters:
- l: $\mathcal{L}$
- (_, p, o): (_, $p \in \mathcal{P}$, $o \in \mathcal{O}$)

Output:
- msg: $\mathcal{M}$

$ans \leftarrow$ `true`
forall $l \in l.shapes$ **do**
- $ans \leftarrow ans \wedge$ `validate`(l, g)

if ans **then**
- **return** l

The `ShapeAnd` constraint checks whether *all* the Shapes wrapped by it are valid. Having all the children already evaluated, it is going to be checked if all of them hold for every node in the graph. This is, the `ShapeAnd` acts as a *logical and* for a grouping of Shapes. Refer to Algorithm 4.

Algorithm 5: Validate method for the `ShapeOr` Shape

Input parameters:
- l: $\mathcal{L}$
- (_, p, o): (_, $p \in \mathcal{P}$, $o \in \mathcal{O}$)

Output:
- msg: $\mathcal{M}$

$ans \leftarrow$ `false`
forall $l \in l.shapes$ **do**
- $ans \leftarrow ans \vee$ `validate`(l, g)

if ans **then**
- **return** l

The `ShapeOr` constraint checks whether *any* Shape wrapped is valid. Having all the children already evaluated, it is going to be checked if any of them holds for every node in the graph. This is, the `ShapeOr` acts as a *logical or* for a grouping of Shapes. Refer to Algorithm 5.

Algorithm 6: Validate method for the `Cardinality` Shape

```
Input parameters:
    l: ℒ
    (_, p, o): (_, p ∈ 𝒫, o ∈ 𝒪)
    prevMsg: ℳ
Output:
    msg: ℳ

count ← prevMsg.count()
match l.min
    case Inclusive(min)
        if count ≤ min then
            min ← true
    case Exclusive(min)
        if count < min then
            min ← true

match l.max
    case Inclusive(max)
        if count ≥ max then
            max ← true
    case Exclusive(max)
        if count > max then
            max ← true

if min ∧ max then
    return l
```

The `Cardinality` constraint is in charge of checking whether the Shape referenced is valid a certain number of times in the neighborhood of every node in the graph. That is, the number of times the neighboring nodes are valid for a certain Shape. The concept of inclusivity or exclusivity allows for the range to be closed or open, respectively. Refer to Algorithm 6.

Lemma 1 (Convergence of the PSchema algorithm). *Given a* ***`Shape Expression`*** *tree* $\mathcal{T}$ *and a Knowledge Graph* $\mathcal{G}$*, let* h *denote the height of* $\mathcal{T}$*; then the* ***`PSchema`*** *algorithm is going to converge in* h *supersteps. This is, the algorithm is going to validate all the* ***`Shapes`*** *of* $\mathcal{T}$ *in* h *supersteps.*

Example 7 (Example of the subset generated by the ***`PSchema`*** *algorithm).* Given the RDF-based Knowledge Graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ from Example 1 and the ShEx schema *se* defined in Example 2, the *ShEx-based matching subgraph* of $\mathcal{G}$ from *se* is the RDF-based Knowledge graph $\mathcal{G}'$ from Example 5, which is represented in `Turtle` syntax as follows, refer to RDFShape for more information:

```
PREFIX :        <http://example.org/>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>

:alan       :placeOfBirth     :warrington             ;
            :dateOfBirth      "1912-06-23"^^xsd:date  ;
            :employer         :GCHQ                   .

:warrington :country          :uk                     .

:wilmslow   :country          :uk                     .
```

5.3 Optimizations

Columnar storage [1] is a data warehousing technique used in databases and data processing systems. Unlike traditional row-based storage, where data is stored in rows, columnar storage organizes data in columns, grouping values of the same attribute. Columnar storage enables better data compression, as similar data types are stored together, reducing the storage footprint. This leads to faster data retrieval and reduced I/O operations when querying specific columns. It is advantageous for analytical workloads that involve aggregations or filtering on specific attributes, as they only need to read the relevant columns. It also enhances query performance by leveraging vectorized processing, as modern CPUs can perform operations on entire sets of data (vectors) more efficiently than on individual elements, further improving query speeds.

Caching. Dictionary encoding [15] is a data compression technique where unique values in a column are assigned numerical identifiers (dictionary indices) and stored in a separate data structure. The actual data in the column is replaced with these compact numerical representations. This technique significantly reduces the storage footprint, especially when columns contain repetitive or categorical data with limited distinct values, as predicates in real-life scenarios.

In data processing scenarios with repeated patterns and aggregations, caching with columnar storage and dictionary encoding can lead to performance improvements. The reduced data size allows more data to be cached within the same memory capacity, maximizing cache utilization. Additionally, the focused access ensures that only the necessary data is retrieved, further enhancing cache efficiency. The cache can hold a large amount of relevant data, minimizing the need for costly disk accesses and accelerating query response times, ultimately resulting in a more efficient and responsive data processing system.

6 Experiments and Results

Two different Knowledge Graphs are going to be used to test the algorithm, namely, Uniprot[6] and Wikidata. The former is a database that contains information about proteins [3], while the latter is a general-purpose knowledge base

[6] https://ftp.uniprot.org/pub/databases/uniprot/current_release/rdf/.

having a dump created the 21^{st} August 2017. As the serialization format of *Uniprot* is `RDF/XML`, the `riot` utility from *Apache Jena* is used to convert from `RDF/XML` to `N-Triples`. Refer to the examples in the GitHub repository[7].

As it is seen, the results obtained are similar regarding the size of the subsets. That is, the optimizations have no impact on the *validity* of the tool; this is, the subsets are correct for the `Shape` defined. For this, two Shapes were created during the Japan BioHackathon 2023 [8], namely, `protein` and `subcellular_location`. When comparing the optimized version against its counterpart, the time consumption is reduced by 38% and 35%; while the memory consumption is decreased by 43% and 38%, respectively. Hence, the optimizations are effective both time and memory-wise. The next experiment will focus on how the algorithm behaves when its parameters are modified (Fig. 3).

Shape Expression	Initial triples	Resulting triples	Time (s)	Memory (GB)
protein	7,346,129	226,241	23.35	6.74
subcellular_location	7,346,129	1,084,151	57.56	6.04

(a) Execution of the `PSchema` algorithm with no optimization enabled

Shape Expression	Initial triples	Resulting triples	Time (s)	Memory (GB)
protein	7,346,129	226,241	14.58	3.87
subcellular_location	7,346,129	1,084,151	37.76	3.75

(b) Execution of the `PSchema` algorithm with all the optimizations enabled

Fig. 3. Time and memory consumption to create `Uniprot`'s *subsets*

In the second experiment, the number of `Wikidata` entities, the depth, and the width of the `ShEx` tree were modified. The results are depicted in Fig. 4. It was observed that the execution time followed a linear trend in all scenarios. This indicates that as the number of `Wikidata` entities increased, the execution time increased at a consistent rate. Additionally, the depth and width of the `ShEx` tree influenced the execution time similarly, displaying a linear correlation.

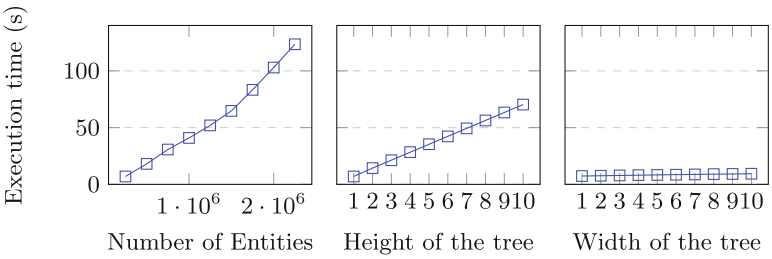

Fig. 4. Time to create the *subsets* of Wikidata with `pschema-rs`

[7] https://github.com/angelip2303/pschema-rs/tree/main/examples/from_uniprot.

7 Conclusions and Future Work

A novel approach for creating subsets of RDF-based Knowledge Graphs using Shape Expressions and the Pregel framework is presented. The `PSchema` algorithm is described, including the different steps followed in the Pregel implementation. Moreover, the support data structure and the optimizations applied are also described. Two Shapes were created during the Japan BioHackathon 2023 [8] for testing the tool and its validity regarding the optimizations applied. The subsets resulting subsets have the same size in both scenarios. When comparing the optimized version against its counterpart, the time consumption is reduced by 38% and 35%; while the memory consumption is decreased by 43% and 38%, respectively. Hence, the optimizations are effective. The next experiment focused on how the algorithm behaves when its parameters are modified. It was observed that the execution time followed a linear trend in all cases.

`PSchema` could be extended to support more complex ShEx features like recursive Shapes, and an early-prune strategy to reduce the cost of the local computation. The algorithm should receive the `ShEx` schema as an input, rather than programmatically creating desired Shape instance. It is planned to give support for WShEx [5], a ShEx-inspired language for describing Wikidata entities, where qualifiers about statements and references can be used for validating purposes. This would allow the algorithm to be used in a wider range of scenarios.

To conclude, `PSchema`, being a Pregel-based Knowledge Graph validating algorithm, allows the processing of large-scale Knowledge Graphs. This is especially relevant in the *Bioinformatics*, where the integration of data from multiple sources is needed. What's more, inference algorithms can be applied to the subsets generated, which is not possible in larger Graphs due to their sizes.

Acknowledgements. This project has received funding from NumFOCUS, a nonprofit organization promoting open-source scientific projects, and has been supported by the ANGLIRU project, funded by the Spanish Agency for Research. The opinions and arguments employed herein do not reflect the official views of these organizations.

References

1. Abadi, D.J., Madden, S.R., Hachem, N.: Column-stores vs. row-stores: how different are they really? In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 967–980. Association for Computing Machinery, New York (2008). https://doi.org/10.1145/1376616.1376712
2. Beghaeiraveri, S.A.H., et al.: Wikidata subsetting: approaches, tools, and evaluation (2023). https://www.semantic-web-journal.net/system/files/swj3491.pdf
3. The UniProt Consortium: UniProt: the Universal Protein Knowledgebase in 2023. Nucleic Acids Res. **51**(D1), D523–D531 (2022). https://doi.org/10.1093/nar/gkac1052
4. Gayo, J.E.L.: Creating knowledge graphs subsets using shape expressions (2021). https://doi.org/10.z8550/ARXIV.2110.11709. https://arxiv.org/abs/2110.11709
5. Gayo, J.E.L.: Wshex: a language to describe and validate wikibase entities (2022). https://arxiv.org/abs/2208.02697

6. Hogan, A., et al.: Knowledge graphs. CoRR abs/2003.02320 (2020). https://arxiv.org/abs/2003.02320
7. Labra-Gayo, J.E., et al.: Knowledge graphs and wikidata subsetting (2021). https://doi.org/10.37044/osf.io/wu9et. http://biohackrxiv.org/wu9et
8. Labra-Gayo, J.E., et al.: RDF Data integration using Shape Expressions (2023). https://biohackrxiv.org/md73k
9. Labra Gayo, J.E., Prud'hommeaux, E., Boneva, I., Kontokostas, D.: Validating RDF Data. No. 1 in Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers LLC (2017). https://doi.org/10.2200/s00786ed1v01y201707wbe016
10. Malewicz, G., et al.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 International Conference on Management of Data, New York, NY, USA, pp. 135–146 (2010). https://doi.org/10.1145/1807167.1807184
11. McCune, R.R., Weninger, T., Madey, G.: Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. ACM Comput. Surv. **48**(2) (2015). https://doi.org/10.1145/2818185
12. Prud'hommeaux, E., Labra Gayo, J.E., Solbrig, H.: Shape expressions: an RDF validation and transformation language. In: Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, pp. 32–40. ACM (2014)
13. Reutter, J.L., Soto, A., Vrgoč, D.: Recursion in SPARQL. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9366, pp. 19–35. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25007-6_2
14. Thornton, K., Solbrig, H., Stupp, G.S., Labra Gayo, J.E., Mietchen, D., Prud'hommeaux, E., Waagmeester, A.: Using shape expressions (ShEx) to share RDF data models and to guide curation with rigorous validation. In: Hitzler, P., et al. (eds.) ESWC 2019. LNCS, vol. 11503, pp. 606–620. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21348-0_39
15. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann, San Francisco (1999)
16. Xu, Q., Wang, X., Li, J., Zhang, Q., Chai, L.: Distributed subgraph matching on big knowledge graphs using pregel. IEEE Access **7**, 116453–116464 (2019). https://doi.org/10.1109/ACCESS.2019.2936465