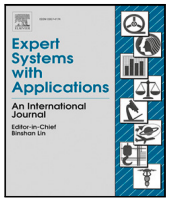
Expert Systems With Applications 258 (2024) 125115

Contents lists available at ScienceDirect

Expert Systems With Applications

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



Querying large-scale knowledge graphs using Qualitative Spatial Reasoning

Matthew Mantle [a,*], Sotirios Batsakis [b,a], Grigoris Antoniou [c]

[a] University of Huddersfield, Huddersfield, UK
[b] Hellenic Mediterranean University, Heraklion, Greece
[c] Leeds Beckett University, Leeds, UK

ARTICLE INFO

Keywords:
Spatial reasoning
RCC
Knowledge graphs
Spatial queries
Distributed computing
GeoSPARQL

ABSTRACT

In this paper we consider how Qualitative Spatial Reasoning (QSR) can be used to answer queries over large-scale knowledge graphs such as YAGO and DBPedia. We describe the challenges associated with spatially querying knowledge graphs such as point based representations, sparsity of qualitative relations, and scale. We address these challenges and present a query engine, Parallel Qualitative Reasoner-Query Engine (ParQR-QE), that uses a novel distributed qualitative spatial reasoning algorithm to provide answers to GeoSPARQL queries. An experimental evaluation using a range of different query types and the YAGO knowledge graph shows the advantages of QSR techniques in comparison to purely quantitative approaches.

1. Introduction

Qualitative Spatial Reasoning (QSR) is an area of artificial intelligence research concerned with reasoning over non-numeric spatial information. Instead of working with quantitative representations — points, linestrings and polygons, QSR focusses on topological relations between spatial entities. For example 'Scotland *borders* England', or 'England *contains* London'. Qualitative Constraint Calculi (QCC) formalise these relations and provide operations for deriving additional spatial facts. QSR is a well established topic of academic interest, constraint calculi have been studied extensively and their properties are well understood (Dylla et al., 2017). However, despite the long standing interest in the area of QSR, there has been relatively little research that considers applications of QSR or considers how exactly qualitative spatial reasoning techniques might be used in practice. One proposed application area for QSR is GIS and querying spatial datasets (Cohn & Renz, 2008). There are a number of reasons for thinking QSR is well suited to this task:

- Spatial information is often only available in qualitative form. This is especially true for data that originates in natural language; humans typically communicate spatial information by describing one location in relation to another, rather than by stating a location's coordinates (Vasardani, Winter, & Richter, 2013). A query engine that only uses quantitative data cannot utilise this information. Conversely, in a qualitative approach, quantitative data can be converted to qualitative relations and combined with qualitative facts to provide a richer dataset. Even when spatial queries are based around quantitative data e.g. k-nearest neighbour or range queries, QSR can have a role. A hybrid approach can be used where quantitative reasoning (by testing geometries) can be used as a starting point for providing answers. QSR can then enhance these results using qualitative relations (Bennett, Cohn, & Isli, 1997).
- Many spatial datasets feature incomplete and/or imprecise records. For example, geospatial datasets may be incomplete due to cloud cover affecting satellite images or legal restrictions affecting drone access (Buckley, 2017). In other cases, privacy concerns e.g. when collecting location data from smart phones or IoT devices, results in users' precise locations being obfuscated (Duckham & Kulik, 2005). Quantitative spatial reasoning requires complete geometric representations in order to make decisions. On the other hand, being able to represent unknown or imprecise spatial information is a key feature of constraint calculi, making QSR well suited to situations where comprehensive spatial information is not available.
- Algorithms that determine topological relations using quantitative representations have runtimes that are dependent on the size of the geometries being tested (Rigaux, Scholl, & Voisard, 2002). Geometries can feature multi-polygons consisting of millions of pairs of coordinates. Therefore, even if accurate, high resolution geometries are available, these can prove challenging to process in a timely manner (Long, Schockaert, & Li, 2016). In contrast, qualitative spatial reasoning is free from constraints related to the size of geometries.

* Corresponding author.
E-mail addresses: m.e.mantle@hud.ac.uk (M. Mantle), sbatsakis@hmu.gr (S. Batsakis), G.Antoniou@leedsbeckett.ac.uk (G. Antoniou).

https://doi.org/10.1016/j.eswa.2024.125115
Received 22 August 2023; Received in revised form 11 July 2024; Accepted 13 August 2024
Available online 21 August 2024

Intuitively, these points suggest QSR can have an effective role to play in spatial query engines and similar arguments have been made previously e.g. Nikolaou and Koubarakis (2013). Therefore it is reasonable to ask why have not we seen many real world applications of QSR? We believe there are two main obstacles to the adoption of QSR techniques. The first is that real world datasets featuring qualitative spatial relations tend to be quite sparse. Even when datasets feature a greater number of relations, these typically have a scale-free structure where a small number of spatial entities have a large number of relations and many entities have few (Nikolaou & Koubarakis, 2014). The second issue is one of scale. QSR reasoners typically maintain a complete list of relations between spatial entities, resulting in n^2 relations (where n is the number of spatial entities). Even comparatively small datasets of 10,000 spatial objects become challenging to reason over (Sioutis, 2014). Of course, the two issues are related. Denser, richer datasets have the greatest utility because they afford a greater number of inferences, but these datasets also pose the greatest challenge for reasoners.

This paper considers how these challenges can be addressed, and goes on to demonstrate the use of QSR to answer spatial queries over large-scale real world knowledge graphs. The use case we focus on concerns the use of Linked Data (Heath & Bizer, 2011) and spatially querying the YAGO[1] knowledge graph. We chose this example because it illustrates many of the obstacles to implementing QSR. Although the YAGO knowledge graph features millions of spatial facts, without enhancement it is not dense enough for QSR to be useful in query answering. Secondly, it is a large scale knowledge graph and therefore presents challenges for reasoning at scale. Furthermore, there is real value in implementing QSR over YAGO. YAGO is a rich source of knowledge about real world entities such as movies, people and organisations. Integrating this knowledge with qualitative reasoning over spatial properties allows us to execute queries that are not possible using existing approaches. Our basic approach is to enhance the knowledge graph with additional spatial information, and then use parallel distributed computing to deal with the issue of scale. We make the following contributions:

- Practical demonstration of techniques to develop an enhanced large-scale knowledge graph that features both extensive qualitative relations, and high resolution geometries.
- The design and implementation of ParQR-QE (Parallel Qualitative Reasoner - Query Engine), a distributed query answering system that can reason over large-scale knowledge graphs to answer GeoSPARQL queries. To our knowledge, this is the first time QSR techniques have been used at scale as part of a query answering system.

The remainder of this paper is organised as follows. Section 2 provides an overview of qualitative spatial reasoning and the fundamental path consistency algorithm that is used to reason over qualitative spatial data. Section 3 describes the challenges of implementing spatial queries over large-scale knowledge graphs. Section 4 describes the approach taken in preparing the YAGO dataset so it can support spatial querying. This involves the use of Linked Data to integrate YAGO with other sources of geographic data. Section 5 provides an overview of ParQR-QE. Section 6 describes the backward chaining algorithm that forms the fundamental reasoning mechanism of ParQR-QE. Section 7 presents an evaluation of ParQR-QE using a variety of queries, a comparison to a query answering system that uses a purely quantitative reasoning approach, and a comparison to other question answering systems. Section 8 discusses related work. Finally, Section 9 provides a conclusion and considers future work.

[1] https://yago-knowledge.org/

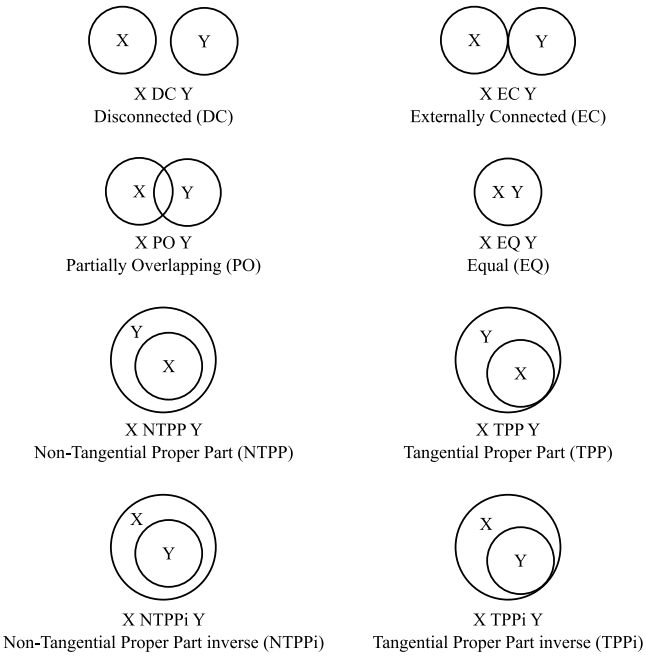


Fig. 1. The basic relations of RCC-8.

2. Preliminaries

2.1. Qualitative spatial reasoning

Qualitative spatial constraint calculi define relations between spatial objects and a means to reason about these relations. We focus on the Region Connection Calculus (RCC) (Randell, Cui, & Cohn, 1992), and specifically the RCC-8 variant. RCC-8 describes eight possible topological relations between regions. These are shown in Fig. 1 and are known as the basic relations of the calculus. The relations are jointly exhaustive and pairwise disjoint (JEPD) i.e. the relation between any two regions must be one of these eight, and only one relation can exist between two regions. The set of basic relations is denoted with the symbol $\mathcal{B}$. Each basic relation has an inverse e.g. the inverse of TPP is $TPPi$. If the exact relation between two regions is not known, a disjunction of basic relations is used to specify this indefinite knowledge e.g. $x\{PO, EC\}y$ - region x either *overlaps* region y or is *externally connected* to it. Often no information is known about how two regions are related in which case the relation could be any one of the eight basic relations. This is referred to as the universal relation.

Collectively, a group of regions and the relations defined between these regions, form a qualitative constraint network (QCN), specifically an RCC-8 network. This is a directed graph (V, C), where V is the set of regions and C is a mapping between a pair of regions and a relation r, where $r \subseteq \mathcal{B}$. Furthermore, for all v in V, $C(v_i, v_i) = EQ$, the identity relation, and the inverse of $C(v_i, v_j)$ is $C(v_j, v_i)$.

Fig. 2 show an example of an RCC-8 network. Note that for clarity inverse relations are not shown, neither are self relations.

The basic mechanism for reasoning over a QCN involves the weak composition operation ($\diamond$) which considers how the relations $C(v_i, v_j)$ and $C(v_j, v_k)$ constrain the relation $C(v_i, v_k)$. For a specific example, and referring to Fig. 2, what do the relations $u\{NTPP\}z$ and $z\{EC\}y$ tell us about the possible relation between regions u and y? If u is a *non-tangential proper part* of z, and z is *externally connected* to y, the only possible relation that can hold between u and y is DC, they are disconnected. A composition table, a $|B| \cdot |B|$ matrix showing all compositions between basic relations, can be used to look up these inferred relations. Part of the composition table for RCC-8 can be seen

Table 1
Part of the Composition Table for RCC8.

⋄	EC	PO	TPP	NTPP
EC	{DC, EC, PO, TPP, TPPi, EQ}	{DC, EC, PO, TPP, NTPP}	{EC, PO, TPP, NTPP}	{PO, TPP, NTPP}
TPP	{DC, EC}	{DC, EC, PO, TPP, NTPP}	{TPP, NTPP}	{NTPP}
NTPP	{DC}	{DC, EC, PO, TPP, NTPP}	{NTPP}	{NTPP}
PO	{DC, EC, PO, TPPi, NTPPi}	*	{PO, TPP, NTPP}	{PO, TPP, NTPP}

* denotes the universal relation.

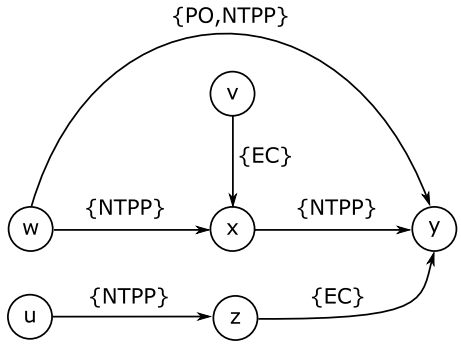


Fig. 2. Example RCC-8 Network.

in Table 1. The full table can be found in Renz and Nebel (1999). Note that the result of composition could be a disjunctive relation e.g. if the relation between u and z had been TPP, then $u\{TPP\}z \diamond z\{EC\}y \rightarrow u\{EC, DC\}y$. Composition can also be applied to disjunctive relations by taking the union of the composition of each basic relation in $C(v_i, v_j)$ with each basic relation in $C(v_j, v_k)$.

The weak composition operation restricts the possible relations that can exist between a pair of regions. However, the RCC-8 network may already feature a relation for this pair of nodes. For example, in Fig. 2, before any reasoning has taken place, the relation between the regions w and y is $\{PO, NTPP\}$. Applying composition for $w\{NTTP\}x$ and $x\{NTPP\}y$ results in the relation $NTPP$. We can then update the relation between w and y by taking the intersection of the initial relation and the result of composition, resulting in $w\{NTPP\}y$. If this intersection results in an empty set this indicates an inconsistency in the network. These two operations, using weak composition to derive new relations between regions, and then taking the intersection to check consistency, forms the basis for reasoning and can be summarised as:-

$$\forall v_i, v_j, v_k \in V, C(v_i, v_j) \cap (C(v_i, v_k) \diamond C(v_k, v_j)) \rightarrow C(v_i, v_j)$$

The operations execute iteratively until a fixed point is reached, at which point we describe the RCC-8 network as being algebraically closed or $\diamond$-consistent. Determining $\diamond$-consistency for an RCC-8 network is a type of constraint satisfaction problem. The algorithm, that checks three node subsets of the network, can be considered a form of path consistency.

In the general case this path consistency algorithm is an approximation i.e. it does not guarantee to discover all inconsistencies. Determining consistency for RCC-8 networks is known to be $\mathcal{NP}$-hard (Renz & Nebel, 1998). However, there are tractable subsets of RCC-8 for which path consistency can effectively decide consistency for a network. For example, the $\widehat{H}_8$ maximal subset (Renz & Nebel, 1999) is of practical use as it contains 148 relations including all of the basic relations, the more general *contains* ($\{TPP, NTPP\}$) relation, and the universal relation.

Although the focus in much of the literature is on using path consistency to determine consistency, the execution of the algorithm also has the benefit of pruning relations between regions so that spatial queries can be answered. For example, before reasoning, if we asked the question does region y contain region w? We could not be sure. After executing path consistency we can definitively answer 'yes'.

2.2. Knowledge graphs, RDF, SPARQL and GeoSPARQL

Knowledge graphs are datasets that have a graph structure where nodes in the graph represent entities and edges between nodes represent relations between entities. In comparison to the relation or NoSQL model, the graph structure affords greater flexibility as entities can be linked via many different relations, free from the constraints of a rigid schema or hierarchical structure (Hogan et al., 2021). See Fig. 3 for an example of a knowledge graph. Prominent knowledge graphs such as YAGO, DBPedia and Wikidata use the Resource Description Framework (RDF) data model for implementation. In RDF, knowledge graphs are constructed using *triples* that describe a pair of nodes and the relation between them. For example, `yago:Belgium rdfs:type schema:country`. Querying RDF knowledge graphs is accomplished using the SPARQL language which specifies a structure or basic graph pattern to search for within the target knowledge graph. It is also possible to add a spatial element to queries using GeoSPARQL, an Open Geospatial Consortium (OGC) standard that extends SPARQL to support querying for topological relations such as *contains*, *touches* and *overlaps* (OGC, 2024). For example, Query C1 shows a GeoSPARQL query that finds all the museums in Belgium.

Query C1

```
SELECT ?museum
  WHERE {
  ?museum rdf:type schema:Museum.
  ?museum geo:sfWithin yago:Belgium.
}
```

SPARQL queries are composed of a number of triple patterns e.g. `?museum rdf:type schema:Museum.` is a triple pattern. Variables, prefixed with a question mark, are substituted for entities from the knowledge graph at query time to generate the results. The second query triple demonstrates GeoSPARQL and the *geo:sfWithin* property. This is executed by testing whether the entities of type museum are spatially within the boundary of Belgium. GeoSPARQL also supports querying using a user specified geometry for example Query R1.

Query R1

```
SELECT ?museum
WHERE{
  ?museum rdf:type schema:Museum.
  FILTER(geof:sfWithin(?museum,
    "POLYGON((4.321423...))"^^geo:wktLiteral))
}
```

Here the FILTER clause restricts results to museums that can be found within the given polygon.

3. Spatially querying knowledge graphs

Before presenting our implementation of QSR for querying large-scale knowledge graphs, it is first necessary to describe how spatial information is typically represented in many knowledge graphs and the challenges these representations present. We used YAGO as our example knowledge graph. Specifically, we used the English Wikipedia version of YAGO 4 which features over 200 million RDF triples. Although the following discussion focusses on YAGO, it is important

to note that the issues we raise here are applicable to many other knowledge graphs e.g. DBpedia.

YAGO features significant amounts of quantitative spatial data where locations of places are represented as points, using the *schema:geo* predicate. For example, the location of Belgium is represented using its centroid as `yago:Belgium schema:geo geo:50.641,4.668`. Similarly, the location of the Hergé Museum is represented as `yago:Musée_Hergé schema:geo geo:50.6678,4.6116`. YAGO also contains qualitative spatial information where some locations have a *schema:containedInPlace* property e.g. `yago:Wellington_Museum,_Waterloo schema:containedInPlace yago:Waterloo,_Belgium`. Using spatial predicates and GeoSPARQL it should be possible to spatially query the YAGO knowledge graph in a variety of ways:

- Containment queries e.g. Query C1 which finds museums in Belgium.
- Adjacency queries e.g. Query A1 which finds Argentinian provinces that border Chile.
- Spatial joins e.g. Query J1 which finds Oscar winning actors and the US states they were born in.
- Range queries e.g. Query R1 which finds museums within a user specified polygon.

We have selected these queries because they demonstrate the use of different RCC-8 relations, spatial joins introduce the challenge of querying using multiple spatial entities, and range queries allow us to investigate the potential of QSR in hybrid reasoning that uses both quantitative and qualitative data.

Query A1

```
SELECT ?province
  WHERE {
  ?province rdf:type yago:Provinces_of_Argentina.
  ?province geo:sfTouches yago:Chile.
}
```

Query J1

```
SELECT ?actor ?birthplace ?state
  WHERE {
  ?actor schema:Award
              yago:Academy_Award_for_Best_Actor.
  ?actor schema:birthPlace ?birthplace.
  ?state rdf:type yago:U.S._state.
  ?birthplace geo:sfWithin ?state.
}
```

A query such as Query C1 could be executed quantitatively by looking up the *schema:geo* properties, and qualitatively by reasoning using *schema:containedInPlace* predicates. However, such a query would fail to return useful results for a number of reasons. Attempting to answer Query C1 using quantitative reasoning would fail due to Belgium being represented as a simple point. For smaller spatial features e.g. museums or railway stations points can suffice, but for larger geographical features that might form the basis of a containment query, a simple centroid is of limited value.

Qualitative reasoning is also going to fail to return comprehensive results due to the sparsity of qualitative relations in the knowledge graph. Even though YAGO features over a million qualitative spatial triples, these are not comprehensive. For example, *yago:Waterloo,_Belgium* has a *containedInPlace* property of *yago:Walloon_Brabant* (the province it is located within), but *yago:Walloon_Brabant* does not have a *containedInPlace* predicate. Consequently, Query C1 would fail to identify *yago:Wellington_Museum,_Waterloo* if executed qualitatively.

Considering a range query such as Query R1 raises further issues. In theory, this query could be answered using a hybrid approach. Quantitative reasoning could be used to identify places within the query polygon. Qualitative reasoning could then be used to find museums located within these places. Again, the point based representation becomes an obstacle. The polygon defined in Query R1 covers a small part of Belgium. However, the centroid for Belgium is within this polygon. Therefore, all museums with a *schema:containedInPlace* relation with Belgium would be returned as results even though only some of them are actually within the query polygon.

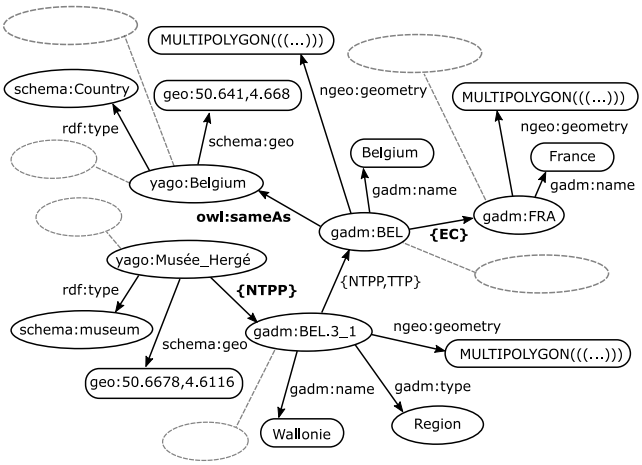


Fig. 3. Enhanced knowledge graph.

4. Generating an enhanced knowledge graph

To address the issues described in Section 3 we took two actions. First, we linked YAGO resources to complex multi-polygon geometries. Second, we computed additional qualitative relations between spatial entities.

A number of open datasets such as OpenStreetMap[2] (OSM) and Database of Global Administrative Areas (GADM)[3] provide geometries in the form of polygons/multi-polygons for places found in the YAGO dataset. Using the principles of Linked Data, a query such as Query C1 can be executed by following *owl:sameAs* links between places in YAGO and a linked polygon based geometry. For this project we used GADM which describes the boundaries of countries and their administrative areas. GADM is comprehensive, it provides complete coverage of the globe, regions are arranged hierarchically (which lends itself to easy mapping to RCC-8 relations), and the geometries are high resolution and consistent i.e. we do not get bordering regions with geometries that overlap because of differences in granularity. We used the first two layers of GADM, layer 0 (countries) and layer 1 (first level administrative areas).

Then, using both the GADM geometries and the point based representations in YAGO, we pre-computed a number of additional RCC-8 relations. Combined with the existing qualitative relations already present in YAGO and GADM this provided a richer RCC-8 network capable of answering queries qualitatively.

Fig. 3 shows a subset of the resulting enhanced knowledge graph after these two tasks have been executed. The relations displayed in bold are newly added relations. The following describes exactly how these two tasks were accomplished.

[2] https://www.openstreetmap.org/
[3] https://gadm.org/

Table 2
Results of linking GADM regions to YAGO.

	Layer 0 (Countries)	Layer 1 (Admin Areas)
GADM Regions	255	3609
Matches Identified	200	2285
Correct Matches	198	2271
Precision	99%	99.38%
Recall	77.64%	62.92%

4.1. Linking GADM regions to YAGO

Identifying equivalent spatial entities in separate datasets has been done previously in a number of projects such as LinkedGeoData (Stadler, Lehmann, Höffner, & Auer, 2012). We used similar techniques, with our exact approach using the following process:

- Using *rdf:type* information, relevant points in the YAGO knowledge graph were obtained e.g. places of the type *schema:Country* or places of the type *schema:AdministrativeArea*. For administrative areas we were also able to use additional, more specific type information e.g. whether a region is province, canton, oblast etc. as this information was often present in both YAGO and GADM.
- By comparing the YAGO point based location to the MBB (Minimal Bounding Box) of GADM regions, potential matches were identified.
- The text similarity of the candidate YAGO resource's label was compared to the GADM region name.

For example, if a GADM country had an MBB that contained a YAGO place of type *schema:Country*, and the GADM *gadm:name* property had a high text similarity to the YAGO place's label, it was assumed to be a match. The Jaro–Winkler distance (Winkler, 1990) was used to obtain the text similarity score. Table 2 shows the results for the linking of resources. In order to calculate precision and recall, we assumed that matches with a similarity score ≥98% were accurate, we then manually checked the remaining matches. Table 2 shows that we were able to match with a high level of precision, but without complete coverage of GADM regions, especially the administrative areas. The recall is low for a number of reasons. In some cases there were differences in how regions are classified in GADM and YAGO. However, the biggest reason was simply a lack of equivalent resources in YAGO for GADM regions. For example, Australian states and territories are not represented in YAGO.

4.2. Computing additional qualitative relations

In computing additional RCC-8 relations, we were looking to strike a balance between generating enough relations so that useful reasoning could take place, but at the same time limit the scale of this task. Furthermore, our focus was on relations that would help in answering the types of queries that we described in Section 3. The following relations were computed:

EC relations between GADM regions within layers. Computing for individual layers made the computation feasible, and was sufficient to allow for effective reasoning to follow. For example, reasoning can infer that the Chubut province of Argentina borders Chile using EC relations between countries/regions within a layer and the $\{TPP, NTPP\}$ relations between regions and their parents. Computing the EC relations was handled by Apache Sedona,[4] a library for processing large-scale spatial data, that is built on top of the Apache Spark[5] distributed programming framework. These were computed using standard library functions, the details of which we do not provide here.

$NTPP$ Relations between YAGO places and GADM regions. It was not necessary to find all regions that contained a given YAGO place, only the 'highest layer' region. At query time, reasoning can be used to infer relations between a YAGO place and parent regions. For example, referring to Fig. 3 we computed that Musée Hergé has an $NTPP$ relation with Walloonie. Reasoning executed for Query C1 will return Musée Hergé as a result because Walloonie has an existing $\{NTPP, TPP\}$ with Belgium. Computing these relations equated to executing point in polygon tests for each relevant YAGO point (YAGO places that we had already linked to GADM geometries were not included). Again the Sedona library was employed to perform these computations.

These computed EC and $NTPP$ relations were combined with existing qualitative relations to generate a final RCC-8 network. There were two sources of existing qualitative relations. Implicitly, the GADM dataset already featured containment relations. These were extracted using the hierarchical ids used for regions e.g. BEL.3_1 indicates a $\{NTPP, TPP\}$ relation with Belgium. As described previously, the YAGO knowledge graph features over a million *schema:containedInPlace* triples. These were mapped to $NTPP$ relations.

We found that the resulting RCC-8 network featured several inconsistencies. There were two main reasons. The first source of inconsistencies was when the computed $NTPP$ relation contradicted a *schema:containedInPlace* triple, either because the coordinates for the YAGO place were incorrect, or the *schema:containedInPlace* triple was erroneous. The second was ambiguity over the meaning of 'contained in place'. For example, rivers in YAGO often have *schema:containedInPlace* relations with multiple YAGO places when in fact they overlap them. To solve these issues we simply limited each YAGO place to a single $NTPP$ relation. Similar projects e.g. Ragalia et al. (Regalia, Janowicz, & McKenzie, 2019) also found inconsistencies between computed relations and topological relations present in DBpedia. Despite the limitations of our approach, we still found that the resulting knowledge graph was a rich source of qualitative spatial data with the final knowledge graph featuring 1,102,691 RCC-8 relations.

5. ParQR-QE: Parallel qualitative query engine

ParQR-QE[6] is a query engine that is able to use knowledge graphs such as the one generated in Section 4 to answer GeoSPARQL queries. A reasoner called ParQR (Mantle, Batsakis, & Antoniou, 2019) has been presented previously. Although we have used some of the same techniques, ParQR-QE is a completely separate application that provides query answering functionality, and a fundamentally different reasoning algorithm.

ParQR-QE has been built using the Apache Spark framework. Spark is an industry standard distributed programming framework for working with large-scale datasets (Zaharia et al., 2016). The Spark framework takes care of low level distributed programming tasks such as splitting input files into separate partitions and distributing data to machines in a cluster. The framework also offers a number of different interfaces for working with datasets. In ParQR-QE, the lower level RDD API was used to implement the QSR algorithm (Section 6). Other aspects of the query engine use the Spark SQL interface that supports execution of SQL queries over distributed datasets. The Sedona library, which is an extension for Spark, was used to execute quantitative spatial functions.

The justification for taking a distributed approach to processing is the size of the enhanced knowledge graph generated in Section 4. Even without considering the spatial elements, previous work e.g. Galárraga, Hose, and Schenkel (2012) and Schätzle, Przyjaciel-Zablocki, Skilevic, and Lausen (2016) demonstrate the benefits of using a distributed

[4] https://sedona.apache.org/
[5] https://spark.apache.org/

[6] Full code listings can be found at https://github.com/mmantle-hud/ParQR-QE.

Table 3
Vertically partitioned *hasOccupation* table.

Subject	Object
yago:Soe_Win_(prime_minister)	yago:Politician
yago:Chimaobi_Nwaogazi	yago:Football_player
yago:Finian_McGrath	yago:Politician
...	...

approach when querying large-scale RDF graphs. Furthermore, traditional approaches to reasoning over RCC-8 networks such as the GQR reasoner (Westphal, Wölfl, & Gantner, 2009) do not scale to networks featuring millions of relations. Among the most effective approaches for large-scale QSR are those that use a distributed, parallel approach.

The following provides an overview of how data is stored and queries are executed in ParQR-QE. In Section 6 we describe the distributed backward-chaining spatial reasoning algorithm used in ParQR-QE that allows queries to be answered qualitatively.

5.1. Data storage

ParQR-QE stores triples from the input knowledge graph in three ways:

Non-spatial triples. These are stored using a Vertical Partitioning (VP) approach (Abadi, Marcus, Madden, & Hollenbach, 2007) where a separate table is created for each predicate in the knowledge graph, see Table 3 for an example. Parquet,[7] a distributed column based file format, is used to store the tables.

RCC-8 relations. These are extracted from the knowledge graph and dictionary encoded. The text based representations e.g. *yago:Belgium* are replaced with primitive integers. The simpler representation reduces the memory requirements and affords faster processing e.g. when used as keys during the reasoning process.

Quantitative spatial data. Range queries require a hybrid approach involving quantitative and qualitative reasoning. Therefore it is necessary to store the geometries of spatial entities. Multi-polygon geometries are split into polygons (to support greater parallelisation). These polygons are combined with points and stored in a *geometries* Parquet table.

5.2. Query execution

Fig. 4 shows the query execution for Query A1. Executing the non-spatial part of the query involves mapping GeoSPARQL query triples to SQL queries. These queries are then executed using the vertically partitioned tables. For example, the non-spatial element of Query A1 is mapped to an SQL statement that selects Argentinian Provinces from a *type* table. The query parsing and mapping to Spark SQL is handled with the assistance of the Apache Jena[8] semantic web framework.

The spatial part of the query is executed separately using qualitative spatial reasoning that derives all possible relations for the given query objects. This is shown in Fig. 4 as a call to the *reason* function. After reasoning has completed, these relations are filtered to only keep relations that match the spatial predicate of the query. In this example only locations that have an EC relation with *yago:Chile* will be retained. These results are joined to the results from the non-spatial part of the query to provide the final query response. Containment queries follow an identical plan, but with filtering on the basis of $\{TPPi, NTPPi\}$. The execution of joins is similar, but multiple spatial objects are passed to the *reason* function e.g. an array of US States for Query J1.

The execution of range queries is more involved, implementing Query R1 is shown in Fig. 5. The spatial part of the query starts by finding all spatial entities that are quantitatively located within the query polygon. This is accomplished using the *geometries* table, both polygons and points are tested using the Sedona library's *ST_Contains()* function. These matching objects are then passed to qualitative spatial reasoning where RCC-8 relations are derived for these entities. These relations are then filtered on the basis of containment and combined with the quantitative results. Finally, the spatial results are joined to the non-spatial part of the query to provide the overall results.

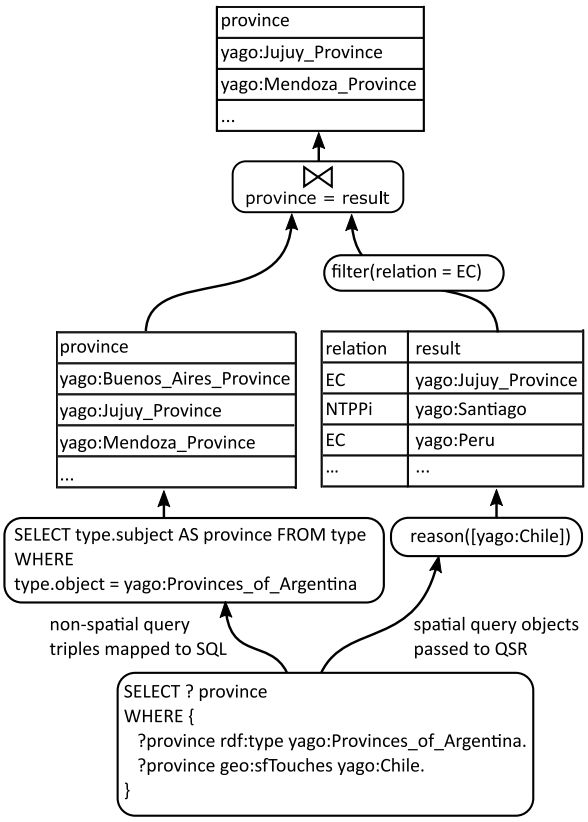


Fig. 4. Query execution for query A1.

6. Backward chaining query answering

In theory, it would be possible to use the path consistency algorithm described in Section 2 to reason over an RCC-8 network like the one generated in Section 4 and materialise relations. The spatial part of queries could then be answered simply by looking up relations in the resulting network. The obstacle to this is the size and topology of the RCC-8 network generated in Section 4. We found that using ParQR, a state of the art reasoner designed for working with large-scale datasets, we were unable to successfully compute closure for this network. This is consistent with the results presented in Mantle et al. (2019) where current state of the art reasoners struggled with similar networks that allow for a very large number of inferences to be made.

An alternative approach to inferring query answers is to backward chain and only derive those relations that are needed to provide an answer to the query. Limiting reasoning to spatial objects specified in the query can still be challenging, however, the volume of data generated becomes much more manageable. Furthermore, although not a factor for the use case presented here, backward chaining is useful when reasoning with knowledge graphs that are dynamic and frequently change. New information can be added to a constraint network and inferences made without having to re-compute the full closure.

[7] https://parquet.apache.org/
[8] https://jena.apache.org/index.html

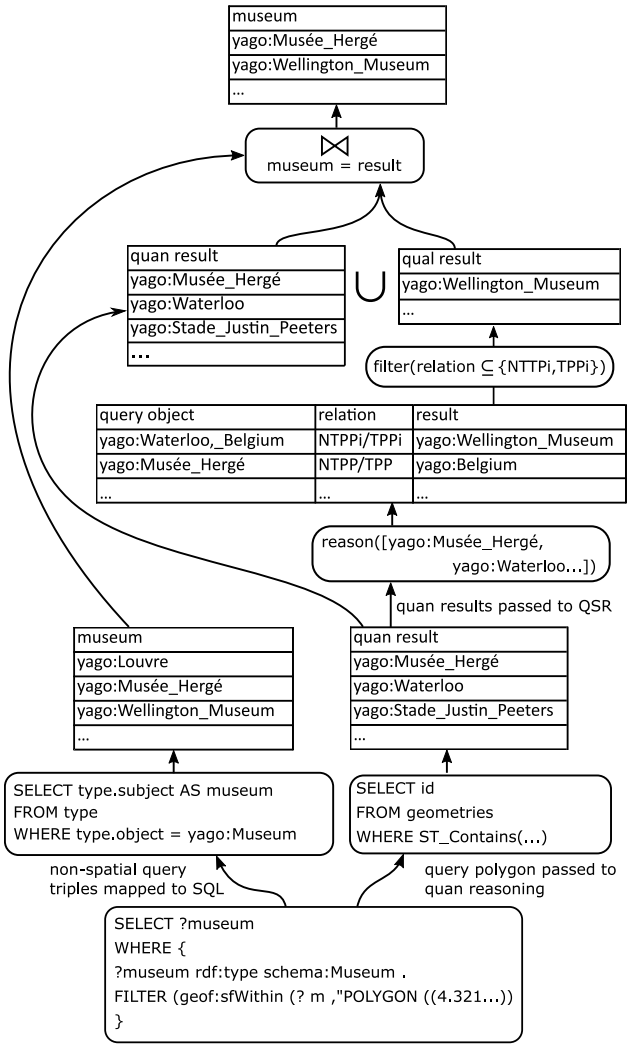


Fig. 5. Query execution for query R1.

6.1. Overview and parallelisation

Fig. 6 visually depicts the backward chaining algorithm used in ParQR-QE. Although the example presented here focusses on RCC-8, the reasoner is a general one. The inputs are simply a composition table and a qualitative constraint network. The reasoner can work with any qualitative constraint calculi. For example, reasoning over qualitative temporal relations using Interval Algebra (Allen, 1983), or the relative orientation of objects using Cardinal Direction Calculus (Ligozat, 1998). Edges in the input network are represented as tuples in the form *(head node, relation, tail node, distance)*. The *distance* value represents the distance between the nodes of the edge, and is needed to prevent unnecessary duplicate derivations.

ParQR-QE only stores edges where the relation between nodes is not the universal relation. This reduces the memory requirements for the reasoner. The universal relation represents an absence of information, and is of no benefit in refining the network. Also, if not already present, the reverse of each edge is added to the network as a pre-processing step. This is to ensure that all possible chains inference can be followed. However, for clarity the reverse edges are not shown on Fig. 6.

At query time, an array of spatial objects from a GeoSPARQL query (query objects) are passed to the reasoner. See the query execution plans, Figs. 4 and 5, where *reason*(...) indicates a call to the backward chaining algorithm.

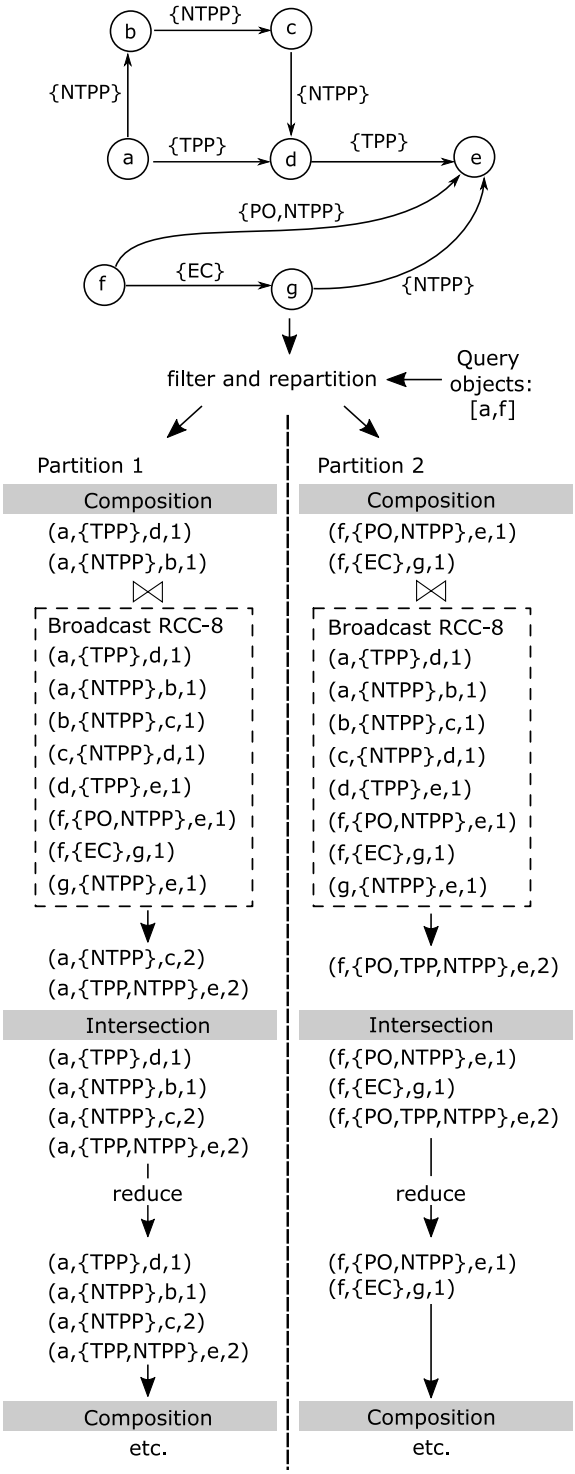


Fig. 6. Overview of the backward chaining algorithm.

The RCC-8 network is filtered to obtain only those edges that feature a query object as the head node. These query edges are partitioned so that all edges for the same query object reside in the same partition. For example, In Fig. 6, we assume that two objects, a and f have been passed from the GeoSPARQL query. The RCC-8 network is filtered and split into two partitions, one for edges featuring a as the head node and one for those featuring f. Reasoning then proceeds in parallel, and only once reasoning has completed are the results from different partitions combined.

A key design choice that facilitates this is to use a broadcast or map-side join. Ahead of query time, the entire RCC-8 network is copied to each machine in the computing cluster. This is shown on the figure as *Broadcast RCC-8*. This allows reasoning to execute completely within partitions, which circumvents the need for costly shuffling of data between machines, which would take place if the default join strategy was implemented.

Reasoning is an implementation of path consistency, but only for objects specified in the query. This is accomplished by iteratively deriving new relations for the query objects. Fig. 6 shows the first iteration of the algorithm. At each iteration there are two stages, *composition* and *intersection*. During *composition*, query edges are joined to edges in *Broadcast RCC-8*, and new relations are inferred. The join is made by matching the tail nodes of query edges to the head nodes of edges in the broadcast RCC-8 network. A composition table (as described in Section 2) is used to look-up the inferred relation. The outputs from the *composition* stage are combined with the existing query edges and form the input to the *intersection* stage where RCC-8 relations between pairs of nodes are pruned by taking the intersection of the relations. This reasoning continues iteratively, until no more new relations can be inferred.

Algorithm 1 shows the *reason* function. The *mapPartitions* operation executes over partitions, consequently a number of while loops (lines 12–17) execute in parallel, each performing reasoning for a subset of the query objects. At each iteration the *composition* and *intersection* stages are delegated to sub-routines. These are shown in Algorithms 2 and 3.

Algorithm 1: Reason.

```
1: reason(RCC8, queryObjects) {
2:
3:    //filter to obtain edges featuring query objects
4:    queryEdges = RCC8.filter(edge.head ∈ queryObjects)
5:       .map(edge ⇒ (edge.head, edge))
6:       .repartition()
7:
8:    //reason for the selected edges
9:    result = queryEdges.mapPartitions(queryEdges ⇒ {
10:      count = 0
11:      i=1
12:      while queryEdges.count() ≠ count {
13:         count = queryEdges.count()
14:         newEdges = composition(queryEdges, i)
15:         queryEdges = intersection(queryEdges ∪ newEdges, i)
16:         i++
17:      }
18:      return queryEdges
19:   })
20:   return result
21: }
```

6.2. Preventing duplicate derivations

In order to prevent duplicate derivations it is necessary to limit which edges can participate in the join at a given iteration. This is shown in Algorithm 2. The right side of the join always remains the same, this is the broadcast RCC-8 network. The left side of the join varies with each iteration. At iteration 1 edges from the input network are used. At iteration 2, to prevent duplicate derivations being made, we only want the newly generated edges from the previous iteration to form the left side of the join. To do this we assign an integer value to each edge (see Algorithm 2, line 24). We call this value distance (*dist* on the algorithms), as it represents the distance between the nodes in the original RCC-8 network. This value is then used at subsequent iterations to determine which edges can participate in the join, see Algorithm 2, line 4. Without this filtering the join size would grow with each iteration, generating duplicate derivations, and soon become unmanageable.

Algorithm 2: Composition.

```
1: composition(queryEdges, i) {
2:
3:    // filter to prevent same derivations as previous iterations
4:    lhsEdges = queryEdges.filter(edge ⇒ edge.dist = i)
5:
6:    // join edges
7:    joinedEdges = lhsEdges.map(lhsEdge ⇒ {
8:       tailNode = lhsEdge.tail
9:       //get edges from the broadcast RCC-8 network
10:      rhsEdges = broadcastRCC8(tailNode)
11:      //join to lhsEdge
12:      joinedEdges = rhsEdges
13:         .map(rhsEdge ⇒ {
14:            return (lhsEdge,rhsEdge)
15:         })
16:         .filter( lhsEdge.head ≠ rhsEdge.tail) //no loops
17:      return joinedEdges
18:   })
19:
20:   //use composition to infer new relations
21:   newEdges = joinedEdges.map((lhsEdge, rhsEdge) ⇒ {
22:         //look-up inferred relation using composition table
23:         rel = lookUp(lhsEdge.relation, rhsEdge.relation)
24:         dist = i+1
25:         return (lhsEdge.head, rel, rhsEdge.tail, dist)
26:   }).filter(newEdge.relation ≠ universal relation)
27:   return newEdges
28: }
```

6.3. The algorithm makes all possible inferences with respect to the query nodes

The algorithm follows all paths from the starting query nodes, one edge at a time, at each iteration. This continues until all reachable nodes have been discovered, and all constraints are considered, making the algorithm sound and complete under the given constraint calculi for the given query nodes. For example, Fig. 6 shows a relation between a and e being inferred at iteration 1. However, by following the path through b, c and d further constraints will be put on this relation at iteration 3, when it will be updated to $NTPP$.

By following every path the reasoner utilises all available spatial information for the given query objects. There are only two conditions that halt the following of a path. The first is if the inferred relation between two nodes is the universal relation. In which case this edge is not added, see Algorithm 2, line 26. The second is if a newly inferred relation is the same or weaker than an existing relation. This is determined in the *intersection* stage (Algorithm 3) where relations between pairs of nodes are compared and only if the newly inferred relation is stronger will the distance value be updated. This is implemented by generating a key for each edge so that different edges for the same pair of nodes can be compared. The comparison is then executed using a *reduceByKey* operation that takes the intersection of the relations, tests for consistency, and determines if the new relation is stronger.

6.4. Complexity analysis

The *reason* algorithm runs in polynomial time with respect to the number of nodes in the RCC-8 network. In the worst case the input RCC-8 network is a complete graph with n^2 edges, and all the edges are query edges i.e. it is necessary to infer relations for all nodes in the network. The key part of the algorithm from a time complexity perspective is the *composition* function; this uses nested iteration to implement a join for deriving new relations. Assuming the worst case scenario, one side of this join ($lhsEdges$) features every edge in the network, and the join is executed between each of these edges and their adjacent edges giving

$n^2 \cdot n = n^3$ operations. *composition* is executed from within the while loop of the *reason* function. In the case of a complete graph, a relation between every pair of nodes will be derived in a single iteration of this while loop. Each relation can be updated a maximum of eight times (for each of the basic RCC-8 relations), consequently the loop runs in constant time and the overall time complexity of the algorithm is $\mathcal{O}(n^3)$. Other aspects of the algorithm involve simpler, flat linear time operations such as *map* and *filter*.

In terms of space complexity, *composition* can return n^3 edges. These results are combined with the input network (Algorithm 1, line 15), potentially resulting in $n^2 + n^3$ edges being passed to *intersection* where they are reduced to no more than n^2 edges. Therefore the overall space complexity is also $\mathcal{O}(n^3)$.

Of course we do not expect to see this worst case scenario in practice. As described above, real world knowledge graphs tend to be fairly sparse and queries tend to focus on specific spatial objects or spatial objects of a particular type. Furthermore, the algorithm is executed in a distributed setting with the query edges split over multiple partitions allowing processing and memory requirements to be shared between machines in a cluster. However, there are limits to the parallelisation. A complete copy of the $broadcastRCC8$ variable is sent to every machine in the computing cluster. Therefore, each machine needs sufficient memory to be able to store this potentially n^2 data structure in memory.

Algorithm 3: Intersection.

```
1: intersection(edges, i) {
2:
3:   //generate a key for each edge
4:   keyedEdges = edges.map(edge⇒ {
5:            return (edge.head+'#'+edge.tail, edge)
6:   })
7:
8:   //reduce by key to compare relations
9:   conEdges = keyedEdges.reduceByKey((edgeA,edgeB)⇒ {
10:    isect = edgeA.rel ∩ edgeB.rel
11:    if |isect| = 0
12:      stop() //inconsistency detected
13:    end if
14:    dist = if(edgeA.dist = (i+1) and |edgeB.rel| > |isect|)
15:      edgeA.dist
16:    else if (edgeB.dist = (i+1) and |edgeA.rel| > |isect|)
17:      edgeB.dist
18:    else
19:      Math.min(edgeA.dist, edgeB.dist)
20:    endif
21:    return (edgeA.head, isect, edgeA.tail, dist)
22:  })
23:  return conEdges
24: }
```

7. Evaluation

We evaluated ParQR-QE in a number of ways. First, we compared the ParQR-QE to a quantitative approach to resolving spatial queries. To do this we developed a separate query engine, named Quan-QE, that used the same input knowledge graph, however, instead of using QSR, the spatial parts of queries were executed using the geometries of spatial objects. This was implemented using functions from the Sedona library. Secondly, we were interested in ParQR-QE's wider utility as a query answering system therefore we compared it to alternative approaches to answering the types of query described in Section 3. Finally, seeing as ParQR-QE is a distributed application, we also wanted to test the scalability of the reasoner.

Queries. The evaluations focussed on four different types of queries, containment queries (C1–C4), adjacency queries (A1–A4), spatial joins (J1–J4) and range queries (R1–R4). Query response times do not include pre-processing and set-up tasks such as dictionary encoding, broadcasting of the QCN, and generating the VP tables. Queries were executed cold and warm. For the cold execution the queries were run immediately after system start-up. For warm execution, the same query was executed an additional five times and the mean average response time was recorded. The warm response times are significantly faster due to characteristics of the Spark framework that speed-up subsequent operations on the same dataset. For example, by default and beyond the end users control, shuffled data is stored on worker nodes ready to be reused in needed.

The experiments were run using a 4-node computing cluster on the Google Dataproc[9] cloud computing platform. Each machine in the cluster had 8vCPUs and 52 GB of memory. We set a limit of five minutes for query execution time.

7.1. Comparison with quantitative reasoning

Table 4 shows the results for the queries using both ParQR-QE and Quan-QE. As expected there are differences in the number of results returned for many of the queries. ParQR-QE was often able to return additional results. This is not surprising, qualitative relations were generated for all YAGO points, therefore any result found by quantitative reasoning could also be found through qualitative reasoning. In addition, ParQR-QE was often able to find extra, qualitative only based, results. For example, Query C2 (*find Barcelona footballers that were born in Catalonia*) returns 268 results for ParQR-QE, but only 263 results for Quan-QE. Closer analysis of these results shows that there are places, such as Navarcles, that are located in Catalonia, but do not have a *schema:geo* predicate. As a consequence, footballers born in Navacles do not appear in the results of quantitative reasoning. However, Navarcles does have a qualitative *schema:containedInPlace* predicate so it can be found using qualitative reasoning. Mataró is a coastal town whose YAGO *schema:geo* predicate inaccurately locates it in the Mediterranean Sea, outside the geometry of Catalonia. Therefore, footballers born in Mataró do not appear in the results of quantitative reasoning. Again, Mataró has a *schema:containedInPlace* property so it is included in the results from ParQR-QE. These two factors, incomplete and inaccurate quantitative data, explain the differences in the number of results returned for the queries shown in Table 4 and show the advantages of using a qualitative approach.

The query response times for ParQR-QE are largely dependent on the volume of derivations that are made during reasoning. For example, the fastest executing query A4 (*countries that border Chad*) results in ParQR-QE outputting 17,140 relations. In comparison, the longest running Query J4 (*Nuclear power plants and the countries they are located within*) results in 61,101,239 relations. Query J4 is a join where reasoning is executed for all spatial objects of the type *schema:Country*. In comparison, Query A4 involves a single object in a comparatively sparse part of the RCC-8 network, resulting in fewer derivations being made. With range queries, again multiple spatial entities are passed to ParQR-QE, which creates the possibility of more relations being inferred, and a larger volume of data being processed. This can be seen in the runtimes of the range queries, especially queries R1 and R4.

When comparing runtimes to Quan-QE we are not looking to make definitive statements about the speed of qualitative vs quantitative spatial reasoning. The Quan-QE system we developed simply uses the standard functions of Sedona. We have made little attempt to fine tune the Quan-QE implementation or consider query optimisation in-depth.

However, it is still worth considering where query times differ and the reasons why. Quan-QE results show much greater variability in

[9] https://cloud.google.com/dataproc

Table 4
Experimental Results comparing ParQR-QE and Quan-QE.

Query	ParQR-QE			Quan-QE		
	Time (cold)	Time (warm)	No. of results	Time (cold)	Time (warm)	No. of results
C1	21.896	11.068	149	21.113	9.804	137
C2	24.501	10.939	268	11.901	2.675	263
C3	25.552	11.564	9	–	–	–
C4	24.959	14.602	4	9.765	2.280	4
A1	19.234	8.255	10	39.361	16.759	10
A2	30.718	12.262	4	220.520	166.345	4
A3	24.827	12.535	2	8.559	1.975	2
A4	27.684	7.820	6	22.602	7.891	6
J1	41.983	24.762	49	16.419	4.076	48
J2	28.915	11.822	45	–	–	–
J3	25.768	9.083	52	28.653	10.604	21
J4	126.730	83.231	305	112.532	45.506	279
R1	40.922	21.056	4	11.404	3.456	3
R2	25.688	15.248	53	13.261	3.632	53
R3	28.894	12.457	2	13.559	3.825	2
R4	56.272	40.390	50	14.598	4.008	41

comparison with ParQR-QE. The runtimes for queries that successfully completed varied between 1.975 and 166.345 s. The runtimes are heavily dependent on the size and complexity of the geometries being processed. For example, Query A2 (*US States that border Mexico*) involves testing the multi-polygon geometry of Mexico against the multi-polygon geometries of US States. Queries involving geometries with fewer coordinates or queries where one side of the join are spatial objects represented by simple points, run much faster. Two queries, C3 and J2, failed to complete within the five minute time limit. This is a consequence of some very large geometries being tested. For example, Query C3 *video game developers that were founded in Canada* requires the geometries for Canada and Japan to be compared. The GADM geometry for Canada is a multi-polygon comprised of 24,482 polygons and 3,889,947 coordinates which Quan-QE struggled to process efficiently. In comparison, the query response times for ParQR-QE show less variance. This indicates one possible utility of QSR in query answering. Quantitative reasoning approaches are dependent on the scale and complexity of the geometries being processed. Free of such constraints, qualitative reasoning can prove advantageous in scenarios where large-scale datasets featuring objects with high resolution, complex geometries are being used.

7.2. Comparison with other question answering systems

To our knowledge there are not any existing applications capable of answering GeoSPARQL over large scale knowledge graphs therefore we consider alternative systems for answering the queries shown in Table 4. We ran the queries using ChatGPT 3.5,[10] DBPedia's SPARQL endpoint[11] and Open Street Map's (OSM) Overpass API.[12] Table 5 shows the results of running the same queries using these comparison systems. Clearly, because the queries are being answered using different datasets, for many queries we expect different results. We are more interested in the types of queries that can be answered, and any issues associated with running the queries.

The family of GPT large language models that ChatGPT is built on have shown impressive performance in question answering (QA) tasks (Tom Brown et al., 2020) and therefore provide a useful comparison for ParQR-QE. As ChatGPT is not capable of answering GeoSPARQL queries directly, the queries were re-phrased as natural language questions. For example, Query C1 was posed as *'List all the museums in Belgium'*. A complete list of questions asked and responses from ChatGPT can be found at,[13,14] and[15] ChatGPT was unable to provide answers for the range queries, but provided useful responses to all the other queries. Despite being asked to provide comprehensive results, it often had to be repeatedly prompted to provide additional results. For queries with lots of results, eventually it started producing duplicates, and we stopped prompting any further, see Table 5. In some cases ChatGPT provided factually incorrect answers. However, these were always accompanied by an explanation. For example, when asked for actors born in Germany that appeared in the film Casablanca (Query C4), it included actors considered German that were not actually born in Germany, but explained this was the case. It is also interesting that despite being trained on vast quantities of data (Tom Brown et al., 2020), it often failed to provide a complete list of correct answers e.g. Query J1 *Argentinian Provinces that border Chile*. These results demonstrate the limitation of LLMs for QA tasks (Lazaridou, Gribovskaya, Stokowiec, & Grigorev, 2022) (Maynez, Narayan, Bohnet, & McDonald, 2020), namely that they can experience hallucinations. Furthermore, when question answering using LLMs it is not easy to determine the basis of the answer given (Pörner, Waltinger, & Schütze, 2019) or to update factual information without re-training the model (Verga, Sun, Soares, & Cohen, 2021). In comparison, when using a symbolic knowledge base, such as the enhanced knowledge graph used for ParQR-QE's results, the exact source of answers can be identified, and the knowledge graph can be easily modified to add, remove and update triples.

Although GeoSPARQL are not supported, it was possible to execute some queries against the DBPedia knowledge graph by mapping to plain SPARQL and using DBPedia properties such as *dbo:location* to test for containment instead of the GeoSPARQL *geo:sfWithin* property. For example Query C1 was re-written as:

Query C1 in SPARQL

```
select ?museum WHERE {
  ?museum rdf:type dbo:Museum.
  ?museum dbo:location dbr:Belgium.
}
```

See[16] for a full listing of all the queries. The results for DBPedia expose many of the points listed in Section 3. For example, it was not possible to run adjacency or range queries. For the containment type queries that were possible, many useful results were missed. For example, many Belgian museums have a *dbo:location* property for a town or city they are based in, but not directly with Belgium. Without spatial reasoning, it was not possible to return these as results.

OpenStreetMap is an open geographic database that features a wide range of features including buildings, roads, amenities and boundaries. The OSM Overpass API allows for the execution of spatial queries against OSM data. Again, GeoSPARQL queries are not supported so queries were mapped to the Overpass Query Language. For example Query C1 was translated as:

Query C1 in Overpass QL

```
[out:csv(name)];
(area["ISO3166-1"="BE"][admin_level=2];)->.b;
node["tourism"="museum"](area.b);
out;
```

[10] https://chatgpt.com/
[11] https://dbpedia.org/sparql/
[12] https://overpass-turbo.eu/

[13] C1–C4: https://chatgpt.com/share/947382f6-3767-4bb2-b3dc-8e42515f3f8e.
[14] A1–A4: https://chatgpt.com/share/be4526d1-443c-41af-9396-fc0b25c073d5.
[15] J1–J4: https://chatgpt.com/share/126b48d2-1682-4825-a87a-aec1a94172d2.
[16] https://github.com/mmantle-hud/ParQR-QE/tree/master/evaluation_queries

Table 5
No. of query results for ParQR-QE, ChatGPT, DBpedia + SPARQL and OSM + Overpass API.

Query	ParQR-QE	ChatGPT	DBPedia + SPARQL	OSM + Overpass API
C1	149	$> 66^{+\star}$	50	446
C2	268	$> 85^{+\star\dagger}$	369	–
C3	9	$> 25^{+\dagger}$	–	–
C4	4	$3^{+\dagger}$	1	–
A1	10	8^{+}	–	11
A2	4	4	–	4
A3	2	2	–	2
A4	6	6^{+}	–	6
J1	49	$> 36^{+\star\dagger}$	28	–
J2	45	$33^{+\star}$	–	–
J3	52	$> 47^{+\star}$	40	794
J4	305	$> 127^{+\star}$	145	167
R1	4	–	–	5
R2	53	–	–	165
R3	2	–	–	13
R4	50	–	–	273

– = Not possible to run the query.
+ = Needed to be prompted for additional answers.
★ = Produced duplicate answers.
† = Produced incorrect answers, but with explanation.

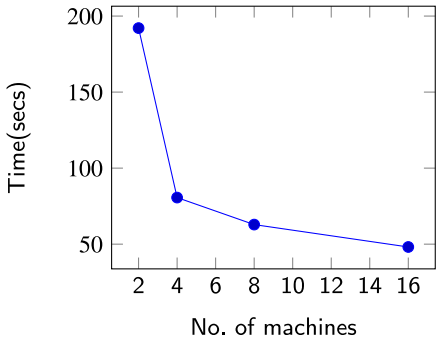


Fig. 7. Query Response time for ParQR-QE as a function of the no. of machines in the computing cluster.

Again, see[16] for a full list of queries. Table 5 shows it was possible to execute a variety of different queries including adjacency and range queries. OSM is a rich source of geographical data featuring over 9.2 billion nodes many of which represent points of interest (OpenStreetMap Statistics, 2024), and queries often returned a greater number of results than ParQR-QE using YAGO and GADM. In comparison to using a knowledge graph such as YAGO, the limitation of OSM is that features are tagged with basic properties such as their type and name, but lack a richer set of properties and are not linked to other entities. Therefore it was not possible to run queries that rely on anything other than geometric properties and basic type information. In comparison ParQR-QE was able to use both qualitative spatial information and non-spatial properties to answer queries.

7.3. Scalability

ParQR-QE is a parallel distributed reasoner therefore we also evaluated its ability to parallelise processing effectively. For this experiment we ran Query J4 repeatedly, each time using different sized computing clusters. Fig. 7 shows that as we add more machines to the computing cluster, greater parallelisation is achieved and query response time decreases significantly. A standard metric used to evaluate parallel, distributed systems is scaled speed-up, $\frac{t_1}{t_n} \div n$ where t_1 is the execution for a single machine, n is the number of machines, and t_n the execution time for the n-machine cluster. A single machine was unable to handle the size of the RCC-8 network so a two machine cluster was used as the baseline and the scaling factor adjusted accordingly. Fig. 8 shows scaled speed-up. Ideally, when the number of machines in the cluster doubles the reasoning time will halve, giving linear speed-up. Moving from two to four machines, ParQR-QE shows super linear speed-up, before dropping to sub-linear speed-up when eight and sixteen machines are used. This is not unusual for distributed applications where at some point all the necessary tasks can be launched simultaneously and the benefit of further parallelisation reduces. Although we have demonstrated the capability of ParQR-QE to reason over large-scale networks and parallelise a workload, it is worth noting that the broadcast approach we have taken, which requires each machine in the cluster to have sufficient memory to store the entire network, could at some point be a limitation for the reasoner.

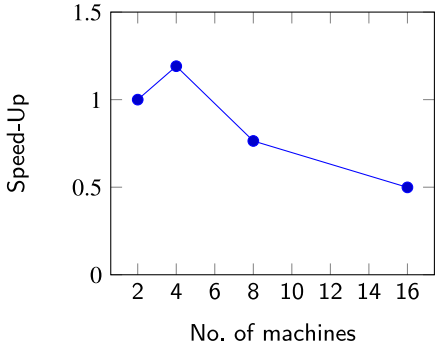


Fig. 8. Scaled speed-up for ParQR-QE.

7.4. Potential and limitations

Section 7 evaluated ParQR-QE in answering a number of different queries using an enhanced YAGO knowledge graph. However, it is worth considering the wider applicability, and the potential and limitations of the work presented here.

Using alternative knowledge graphs. Obtaining useful results was largely dependent on the enhanced dataset generated in Section 4. The knowledge graph was carefully crafted to ensure all YAGO points had a relation with a GADM region, and that relations between GADM regions within layers were present. Without this pre-processing, chains of reasoning could not have been executed by the reasoner. Given the often sparse nature of real world constraint networks, in order for ParQR-QE to have wider applicability, it is likely that similar pre-processing would be necessary to ensure the topology of the constraint network will support useful reasoning. However, the methods described in Section 4 can be easily applied to other knowledge graphs. For example, the same process can be applied to DBpedia, where spatial entities are also represented using points and *dbo:location* properties could be used for mapping to RCC-8 instead of *containedInPlace*.

Choice of queries. We have focussed on four types of query. These cover a wide range of query types, and utilise a range of RCC-8 relations ($\{EC, TPP, NTPP, iTPP, iNTPP\}$). Furthermore, the range query shows how QSR can be used as part of a hybrid approach that combines quantitative and qualitative representations. The query implementation can be extended for other types of query. For example a spatial k-nearest neighbour query (or distance query) can be implemented using an approach similar to the query plan shown in Fig. 5. Quantitative reasoning can be used to find candidate spatial objects, qualitative reasoning to find additional spatial objects contained within these candidate objects, and a final filtering to find the k nearest objects to the point in question.

The obvious omission in terms of RCC-8 relations is querying for overlapping (PO) relations. YAGO does not feature PO type relations.

Furthermore, the point based representations in YAGO meant that when generating the RCC-8 network in Section 4 no *PO* relations were computed. The enhanced knowledge graph generation described in Section 4 could be extended to include other spatial features e.g. computing *PO* relations between GADM regions and the geometries of rivers and national parks. Further 'overlaps' type queries would then be possible such as *find all the countries that the river Rhine flows through*. It should also be noted that there was a benefit to the absence of *PO* relations. Reasoning using *PO* often results in a disjunctive relation, see Table 1. The absence of *PO* relations meant that reasoning resulted in the derivation of one of the basic relations, and we could provide definitive answers to queries. Including *PO* relations would not render the application impractical, it would simply mean that querying may present answers in the form of a disjunction of relations.

Using other qualitative constraint calculi. It is important to emphasise the reasoning element of the query engine is not restricted to RCC-8. Provided with a suitable composition table and constraint network, it can work with any qualitative constraint calculi. For example, by using a different query execution layer, it would be possible to answer temporal queries using Interval Algebra without having to make any changes to the reasoner. Although the reasoner is not capable of reasoning over a combination of two or more calculi (often a challenging task (Liu, Li, & Renz, 2009)), it would be possible to reason separately using two different calculi and take the intersection of the results to provide query answers. For example, to answer spatio-temporal queries (RCC-8 and Interval Algebra) *'all the Oscar winning actors born in England during the Second World War'* or using two different spatial constraint calculi (RCC-8 and CDC) *'find museums in Belgium that are south of Brussels'*.

8. Related work

There are number of semantic web frameworks that support the parsing and execution of GeoSPARQL queries e.g. Apache Jena,[17] Ontop.[18] However, such frameworks do not implement spatial reasoning in order to answer queries.

There are also distributed approaches to storing and querying RDF data e.g. S2RDF (Schätzle et al., 2016) is a distributed RDF query engine built using Spark. The limitation of these distributed RDF systems is they are not capable of answering GeoSPARQL queries, they can only provide answers to non-spatial queries. Therefore we focus our analysis of related work on two areas (1) spatial query answering systems that use QSR and (2) systems that utilise Linked Data and a combination of qualitative and quantitative data to answer spatial queries.

8.1. Answering spatial queries using QSR

Bennett et al. first proposed the use of QSR for query answering in GIS (Bennett et al., 1997). A prototype system was described that featured quantitative polygon data and RCC-8 relations, and used QSR to evaluate queries. The system was limited in scale, showed no real world application, and no evaluation was presented. It was presented to *"explore possible ways to exploit qualitative spatial reasoning"*.

Since this time a number of reasoners have been developed that use QSR to answer spatial queries, notably PelletSpatial (Stocker & Sirin, 2009) and CHOROS (Christodoulou, Petrakis, & Batsakis, 2012). CHOROS is an extension of PelletSpatial, which itself extends the Pellet reasoner. These applications have some similarities to ParQR-QE. Both systems are centred around using semantic web technologies, and the spatial part of queries is handled by QSR reasoning. However, there are also significant differences. These are non-distributed systems that run on a single machine. Furthermore, in both reasoners, path consistency is computed for the entire RCC-8 network ahead of query time. These factors mean both systems lack the ability to scale. Experiments presented in Stocker and Sirin (2009) and Christodoulou et al. (2012) contained no details on queries, and focussed on using comparatively small datasets consisting of hundreds of relations.

8.2. Qualitative knowledge in geospatial query answering over linked geospatial datasets

A number of systems have been developed that use linked geospatial datasets and a combination of quantitative and qualitative knowledge to answer spatial queries.

The work by Regalia et el. Regalia et al. (2019) has many similarities with the work we have presented in Section 4. They enriched DBpedia with qualitative relations by matching DBpedia places with geometries from Open Street Map, and then computed topological relations between entities using the polygons and polylines from OSM. Compared with the enhanced knowledge graph presented in Section 4, this was a smaller example. They focussed on a single country, the United States, a limited set of spatial entities, and generated a smaller dataset consisting of 120,681 relations. However, the dataset they generated had greater complexity. They used polygons for cities, counties and parks, and polylines for streams and roadways. As a result, they were able to utilise a wider range of relations, this included the *PO* relation in addition to EC, TPP and $NTTP$ RCC-8 relations, but also relations from other formalisms. For example approximate topological relations, such as *nearly meets* or *nearly contains*.

The focus of their research was on computing the relations and generating an enhanced dataset. They presented a small number of example queries that demonstrated both the performance benefits of their approach and that additional results are generated. They also provided an example of how reasoning over these relations could be used to infer additional facts. However, reasoning was not integrated into a wider query answering system.

Younis, Jones, Tanasescu, and Abdelmoty (2012) developed a system for answering spatial queries using data from DBPedia and the Ordnance Survey. The Ordnance Survey data, comprised of polygons representing UK regions was stored in a PostGIS database. DBPedia spatial entities represented by points were stored separately. Similar to the range queries described above, spatial queries were answered by first accessing this quantitative data, and then executing queries using a SPARQL endpoint to enhance these results with qualitative knowledge from DBPedia e.g. using *dbpo:locatedInArea* properties. They did not provide any detail on query response times. But, like our work, they found that integrating qualitative data returned richer results.

A similar approach is taken by GeoQA (Punjani et al., 2020) a geospatial question answering system that utilises data from GADM, Open Street Map and DBPedia. Resources in these different datasets were semantically interlinked. Queries were then answered by interrogating the three datasets and combining quantitative and qualitative spatial knowledge.

These three examples are significantly different from the work presented here. These systems simply combine data from different sources to answer queries. There is not any reasoning to infer additional results that are not already present. Furthermore, these approaches do not use a distributed architecture in the way ParQR-QE does. As a result we would not expect them to handle large-scale knowledge graphs. In each case, the datasets used for testing were limited in scale, focussing on one or two countries.

9. Conclusions

In this paper we have presented ParQR-QE, a distributed query engine that uses qualitative reasoning to provide answers to spatial queries for large-scale knowledge graphs. To our knowledge, this is the first demonstration of QSR techniques being used in this way. We have evaluated ParQR-QE using an enhanced YAGO knowledge

[17] https://jena.apache.org/
[18] https://ontop-vkg.org/

graph by running a variety of queries and comparing to a purely quantitative approach. Our evaluation has shown that ParQR-QE is able to parallelise workloads effectively and deal with large-scale datasets. We have also shown the benefits of using QSR in query answering. Using ParQR-QE we were able to generate a greater number of results compared to quantitative only techniques, and in some cases provide faster query times.

There are some limitations to the work we have presented here. For example, we took a simple approach to dealing with inconsistencies encountered in the YAGO knowledge graph. Also we chose to focus on adjacency and containment queries, and generated an RCC-8 network that was limited to EC, TPP and $NTPP$ relations. As a consequence QSR was often able to infer basic, rather than disjunctive relations, and could therefore provide definitive answers to queries. Less controlled scenarios featuring a wider range of RCC-8 relations and query types are unlikely to result in similarly neat results. Furthermore, as noted in Section 6, in some situations the scalability may be limited by the broadcast join approach we have employed.

There are a number of directions for future work. There is room to optimise the query execution. For example we expect that early filtering for transitive relations, applying selectivity for joins, and materialising a greater number of relations in advance of query time will speed up response times.

Furthermore, the techniques we have presented can be generalised to other situations. For example, we plan to look at using ParQR-QE in scenarios which are dynamic e.g. natural disaster situations where spatial information is likely to be imprecise, incomplete and rapidly changing. We believe an approach like we have described here, that can integrate quantitative and qualitative data, represent imprecise and incomplete information, and provide reasoning at query time, will be beneficial.

CRediT authorship contribution statement

Matthew Mantle: Conceptualization, Methodology, Investigation, Software, Writing – original draft. **Sotirios Batsakis:** Writing – review & editing. **Grigoris Antoniou:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Article contains links to data/code.

References

Abadi, D. J., Marcus, A., Madden, S., & Hollenbach, K. J. (2007). Scalable semantic web data management using vertical partitioning. In C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C. Kanne, W. Klas, & E. J. Neuhold (Eds.), *Proceedings of the 33rd international conference on very large data bases* (pp. 411–422). ACM, URL: http://www.vldb.org/conf/2007/papers/research/p411-abadi.pdf.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM, 26*(11), 832–843. http://dx.doi.org/10.1145/182.358434.

Bennett, B., Cohn, A. G., & Isli, A. (1997). Combining multiple representations in a spatial reasoning system. In *9th international conference on tools with artificial intelligence* (pp. 314–322). IEEE Computer Society, http://dx.doi.org/10.1109/TAI.1997.632271.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. CoRR arXiv:2005.14165.

Buckley, A. (2017). Dealing with incomplete data for mapping and spatial analysis. URL: https://www.slideshare.net/aileenbuckley/dealing-with-incomplete-data-for-mapping-and-spatial-analysis.

Christodoulou, G., Petrakis, E. G. M., & Batsakis, S. (2012). Qualitative spatial reasoning using topological and directional information in OWL. In *IEEE 24th international conference on tools with artificial intelligence* (pp. 596–602). IEEE Computer Society, http://dx.doi.org/10.1109/ICTAI.2012.86.

Cohn, A. G., & Renz, J. (2008). Qualitative spatial representation and reasoning. In F. van Harmelen, V. Lifschitz, & B. W. Porter (Eds.), *Foundations of artificial intelligence*: *vol. 3*, *Handbook of knowledge representation* (pp. 551–596). Elsevier, http://dx.doi.org/10.1016/S1574-6526(07)03013-1.

Duckham, M., & Kulik, L. (2005). A formal model of obfuscation and negotiation for location privacy. In H. Gellersen, R. Want, & A. Schmidt (Eds.), *Lecture notes in computer science*: *vol. 3468*, *Pervasive computing, third international conference, PERVASIVE 2005, munich, Germany, May 8-13, 2005, proceedings* (pp. 152–170). Springer, http://dx.doi.org/10.1007/11428572_10.

Dylla, F., Lee, J. H., Mossakowski, T., Schneider, T., van Delden, A., van de Ven, J., et al. (2017). A survey of qualitative spatial and temporal calculi: Algebraic and computational properties. *ACM Computing Surveys*, *50*(1), 7:1–7:39. http://dx.doi.org/10.1145/3038927.

Galárraga, L., Hose, K., & Schenkel, R. (2012). Partout: A distributed engine for efficient RDF processing. CoRR arXiv:1212.5636.

Heath, T., & Bizer, C. (2011). *Synthesis lectures on the semantic web*, *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool Publishers, http://dx.doi.org/10.2200/S00334ED1V01Y201102WBE001.

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., et al. (2021). Knowledge graphs. *Synthesis lectures on data, semantics, and knowledge*, Morgan & Claypool Publishers, ISBN: 978-3-031-00790-3, http://dx.doi.org/10.2200/S01125ED1V01Y202109DSK022.

Lazaridou, A., Gribovskaya, E., Stokowiec, W., & Grigorev, N. (2022). Internet-augmented language models through few-shot prompting for open-domain question answering. CoRR arXiv:2203.05115.

Ligozat, G. (1998). Reasoning about cardinal directions. *Journal of Visual Languages and Computing*, *9*(1), 23–44. http://dx.doi.org/10.1006/JVLC.1997.9999.

Liu, W., Li, S., & Renz, J. (2009). Combining RCC-8 with qualitative direction calculi: Algorithms and complexity. In C. Boutilier (Ed.), *Proceedings of the 21st international joint conference on artificial intelligence, pasadena, california, USA, July 11-17, 2009* (pp. 854–859). URL: http://ijcai.org/Proceedings/09/Papers/146.pdf.

Long, Z., Schockaert, S., & Li, S. (2016). Encoding large RCC8 scenarios using rectangular pseudo-solutions. In C. Baral, J. P. Delgrande, & F. Wolter (Eds.), *Principles of knowledge representation and reasoning: proceedings of the fifteenth international conference* (pp. 463–472). AAAI Press, URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12837.

Mantle, M., Batsakis, S., & Antoniou, G. (2019). Large scale distributed spatio-temporal reasoning using real-world knowledge graphs. *Knowledge-Based Systems*, *163*, 214–226. http://dx.doi.org/10.1016/j.knosys.2018.08.035.

Maynez, J., Narayan, S., Bohnet, B., & McDonald, R. T. (2020). On faithfulness and factuality in abstractive summarization. In D. Jurafsky, J. Chai, N. Schluter, & J. R. Tetreault (Eds.), *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 1906–1919). Association for Computational Linguistics, http://dx.doi.org/10.18653/V1/2020.ACL-MAIN.173.

Nikolaou, C., & Koubarakis, M. (2013). Incomplete information in RDF. In W. Faber, & D. Lembo (Eds.), *Lecture notes in computer science*: *7994*, *Web reasoning and rule systems - 7th international conference, RR 2013, mannheim, Germany, July 27-29, 2013. proceedings* (pp. 138–152). Springer, http://dx.doi.org/10.1007/978-3-642-39666-3_11.

Nikolaou, C., & Koubarakis, M. (2014). Fast consistency checking of very large real-world RCC-8 constraint networks using graph partitioning. In C. E. Brodley, & P. Stone (Eds.), *Proceedings of the twenty-eighth AAAI conference on artificial intelligence* (pp. 2724–2730). AAAI Press, URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8234.

OGC (2024). OGC GeoSPARQL - A geographic query language for RDF data. (Accessed 1 July 2024), https://www.ogc.org/standard/geosparql/.

OpenStreetMap Statistics (2024). (Accessed 1 July 2024), https://wiki.openstreetmap.org/wiki/Stats.

Pörner, N., Waltinger, U., & Schütze, H. (2019). BERT is not a knowledge base (yet): Factual knowledge vs. Name-based reasoning in unsupervised QA. CoRR arXiv:1911.03681.

Punjani, D., Iliakis, M., Stefou, T., Singh, K., Both, A., Koubarakis, M., et al. (2020). Template-based question answering over linked geospatial data. CoRR arXiv:2007.07060.

Randell, D. A., Cui, Z., & Cohn, A. G. (1992). A spatial logic based on regions and connection. In B. Nebel, C. Rich, & W. R. Swartout (Eds.), *Proceedings of the 3rd international conference on principles of knowledge representation and reasoning* (pp. 165–176). Morgan Kaufmann.

Regalia, B., Janowicz, K., & McKenzie, G. (2019). Computing and querying strict, approximate, and metrically refined topological relations in linked geographic data. *Transactions in GIS*, *23*(3), 601–619. http://dx.doi.org/10.1111/tgis.12548.

Renz, J., & Nebel, B. (1998). Spatial reasoning with topological information. In C. Freksa, C. Habel, & K. F. Wender (Eds.), *Lecture notes in computer science*: *1404*, *Spatial cognition, an interdisciplinary approach to representing and processing spatial knowledge* (pp. 351–372). Springer, http://dx.doi.org/10.1007/3-540-69342-4_17.

Renz, J., & Nebel, B. (1999). On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, *108*(1–2), 69–123. http://dx.doi.org/10.1016/S0004-3702(99)00002-8.

Rigaux, P., Scholl, M., & Voisard, A. (2002). Spatial databases - with applications to GIS. Elsevier, ISBN: 978-1-55860-588-6.

Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., & Lausen, G. (2016). S2RDF: RDF querying with SPARQL on spark. *Proceedings of the VLDB Endowment*, *9*(10), 804–815. http://dx.doi.org/10.14778/2977797.2977806.

Sioutis, M. (2014). Triangulation versus graph partitioning for tackling large real world qualitative spatial networks. In *26th IEEE international conference on tools with artificial intelligence* (pp. 194–201). IEEE Computer Society, http://dx.doi.org/10.1109/ICTAI.2014.37.

Stadler, C., Lehmann, J., Höffner, K., & Auer, S. (2012). LinkedGeoData: A core for a web of spatial open data. *Semantic Web*, *3*(4), 333–354. http://dx.doi.org/10.3233/SW-2011-0052.

Stocker, M., & Sirin, E. (2009). PelletSpatial: A hybrid RCC-8 and RDF/OWL reasoning and query engine. In R. Hoekstra, & P. F. Patel-Schneider (Eds.), *CEUR workshop proceedings*: *529*, *Proceedings of the 5th international workshop on OWL: experiences and directions*. CEUR-WS.org, URL: http://ceur-ws.org/Vol-529/owled2009_submission_20.pdf.

Vasardani, M., Winter, S., & Richter, K. (2013). Locating place names from place descriptions. *International Journal of Geographical Information Science*, *27*(12), 2509–2532. http://dx.doi.org/10.1080/13658816.2013.785550.

Verga, P., Sun, H., Soares, L. B., & Cohen, W. W. (2021). Adaptable and interpretable neural MemoryOver symbolic knowledge. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tür, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, & Y. Zhou (Eds.), *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies* (pp. 3678–3691). Association for Computational Linguistics, http://dx.doi.org/10.18653/V1/2021.NAACL-MAIN.288.

Westphal, M., Wölfl, S., & Gantner, Z. (2009). GQR: A fast solver for binary qualitative constraint networks. In *Benchmarking of qualitative spatial and temporal reasoning systems, papers from the 2009 AAAI spring symposium*: *Technical report SS-09-02*, (pp. 51–52). AAAI, URL: http://www.aaai.org/Library/Symposia/Spring/2009/ss09-02-011.php.

Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage..

Younis, E. M. G., Jones, C. B., Tanasescu, V., & Abdelmoty, A. I. (2012). Hybrid geo-spatial query methods on the semantic web with a spatially-enhanced index of DBpedia. In N. Xiao, M. Kwan, M. F. Goodchild, & S. Shekhar (Eds.), *Lecture notes in computer science*: *7478*, *Geographic information science - 7th international conference, GIScience 2012, columbus, OH, USA, September 18-21, 2012. proceedings* (pp. 340–353). Springer, http://dx.doi.org/10.1007/978-3-642-33024-7_25.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., et al. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, *59*(11), 56–65. http://dx.doi.org/10.1145/2934664.