A Computational Framework for Organizing and Querying Cultural Heritage Archives

JAN DE MOOIJ, CAN KURTAN, JURIAN BAAS, and MEHDI DASTANI, Utrecht University

Now that within the humanities more and more data sources have been created, a new opportunity is within reach: the searching of patterns spanning across data sources from archives, museums, and other cultural heritage institutes. These institutes adopt various digitization strategies based on differences in selection procedures. This results in heterogeneous data sources with a huge impact on the accessibility and interoperability of data within and between these distributed collections. We identify three interrelated challenges that researchers may encounter when querying such distributed data sources, namely *query formulation*, *source selection*, and *alignment of data sources*. We present a multiagent architecture to overcome these challenges and discuss a prototype implementation of the architecture by developing and integrating various technologies. To measure and validate the performance of integrated technologies that meet these three interrelated challenges, we propose a methodology for setting up and conducting experiments. We take an existing data source for which we can establish a baseline query result, against which we measure the precision and recall performance, and create various sets of data sources with realistic characteristics. We report on the results of a number of experiments that show the performance of the developed and integrated technologies.

CCS Concepts: • **Information systems** → **Distributed retrieval**; Users and interactive retrieval; Search interfaces; **Retrieval effectiveness**; **Combination, fusion and federated search**; **Entity resolution**; **Federated databases**; **Digital libraries and archives**; *Resource Description Framework (RDF)*; • **Computing methodologies** → *Multi-agent systems;*

Additional Key Words and Phrases: Multi-agent systems, entity disambiguation, alignment, validation by simulation

ACM Reference format:
Jan de Mooij, Can Kurtan, Jurian Baas, and Mehdi Dastani. 2022. A Computational Framework for Organizing and Querying Cultural Heritage Archives. *J. Comput. Cult. Herit.* 15, 3, Article 45 (September 2022), 25 pages.
https://doi.org/10.1145/3485843

1 INTRODUCTION

In the near past, numerous museums, cultural heritage archives, and research institutes have created new or digitized existing data sources, with the purpose of creating the opportunity to search for patterns in the production and consumption of culture. However, since these patterns often span across the broad domain of cultural heritage, data from outside collections should be considered to expose the broad picture. For this reason, more and more of these institutes try to integrate data from outside data sources, which have been created with different

This work was done in the context of the Golden Agents project (www.goldenagents.org), funded by the Netherlands Organization of Science NWO—Large Investments program.
Authors' address: J. de Mooij, C. Kurtan, J. Baas, and M. Dastani, Utrecht University, Utrecht, The Netherlands; emails: a.j.demooij@uu.nl, a.c.kurtan@uu.nl, j.baas@uu.nl, m.m.dastani@uu.nl.

1556-4673/2022/09-ART45 $15.00
https://doi.org/10.1145/3485843

selection procedures and with different audiences and uses in mind. The resulting heterogeneity has an enormous impact of the accessibility and interoperability of these distributed data sources. Although Digital Heritage and Semantic Web aim at making it possible to use data from multiple providers, there are still many challenges in organizing and integrating these data sources and making them accessible and searchable for non-technical users.

We aim at enabling integrated search on cultural heritage data sources while keeping them distributed and without forcing changes to the source of the data. We thus assume that various data sources are owned and maintained by various parties while they are accessible for search from outside. Moreover, we aim at enabling non-technical users to use these tools. To achieve these objectives, we focus on the following challenges:

(1) *Query formulation*: The users of the system should be supported in formulating queries, as they cannot be assumed to be able to formulate the correct query right away. Users may be non-technical, may not know exactly what to look for, or may realize that a query can be improved upon encountering new data.
(2) *Source selection*: Complex queries that need data from various collections require careful selection of data sources since the quality of answers is highly related to the relevance of particular data sources in relation to their research questions. This requires automatically deciding and querying relevant data sources, and combining the resulting data. These require in turn some meta-knowledge about the data sources.
(3) *Alignment*: Data sources may use different schemes to represent data even for overlapping content in the same domain. This problem may appear at both the data records and schema representation of the data source. For a large number of data sources, it is not feasible to solve these interoperability problems manually, and therefore semi-automatic alignment approaches are required.

This work has been performed in the context of the Golden Agents project.[1] The project focuses on data sources regarding the cultural production and consumption in the so-called Dutch Golden Age. A great number of creative industries came to blossom in that age (painting industry, book industry, silverware industry, etc.). What mechanism (on the part of the buyers as well as the sellers) set and kept this blossom in motion is a question that could not be answered in pre-digital times. These are some of the research questions that the infrastructure the Golden Agent project builds and is aiming to resolve. For the types of data sources at hand, the **Resource Description Framework (RDF)** is commonly used. We thus assume that digital data sources can be queried using the SPARQL query language, but that these data sources cannot be expected to use consistent and compatible ontologies and identifiers. We propose a multiagent system to meet the listed challenges in a structured manner. The system consists of three agent types organized in a hierarchical structure. The *user agent* supports a user to formulate queries and get results back. The *broker agent* selects data sources, distributes subqueries, and aggregates the results. Finally, the *data source agent* encapsulates a digital data source to provide data for a request coming from the broker.

To measure and validate the performance of the developed multiagent technology, an accurate estimate of the expected performance of the distributed search must be available. Since such an estimate for a set of existing data sources is hard to determine, if not impossible, we propose a methodology for setting up experiments that measure the performance of the integrated technologies in terms of precision and recall. We take an existing data source and create various sets of data sources with some real characteristics. We distribute the single data source, rather than using a set of existing data sources, to gain control over the experimental setup. In particular, the distribution of one single data source allows us to use the results obtained through queries on the single data source as a baseline to determine the expected results for any given query. We report on a number of experiments and analyze the performance of the adopted technologies in the developed multiagent system. We would like to emphasize that the employed methodology to use distributions within one data source is just to evaluate

[1] https://www.goldenagents.org.

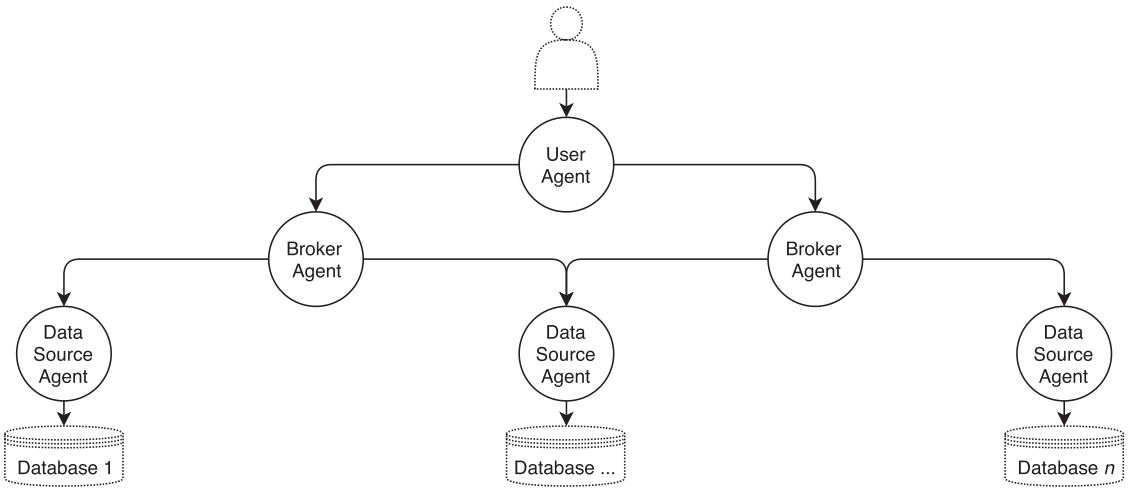


Fig. 1. An overview of the multiagent system.

the performance of the developed technology, and that the developed multiagent technology can be applied to existing distributed heterogeneous data sources.

The structure of this article is as follows. Section 2 presents the multiagent architecture, and Section 3 explains the details of its implementation in terms of the technologies that are developed and deployed in the multiagent system. Section 4 presents the proposed methodology for setting up experiments for validating different aspects of the developed multiagent system, and Section 5 reports the results of the conducted experiments. Section 6 discusses and compares our approach with some existing related work, and Section 7 concludes the article.

2 A MULTIAGENT SYSTEM ARCHITECTURE

In this section, we describe the multiagent system architecture consisting of user, broker, and data source agents. A preliminary version of this architecture was presented in our earlier work [7]. The general multiagent system architecture is illustrated in Figure 1. We emphasize that this architecture is schematic. A particular multiagent system can consists of various user agents, each representing one particular user, one or more broker agents that can communicate with each user agent, and one or more data source agents, each with access to one data source. We assume a *general ontology* that fully describes the relevant domain in a consistent and unambiguous manner. Moreover, we assume that the ontologies used in the participating data sources may be different from the general ontology, but they can be mapped to the general ontology.

The query process starts with a user who is interested in a question relating to facts in one or more connected data sources. The user interacts with the system through the user agent, which among other things supports the user to express the question in a natural and intuitive manner. The user agent does this by providing an interface through which the user can formulate queries by means of options or suggestions (e.g., properties) that the user has to select from. Moreover, the user can at any time re-formulate a query by constraining or relaxing the query. The user agent informs the user what data sources are available, and allows the user to use all (by default) or specify exactly what data sources to consult. The automatically generated formal query is then passed on by the user agent to one or more broker agents. The broker agent then sends the query to the relevant data source agents, collects the answers, and presents the results back to the user agent. At this stage, the user can view and download the results for further analysis, work together with the user agent to further improve the query based on the provided results, or ask the user agent to visualize the agents' decisions underlying the constructed

answer, including tracing the obtained results back to the data sources. The main purpose of the user agent is to hide the complexity of the formal query formulation for distributed data sources from novice users.

The main task of a broker agent is to organize the query distribution process and to automate data integration. The broker agent uses the general ontology to learn which concepts each individual data source agent can answer. The concepts from the general ontology that can be answered by a data source agent make up its *capabilities*. The broker agent uses these capabilities to select data source agents, which can provide results to incoming queries. Thus, when a new query is received, the broker agent splits it into one or more subqueries based on its knowledge of the capabilities of the data source agents. These data source agents send the answers for their assigned subqueries back to the broker agent, which joins these intermediate results to form a final answer to the original user query. Additionally, multiple broker agents may collaborate, for instance, on distributing the workload, or each broker can be more knowledgeable about certain aspects of a query.

Each data source is managed by a data source agent that can be understood as a representative of the data source it encapsulates. Data source agents are capable of querying SPARQL endpoints—making them work with existing RDF data structures—or raw database files. The main task of the data source agents is to answer specific subqueries sent to them by the broker agents. The broker agents are assumed to be agnostic about the ontologies used in the local data sources. To this end, each data source agent is capable of translating its received subquery to the local ontology, and makes sure results that are sent back are presented using concepts from the general ontology. To be included in the query process, a data source agent constructs a local expertise model, representing the capabilities of the data source it encapsulates, and shares that expertise with the broker agents when requested. The model is constructed by indexing concepts used in the local data source through automatically generated queries, and communicated in the vocabulary of the general ontology.

Data providers can instantiate data source agents for their own data sources, and announce that agent to the multiagent system to make their data source available. They can use the default agent implementation or make changes to reflect their own values. For example, fine-grained access control not possible with regular SPARQL endpoints, or domain-specific reasoning to improve result quality, could be included. Most importantly, however, a data source agent can run on a server maintained by the data provider itself such that the data provider remains in full control.

3 MULTIAGENT SYSTEM IMPLEMENTATION

In this section, we explain an integrated implementation of the multiagent system architecture. For the sake of readability, the implementation is explained by means of a simplified example of the life and work of the well-known Dutch poet, writer, and playwright Joost van den Vondel. For this example, we use three data sources shown in Table 1. The overall domain of these three data sources is writers and their creative works, but each data source provides only part of that domain. In this example, the first data source, DB1, provides the biographical data of persons, whereas the third source, DB3, provides details on the creative works. The second data source provides both biographical data of persons and of their works. Integrating this information will help us answers questions such as "In which stage of their lives were authors the most productive, or most well received by their buyers?" It should be noted that the example used in Table 1 is simplified and shows only the triples in each data source, without using the full IRIs. In RDF data sources, an IRI is placed in angle brackets (e.g., <http://www.vondel.humanities.uva.nl/ecartico/persons/9696>). When many IRIs from the same domain are used, a prefix can be defined (e.g., `PREFIX ecartico: <http://www.vondel.humanities.uva.nl/ecartico/>`) after which each IRI from that domain can be denoted as, for example, `ecartico:persons/9696`. To simplify the presentation of the triples, here we use the prefixes `db1`, `db2`, and `db3` to indicate the different data sources using different IRIs, without specifying what those IRIs are.

The multiagent system has been implemented in the Java-based edition of the agent programming language 2APL [5, 6]. All agent communication, including query requests and the sending of query answers, takes place

Table 1. Example of three Data Sources: DB1, DB2, and DB3

DB1	DB2	DB3
person1 db1:name ‘‘Vondel’’.	person4 db2:authored work1.	work1 db3:title ‘‘Gijsbrecht van Aemstel’’.
person2 db1:name ‘‘Scriverius’’.	person2 db2:authored work2.	work2 db3:title ‘‘Batavia Illustrata’’.
person1 db1:birth ‘‘1587’’.	person4 db2:bornOn ‘‘1586’’.	work1 db3:published ‘‘1637’’.
person2 db1:birth ‘‘1576’’.	person4 db2:hasName ‘‘Vondel’’.	work2 db3:published ‘‘1609’’.
		work1 db3:authoredBy person1.

Table 2. Mappings of the Data Sources in the Example

Data Source	General Ontology	Mapping	Local Ontology
DB1	ga:name	owl:equivalentProperty	db1:name
	ga:birth	owl:equivalentProperty	db1:birth
DB2	ga:authored	owl:equivalentProperty	db2:authored
	ga:birth	owl:equivalentProperty	db2:bornOn
	ga:name	owl:equivalentProperty	db2:hasName
DB3	ga:title	owl:equivalentProperty	db3:title
	ga:published	owl:equivalentProperty	db3:published
	ga:authored	owl:inverseOf	db3:authoredBy

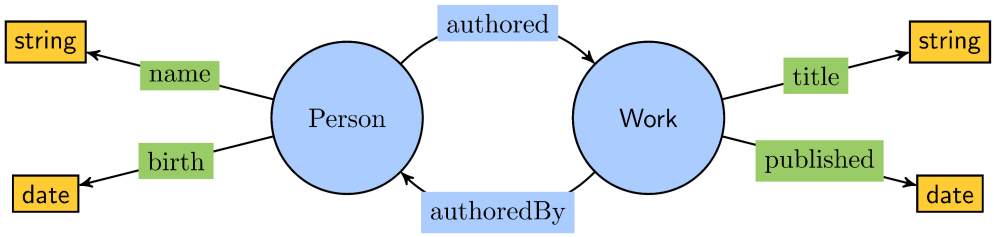


Fig. 2. Example subset of the general ontology used in the Golden Agents project.

through message passing. In this implementation, the general ontology we use is specific to the domain of cultural heritage [38]. The part of this ontology relevant to our example is visualized in Figure 2. For all participating data sources, a mapping to this ontology is provided in separate files, which use the conventional OWL[2] (Web Ontology Language) semantics for specifying relations between concepts. We have given three example mappings, one for each data source in our example, in Table 2. Additionally, *linksets* containing equivalence relations between entity IRIs across data sources are provided [20]. The linkset in the running example would consist of the single triple: `person1 owl:sameAs person4`. By means of the general ontology and linksets, the broker and data source agents align the data at both ontology and instance levels during query processing. How the ontology, the mappings, and the linksets are used will be discussed later in this section.

For evaluating SPARQL queries on RDF graphs or materializing OWL reasoning rules, the broker and data source agents use Apache Jena.[3] We restrict the class of allowed queries to a subset of the SPARQL query language. Specifically, we do not currently support the Basic Graph Patterns `OPTIONAL`, `NOT` (negative lookup), and `UNION`. Negative lookup requires exhaustive search of the data, which is not feasible for distributed querying. `OPTIONAL` and `UNION` are currently not propagated to the data source agents. In the case of `OPTIONAL`, this would already drop candidate results in the participating data sources before they can be accepted as optional in the aggregate

[2]https://www.w3.org/TR/owl-ref/.
[3]https://jena.apache.org/.

results. Note that both of these keywords increase the expressivity of SPARQL, but the same results can be obtained by constructing multiple well-crafted queries without these keywords.

3.1 User Agent

A user interacts with the user agent through a web interface. Expert users with technical background are given the opportunity to write formal SPARQL queries, similar to interfaces to existing SPARQL endpoints such as the Virtuoso interface of DBpedia.[4] This allows expert users to write SPARQL queries over distributed data sources without knowledge of availability or content of individual data source.

Example 3.1 (Query). Consider the expert user wants to query the integrated data sources to find the names of all authors who have published some work, as well as the title of those works. They can then construct the query shown here on the left, with the answers we would expect from the data sources from Table 1 on the right:

```
SELECT * WHERE {
    ?person ga:name ?name .
    ?person ga:authored ?work.
    ?work ga:title ?title
}
```

?person	?name	?work	?title
person1	“Vondel”	work1	“Gijsbrecht van Aemstel”
person2	“Scriverius”	work2	“Batavia Illustrata”

For users less familiar with SPARQL, we have implemented a search browser based on the concepts of SPARKLIS [9]. SPARKLIS makes use of an abstraction of SPARQL, called *LISQL* [10], which instead of a set of triple patterns uses a tree-like structure to represent a query. Instead of having to write a query manually from scratch, users are allowed to build their query step-by-step—users are given the opportunity to inspect results for each intermediate query—by selecting options provided by the search interface. These options come in the form of classes and properties, and are carefully selected and updated each time the user extends the query to ensure no suggestion will result in a query for which there are no answers. SPARQL finds these suggestions automatically by consulting the data source under consideration, but only works on a single RDF graph. We have extended the approach to work on multiple data sources with different ontologies by having the broker request and combine search suggestions from the individual data source agents.

The LISQL abstraction of the query is typically easier to read than formal SPARQL, because all variables are made implicit and hidden away. This allows the query to be represented in a near-natural language conjunction of selection criteria. For example, the LISQL representation of the query from Example 3.1 would be

```
? and ( (name : ?) and ( authored : ( ? and (title : ?) ) ) )
```

(with each question mark representing each of the variables in the query from Example 3.1), which is easily translated to its natural language representation that can be displayed to the user:

```
Give me everything
    that has a name
    and that has authored something that has a title
```

Any LISQL query can recursively be unambiguously translated to a SPARQL query, at which point the implicit variables are made explicit. The key contribution of LISQL is the notion of a *focus*, which represents the node in the LISQL query tree at which suggestions are applied or, equivalently, what variable the added selection criterion is applied to. When the search suggestion selected by the user contains a property, the focus will automatically change from the subject of that property to the object, which allows exploring complex chains of relations in the RDF graph. At any time, the user can freely change the focus by clicking on the relevant part of the query. This greatly improves the expressivity of the query language over faceted search browsers, because the user may go

[4] https://dbpedia.org/sparql.

back to apply additional selection criteria to earlier parts of the query. In traditional faceted search browsers, such additions would require the user to rebuild the query from scratch. For example, to obtain the LISQL query given earlier, it does not matter if the user first selects the name as a query extension or starts with the book and its title when building the query. Moreover, focus change allows constructing queries that are not possible with traditional faceted search browsers, without trading in expressivity.

When the query is ready, the user agent starts the querying process. The user agent verifies the syntax of the query before it is evaluated. If any errors in the query syntax are found, suggestions for repairing the query are presented to the user. Otherwise, the user agent sends the query to the broker agent, which then starts processing the query. By default, all available data sources are considered, but if a user is knowledgeable about the data sources in question, and wants to exclude some, they have the option to do so before sending the query. When the broker agent has finished the querying process, it sends the result back to the user agent. The user agent displays these results to the user in a table where they can be inspected or exported as a CSV or XML file for further analysis. For each variable in the query, all data sources that contributed some values are listed, allowing the user to trace answers back to the data sources. If the user is unsatisfied with the results, they can edit their query and evaluate it again, optionally specifying different data sources to consult.

3.2 Broker Agent

When the broker agent receives a user query, it uses the process given in Algorithm 1 to answer the query. First, for each triple pattern in the query, it creates a set of candidate data source agents. Note that queries are restricted to non-nested conjunctive queries such that each conjunct corresponds with a triple pattern. A data source agent is marked as a candidate for a triple pattern if the predicate and class that occur in that triple match the capabilities of the data source agent (i.e., a data source agent is a candidate for a triple pattern if it can answer the triple pattern). Note that a data source agent can be marked as a candidate for several triple patterns and that a triple pattern may have several data source agents as candidate. For the running example, this step results in marking `DB1` and `DB2` as the candidates for the triple pattern including `ga:name` and, similarly, `DB2` and `DB3` as the candidates for the one including `ga:authored`.

The next step in the process is to use a split function that for each triple pattern from the user query decides which data source agents that are marked as a candidate for the triple pattern should answer the triple pattern (line 1). Note that the split function can follow different approaches by deciding one or more candidate data source agents to provide answers for a triple pattern (see the following for two different approaches). Each data source agent receives then a (sub)query that is formulated as the conjunction of the triple patterns that are decided for the data source agent (see Listing 1 in Section 3.3 for an example subquery). Each data source agent is expected to reply with a collection of RDF triples that answer the (sub)query (line 3). In the next step, the broker agent collects all of these triple patterns in a Jena model to create a temporary virtual graph containing the aggregated data (lines 5–8). For the running example, a virtual graph can contain RDF triples `person2 ga:name ''Scriverius''` and `person2 ga:authored work1` together by collecting data from both `DB1` and `DB2`. The broker agent then adds the so-called linksets of all participating data sources to this model (line 10) and uses the Jena reasoner to materialize the equivalence relations (line 11). As will be detailed in the next section, adding the linkset ensures that when different identifiers (IRIs) are used across data sources, they resolve to the same entity. Finally, the broker evaluates the user SPARQL query on the temporary model (line 12) and streams the result back to the user agent as shown in Example 3.1.

Once the broker agent lists data source agents as candidates for triple patterns of a given query, it splits the query by assigning data source agents to each triple pattern. How the broker agent splits the query and assigns subqueries to the data source agents plays a crucial role for the query answering function. We implement two different approaches for splitting a query: a *naive approach* and an *expertise approach*.

3.2.1 Naive Approach. The naive approach assigns to a data source agent all triple patterns for which the data source agent is marked as a candidate. The triple patterns that are assigned to a candidate data source agent are

ALGORITHM 1: *answerQuery*(Q, A, L, R)

```
Input: Query Q, Set of Agents A, Linkset L, Reasoner R
Result: Resultset S
SQ ← split(Q, A)                                   // create subqueries
foreach sq ∈ SQ do
    send(sq)                                       // send to agent
end
M ← init()                                         // initialize a model
/* async partial data source                                        */
foreach P ← collect() do
    M.add(P)                                       // add partial model
end
M.add(L)                                           // add linkset
M.reason(R)                                        // reason over the collected data
S ← M.execute(Q)                                   // execute query
return S
```

combined into a conjunctive (sub)query and sent to the candidate data source agent. This approach is expected to produce duplicates in the results because more than one data source agent can provide RDF triples for the same triple pattern in their answers to the subqueries. In the case of the running example, both DB1 and DB2 would provide the name "Vondel" for the same person, and this would duplicate the result rows. However, asking for many triple patterns in one subquery may cause loss of information because each triple pattern in a query is a restriction on the entities that the agent returns as the result. For instance, the subquery of DB2 does not return RDF triples for person2 because the subquery consists of triple patterns with ga:authored and ga:name.

3.2.2 Expertise Approach. The expertise approach assigns each triple pattern to only one of the data source agents marked as a candidate for the triple pattern. This approach selects one data source agent for a triple pattern based on its expertise on the triple patterns. To enable this method, a broker agent needs to build an expertise model [22] that stores information about the performance of individual as well as pair-wise collaboration of the data source agents. For example, how many entities a data source agent can individually provide for a single triple pattern is considered as a performance measure of the data source agent. In addition, how many entities two data source agents can cooperatively/jointly provide for two separate single triple patterns can be considered as the (pair-wise) cooperation performance measure of the data source agents. In Figure 3, we present a part of the expertise graph of the running example, where node $v1$ represents the assignment of ga:name to DB1 and its value is equal to the individual performance of DB1. Similarly, the node $v3$ represents the assignment of ga:authored to DB2 and the edge between these two nodes represents their cooperation performance for the corresponding assignments.

In this work, we use the expertise graph to assign the best-performing data source agent for each triple pattern based on the average cooperation performance for the corresponding triple pattern. This is one of the proposed metrics for the expertise graph (see the work of Kurtan et al. [22] for more information on other metrics). The cooperation performance value is computed as follows. For two different capabilities, let $pf\ (e)$ represent either the joint work of two data source agents d_1 and d_2 connected by a j-edge e or the work of one data source agent connected by a c-edge e in the expertise graph. The average cooperation performance of a data source agent d for capability c, denoted by cpt (d, c), is determined as

$$\text{cpt}\ (d, c) = \sum_{e \in E} \text{pf}\ (e)/|E|,$$

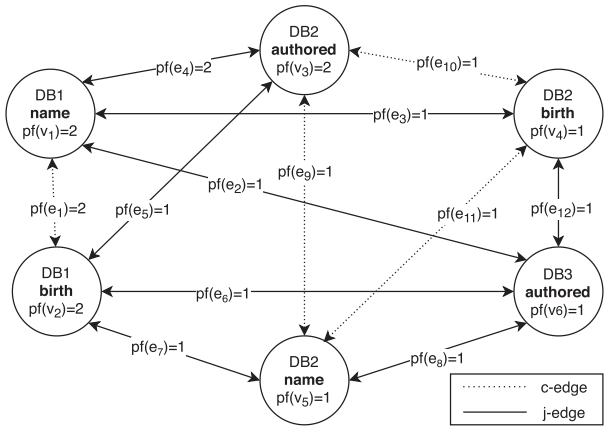


Fig. 3. Expertise graph of the data sources in the running example.

where E denotes the j-edges of the node that represents the assignment of data source d to capability c in the expertise graph. In the running example, the cooperation performance value of data source DB1 for capability ga:name is calculated through the j-edges of $v1$ and the result is 1.33, whereas the cooperation value of DB2 for the same capability (node $v5$) is 1. Note that the expertise approach does not produce duplicates in the results, but it may lose information because it selects a single data source agent for a triple pattern. For the running example, DB1 and DB2 provide different ga:birth information for the same person (person1 in DB1 and person4 in DB2), but the expertise approach selects only one of them guided by the expertise graph. It is important to note that both approaches suffer from information loss, but in two different ways. The expertise approach may lose information that multiple data source agents can produce together, whereas the naive approach may lose information that a single data source agent can send because of the excessively constrained queries.

3.3 Data Source Agent

When a data source agent receives a query request, it generates a CONSTRUCT query that uses all triples specified by the broker. With the CONSTRUCT keyword, SPARQL allows aggregating results as an RDF graph, which is a collection of triples constructed based on the query. By using concepts of the local data source in the query body, but taking the translations of those concepts from the general ontology and placing them in the head, results are automatically returned to the broker agent in the vocabulary of the general ontology. More complex mappings can be dealt with this way as well. An example of this approach is provided in Listing 1, in which we show a possible query that can be sent to DB3 (Table 1). Note that the ontology from Figure 2 contains a property authored that occurs as as the inverse variant authoredBy in DB3. The mapping from Table 2 specifies how to translate those concepts, which is used in the example CONSTRUCT query.

```
CONSTRUCT {
    ?person ga:authored ?book .
    ?book ga:title ?title
} WHERE {
    ?book db3:authoredBy ?person .
    ?book db3:title ?title
}
```

Listing 1. Example CONSTRUCT query used for translation.

A data source can be created either by specifying the URL of a SPARQL endpoint, allowing existing infrastructure of knowledge institutes to be used, or by giving it direct access to the data source files formatted as, for example, Turtle, TDB(2), N-Triples, or JSON-LD. The data source agent autonomously determines query strategies based on the method with which the data is provided. For example, an edge case of SPARQL endpoints that the data source agent deals with is that some of these servers limit the number of results that a single query can yield. If the result size is larger than this number, the SPARQL endpoint will silently drop results beyond that limit without notice. Since the limit is usually constant for a given endpoint, the data source agent can determine if such a limit exists and, if so, what that limit is in advance. When querying the endpoint, it iteratively requests batches of data smaller than this maximum result size by including a `LIMIT` clause, and keeps track of the progress to continue querying until all results are collected.

3.4 Linkset Generation

In existing cultural heritage data sources, different IRIs may be used to represent the same real-world entity. This can, for instance, be caused by the use of different schemas among data sources, or because identical counterparts in another data source were not known such that another IRI is generated for it. The process of linking all identical entities is called *entity alignment*. The output of the entity alignment process is a linkset: a set of entity pairs where each pair represents an equivalence relationship. In the following, we explain the linkset generation process without going into technical details while providing reference to the technologies we use. First, we would like to emphasize our working assumption that an entity and its identical counterparts have certain properties (e.g., names and birthdays) in common. Since the data is stored as an RDF graph, this directly translates to two nodes both having edges to the same literal-node in the graph. We assume that nodes that are close by to an entity should also be close by to any identical counterpart entities.

In the linkset generation process, we may first remove from the RDF graphs those nodes and predicates that are not relevant for the entity alignment. Moreover, for all predicates, we ensure that a single literal-node exists per unique literal value. For example, for all triples of the form (?subject `schema:name` "Vondel"), in the RDF graph all subject nodes have an edge to the same "Vondel" literal-node. In the running example, this would equate to the `person1` entity from DB1 and `person4` entity from DB2 both having an edge to the same "Vondel" literal-node.

We then generate a neighborhood (context) for each entity we wish to potentially align, such as all persons. This neighborhood consists of other nodes, such as other entities and properties, that can be reached from the starting node by traversing the RDF graph. We use a modified version of the Bookmark Coloring Algorithm [4] to accomplish this. The neighborhoods of the entities to be aligned are then used as input for the GloVe [25] algorithm that creates an embedding of the entities. The embedding assigns to each entity, such as a person, a coordinate in a multi-dimensional Euclidean space in such a way that the coordinates of the entities that have a strongly overlapping context are moved closer to each other. This allows us to compare the coordinates of any pair of entities and to suggest linking entities that are significantly close to each other. As the number of possible links grows quickly, we create for each entity e from the embedding the set of potential candidate links, described in terms of their mutual Euclidean distances. In the running example, since `db1:person1` and `db2:person4` share a common context (the "Vondel" literal-node), both entities would point to each other as their respective candidate link.

Finally, we assume that a small number of candidate links are labeled as being either correct or incorrect. This can be done manually or, for instance, guided by software such as Lenticular Lenses [21]. In the case of the running example, a domain expert could judge the `person1`, `person4` candidate link as correct. These labeled links are then used to train a machine learning algorithm to classify the remaining candidate links. The final linkset consists of all candidate links that the classifier judges as correct. Note that this method leaves the manner of classification open to any machine learning algorithm that can perform binary classification.

4 EXPERIMENT SETUP

To validate the proposed multiagent system for querying distributed data sources, a number of experiments have been conducted. These experiments are designed to measure the performance of each adopted technology as well as their integration in the multiagent system in terms of precision and recall. To measure and validate the performance of distributed querying mechanisms, we set up experiments by taking an existing data source, represented as an RDF graph, and creating various sets of data sources, called *data distributions*. We distribute a single data source, rather than using a set of existing data sources, to gain control over the experimental setup. In particular, the distribution of one single data source allows us to use the results obtained through conventional SPARQL queries on the single data source as a baseline to determine the expected results for any given query. In this section, we discuss the existing data source that is used for these experiments and explain how the data source is distributed into multiple data sources. We further explain how entities are delinked, as to allow validating the use of machine learning for linkset generation. Last, we discuss how the experiment are performed using the multiagent system.

4.1 Creating Data Sources and Queries

Ecartico[5] is a data source that consists of biographical information on painters, engravers, book sellers, and gold- and silversmiths. These people worked in the Low Countries (the Southern as well as Northern Netherlands; later on "the Dutch Republic") at the time of the 16th and 17th centuries. When it comes to patterns in production, this is one of the key data sources in the Golden Agents project, for it contains data about products and producers. For data on consumption, we mostly rely on archival data sources. Ecartico satisfies a number of properties important for our purposes. First, it is actively curated, so we can be sure that there are no duplicate records of persons present, which would be a source of potential errors we cannot account for. Second, person records have a sufficient number of properties to be split across multiple subsets while still maintaining some overlap (i.e., allowing for properties that occur in more than one subset). Third, the number of person records is large enough to make a manual approach for linkset generation challenging.

From the Ecartico graph, we have constructed a new graph containing all `schema:Person` entities that have values for the properties `schema:name`, `schema:workLocation`, and `schema:hasOccupation`. For each of these entities, we also copy the property-value pairs for `schema:birthDate`, `schema:deathDate`, `schema:birthPlace`, and `schema:deathPlace` when they are present. For all properties, we follow the syntax specified by the schema ontology. This resulted in a graph with—including `rdf:type` for the class `schema:Person`—eight properties, all centered around that one RDF class. This process ensured that we have a fixed domain to work with. We refer to this graph as the *baseline graph*.

The baseline graph can be distributed in various ways; however, we aim at doing this in a way that the resulting data distributions reflect some aspects of real-world scenarios. One such scenario is when the institutions have focused on different aspects of a persons life—for example, one institution's data source may have detailed information on artwork of the artists, whereas another focuses on biographical information. We consider two aspects in such a scenario: (1) the focus of individual data sources and (2) the extent to which properties of various data sources overlap. We see the focus of an individual data source as being determined by the importance/priority of the properties in the data source, together with the minimum number of properties for which each entity in the data source has a value. A property is considered as being more important than another one in the sense that an entity cannot have a value for the less important property unless it has also a value for the more important properties. The existence of an importance ordering together with a minimum number ensures that a data source has a focus on the minimum number of properties with the highest priority. In the rest of this article, a data source with a focus (specified by an importance ordering and a minimum number) is called

[5] http://www.vondel.humanities.uva.nl/ecartico/.

an *ordered data source*. Conversely, a data source is called *random* if it is specified by only a minimum number, without an importance ordering. Each entity in such a data source has values for at least a minimum number of properties selected randomly. Finally, we call a data source *complete* for a property if all entities in the data source have values for the property. Note that an ordered data source is complete for the minimum number of properties associated to the ordered data source.

We use the baseline graph and generate a total of six different data distributions, each containing four data sources. In particular, we generate three so-called ordered data distributions consisting of ordered data sources and three so-called random data distribution consisting of random data sources. The extent to which the data sources within ordered- and random data distributions share information is determined by the minimum number of properties assigned to the data sources in that data distribution: the higher the minimum number is, the larger the overlap of properties between data sources, and thus the more information they share. The data distributions are generated based on a configuration file specifying the parameters of each of the six data distributions. For each data distribution, the first parameter specifies whether the data sources should be ordered or random, the second parameter specifies the minimum number of properties that should have values in each data source, and the final four max values specify the number of properties assigned to the four data sources. The configuration file for the six generated data distributions is specified as follows:

$\langle ordered\ ,\ min : 3\ ,\ max_{d_1} : 5\ ,\ max_{d_2} : 6\ ,\ max_{d_3} : 7\ ,\ max_{d_4} : 8\rangle$
$\langle ordered\ ,\ min : 4\ ,\ max_{d_1} : 5\ ,\ max_{d_2} : 6\ ,\ max_{d_3} : 7\ ,\ max_{d_4} : 8\rangle$
$\langle ordered\ ,\ min : 5\ ,\ max_{d_1} : 8\ ,\ max_{d_2} : 8\ ,\ max_{d_3} : 8\ ,\ max_{d_4} : 8\rangle$
$\langle random\ ,\ min : 3\ ,\ max_{d_1} : 5\ ,\ max_{d_2} : 6\ ,\ max_{d_3} : 7\ ,\ max_{d_4} : 8\rangle$
$\langle random\ ,\ min : 4\ ,\ max_{d_1} : 5\ ,\ max_{d_2} : 6\ ,\ max_{d_3} : 7\ ,\ max_{d_4} : 8\rangle$
$\langle random\ ,\ min : 5\ ,\ max_{d_1} : 8\ ,\ max_{d_2} : 8\ ,\ max_{d_3} : 8\ ,\ max_{d_4} : 8\rangle$

In the rest of this article, we use the abbreviations ordered-3, ordered-4, ordered-5, random-3, random-4, and random-5 to refer to the preceding six data distributions, respectively. Note that for ordered-5 and random-5 data distributions, the maximum value for every data source is equal to 8. Each data source within a data distribution is assigned a number of properties equal to its specified max value, which are randomly taken from the set of the eight available properties. For example, a data distribution (either random or ordered) with a specified min number of properties equal to 3 may result in four data sources $d_1, \ldots, d_4$, with the following property assignment (note that the prefix `schema:` is omitted):

d_1: `name`, `birthDate`, `deathDate`, `workLocation`, `hasOccupation`
d_2: `birthDate`, `workLocation`, `birthPlace`, `deathDate`, `hasOccupation`, `type`
d_3: `type`, `name`, `birthPlace`, `deathDate`, `workLocation`, `deathPlace`, `birthDate`
d_4: `hasOccupation`, `birthPlace`, `name`, `birthDate`, `type`, `deathDate`, `deathPlace`, `workLocation`

For each data source, we take all person entities from the original data source, in this case Ecartico, and decide for each entity the properties that should have a value. We decide a subset of the assigned properties, with the size of that subset (i.e., the number of properties assigned to that entity) being at least the minimum value. For the ordered data distributions the properties are decided from the start of the list, whereas for the random data distributions the properties are decided randomly. For example, if the specified minimum value for aforementioned data source d_1 is 3, we may take for the person entity Vondel a subset of four properties from the five properties assigned to d_1. If the ordered data distribution is used, Vondel will have values for the first four properties: `schema:name`, `schema:birthDate`, `schema:deathDate`, and `schema:workLocation`; otherwise, if the random data distribution is used, Vondel can have values for any four of the five assigned properties.

As in practice different data sources use different IRIs for the entities they describe, we delink a part of the entities in the generated data sources by replacing some of the entity IRIs. For each data source, we randomly sample a percentage of the available person IRIs independently, and append the ID of the data source to these IRIs. For example, when processing the second data source, we append '/2' to all sampled IRIs. This way, we know that the IRI's `http://ecartico.nl/1234`, `http://ecartico.nl/1234/1`, and `http://ecartico.nl/1234/3` all represent the same entity from the original data source. We have chosen three levels of sample size (10%, 25%, and 50%), where each level determines the size of the random sample of IRIs that will be altered. Note that when the sample size increases, not only will the number of altered IRIs be increased but also the probability that an IRI alters in more than one data source. It therefore becomes increasingly difficult to generate an accurate linkset.

Finally, we generate the queries to be used for testing. For each non-empty subset of the eight properties (255 in total), in the baseline graph (including the `rdf:type` of `schema:Person`), we generate one query. For each property, we create a triple pattern using the ?x variable at the subject position and a fresh variable at the object position. We concatenate all triple patterns in the set using the SPARQL dot ('.') operator—which is used for conjunction—and place the result in a `SELECT * WHERE` query. This approach yields a total of 255 queries. We evaluate each query on the baseline graph using Apache Jena to obtain the baseline results and store the results alongside the queries. To test the performance of the mappings, we translate all 255 reference queries to the general ontology as used in the Golden Agents project by replacing the class and property IRIs in the reference queries with their Golden Agents counterparts. These queries are the *test* queries, which will be evaluated with the multiagent system, and are stored alongside the reference queries and results. This translation step will allow us to verify whether the data source agent level translations described in Section 3.3 are effective.

4.2 Running Experiments

We set up and conducted experiments to evaluate the performance of various technologies used in the implemented multiagent system. In particular, we focused on experiments to evaluate the expertise query decomposition in the multiagent system for distributed query answering, the performance of the machine learning approach for linkset generation, and the integrated approach.

To isolate and evaluate the query split approaches as discussed in Section 3 (i.e., without considering the necessity of linksets for now), we instruct the user agent to send each test query to the broker agent, which first adopts the naive approach and then repeats it by adopting the expertise approach, and collect the results. This process is repeated for each of the six data distributions we generated.

After delinking some of the entities in the generated data distributions, we use the link generation approach as explained in Section 3.4 to embed entities and create candidate links. A small portion of the candidate links are then used to train a classifier by means of a machine learning algorithm. The classifier is used to identify links as either correct or incorrect. The final generated linkset consists of all candidate links that the classifier judged as correct.

Finally, to evaluate the performance of the integration of both linkset generation and query decomposition in the overall multiagent system, we again instruct the user agent to send each test query to the broker agent (again adopting first the naive and then the expertise approach) and collect the new results obtained. However, this time, the data distributions in which the sampled entities have been delinked are used to answer these queries, and the broker adopts the embedding generated linkset corresponding to the data distribution at hand. To evaluate the influence of the quality of the linksets, this process is also repeated using "ground-truth" linksets, which contain all and only valid IRI pairs, and was obtained by placing an equivalence pair for each replaced IRI with the IRI it originated from in the linkset.

5 RESULTS

The results for the experiments on the six data distributions (where no entity delinking has yet taken place) as described in Section 4 are plotted in Figure 4. We have plotted the performance of the expertise approach

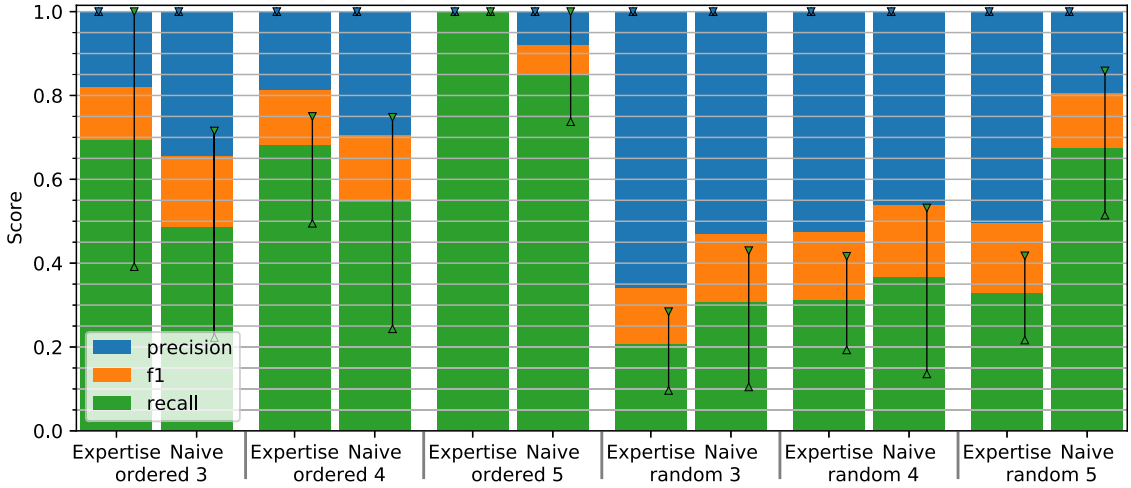


Fig. 4. Performance of the expertise approach vs. the naive approach on the three ordered and three random data distributions.

against that of the naive approach for all six data distributions, with the three ordered data distributions with all data sources having a minimum of 3, 4, and 5 properties respectively on the left, and the three random data distributions with the respective same minimum number of properties on the right. The bars indicate the average precision and recall, with the error bars displaying the 25th and 75th percentile, and the f1-score calculated from those averages. Because we calculate the f1-scores on the average recall and precision, this represents just a single data point. As such, the f1-scores do not have error bars. Note that the bars are overlaid and not stacked, which means that the bar heights indicate the absolute scores and not the aggregate score. For example, in Figure 4, the precision for all experiments was exactly 1, and the recall on the ordered-3 data distribution was 0.69 with the expertise approach and 0.49 with the naive approach.

In the rest of this section, we will discuss these results in detail, and show how this performance changes with the addition of delinked entities and the presence of the linksets.

5.1 Performance of Data Distributions

In all of the data distributions, the multiagent system reached a precision of 1 on all tested queries, indicating that the system did not introduce results that were not in the baseline. This was expected, since we use established SPARQL utilities provided by Apache Jena for combining the results from the individual data sources into the final result of the original query. Specifically, the query decomposition algorithm creates what are essentially relaxed versions of the original query and collects the results as triple patterns. The broker uses the Jena SPARQL engine to evaluate the original query on the aggregated results. It stands to reason, then, that the subqueries do neither introduce new relations to the data nor add properties or values to what is in the individual data sources. Moreover, the original query executed by the broker agent behaves similarly to how the baseline query behaves on a single graph.

Regarding the recall performance, Figure 4 shows that both the naive and the expertise approaches have a better recall performance for the ordered data distribution when compared to the random data distribution of the same minimum number of properties (e.g., compare ordered-3 with random-3). This is due to how properties were assigned to the entities when creating the data distributions and how the subqueries are constructed by the broker agent. Note that for any subquery, a data source can only return those entities that have a value for each property in that subquery and that a broker assigns a subquery to a data source agent if it can answer the subquery. In the ordered data distributions, a data source is complete for at least the given minimum number

of properties, so for queries that involve those properties, the data sources returns all entities it contains. We further observe the larger the minimum number of properties assigned to a data distribution (either ordered or random), the higher the recall is likely to be. In other words, if the entities in a data source have values for more properties, it is more likely to find more entities that have values for all triple patterns in any given query. This in turn means that for any given query, a data source with a higher minimum number of properties will be able to provide more results than a data source with a lower minimum number of properties.

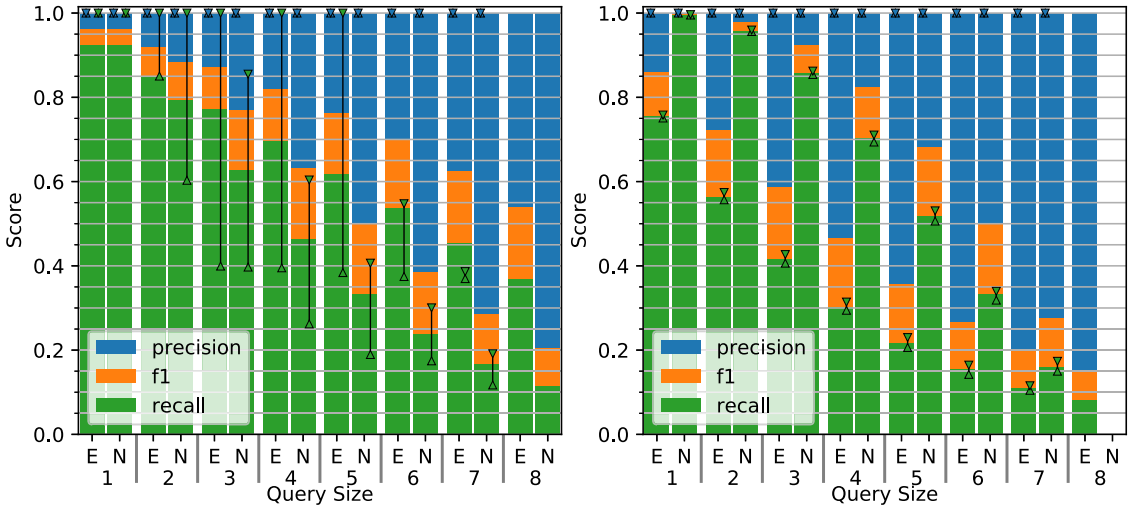
Focusing on the ordered data distributions in the left half of Figure 4, we observe that the expertise approach performs significantly better than the naive approach (with the p-value being $p < 0.01$). This is due to the fact that the expertise approach attempts to split a query in such a way that each subquery can be assigned to the one data source that is expected to provide the largest number of entities. This is done, when possible, by using in each subquery to be assigned to a data source only those properties for which that data source is complete. The naive approach, however, will over-constrain the subqueries by adding properties for which the data source is not complete (but still contains some values), leading to a lower number of results overall. One special point of attention is the recall of the expertise approach on the ordered-5 data distribution, which was exactly 1 for all 255 tested queries. This tells us that each of the 255 queries could be decomposed in such a way that each data source was complete for all properties occurring in its assigned subquery. In generating the data distributions, the properties assigned to each data source were sampled randomly from all eight available properties; we did not generate the data distributions so that in each data distribution each property was complete in at least one data source, so we believe that this is just a coincidence. This chance effect did not occur in the ordered-3 or ordered-4 data distributions, which is why some of the queries tested on those data distributions did not achieve a perfect recall. However, with only eight properties, and each data source in the data distribution being complete for at least five of those, this chance effect in the ordered-5 data distribution should not be surprising.

On the random data distributions in the right half of Figure 4, this trend is reversed. Although the overall recall is lower than on the ordered data distributions, the naive approach clearly outperforms the expertise approach. The reason is that each entity is assigned a number of properties at random, which means that the expertise approach can no longer find a single data source that is complete for all properties in the query. However, the naive approach, like on the ordered data distributions, overly constraints the query by adding all properties occurring in a source to its subquery. In short, where the expertise approach consults only one data source, and thus will be able to retrieve only a subset of the entities in that source, the naive approach consults *all* data sources. Although the number of results from each individual data source will likely be slightly lower than is the case for the expertise approach, it combines these smaller subsets for the final results, resulting in an overall better recall than that of the expertise approach.

5.2 Performance Per Query Size

Figure 5 provides further insight into the preceding analysis by showing the performance of the expertise (E) and naive (N) approaches against the number of triple patterns in the queries. We have analyzed all six data distributions in this manner but opted to show only the deconstruction of the ordered-3 (Figure 5(a)) and random-5 (Figure 5(b)) data distributions here due to space. (In Appendix A, Figure 7 presents the other four approaches.) Both ordered-4 and ordered-5 show similar trends to the plotted ordered-3 data distribution, but ordered-5 achieved a perfect recall for each of the queries using the expertise approach. We have selected the ordered-3 data distribution over the ordered-4 data distribution because the difference between the expertise and naive approach is more pronounced. Similarly, both random-3 and random-4 show similar trends to that of the plotted random-5 data distribution, but for random-3 and random-4, neither the naive approach nor the expertise approach found any results for the largest query that contained all eight properties, which is why we have opted to plot random-5 here.

It is important to note that the number of queries in each size group differs. For example, there were only eight queries of size 1, and only one query of size 8 (for the latter of which we consequently have not plotted



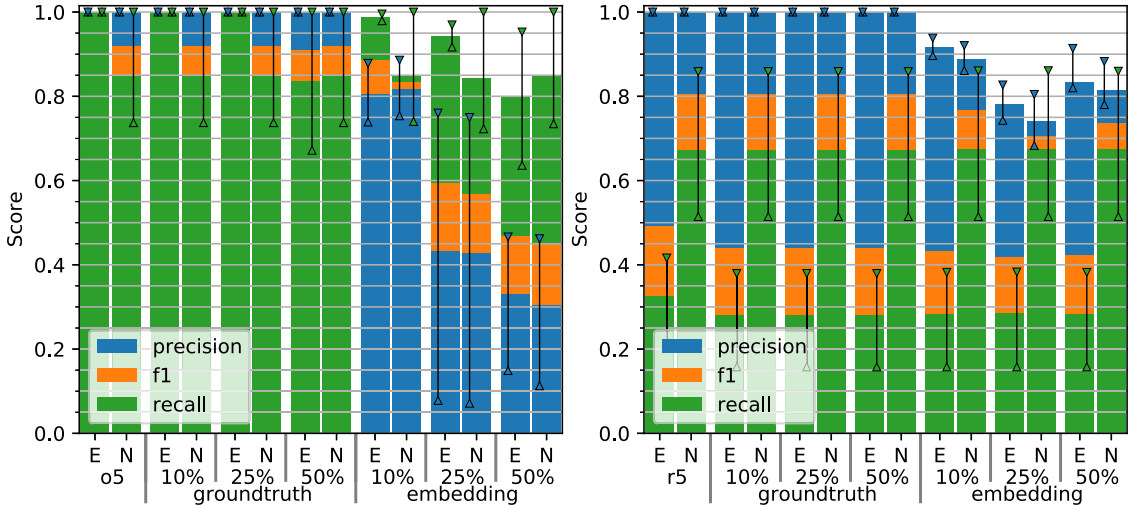
(a) ordered-3 (b) random-5

Fig. 5. Precision, recall, and f1 scores of the expertise (E) and naive (N) approaches, split out by the size of the query.

error bars). As can be seen, the recall drops as the size of the query increases. This was the case in all six data distributions we tested.

Since the plots in Figure 5 are a deconstruction of the results shown in Figure 4, there is some analogous variance in the averages between data distributions. Our experiments have shown that the larger the minimum number of properties of a data sources, the better the odds of finding most or all results. We found an almost linear decrease in recall with an increase in query size, both for the expertise and the naive approach, with the only exception being the ordered-5 data distribution, where the naive approach achieved a perfect recall on all tested queries.

We can dissect the result by looking at the performance of both approaches on the eight queries of size 1 (containing just one property in a single triple pattern). In the ordered data distribution (Figure 5(a)), we can see that the scores are exactly the same for the naive approach as for the expertise approach. In fact, both approaches found the same answers for all queries, and this was also the case in the ordered-4 and ordered-5 data distributions, on the latter of which the naive approach achieved a perfect recall as well. In all random data distributions, however, the naive approach scored significantly better than the expertise approach ($p < 0.01$) on the queries of size 1. When a query consists of only one triple pattern, it can be "decomposed" in only one way, namely into a query containing just that triple pattern. This means that both the expertise and naive approach necessarily construct the same subquery. The only difference is that the naive approach sends that subquery to all available data sources, whereas the expertise approach selects the single best source for that query. If for each property there exists a source that is complete for it, the recall will be perfect when that source is consulted. The naive approach also consults that best source, but in addition performs a lot of redundant work by also consulting all other data sources, which cannot contribute further to the result. These perfect sources existed for all properties in the case of our ordered-5 data distribution, but not for the ordered-3 and ordered-4 data distributions, on which, for some properties, both the expertise and naive approach missed a few results. In the random data distributions, no perfect source existed for any property. In this case, multiple data sources can complement each other, which is leveraged by the naive approach, whereas the expertise approach is limited to the single data source that cannot deliver all results. On the random-3 and random-4 data distributions, neither the naive nor the expertise approach found any results for the query of size 8, whereas in the random-5 data



(a) ordered 5

(b) random 5

Fig. 6. Precision, recall, and f1 scores of the expertise (E) and naive (N) approaches on the data distributions with entities delinked, using the ground-truth and embedding-generated linksets.

distribution (plotted in Figure 5(b)), the naive approach did not find any results, and the expertise approach only found a small portion. Since the query of size 8 contains all properties, this suggests that in these data distributions, at least one source was unable to provide a single entity that contained all properties that exist in that data source.

5.3 Performance of Generated Linksets

Last, we discuss the results of the experiments with the data distributions where some of the entities in the involved data sources have been delinked as described in Section 4.1. We first discuss the performance of the multiagent system using the ground-truth linksets, which essentially "undo" the delinking process. We then discuss the performance of the multiagent system using embedding generated linksets as described in Section 3.4.

Figure 6 shows the performance of the multiagent system when using linksets on the ordered-5 (Figure 6(a)) and random-5 (Figure 6(b)) data distributions, again with the performance of the expertise approach (E) set out against that of the naive approach (N). The first two bars in both of these graphs show the same scores as plotted in Figure 4 for the corresponding data distribution, so we can more easily compare the performance of the integrated system. The next three pairs of bars show the performance of the system on the data distributions where we sampled and replaced 10%, 25%, and 50% of the IRIs respectively in each data source, and where we use the ground-truth linksets to resolve these IRIs. The last three pairs of columns show the performance on the same sampled data sources, but this time with the embedding generated linksets.

Figure 6(a) shows that the performance of the naive approach did not decrease on the data distributions with the delinked entities (second to fourth pairs of bars), compared to the original data distribution (first pair of bars), when the ground-truth linksets were used. This is due to the fact that the naive approach assigns the subqueries only on the basis of the properties present in the data sources (which do not change with entity delinking) and therefore consults the same data sources, using the same subqueries, for each of the 255 queries. This shows that the process employed for resolving entity linkage in the presence of a linkset works as expected and does not introduce errors that are not in the linkset. The expertise approach achieved the same recall and precision with 10% and 25% of the entity IRIs sampled for delinking (second and third pair of bars) as it was for the original data

distribution (first pair of bars), but the recall decreased slightly on the data distribution with 50% of the entity IRIs sampled for delinking (fourth pair of bars). This is because the expertise approach estimates how many results can be combined from the various data sources but does not take into account yet the equivalence relations for entities provided by the linksets. In this case, that estimation was too low, because some results could be combined after the entity delinkage was resolved by the reasoner. In the case of the data distribution in which 50% of the IRIs were replaced, this resulted in a slightly different assignment of subqueries which, when the Jena reasoner resolved the entity IRIs, would be slightly sub-par to the original assignment. On the random data distribution, we can see that both the expertise approach and the naive approach achieved the same performance under the ground-truth links on all three data distributions with entity delinking (the second, third, and fourth pairs of bars), but for the expertise approach, this score was slightly lower than for the original data sources (the first pair of bars). This is again because the expertise approach underestimates results that can be combined when linksets are used.

The three rightmost pairs of bars in Figures 6(a) and 6(b) denote the performance when the linksets are automatically generated using the embedding techniques as described in Section 3.4. As can be observed, the performance of the multiagent system drops as more entities are delinked. This is due to the fact that as the number of delinked entities increases, more potential links can be generated. As a consequence, the number of incorrectly identified links increases much faster than the number of correctly identified links, thus causing the performance to drop.

Additionally, the type of data distribution (i.e., ordered vs. random) has a strong effect on the performance, especially the precision (blue bars). In the case of ordered data distributions, there is much less variation between the contexts of entities due to the way the properties are distributed. For instance, in a single data source, all entities share many properties, some of which can have a small range of possible values. This causes their contexts to become more similar. Therefore, when we later observe that the contexts of two entities are quite similar, this can be an indication of them originating from the same data source instead of them representing the same person. Another issue that affected the performance of the ordered-5 data distribution is the presence of clusters in the generated linkset that are larger than the expected maximum number (four) of duplicates per entity. These clusters can consist of 10 or more entities that are only loosely connected with the `owl:sameAs` relationship. Due to the assumption of the SPARQL reasoner that each cluster in the linkset is an equivalence class, the entire loosely connected cluster is treated as a single entity. This causes unwanted triples to appear in the query results, thereby lowering the precision scores. One way of dealing with this issue is to discard any clusters that are larger than the reasonable maximum size in a specific domain.

However, as seen in the three rightmost pairs of bars in Figure 6(b) related to random data distributions, when the assumption of the linkset generation algorithm is met (i.e., there exists an overlap in the contexts of entities between data sources), we see that there is only a small drop in precision, whereas recall remains constant. This indicates that we continue to find almost all relevant entities in the queries compared to the baseline.

The results show that in general, the expertise approach performs better when the data sources have a clear focus, whereas the performance of the naive approach is slightly better than the expertise approach when this focus is lacking. The results further show that, regardless of the used approach, consulting focused data sources is beneficial compared to consulting data sources without such a focus. A combination of the expertise and naive approaches may be beneficial when both focused and unfocused data sources are queried together. When data sources use different identifiers in referring to the same entities, the results show that the integrated system is dependent on the quality of the available linksets, and that the unfocused data sources, which provide more overlap in the context of entities, are more suitable for the automatic generation of such linksets.

6 RELATED WORK

The MuseumFinland project [19] aimed at making data from multiple museums available to non-technical users in an integrated manner. Its members have developed a general ontology describing the common domain

of the museums, and a faceted search browser providing easy access to the collections. Where our approach aims at facilitating researchers, their approach was primarily aimed at disclosing their digital collections, including relations between objects from multiple museums, to museum visitors. Moreover, where our approach integrates data *virtually* without requiring changes in the collections themselves, in the MuseumFinland project museums were asked to align their data to the general ontology and to upload their data to a central repository.

Since RETSINA [32] has been proposed, the hierarchical multiagent structure with user, broker, and data source agents has been a common approach to access separated information sources. The authors describe how process models can be used by agent-based Semantic Web services to indicate what inputs are required for the services they provide [33]. Broker agents reason over these required inputs and either use other services to acquire missing information or request that information directly from the user. Although we use a similar organizational structure, our contribution can be distinguished in that it deals with the problems users may face when querying distributed heterogeneous collections and is developed primarily for cultural heritage research.

García-Sánchez et al. [11] noted the variation in standards used in the Semantic Web, and compared those to relatively rigid standards and protocols of the traditional web. They proposed a multiagent system that uses these stricter web standards to discover web services that publish semantic data and use their service descriptions to map those data to a domain ontology automatically. A separate ontology is used to store collections' capabilities, allowing agents to jointly reason over the discovered services. This approach was aimed at the automated integration of semantic data and minimizing the amount of human intervention required rather than at accessible search. Although the application has been developed for the domain of bioinformatics, the authors claim it could be applied to open environments.

In the Semantic Web literature, various solutions to distributed querying have been proposed. By far, the easiest is to consolidate all data in a central data warehouse, as this allows the use of established query technologies. However, this requires a huge maintenance effort if the individual data sources are regularly updated, and is only feasible when the data is relatively static, and a single entity has both ownership permission for hosting the data and insight in the provenance of the data.

With the introduction of the SERVICE keyword in SPARQL 1.1, the W3C formalized a possible solution for querying distributed data [16, 26]. This method allows users to specify different remote data sources for specific parts of a query, and defer joining the results to the query engine of one of those data sources. This approach is a good solution for experienced SPARQL users, but it is very demanding of non-expert users, as they have to know what data source to consult for each part of the data, and resulting queries quickly increase in complexity. Moreover, this approach is intended for data sources that are intrinsically aligned at both the schema level and the entity level. Although SPARQL is sufficiently expressive to allow on-the-fly translation and matching of resources, using SPARQL1.1 for query-time alignment would add even more complexity to the queries. Last, most resources found online seem to suggest the approach is intended to query one main graph, which provides the bulk of the data, augmented with a limited amount of data from the other data sources. This correlates most with the ordered data distributions in our experiments, although in our approach, all data sources in a data distribution are equally important.

To automate the process of relevant data source selection, federation engines [1, 27, 29, 31] have been proposed. These are query processors that allow a user to query a set of *virtually integrated* data sources using conventional SPARQL without specifying which data sources to use. The challenge for this approach is to maintain *global relations*—that is, relations that span multiple data sources.

Most research in federation engines focuses on evaluating queries as fast as possible while maintaining soundness (and, to a lesser extent, completeness) of results. Various methods for optimizing efficiency have been suggested, which can broadly be classified in two categories: limiting communication overhead and efficient joining.

Since queries and results are transmitted over a network, decreasing the amount and size of these transmissions can significantly improve communication speed and thus query execution time. This can be achieved by intelligently selecting only relevant data sources, and constraining the query such that only relevant results

are returned. This decision process can be based on various heuristics or statistics. For example, Splendid [13] depends on remote data sources implementing the Vocabulary of Interlinked Data (VoID), whereas DARQ [27] gathers statistics dynamically from participating data sources. FedX [31] uses heuristics provided by the data sources at query time by sending an ASK query for each triple pattern in the received query to determine if the data source could contribute to the result. Data sources that cannot contribute are filtered to reduce communication overhead. Hibiscus [29] suggests an advanced algorithm that is based on hyper-graphs denoting all participating data sources. This improves early filtering of non-contributing data sources even further. Last, Harth et al. [15] proposed an approach to deal with conjunctive SPARQL queries by using QTree as an indexing structure, which combines histograms and the R-tree multidimensional structure; an idea stemming from Peer-to-Peer (P2P) systems [17, 18], where QTrees have been used to route queries between peers. The approach requires iterating over all of the records, resulting in a high space complexity.

FedX and Splendid reduce requests to remote data sources by grouping triple patterns that can only be answered by a single data source, called *exclusive sub-groups*, in one query. This approach has the advantage of constraining the subquery, reducing the number of intermediate results without losing relevant information. Other approaches for constraining sub-results early on is to use *nested join loops* [13, 31], where unbound variables in a subquery are bound with results from previous subqueries, and *early join*, which allows already dropping results without a join partner before they are sent back to the federation engine.

The other category of optimization, *efficient joining strategies*, improves the speed at which the results can combined. Usually, approaches are taken from existing databases. For example, Splendid [13] and DARQ [27] use dynamic programming strategies developed for SQL databases. Splendid adds the use of hash joins. Groppe et al. [14] studied how different distributed join strategies, accomplished by restructuring a query, compare. Of course, all approaches mentioned previously that reduce the number of false join partners before collecting them in the federation engine also improve joining efficiency, and can therefore also be considered as efficient joining strategies. There are many more proposals on federated search, many of which employ multiagent systems, that we cannot survey and compare with our work due to space limitations, but the reader can refer to other resources (e.g., [8, 23, 24]). Görlitz and Staab [12] provide an excellent overview of the challenges and approaches in federated querying. Saleem et al. [28] investigated the efficiency of various federation engines over multiple endpoints.

In the context of entity alignment, both symbolic and subsymbolic methods have been developed. For example, in the symbolic realm, Idrissou et al. [21] developed *Lenticular Lenses*, a tool for creating context-dependent links between data sources. Lenticular Lenses has already been used for linking between historical data sources from the Amsterdam City Archives [20]. Furthermore, there have been many efforts to use entity alignment techniques with the use of subsymbolic (e.g., machine learning) techniques.—for example, to link entities that occur in multiple knowledge graphs that are written in different languages, such as English and Chinese [34, 36, 37, 39]. These subsymbolic methods, however, do not take the particularities of historical data into account—for example, the need for proxy variables such as using a baptism date as an indicator for a birth date, and uncertainty in dates and names. Our embedding method does provide support for these kinds of issues.

7 CONCLUSION AND FUTURE WORK

In this work, we have acknowledged three challenges that researchers may encounter when querying the distributed data sources of multiple cultural heritage data providers in conjunction with search for patterns spanning multiple datasets: *query formulation*, *source selection*, and *alignment*. To overcome these challenges, we have proposed a multiagent architecture that allows data providers to expose their data without restrictions on the ontology or schema, and users to organize and search these data sources in an integrated manner. To this end, three types of agents—user, broker, and data source agents—collaborate to query the distributed data on behalf of the user. We have implemented a prototype of this architecture in Java and performed experiments to validate

the approach. Our experiments have shown that the proposed methods for combining and translating data from multiple data sources do not introduce errors and thus produce results as reliable as when using conventional SPARQL query methods on single data sources, unless those errors are already present in the provided data or translations. We have shown that automatic entity alignment introduces such errors, so there exists a trade-off between reducing human effort and reliability. Our experiments have further shown that both the nature of the data distribution and the approach for distributing a query influence the amount of answers that can be collected for a single query. If each data source contributes its own distinct domain, the yield of our system is significantly higher. This was proven to be the case for both methods of query distribution—expertise and naive—that we tested. However, when there exists large overlap within the domains of the data sources, or no data source is complete for some of the properties (in the sense it does not have value for those properties), an approach where each part of the query is sent to all participating data sources that may be able to contribute to it is more suitable. These observations suggest that data curators are better off adding highly focused data sources—that is, data sources that have a clearly bounded scope and are highly complete and trustworthy within that scope—that expand the domain rather than trying to improve coverage by adding data sources redundantly contributing data already covered by other sources. Moreover, the quality of answers stands or falls with the quality of the mappings and linksets, which still require some human effort to produce [2, 3]. To facilitate easy integration, we highlight four more (interrelated) considerations for data curators of individual data sources. Some of these are generally recognized as *best practices for publishing (Linked) (Open) Data* [30, 35] and are applicable to general cases of integration beyond this work. First, (re-)using existing, widely adopted ontologies to structure the data reduces the complexity of the required mapping files. Second, although some inconsistency in the data structure can be corrected with (more complex) mapping files, selecting a suitable and sufficiently expressive structure when starting the process of digitizing an archive or creating a data source, and sticking to it, makes it both easier for users to formulate queries on the data source on its own, and to integrate that data source into the architecture proposed in this work. Third, using existing IRIs from other, well-established knowledge graphs (e.g., DBpedia) to denote an entity when available, instead of introducing new IRIs, limits the number of entities that have to be linked explicitly. Last, when entities cannot be referred to with their common IRIs, describing them with common properties, and already linking some to IRIs for the same entities in other data sources (as a ground truth), would significantly help the automatic entity alignment process, which in turn improves the accuracy of automatically generated links.

In our experiments, we have artificially distributed a single data source from the domain of cultural heritage to simulate distributed data in a controlled setting. A more rigorous method for creating artificial data distributions, that takes into account the different types of noise that can be encountered in real-world applications, could provide further insight into the performance of the system. Especially the impact of noise on the automatically generated linkset would be an interesting pursuit for future work.

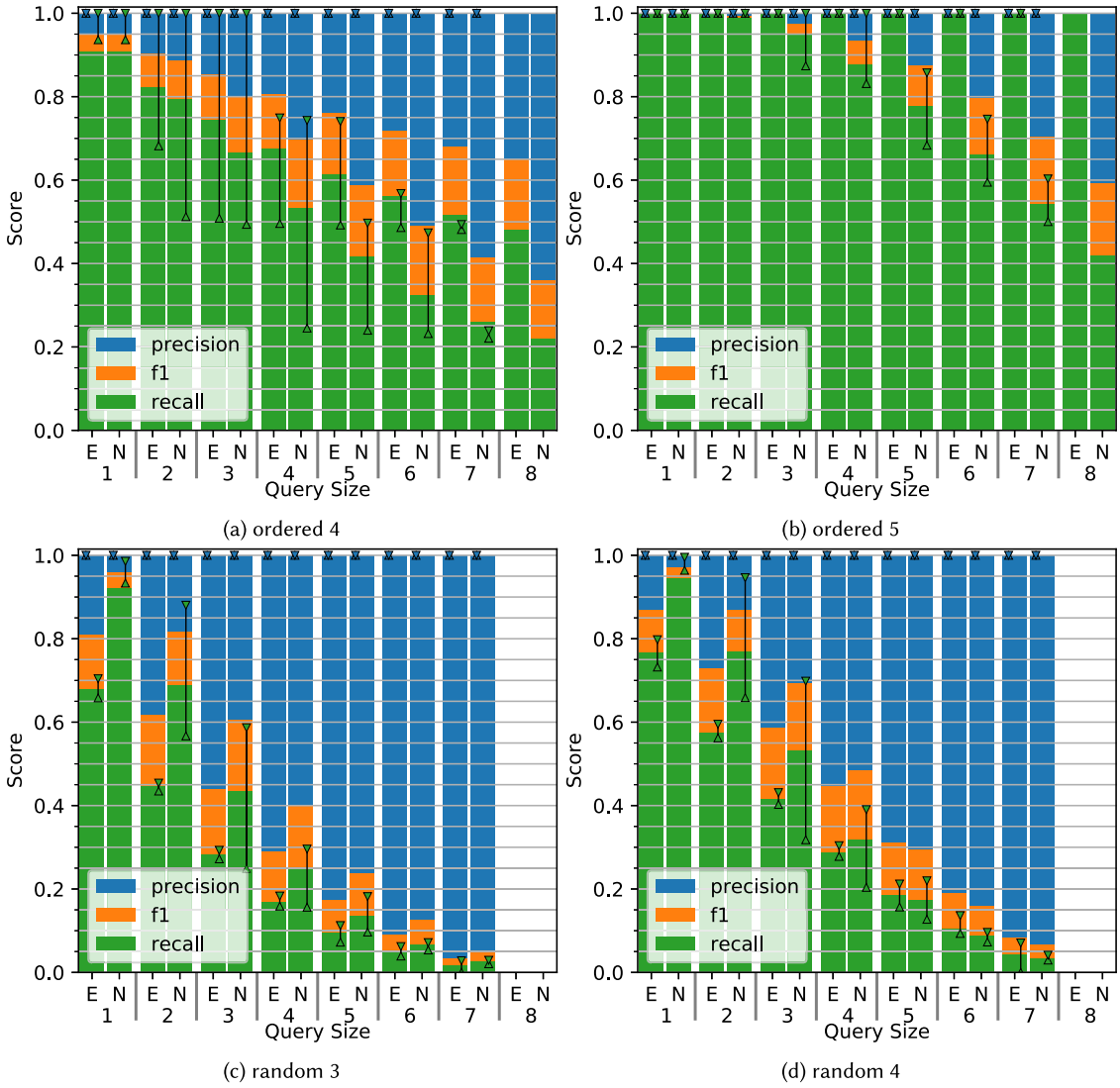
The other limitations of this approach stem from the assumption of the presence a general ontology and mappings to local schema. Although a general ontology is, at least in principle, something that has to be created only once, each additional data source added to the system still requires manual crafting of mappings. Last, the architecture proposed in this work is not tweaked to optimization at the same level as query federation engines. Instead, where federation engines require adherence to global standards that cannot be guaranteed in the domain of cultural heritage, our approach focuses on integrating these types of heterogeneous digital data sources. Moreover, in the proposed work agents are capable to adapt to user intentions as well as structural variation in the data, and handle distributed data more dynamically. Furthermore, agents can run in a physically distributed fashion, allowing data providers to control access to their data by maintaining their own agents. Of course, nothing prohibits incorporating existing approaches from federation engines for query optimization, but this has not been the focus of this work. In ongoing work, we are improving the search interface briefly outlined in Section 3.1 to support a variety of search strategies for non-expert users. A key aspect of this work is serendipitous search and navigating structures that increase in complexity due to their distributed nature.

Although the interface already supports the most common constructs from SPARQL, it is our hope to generate automatic "shortcuts" to the more verbose structures arising from complex mappings. In future work, we intend to include the use of the `OPTIONAL` keyword that SPARQL provides for triple patterns that are sent to multiple data sources, to reduce the constraining effect of a triple pattern added to a subquery. Moreover, we intend to investigate dynamic combinations of the tested expertise and naive approaches. We are further currently investigating two significant extensions to the expertise graph, which should both improve the query distribution. The first is to take linksets into account in the expertise graph, and the second is to take the preference of users into account when assigning importance to a data source for a triple pattern in the user's query. From the experiments, we learned that the OWL reasoning mechanism provided by Apache Jena is a computational bottleneck. Instead of relying on this reasoner, we intend to investigate the possibility of replacing all equivalent entity IRIs with a single "canonical" IRI before transmitting results back to the broker agent.

Last, the quality of the linksets produced by the "embedding" method can further be improved by refining the clustered output of the classifiers. By analyzing how strongly elements in the cluster are connected, false links may potentially be removed automatically.

APPENDIX

A FIGURE 7



(a) ordered 4

(b) ordered 5

(c) random 3

(d) random 4

Fig. 7. The precision, recall, and f1-scores, split out by the size of the query for the four remaining distributions.

ACKNOWLEDGMENTS

We would like to thank Professors Els Stronks, Maarten Prak, and Charles van den Heuvel for their valuable comments, suggestions, and many discussions on various parts of this article.

REFERENCES

[1] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. 2011. ANAPSID: An adaptive query processing engine for SPARQL endpoints. In *International Semantic Web Conference*. Springer, 18–34.

[2] Jurian Baas, Mehdi Dastani, and Ad Feelders. 2020. Tailored graph embeddings for entity alignment on historical data. In *Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications and Services*. 125–133.

[3] Jurian Baas, Mehdi Dastani, and Ad Feelders. 2021. Exploiting transitivity constraints for entity matching in knowledge graphs. *arXiv preprint arXiv:2104.12589*. (2021).

[4] Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Mathematics* 3, 1 (2006), 41–62.

[5] Mehdi Dastani. 2008. 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems* 16, 3 (2008), 214–248.

[6] Mehdi Dastani and Bas Testerink. 2014. From multi-agent programming to object oriented design patterns. In *International Workshop on Engineering Multi-Agent Systems*. Springer, 204–226.

[7] Jan de Mooij, Can Kurtan, Jurian Baas, and Mehdi Dastani. 2020. A multiagent framework for querying distributed digital collections. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence—Volume 1: ARTIDIGH*. INSTICC, SciTePress, 515–521. https://doi.org/10.5220/0009154005150521

[8] Paolo Dongilli, Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris. 2005. A multi-agent system for querying heterogeneous data sources with ontologies. In *Proceedings of the 13th Italian Symposium on Advanced Database Systems (SEBD'05)*. 75–86.

[9] Sébastien Ferré. 2017. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web* 8, 3 (2017), 405–418.

[10] Sébastien Ferré and Alice Hermann. 2012. Reconciling faceted search and query languages for the Semantic Web. *International Journal of Metadata, Semantics and Ontologies* 7, 1 (2012), 37–54.

[11] Francisco García-Sánchez, Jesualdo Tomás Fernández-Breis, Rafael Valencia-García, Juan Miguel Gómez, and Rodrigo Martínez-Béjar. 2008. Combining Semantic Web technologies with multi-agent systems for integrated access to biological resources. *Journal of Biomedical Informatics* 41, 5 (2008), 848–859.

[12] Olaf Görlitz and Steffen Staab. 2011. Federated data management and query optimization for linked open data. In *New Directions in Web Data Management 1*. Springer, 109–137.

[13] Olaf Görlitz and Steffen Staab. 2011. Splendid: SPARQL endpoint federation exploiting void descriptions. In *Proceedings of the 2nd International Conference on Consuming Linked Data—Volume 782*. 13–24.

[14] Sven Groppe, Dennis Heinrich, and Stefan Werner. 2015. Distributed join approaches for W3C-conform SPARQL endpoints. *Open Journal of Semantic Web* 2, 1 (2015), 30–52.

[15] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. 2010. Data summaries for on-demand queries over linked data. In *Proceedings of the 19th International Conference on World Wide Web*. ACM, New York, NY, 411–420.

[16] Olaf Hartig, Maria-Esther Vidal, and Johann-Christoph Freytag. 2017. Federated semantic data management (Dagstuhl seminar 17262). *Dagstuhl Reports* 7, 6 (2017), 135–167. https://doi.org/10.4230/DagRep.7.6.135

[17] Katja Hose, Marcel Karnstedt, Anke Koch, Kai-Uwe Sattler, and Daniel Zinn. 2007. Processing rank-aware queries in P2P systems. In *Databases, Information Systems, and Peer-to-Peer Computing*. Springer, 171–178.

[18] Katja Hose, Daniel Klan, and Kai-Uwe Sattler. 2006. Distributed data summaries for approximate query processing in PDMS. In *Proceedings of the 2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*. IEEE, Los Alamitos, CA, 37–44.

[19] Eero Hyvönen, Eetu Mäkelä, Mirva Salminen, Arttu Valo, Kim Viljanen, Samppa Saarela, Miikka Junnila, and Suvi Kettula. 2005. MuseumFinland—Finnish museums on the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web* 3, 2–3 (2005), 224–241.

[20] Al Idrissou, Veruska Zamborlini, Chiara Latronico, Frank van Harmelen, and Charles van den Heuvel. 2018. Amsterdamers from the golden age to the information age via lenticular lenses: Short paper. In *DH Benelux Conference 6-8 June 2018, International Institute for Social History, Amsterdam*.

[21] Al Koudous Idrissou, Rinke Hoekstra, Frank Van Harmelen, Ali Khalili, and Peter Van Den Besselaar. 2017. Is my:sameAs the same as your:sameAs? Lenticular lenses for context-specific identity. In *Proceedings of the Knowledge Capture Conference*. 1–8.

[22] Can Kurtan, Pınar Yolum, and Mehdi Dastani. 2020. An ideal team is more than a team of ideal agents. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI'20)*. 43–50.

[23] Andreas Langegger, Wolfram Wöß, and Martin Blöchl. 2008. A Semantic Web middleware for virtual data integration on the web. In *European Semantic Web Conference*. Springer, 493–507.

[24] Davy Monticolo, Inaya Lahoud, and Pedro Chavez Barrios. 2020. OCEAN: A multi agent system dedicated to knowledge management. *Journal of Industrial Information Integration* 17 (2020), 100124.

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*. 1532–1543.

[26] Eric Prud'hommeaux and Carlos Buil Aranda. 2013. *SPARQL 1.1 Federated Query*. W3C Recommendation. W3C. http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/.
[27] Bastian Quilitz and Ulf Leser. 2008. Querying distributed RDF data sources with SPARQL. In *European Semantic Web Conference*. Springer, 524–538.
[28] Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga Ngomo. 2018. An evaluation of SPARQL federation engines over multiple endpoints. In *Proceedings of the Web Conference*. ACM, New York, NY.
[29] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. 2014. Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *European Semantic Web Conference*. Springer, 176–191.
[30] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. 2014. Adoption of the linked data best practices in different topical domains. In *International Semantic Web Conference*. Springer, 245–260.
[31] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. 2011. FedX: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference*. Springer, 601–616.
[32] Katia Sycara, Anandeep Pannu, M. Willamson, Dajun Zeng, and Keith Decker. 1996. Distributed intelligent agents. *IEEE Expert* 11, 6 (1996), 36–46.
[33] Katia Sycara, Massimo Paolucci, Julien Soudry, and Naveen Srinivasan. 2004. Dynamic discovery and coordination of agent-based Semantic Web services. *IEEE Internet Computing* 8, 3 (2004), 66–73.
[34] Bayu Distiawan Trisedya, Jianzhong Qi, and Rui Zhang. 2019. Entity alignment between knowledge graphs using attribute embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 297–304.
[35] Boris Villazón-Terrazas, Bernadette Hyland, and Ghislain Auguste Atemezing. 2014. *Best Practices for Publishing Linked Data*. W3C Note. W3C. https://www.w3.org/TR/2014/NOTE-ld-bp-20140109/.
[36] Chi Man Wong, Qiang Chen, Suhui Wu, and Wei Zhang. 2020. Global structure and local semantics-preserved embeddings for entity alignment. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, Vol. 20. 3658–3664.
[37] Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, and Dongyan Zhao. 2020. Neighborhood matching network for entity alignment. *arXiv preprint arXiv:2005.05607* (2020).
[38] Veruska Zamborlini, Arianna Betti, and Charles van den Heuvel. 2017. Toward a core conceptual model for (im)material cultural heritage in the Golden Agents project. In *Proceedings of the SEMANTICS 2017 Workshops Co-Located with the 13th International Conference on Semantic Systems (SEMANTICS'17)*.
[39] Qiannan Zhu, Xiaofei Zhou, Jia Wu, Jianlong Tan, and Li Guo. 2019. Neighborhood-aware attentional representation for multilingual knowledge graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 1943–1949.

Received October 2020; revised May 2021; accepted September 2021