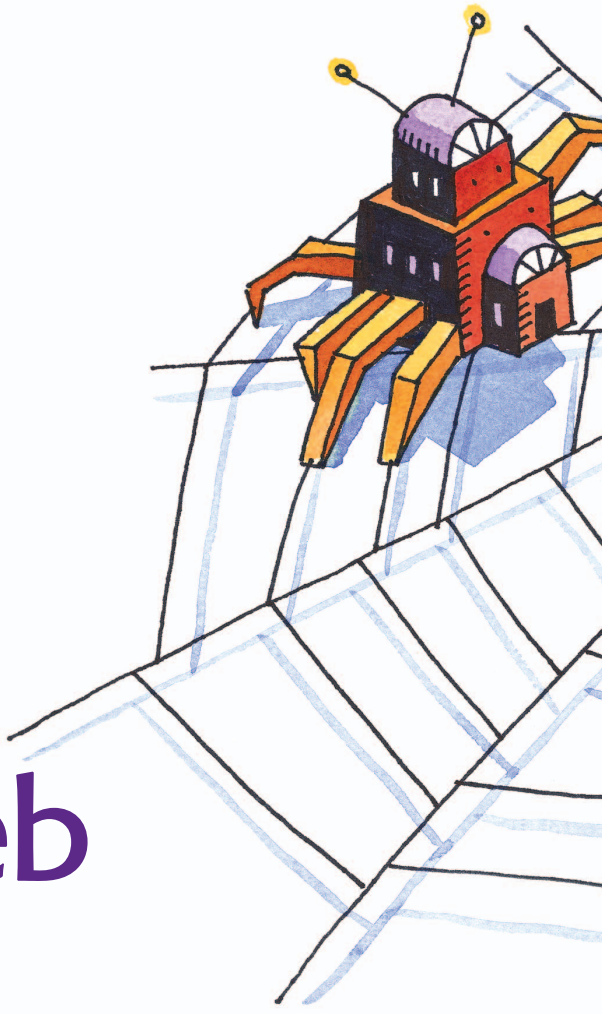
Christophe Guéret, Stefan Schlobach, Kathrin Dentler, Martijn Schut, and Gusz Eiben
Vrije Universiteit Amsterdam, THE NETHERLANDS

Evolutionary and Swarm Computing for the Semantic Web



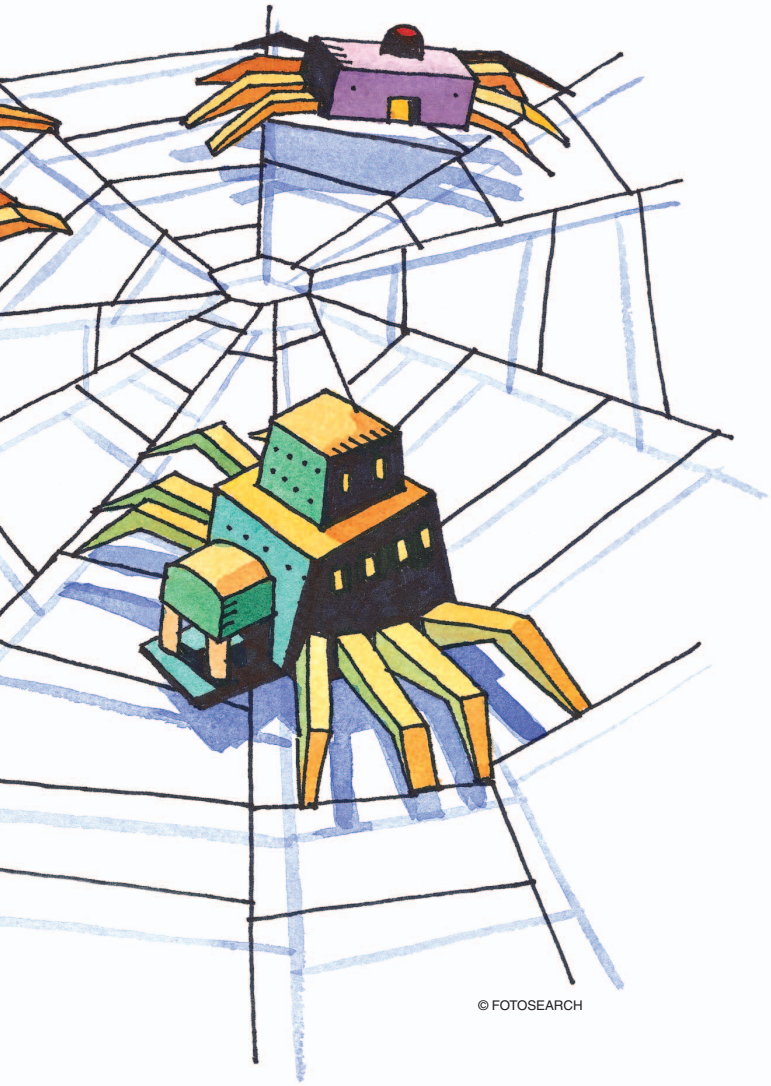
***Abstract*—The Semantic Web has become a dynamic and enormous network of typed links between data sets stored on different machines. These data sets are machine readable and unambiguously interpretable, thanks to their underlying standard representation languages. The expressiveness and flexibility of the publication model of Linked Data has led to its widespread adoption and an ever increasing publication of semantically rich data on the Web. This success however has started to create serious problems as the scale and complexity of information outgrows the current methods in use, which are mostly based on database technology, expressive knowledge representation formalism and high-performance computing. We argue that methods from computational intelligence can play an important role in solving these problems. In this paper we introduce and systemically discuss the typical application problems on the Semantic Web and argue that the existing approaches to address their underlying reasoning tasks consistently fail because of the increasing size, dynamicity and complexity of the data. For each of these primitive reasoning tasks we will discuss possible problem solving methods grounded in Evolutionary and Swarm computing, with short descriptions of existing approaches. Finally, we will discuss two case studies in which we successfully applied soft computing methods to two of the main reasoning tasks; an evolutionary approach to querying, and a swarm algorithm for entailment.**

Digital Object Identifier 10.1109/MCI.2012.2188583
Date of publication: 13 April 2012

I. Introduction

Since its creation some 20 years ago, the Web has been the place of major evolutions. Started as a publication mechanism for connected text, it has now become a rich content platform where everyone can publish and consume various types of media. Yet, all the content published is only accessible to humans, not machines. The next evolution of the Web, sometimes referred to as Web 3.0, will be that of a Web accessible to both machines and humans: a *Semantic Web* (SW) whose content can be interpreted by machines as it is explicitly modeled using languages with clearly defined *Semantics*. In practice, the Semantic Web connects data in a similar way as the WWW connects documents. Atomic data-units called *resources* are connected via labeled links with other resources and together these so called *triples* form a gigantic graph of *Linked Data*. The label of these links is where the Semantics are: links not only connect two resources but also convey the *meaning* of such connection.

1556-603X/12/$31.00©2012IEEE



The meaning of the types can be fixed using standardized schema and ontology languages such as RDFS and OWL [1]. Traditionally, the formal meaning of these languages is based on logical paradigms that were designed for small and hand-made knowledge bases, and come with a classical model-theory assigning truth to formula, and entailment based on this truth. Accordingly, the popular problem solving methods have their origin in database technology and classical logical paradigms, with centralized storage of, and access to, data; and with algorithms aiming at sound and complete results. The search is performed assuming full access to the complete dataset from a single location.

It is increasingly visible that in a highly complex, dynamic, context-dependent, opinionated and contradictory data space such as the Semantic Web, these approaches are insufficient. They are often prone to logical fallacies, usually intractable and can not easily cope with contextual information. The Semantic Web is a decentralized data space, a market place of ideas where contradictions and incoherence are common things for the data it contains. It is widely recognized that new adaptive approaches are required to exploit the ever growing amounts of dynamic Semantic Web data [2]. These insights have triggered research into the Semantic Web as a Complex System [3] and into more expressive formal languages capable of dealing with the multidimensionality and dynamicity [4] to name but a few. There is also a significant body of research about the potential of applying methods from Computational Intelligence to address a diverse set of problems that are inherent to the Semantic Web.

In this paper, we introduce the Semantic Web (Section II) and its typical usage and problem space (Section III). We then describe how evolutionary and swarm computing can be applied to address these problems and illustrate our point with two case studies (Section IV).

II. The Semantic Web

The World Wide Web is a decentralized system enabling the publication of documents and links between these documents on the Internet. A document is a piece of text, usually written in HTML and made available at a particular address (the URI). The links between documents are based on anchors put in these texts (hypertext links) and express a relation whose meaning depends on the interpretation made of the anchoring text. The Semantic Web uses the Web as a platform to publish and interlink data, rather than documents.

A. The Design of the Semantic Web

The Semantic Web replaces the HTML documents found in the documents by descriptions in RDF (Resource Description Framework) [5] about resources and uses typed relations instead of generic hyperlinks. On the Semantic Web, a URI is a **resource** representing a "real world" entity outside the internet or a document within it. For instance, the resource http://dbpedia.org/resource/Amsterdam is a Semantic Web representation of the city of Amsterdam in the Netherlands. The description associated to this resource contains factual information about it, expressed using triples. A **triple** is the combination of a subject, a property and an object. The subject being a resource from the Semantic Web, the property another resource and the object either a resource or a textual value (a **literal**). An example of such description for `dbpedia:Amsterdam` is shown in Table 1. In this table, we follow a common practice of using a compact syntax for the URIs, the interested reader will find the translation of these prefixes on the site http://prefix.cc.

Table 1 shows the two main advantages of the Semantic Web. Firstly, properties are also URIs and, thus, also have a description associated to them. These descriptions are defined in vocabularies (also called "ontologies") that provide a set of properties targeted to specific knowledge representation domains. For instance, the vocabulary "Friend Of A Friend

TABLE 1 Part of the description given for `dbpedia:Amsterdam`.

SUBJECT	PROPERTY	OBJECT
DBPEDIA:AMSTERDAM	RDF:TYPE	DBO:CITY
DBPEDIA:AMSTERDAM	FOAF:NAME	"AMSTERDAM"
DBPEDIA:AMSTERDAM	DBO:COUNTRY	DBPEDIA:NETHERLANDS

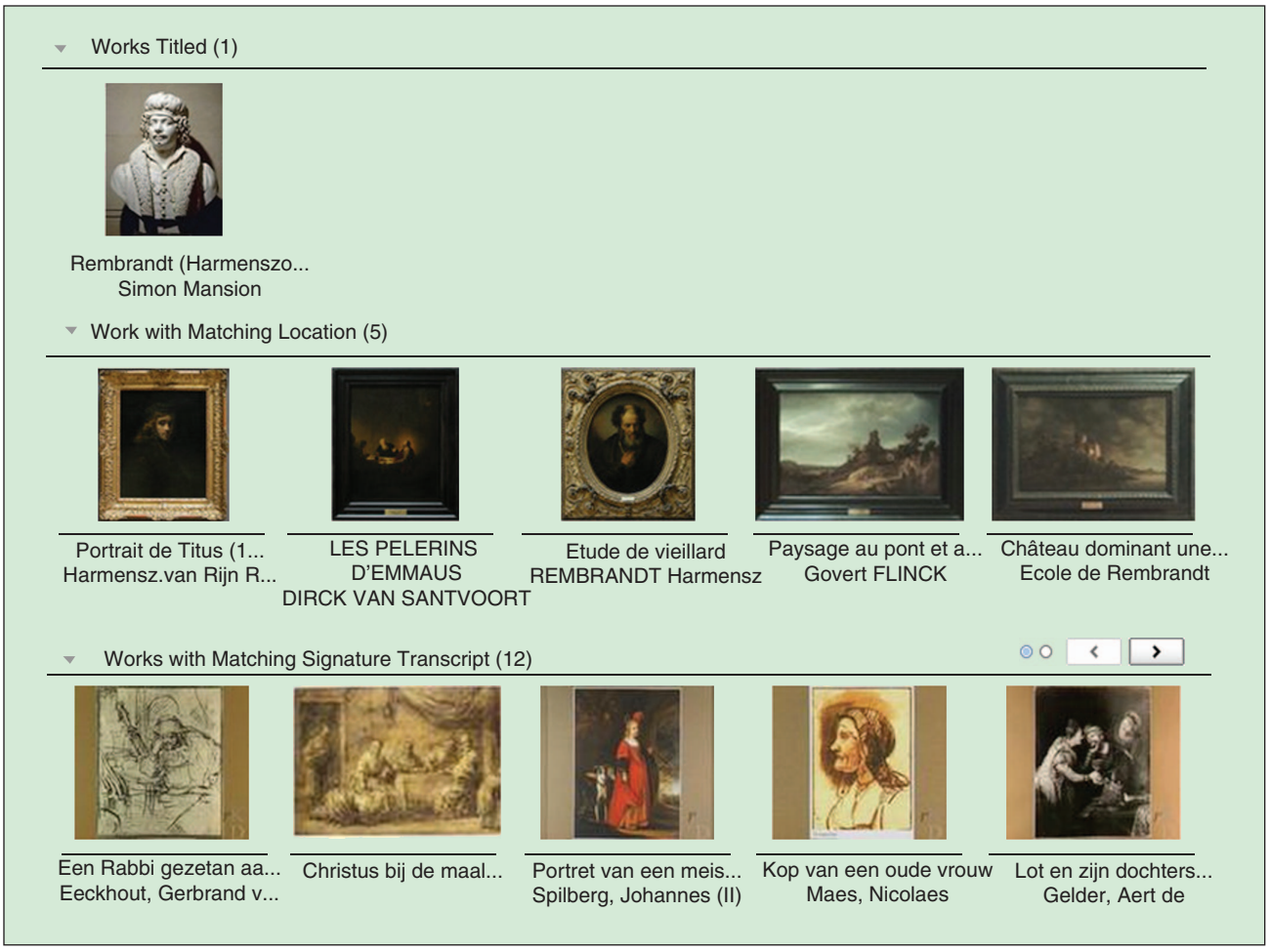


FIGURE 1 The cultural heritage portal "eCulture" shows how the Semantic Web impacts search, browsing and data integration. This image shows example results for "Rembrandt."

(FOAF)" contains properties used to describe social network oriented relationships. Secondly, URIs can be used in subject and object position and yield a network of relationships when URIs that are used as a subject/object in a triple are re-used as a object/subject in an other. For instance, the resource `dbpedia:Netherlands`, used as an object in Table 1, contains the triple ⟨`dbpedia:Netherlands, dbo:currency, dbpedia:Euro`⟩ as part of its description. In addition to this, the Semantics defined for some particular triples allows "reasoners" to derive new information by combining existing data. For instance, a triple expressing the fact that Amsterdam is in the Netherlands and another one saying that the Netherlands are part of Europe can be combined to deduce that Amsterdam is in Europe. This reasoning process will lead to the materialization of an explicit triple for an information that was otherwise implicit. The Semantic Web thus provides a way to express interconnected, structured knowledge about things in a Semantically well understood way.

Having introduced the basic notions needed to get a better understanding of what the Semantic Web is, we now have a closer look at the problems that need to be addressed when one wants to use Semantic Web data in an application. For this, we briefly introduce two web sites which leverage Semantic Web technologies for two different usages.

1) *Access to cultural heritage:* The site "eCulture"[1] (*c.f.* Figure 1) integrates data from different Dutch museums into a common content portal. This application shows how search, browse and data integration tasks are better achieved using Semantic Web technologies. Data integration is performed by re-using identifiers to refer to the same things. In doing so, graphs from different data sources are merged into a single graph. Search and browsing features are provided to explore the content of this graph. The usage of particular properties to describe the resources enables targeted search that can differentiate between finding keywords in the title or in the description of an artwork. Faceted browsing provides the user with a way to find relevant artwork based on sought properties. What cannot be seen in Figure 1 is the underlying usage of

[1]http://eculture.cs.vu.nl/europeana/session/search, accessed January 3, 2012

standards: the thesauri expressed in SKOS [6] and OWL [1], the integration with other Linked Data and the representation of manuscripts as unambiguous resources described in RDF.

2) *Music recommendation:* The site Seevl[2] (*c.f.* Figure 2) shows an example of personalization and recommendation made with Semantic Web technologies. This site provides a search interface to find musical artistes and then propose matching similar artistes. The matching is done using shared properties which are found in the graph. For instance, every band singer born in a particular city will be connected to that city in the data graph. All other artistes connected to that city are similar in the sense they share a common birth place. The usage of Linked Data to encode and link the data makes it possible for the site to integrate data from different sources and perform properties-based comparisons. Different resources can be compared based on their respective, semantically rich, descriptions in RDF.

FIGURE 2 Seevl is a musical search and recommendation system using the RDF data published by DBpedia (an RDF version of Wikipedia) based on dbrec.net.

B. The Nature of the Semantic Web

RDF allows for the creation of triples about anything on the Semantic Web. In fact, anyone is free to combine any subject, predicate and object, eventually coming from three different data sources, to create and publish a new triple. The decentralized data space of all the triples is growing at an amazing rate since more and more data sources are being published as semantic data, in RDF. But the size of the Semantic Web is not the only parameter of its increasing complexity. Its distributed and dynamic character, along with the coherence issues across data sources, and the interplay between the data sources by means of reasoning contribute to turning the Semantic Web into a Complex System.

1) **The massive amount of data and data sources**
Triples can be served by many data sources, eventually in large quantities. Modern triple stores, databases that are optimized to store triples, claim to be able to store and serve as many as a trillion of triples[3]. A popular view of the Semantic Web data space, the "LOD Cloud", referred to around 300 data sources [7], accounting for more than 30 billion of triples as of late 2011; A number that has not taken into account the data published by Facebook and several other web sites using embedded RDF content (RDFa). All this data being served is dynamic and may change on a frequent basis. It is even more so for data that is produced by sensors, such as those found in environmental monitoring networks [8].

These scale and dynamicity properties are similar to that of the Web of Documents, upon which the Semantic Web is grounded, but, unlike it, and because of the Semantics of the links, a triple published by one data provider can have a direct impact in combination with triples provided by many others. This emphasizes the importance of being able to deal with the *entire* Semantic Web, whatever its scale is.

2) **Opaque data locality**
As a set of triples, the content of the Semantic Web can be served by many different data sources at the same time. The global picture of a giant graph of information, is obtained by aggregating all the, possibly duplicated, triples served by all these data sources. From that global point of view, the locality of the triples is opaque and does not (or should not) affect the operation of the tasks making use of that content.

There is currently a large body of work, and an ongoing normalization work lead by the W3C[4], on the definition of provenance on the Semantic Web. The fact that any triple can be potentially served by anyone makes a strong case for tracking the origin of the triples. Opaque data locality also yields data synchronization and coherence issues as several servers can be serving different versions of a similar data set.

3) **Privacy and coherence concerns**
Even if triples from the Semantic Web can be served by many different data sources, some of these may be restricted to only being served by some particular server. Privacy concerns are a first motivation for this restriction: some data providers control their data and prevent any massive data acquisition process on their data set. Provenance is another motivation related to being seen as an authority to the data served. Finally, there is also a technical restriction associated to the dereferencing of the resources (URIs) used to get their description.

These specific constraints are a hiccup for algorithms that require having a local copy of the data they operate on. Whatever the motivation is, incoherence or limitations, the Semantic Web data should not be centralized but be available for use as is, directly from where it is being served.

[2]http://seevl.net, accessed January 5, 2012
[3]http://www.w3.org/wiki/LargeTripleStores, accessed January 6, 2012
[4]http://www.w3.org/2011/prov/wiki/Main_Page, accessed January 6, 2012

TABLE 2 Tasks and traditional solving methods to make use of a set of triples T. In the table, $\vdash$ stands for logical entailment and $t < Q$ implies that t is an instance of Q.

TASK	FORMAL PROBLEM DEFINITION	TRADITIONAL SOLVING METHODS
QUERYING	GIVEN T AND A QUERY Q, RETURN THE SET OF TRIPLES $\{t \in T\}$ SUCH THAT $T \vdash t < Q$	LOOKUP AND JOIN
STORAGE	GIVEN T AND A TRIPLE t RETURN $T \cup t$	CENTRALIZED INDICES, DISTRIBUTED HASH-TABLES
ENTAILMENT	GIVEN T. DERIVE $t \notin T$ WITH $T \vdash t$	CENTRALIZED AND PARALLELIZED DEDUCTION (RULES).
CONSISTENCY	GIVEN T. CHECK WHETHER $T \vdash \bot$ (FALSE)	LOGICAL REASONING
MAPPING	GIVEN T AND A MAPPING CONDITION C. RETURN $S,O \in T \times T$ SUCH THAT $C(S,O)$ LIKELY HOLDS WITH RESPECT TO T	SIMILARITIES SEARCH BETWEEN RESOURCES AND CLASSES. INDUCTIVE REASONING.

C. What Kind of Problem Solving Methods are Required?

To summarize: the Semantic Web data space is distributed, dynamic and inconsistent, sensitive to privacy issues, and opaque. This has immediate impact on typical characteristics expected of problem-solving methods applied to it: adaptivity, scalability and approximation.

- **Learning, adaptation and interactivity:** The Semantic Web is a market place of ideas, filled with contradicting and/or locally relevant facts. Requests expressed over this data are typically contextualized and algorithms should adapt to both the context and the data in order to provide accurate answers. Interactive re-evaluation of the user goal (i.e. the fitness function of the problem at hand) is also a desired capability.
- **Scalability and robustness:** The data is both decentralized and available in large quantities. Algorithms must be scalable to be able to deal with the amount of data available, they must also be robust enough to cope with the potential failure of data providers. The data from the Semantic Web is served by Web servers which may interrupt serving the data without prior notification. They can also change the content being served at any time. Problem solving methods should thus be able to cope with dynamic search spaces.
- **Anytime behavior and approximation:** When considering a distributed publication system such as the Web, one has to give up on completeness, soundness and determinism of answers. These values traditionally ensured by knowledge systems have to be traded for algorithms returning answers as they are found (anytime behavior) on a "best so far" basis (approximation). Additionally, one should not expect to get *all* the answers matching a given query and expect instead a non deterministic subset of them.

An additional desired feature for problem solvers is the ability to be parallelisable. The data model of the Semantic Web is a set of triples, which can easily be split into subsets of triples. Parallelisable strategies can hence benefit from that characteristic to provide increased computational performances.

III. The Semantic Web: A Problem Space for Evolutionary and Swarm Computing

The nature of the Semantic Web, and the typical problems that need to be addressed before one can make use of its richness of information, are precisely those that Computational Intelligence methods are good at solving. We will substantiate this claim in this section by analyzing a number of key tasks required for building Semantic Web applications.

In their systematic analysis [9] Harmelen *et. al* argued that typical Semantic Web applications require a rather restricted set of basic reasoning tasks ("Entailment", "Consistency", "Mapping"). However, the Semantic Web combines data with Semantics, *i.e.* provide extra meaning to data, and thus yields problems related to data management as well. We therefore extend the analysis of [9] with a set of basic data manipulation tasks ("Querying", "Storage"). Table 2 contains a formal description of the different tasks Semantic Web applications have to perform, along with a short explanation of the traditional solving techniques.

Every algorithm currently being put to use on Semantic Web data has been designed as a logic based method operating over a finite set of curated triples T (a "knowledge base"). As discussed in Section II-B, a typical triple set T on the Semantic Web is not static, nor curated, and all the guarantees of logical reasoning (completeness, soundness, determinism, ...) are lost. Instead, one can only *aim* at them and thus *optimize* towards these ideals. The consensual approach is to fit the Semantic Web data into a knowledge base by downloading, aggregating and curating subsets of its content. The problem is therefore adapted to fit the currently available solving methods, rather than being addressed with novel techniques.

In order to find solving methods capable of dealing with the complex character of the Semantic Web we propose to rephrase the logical formulations of the tasks as optimization problems. Proper solving algorithms have then to be found for these problems. Evolutionary and Swarm algorithms are known to perform well on optimization problems with large, and eventually dynamic, search spaces. In the subsections, we go through all the tasks, look at them from an optimization point of view and discuss the approaches that have already been taken to tackle them. A summary of the outcome of this discussion is provided in Table 3.

A. Querying

1) *Problem description:* The task of querying the content of the Semantic Web consists of finding the set of triples $\{t \in T\}$ that matches a given query Q (we denote this by $t < Q$). The query language for Semantic Web data is SPARQL [10]. SPARQL queries are constructed from so called triple patterns usually connected in a graph. Those triple patterns are triples with free variables. Solving such query means looking for parts of the data set that match the given set of triple patterns. Figure 3 shows a given query (on the left) and a matching solution (on the right).

TABLE 3 The interaction with the Semantic Web is made through different tasks which can be phrased either as logic or optimization problems.

TASK	LOGIC PROBLEM	OPTIMIZATION PROBLEM	RELATED WORK
QUERYING	CONSTRAINT SATISFACTION	CONSTRAINED OPTIMIZATION	ERDF [13]
STORAGE	CONSTRUCTION OF SETS	CLUSTERING	SWARMLINDA [20]
ENTAILMENT	LOGICAL DEDUCTION	MULTI-OBJECTIVE OPTIMIZATION	SWARMS [24]
CONSISTENCY	(UN)SATISFIABILITY CHECKING	CONSTRAINED OPTIMIZATION	-
MAPPING	LOGICAL DEDUCTION	CLASSIFICATION	PSO [36], GOSSIPING [31], EVOLUTIONARY STRATEGY [37]

The nodes starting with a question mark are variables that have to be instantiated.

2) *Standard formulation and solving approach:* Solving a SPARQL query is a constraint satisfaction problem. The query defines the constraints and the triple set T is the domain in which the variables can pick their binding from. Taking inspiration from the work done in databases, the resolution of SPARQL queries is made of iterated lookup and join operations. A lookup consists of getting the set of triples that match a single triple pattern (an edge of the query graph) and, typically, loads the results in memory. A join is the finding of overlapping resources amongst two lookup result sets in order to produce a connected result. This construction mechanism ensures that all the possible answers are found and validated against all the constraints. This approach is adopted by the majority of existing triple stores such as Sesame [11] and YARS [12].
3) *Optimization problem:* The constraints expressed by the query can be relaxed into an objective function to optimize. In particular, considering the size of the Semantic Web, one may not expect to get all the answers to a particular query. Furthermore, considering the risk of incoherence and data source high churn rate, one may not assume having all the information at hand at a particular time. We thus relax the constraints of the query patterns into an objective function. The goal is to find bindings validating as many of the constraints as possible. This function has to be optimized under the constraint that the bindings used generate a connected solution graph.
4) *Solving methods:* With "eRDF"[13], [14], [15], we investigated using evolutionary computing to solve this problem. Solutions to the SPARQL query are found in an iterative guessing process based on evolutionary computing. More information on eRDF will be provided in Section IV-A.

B. Storage

1) *Problem description:* Storing a new information into the Semantic Web boils down to inserting a new triple t into a set of triples T. After the insertion, the new set of triples is equal to $T \cup t$. Because of the size of the data sets, this storage has to be achieved so as to facilitate the retrieval of the newly inserted triple. In fact, triples have to be inserted in such a way that their retrieval is optimized for lookup operations (*c.f.* previous section on querying). It is thus desirable that triple sharing subject and/or properties and/or objects are grouped together.
2) *Standard formulation and solving approach:* The standard insertion strategy of databases is to save the data and index it. The indexes ensure efficient lookup operations at the cost of an higher insertion time and more space needed for the data structures. In a distributed setting, a pre-determined indexing strategy applied by a central entity ensures an optimal load balancing between different storage nodes [16].

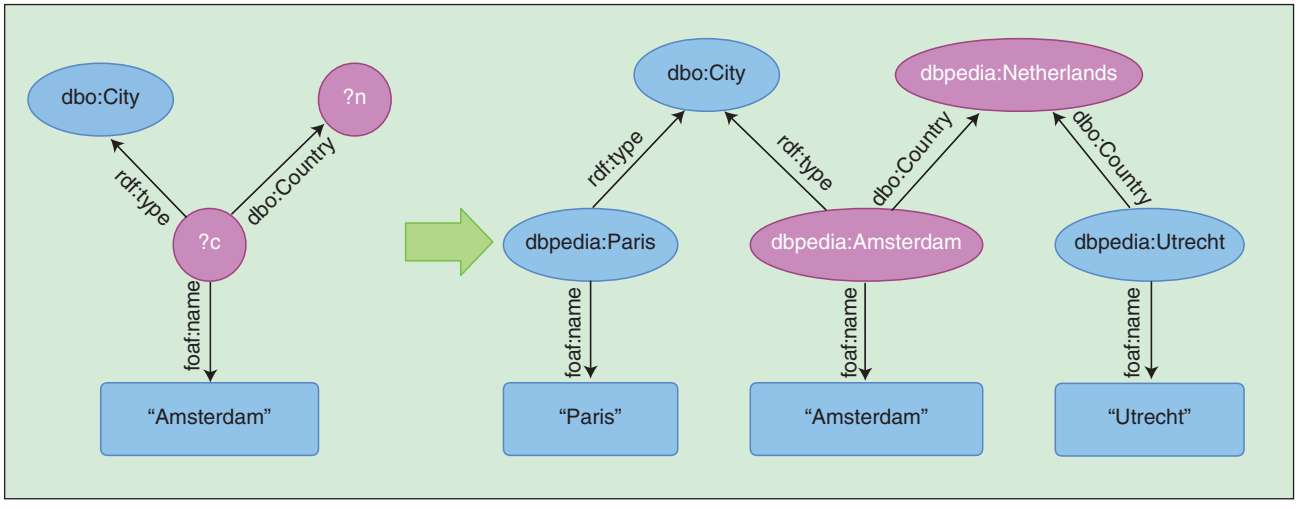


FIGURE 3 Example of a simple query (left part) and a matching sub-graph within an RDF graph (right part). The query reads as "Find a city ?*c* whose name is "Amsterdam" and which is in the country ?*n*", the result found is ?*c* = `dbpedia:Amsterdam` in ?*n* = `dbpedia:Netherlands`.

3) *Optimization problem:* In an optimization setting, the indexing task leaves place to clustering. The single optimum guaranteed of the pre-determined grouping rules defined by the indexes is replaced by a measure of the clustering efficiency. Considering a set of triples and a set of storage nodes, different clusterings (assignment of triples to nodes) of different quality can be established. The problem solver will strive to produce a clustering as efficient as possible by re-allocating data among the storage nodes.
4) *Solving methods:* Collective intelligence has already proven to be efficient for network routing [17] and data clustering [18]. Two features that combined, lead to triples clustering and query routing capabilities. Additionally, swarm algorithms can run continuously and adapt to changes in the data. SwarmLinda [19] shows how an artificial ants algorithm can be leveraged to create a self-organizing system with emerging triple clusters and optimized routes for querying. SwarmLinda applies a sorting method inspired by the brood sorting ability of real ants: artificial ants continuously take triples from one location and move them to another where similar triples are found. Traces left by the triples on the network connections taken during their journey are used for routing purposes at query time [19]. The basic concept has also been extended to provide a self-organized semantic storage service ("S4"), where RDF triples can be stored and queried in a fully distributed fashion [20]. Following a similar idea, PIAF [21] makes use of traces left by facts moved among the peers of a peer-to-peer system. These traces are used to efficiently disseminate data and route queries.

C. Entailment Checking

1) *Problem description:* Inference is the task of extracting information that is implicitly encoded in the formal representations of the data. In most knowledge representation languages, and also in those used to model knowledge on the Semantic Web, entailment is one of the core semantic notions. It is the fact that a formula logically follows from other formulas, *e.g.* a set of triples. As formulas that can be entailed are not necessarily explicit in the original data base, inference algorithms must derive those entailed formulas according to some calculi. The most prominent kind of entailment in Semantic Web is *relation checking* (whether two resources are related by some property), with the special case of *instance checking* (when this property is the `rdf:type` property). The reasoning tasks of realization and retrieval defined in [9] are combinations of the entailment we discuss here.
2) *Standard formulation and solving approach:* The state of the art approach for entailment checking consists of applying a given set of predefined rules over a static set of triples T. This process assumes that T has been curated to remove logical inconsistencies. It is incremental and has to be repeated from scratch every time the content of T changes. The search process is stopped as soon as no new facts are derived and added to the data set. The most efficient entailment checking engine to date, WebPIE, implements this strategy to derive the implications of a large number of triples [22]. A similar approach is also found in more common reasoners [23].
3) *Optimization problem:* In a dynamic and incomplete environment such as the Semantic Web, the goal of creating all the triples that can be derived from T by entailment (the "closure" of T) is actually that of deriving as many new triples as possible. Besides, on the Semantic Web it is unrealistic to consider that T will contain no inconsistencies. A closure will inevitably contain contradicting facts, a situation that has to be avoided. Entailment can thus be defined as an optimization problem around two opposite goals: generate as many new triples as possible and generate few of them in order to avoid deriving inconsistent information. More objectives could also be considered to, for instance, take into account the locality of the data, or its reliability. On the Semantic Web, entailment is hence a multi-objective optimization problem.
4) *Solving methods:* In [24], we applied swarm computing to the generation of the closure of a set of triples. Each of the swarm entity ("agent") is responsible for one reasoning rule used to derive new triples. Agents browse the graph in search of triples that match their rule. They follow properties to move from resource to resource, effectively visiting nodes in the graph. When an agent encounters a matching triple pattern, it fires its rule and writes down the newly derived triple to the graph (creates a new edge). This work is described in more detail in Section IV-B.

D. Consistency and Satisfiability Checking

1) *Problem description:* One of the prominent logical task is consistency checking. That is, checking whether a data set is internally contradictory. Formally, *inconsistency* is defined as the non-existence of a model for a set of triples while a concept or class is called *unsatisfiable* if it is empty in all models. Inconsistencies or the existence of unsatisfiable concepts usually points to modeling errors. Detecting them automatically can thus be very useful and if done early, at modeling time, avoid publishing inconsistent data.
2) *Standard formulation and solving approach:* From a logical point of view, the detection of these modeling errors is a satisfiability (SAT) problem. The inconsistency checking is a model-finding problem where the task is to find a truth assignment that satisfy a given set of assertions, defined by the triples. A set of assertions can be proven unsatisfiable by model-counting if the number of satisfying truth assignments is equal to zero. A truth assignment is a function that associate a boolean value (true or false) to all the variables mentioned by the set of assertions. Logical proof techniques such as the Tableau calculus [25] and the Davis-Putnam-Loveland procedure [26] are commonly used to approach these problems.

3) *Optimization problem:* An optimization formulation of SAT is obtained by, first, relaxing it into a maximum SAT (MAXSAT) problem where the maximal set of satisfiable assertions is sought. Then, that goal is relaxed into an objective. The optimization thus goes about finding a truth assignment that validate as many of the assertions as possible under the constraint that all the logical assertions get a truth value, true or false, assigned to them. This is a constrained optimization problem.
4) *Solving methods:* There seem to be no optimization algorithms that have been applied to address the problem of consistency and satisfiability checking. The literature however provides a wide range of SAT solver based on soft computing techniques. GASAT [27], for instance, uses evolutionary computing to evolve a population of candidate truth assignments. Similar goals have been achieved by Abbass using a swarm of artificial bees [28]. These algorithms should be directly applicable to Semantic Web data and used to tackle the optimization problem.

E. Mapping

1) *Problem description:* The goal of mapping is to find related resources within a set of triples T. The relation sought depends on the mapping use case. For vocabularies (classes, properties, …), the aim is to arrive at a logical organization. The relations established are inclusion, subsumption and specification, to name only but a few. For general resources describing entities, the aim is to establish equivalence. Several resources can be used to refer the same real world entity, which is, by existence, unique. We extract a generic mapping condition c from these two typical usages, to be further specified according to the data at hand and the desired relations.
2) *Standard formulation and solving approach:* State of the art approaches perform an (almost) exhaustive search over the search space, testing every pair of resources against the matching condition c. This condition is either stipulated by human experts or derived by some inductive method. If the condition is met, a new relation is created. Exemplifying this, the linker tool "Silk" [29] uses manually created linking specification to browse and connect resources found in two different data sets.
3) *Optimization problem:* The matching problem can be formulated as a classification problem, where a mapping can be predicted from the similarity between the extensional information (description) of two resources [30]. The quality of the classification can then be measured in different ways, *e.g.* by using a training and control data sets, and be used as a function to optimize. An alternative way of looking at mappings is to optimize the mapping condition itself, *i.e.* find the best similarity measure between the source and target data sets.
4) *Solving methods:* The size and decentralized nature of the Semantic Web makes it unrealistic to establish mappings between every resources it contains. Even if possible, such mappings may not be useful as triples distant from each other in the graph are also likely to be semantically unrelated. Instead, mapping established among neighbor resources are more likely to be both useful and semantically meaningful. Based on this assumption, Semantic gossiping [31] has been proposed to establish local agreements among the peers of a decentralized systems. Using this form of collective intelligence, semantic interoperability can be achieved at network scale [32] and mappings can be established [33].

Other approaches focus on the size of the search space by trying to explore less of it. GAOM [34] and GOAL [35] are two initiatives making use of genetic algorithms to explore the search space. They do it around two different axes. While GAOM evolves a set of mappings, the optimisation process of GOAL is targeted towards the similarity function between the two data sources. The fitness function of GOAL is a similarity function defined as a weighted sum of different features such as the string distance between the labels of the resources or the number of instances of concepts. The MapPSO [36] algorithm follows a strategy similar to that of GAOM but makes use of Particle Swarm Optimization (PSO) instead of evolutionary computing. A set of candidate solutions explore the search space shaped by the similarity function in a quest for positions with the highest correctness.

The feature used in weighted sum similarity functions may have different importance. Because of that, it is relevant to investigate the composition of the weighted sum serving as a similarity function and allow it to adapt to the data. In [37] we compared the results of different algorithms, including an evolutionary strategy, at performing a sound classification while explaining the results with an adjustment of the weights in the similarity function.

IV. Case Studies

Over the past few years, the Vrije Universiteit Amsterdam has been actively promoting the marriage between Computational Intelligence and the Semantic Web. Our main focus were placed on dissemination events (NatuReS workshop[5], SOKS Symposium[6]) and two flagship projects. In the following sections, we highlight these two attempts at tackling Semantic Web tasks with evolutionary and swarm computing. We recall the main features of the algorithms developed, some results and the lessons learned. Interested parties are invited to consult the related papers for more in depth details about this work.

A. Evolutionary Algorithms for Querying: The Erdf Framework

Querying the Semantic Web is defined as the task of finding triples in a huge, and possibly distributed, graph (set of triples). The solutions to the query have to fit a given query defined by a set of triple patterns, a graph with free variables.

[5]http://natures.few.vu.nl, accessed January 6, 2012,
[6]http://www.few.vu.nl/soks/symposium, accessed January 6, 2012

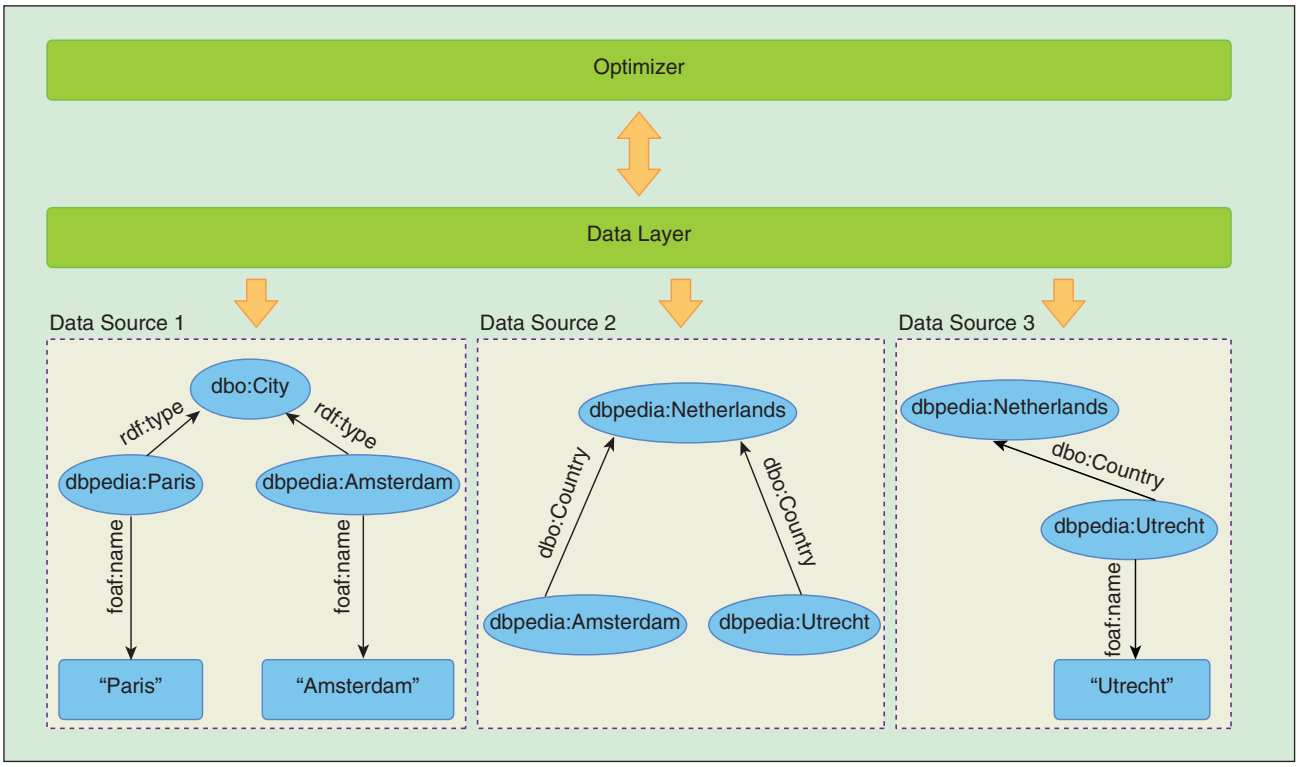


FIGURE 4 eRDF consists in two components: an optimizer and a data layer providing an abstract view over the different data sources.

The traditional way to find answers to the query is to 1) lookup the lists of triples matching the edges of the query and 2) join these partial results. In the example of Figure 3, this consists of retrieving the sets of triples matching $\langle$*, `rdf:type`, `dbo:City`$\rangle$, $\langle$*, `dbo:country`, *$\rangle$ and $\langle$*, `foaf:name`, `"Amsterdam"`$\rangle$. The sets are then joined based on the overlap between their subjects/objects. The result is a final list of every solution to the query. The two shortcomings of this approach are the necessity to pre-fetch all the potentially useful edges and the computational cost of merging the lists.

One can observe that testing if a given solution exists in a graph is easy and only requires checking the presence of the different edges and nodes. The eRDF framework [13], [14], [15] leverage this observation. Its search process is to guess and verify solutions to a SPARQL query. The framework consists of two main components: a data layer and an optimizer[7] (see Figure 4). The data layer provides an abstraction over the data coming from several RDF sources and the optimizer makes use of the data provided by this data layer to build the answers to the queries. Both components are only weakly coupled and may be implemented as two different services. This separation allows the data layer to re-use information across different optimization processes. It also allows the duplication of instances of the optimizer and/or the data layer in order to scale with the number of queries to be solved.

We describe below these two components and then conclude on some results from our prototype implementation.

1) *Data layer:* The data layer is a generic component designed to provide the optimizer with a limited set of primitives over $T = \bigcup_{i=1}^{m} T_i$. With T_i being the triple sets provided by m different data sources. This data layer does not store any "own" data. As in [38], eRDF starts with an empty knowledge base that is filled, at run time, by retrieving the necessary data from the Semantic Web. All the triples are acquired on demand from the different data sources. As a result, the introduced set of triples T is actually an *abstract* set that is never available to the optimizer as such. Data sources may become unavailable and/or require frequent updates. Under these conditions, maintaining a complete, up-to-date, copy of T would be very difficult, if not impossible.

The data layer implements two primitives taking a triple pattern as input. A first one used to verify the validity of the pattern (ASK) and a second one to get a resource matching the triple pattern (GET).

a) *ASK:* A triple $t = \langle s, p, o \rangle$ is considered to be valid if $t \in T$. This first operation is covered by the primitive ASK ($\langle s, p, o \rangle$). Besides telling if the entire triple is valid, the data layer can also be used to test the partial validity of a triple. That is ASK ($\langle *, p, o \rangle$), ASK ($\langle s, *, o \rangle$) and ASK ($\langle s, p, * \rangle$), with * being a wild card allowing any resource to be used. Such ability to test the partial validity

[7]The term "optimizer" is used here in reference to optimization problems, not the optimizer as found in databases systems.

of a triple pattern is used by the optimizer to assign partial rewards to the bindings of candidate solutions.

b) *GET:* The get primitive is used to get a resource that matches a given triple pattern. There are three of them, each returning a result matching one of the possible triple patterns: GET $(\langle *, p, o \rangle)$, GET $(\langle s, *, o \rangle)$, and GET $(\langle s, p, * \rangle)$ while respectively returning a resource to be used as a subject, a predicate or an object. Note that these primitives are aimed at returning one result at a time in a non-deterministic way. This design aspect makes the data layer more robust, allowing the return of results based on the information currently available.

3) *Optimizer:* The optimizer is in charge of generating candidate solutions and checking their validity. We use an evolutionary algorithm [39] for this process. The algorithm is memetic [40], taking care of the evolution of a population of candidate solutions while preserving some knowledge across the different generations. This knowledge is used to gather information about the data being used. It also shares some features with evolutionary strategies as the whole population is seen as parents for the generation of offsprings. A local search and a crossover operator are used to generate new candidate solutions and increase the population by a pre-determined factor. Then, similarly to what is done with "$(\mu + \lambda)$" selection of evolutionary strategies, the parent population and the offspring population are joined and shrunk down to the original population size.

We turned the constraint satisfaction problem of finding a set of bindings that validate all the triple patterns of the query into a constrained optimisation. The problem addressed by eRDF is finding a set of bindings that maximize the number of validated triple patterns. By turning the problem of query resolution into a constrained optimisation problem, eRDF implicitly relaxes all the statements of the query. Basically, the relaxation mimics a query where all the statements would be made optional and for which only the results with a maximum number of statements would be returned. eRDF is non deterministic and does not guarantee completeness. These two traditionally desired features of a query engine are traded for a better response time and an improved robustness. Additionally, the search process can cope with a changing search space (*e.g.* new data available) and can stream back solutions at the end of every generation.

The algorithm is outlined in Algorithm 1. Its three key steps are the *evaluation of the population* (assessing the quality of the candidate solutions, lines 3–4), the *selection of survivors* (selection of solutions to keep for the next generation, lines 5–15) and the *generation of offspring* (creation of new candidate solutions, lines 16–19).

a) *Population evaluation:* The evaluation performs checks on the quality of the population. Based on its bindings β that associate a value to every variable in the query, a candidate solution is given a *fitness* score between 0 and 1. We refer-

ALGORITHM 1 The search starts with a parent population P. The main loop consists in expending the population with new candidate found by a local search and then shrink it down to its original size. Candidate solutions that survived *max_age* generations or have an optimal fitness value are streamed back as results to the request and added to a taboo list τ.

```
1  Initialize population P;
2  while not terminated do
       /* Evaluation */
3      foreach Candidate solution β in P do
4          evaluate(β);
       /* Selection */
5      P ← survivors_selection(P);
6      f* ← best fitness of P;
7      foreach Candidate solution β in P do
8          if fitness(β) = f* then
9              age(β) ← age(β) + 1;
10         else
11             age(β) ← 0;
12         if age(β) = max_age or fitness(β) = 1 then
13             Output β as a result;
14             foreach Triple pattern g in G do
15                 τ ← τ ∪ g|β;
       /* Generation */
16     foreach Candidate solution β in P do
17         β' ← generate(β);
18         if β' ∉ P then
19             P ← P ∪ β';
```

ence by $G|_\beta$ the triple set created by instantiating the triple patterns G of the query with the bindings β. The optimization target is to maximize the size of this set while giving more credit to candidate solutions with connected triples (see Equation 1).

$$\text{fitness}(\beta) = \frac{\text{maximal\_component}(G|_\beta)}{|G|} * \sum_{g \in G} \frac{\text{reward}(g|_\beta)}{|G|}. \quad (1)$$

The function $\text{reward}(g|_\beta)$ credits the bindings β with respect to a single triple pattern g from G. This function can be freely customized to achieve different goals under the conditions

TABLE 4 Rewarding scheme for $g|\beta = \langle s, p, o \rangle$. The conditions are mutually exclusive and tested in the order of the table. A get($\langle s, *, o \rangle$) is typically computationally expensive and is thus tested last.

CONDITION	REWARD($G|\beta$)
$\langle S, P, O \rangle \in \tau$	0.25
ASK($\langle$S, P, O$\rangle$) *IS TRUE*	1
GET($\langle$*, P, O$\rangle$) $\neq \varnothing$	0.5
GET($\langle$S, P, *$\rangle$) $\neq \varnothing$	0.5
GET($\langle$S, *, O$\rangle$) $\neq \varnothing$	0.5

that $\text{reward}(g|\beta) \in [0,1]$. A reward of 1 reflects an optimal binding whereas a value of 0 means a non existing assertion. The ordering of the values has also to be respected, that is $\text{reward}(g|\beta) < \text{reward}(g|\beta')$ is equivalent to saying β' is a better solution than β for g. The notion of a candidate solution being better than another one is subject to interpretation and depends on the semantics of the rewarding scheme. For instance, one could consider a rewarding scheme solely based on size and the correctness of the set of triples created by the candidate solution ($G|\beta$). The rewarding scheme implemented in our prototype is reported in Table 4.

Once the entire new population is evaluated, its members are sorted and the population is trimmed down to the initial population size.

b) *Survivor selection:* At this stage, the population is sorted according to the fitness of its individuals and cut down to its default size. The age of the best individual(s) is increased and their optimality is checked. Candidate solutions that survived a *max_age* consecutive generations or that have a fitness equal to 1 are considered to be optimal. Such solutions are sent back to the user and the triples created by their bindings are added to the taboo list τ. This list penalizes the optimizer for re-using triples that have already been used in previously (locally) optimal solutions. This allows for the finding of diverse answers while not prohibiting the re-use of triples.

c) *Offspring generation:* Two strategies are implemented to generate new candidate solutions: 1) using the graph to propagate new values (local search) and 2) combining two candidate solutions (crossover). Both strategies are applied once for every individual in the parent population leading to a maximum increase of total population by a factor of 2. This value stays a theoretical maximum as duplicate candidate solutions are detected and removed from the population at the end of the generation process.

❑ Local search

The local search strategy is a three step process (see Figure 5). In the following ϵ denotes a small value, typically around 0.01, used to avoid having probabilities equal to zero.

1) Every binding has a probability to be changed proportionally to the expected reward gain such change would yield (*c.f.* Equation 2). This drives the evolution towards finding good assignments for pivot nodes before focusing on the branches of the query graph. A desired behavior for answering SPARQL queries.

$$p(v,\beta) \propto |\{g \in G \mid v \in \text{var}(g)\}| - \sum_{g \in G \mid v \in \text{var}(g)} \text{reward}(g|\beta) + \epsilon \quad (2)$$

2) Once a variable has been elected for a value change, an edge is selected to propagate a new assignment. The probability of using a given edge is proportional to its reward (*c.f.* Equation 3). Only the edges having a connection with the variable to be changed are considered in this process.

$$p(g,\beta) \propto 1 - \text{reward}(g \mid \beta) + \epsilon, \forall g \in \{g \in G \mid v \in \text{var}(g)\} \quad (3)$$

3) A GET query is issued using the selected edge and a new value is assigned to the variable. In the example of Figure 5, "Amsterdam" is a literal constant but the model also works using edges between two variables, thereby leading to a cascade effect of variable assignments and a propagation of their efficiency.

❑ Crossover

The crossover operation takes the candidate solution currently considered and another candidate solution randomly picked, as input. The genotypes of these two candidate solutions are combined in order to generate a third, new, candidate solution. For each variable assignment, the binding credited with the highest reward is picked. The decision is random when ties occurs.

3) *Experiments and conclusion:* A prototype implementation of eRDF is available online, under a free license, at https://github.com/cgueret/eRDF and described on http://www.erdf.nl. This implementation has been tested on answering large queries made up of hundreds of statements compromising tens of variables. The results are promising [15], showing that eRDF can effectively scale to such numbers and provide answers to the submitted queries.

Figure 6 contains an excerpt of the results described in [15]. It shows how eRDF performs on a set of requests of varied complexity (radius) and size (number of triple patterns). What can be observed on these graphs is that eRDF generally performs better on complex queries than on simpler ones. Perfect solutions to queries with at least 20 patterns are always found, under 100 seconds (a bit less than 2 minutes). This finding shows how eRDF turns this adverse complexity to its advantage, by using the constraints as guides for the evolutionary process (*c.f.* the local search process). The number of variables is also directly related to the size of the genetic material of the candidate solutions. A complex query leads to richer candidate solutions undergoing a better guided evolution.

In comparison, traditional implementations of querying engines only suffer from an increasing complexity. The number of patterns in the query is related to the number of lookups to be performed whereas the number of variables relates to the quantity of joins to be achieved. As a result, when compared to eRDF

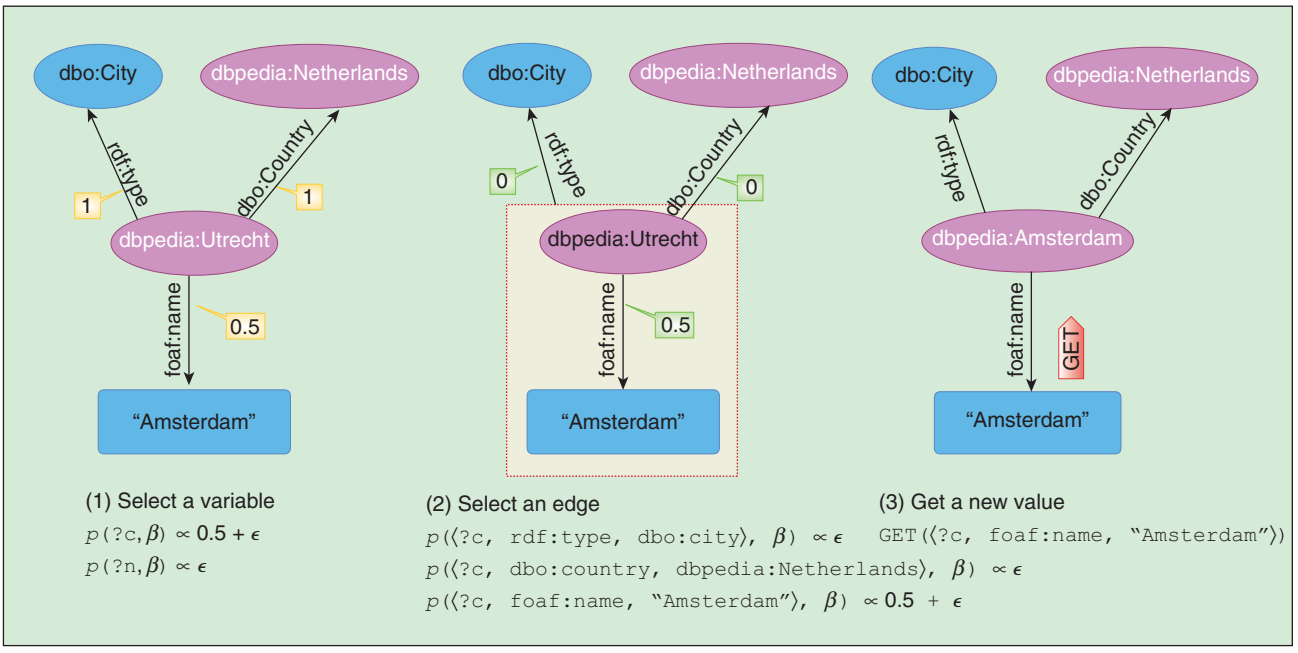


FIGURE 5 The three steps followed by the local search strategy for changing one of the bindings. In this example, the candidate solution {`?c = dbpedia:Utrecht,?n = dbpedia:Netherlands`} is changed into {`?c = dbpedia:Amsterdam,?n = dbpedia:Netherlands`}.

and under a similar time constraint, a standard query engine was found to only be able to solve the simplest queries of the test set.

The development of eRDF is ongoing and new data layers and alternative optimisation strategies are being added and tested as new ideas and use-cases are being found. The main objective of the current investigations is to ensure eRDF performs equally well on both simple and complex queries.

B. Swarm Computing for Logical Entailment

In our second project, we tackled the challenge of distributed Semantic Web entailment with a self-organizing swarm of agents that explore RDF graphs. Our framework is distributed in two ways: the data is stored in distributed dynamic networks of RDF graphs, and the reasoning task is distributed amongst a number of very light-weight agents. In the following, we summarize our work. Interested readers are invited to read [24],[41] and visit http://beast-reasoning.net/ for more details[8].

1) *Semantic Web Entailment:* Logical entailment is defined as the task of finding triples that can be derived based on the semantics of the underlying formalism. We present our reasoning mechanism with the help of RDF(S), but our framework can be extended to OWL 2 RL/RDF rules, or to custom rules that for example add `foaf:knows` relationships between co-authors. The RDF(S) closure of a set of triples T is defined as the materialization of all those entailed triples with respect to RDF(S) model theory [42], and is usually denoted by T^*. It is calculated by applying a set of entailment rules to all triples in the data set until no new triples can be derived. This process can be automated and is (in contrast to our swarm-based alternative) usually performed by an exhaustive search and logical deductions. An RDFS entailment rule consists of two parts: an antecedent with one or two triples as arguments, and a consequent, which is to be added as a new

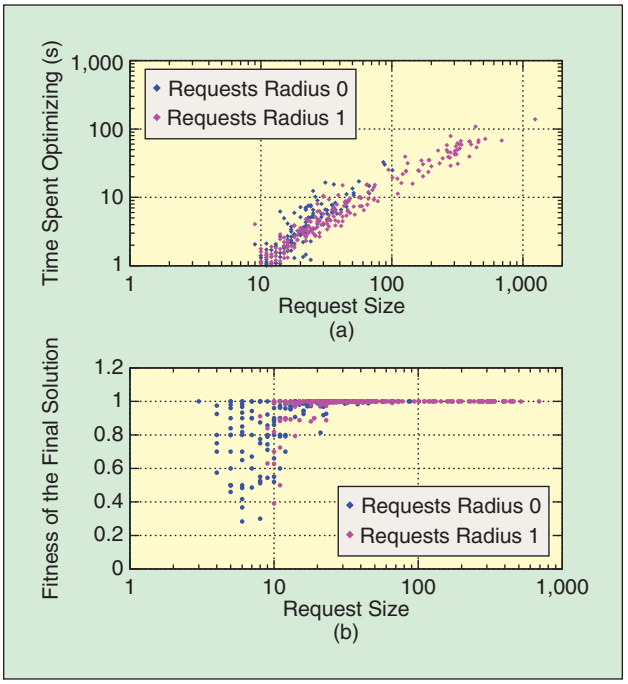


FIGURE 6 Fitness of the first solution streamed back by eRDF and the time the optimization process took to reach it. The results are for requests of different sizes and complexity (radius). (a) Optimization time until a first result is streamed back by the optimizer. (b) Fitness of the first result found, depending on the request size.

[8]Our swarm-based reasoning approach has recently been implemented on top of the fully distributed self-organized semantic storage service "S4" [20]

LISTING 1. Two RDF graphs about publications

```
cg:ISWC08
 pub:title "Anytime Query Answering in
  RDF through Evolutionary Algorithms";
 rdf:type pub:InProceedings;
 pub:author people:Gueret;
 pub:author people:Oren;
 pub:author people:Schlobach;
 pub:cites fvh:SWP.

fvh:SWP
 pub:title "Semantic Web Primer";
 rdf:type pub:Book;
 pub:author people:Antoniou;
 pub:author people:vanHarmelen.
```

triple to the graph. Table 5 lists all RDFS entailment rules with two triples as antecedent (RDFS entailment rules with one triple as antecedent are trivial).

Listing 1 contains two RDF graphs about two publications `cg:ISWC08` and `fvh:SWP`, which are described with the ontology `pub` (for publications) and linked to a `people` data set. On the Semantic Web those two graphs are expected to be administered and served independently by their respective authors.

As shown in Listing 2, the triples in the graphs are enriched with schema information about the ontologies. For example with the information that `pub:InProceedings` is a particular type of `pub:Publication`, the indication that every `people:Person` is also a `people:Agent`, and also that the property `pub:author` can only be used to relate to a `people:Person`.

TABLE 5 Some RDFS entailment rules. These rules are applied to a set of triple to derive and express information which is otherwise implicit.

RULE	IF GRAPH *G* CONTAINS	THEN ADD
rdfs2	*p* `rdfs:domain` *c*. and *s p o*.	*s* `rdf:type` *c*
rdfs3	*p* `rdfs:range` *c*. and *s p o*.	*o* `rdf:type` *c*.
rdfs5	p_1 `rdfs:subPropertyOf` p_2. and p_2 `rdfs:subPropertyOf` p_3.	p_1 `rdfs:subPropertyOf` p_3
rdfs7	p_1 `rdfs:subPropertyOf` p_2. and *s* p_1 *o*.	*s* p_2 *o*.
rdfs9	c_1 `rdfs:subClassOf` c_2. and *s* `rdf:type` c_1.	*s* `rdf:type` c_2
rdfs11	c_1 `rdfs:subClassOf` c_2. and c_2 `rdfs:subClassOf` c_3.	c_1 `rdfs:subClassOf` c_3

LISTING 2. Some RDFS statements

```
pub:InProceedings rdfs:subClassOf
  pub:Publication.
people:Person rdfs:subClassOf
  people:Agent.
pub:author rdfs:range people:Person.
```

Based on the RDFS semantics, it can be derived that `cg:ISWC08` is of type `pub:Publication` (rdfs9), and that all authors are instances of the class `people:Person` (rdfs3), and thus `people:Agent` (rdfs9). The traditional way of calculating these consequences requires the data and the rules to be centralized, and to be treated as if they were one single data set. Given the nature of the Semantic Web as discussed in Section II-B, this assumption is often unrealistic, and undesirable. Instead, we need algorithms that keep the data at their origin and that are able to adapt to dynamic information, with local control.

2) *Reasoning as graph traversal:* The basic idea underlying our approach is that the reasoning task can be decomposed by distributing complementary entailment rules to members of a swarm, so that each individual is responsible for the application of only one rule. Agents thus locally expand the knowledge by traversing RDF graphs and applying their rules on the triples they visit.

Definition 1 (Reasoning as graph traversal): Let T be an RDF graph, N_T the set of all nodes in T and M_T the memory that each agent is associated with. RDF graph traversal reasoning is defined as a triple (t, B_T, M_T), where each $b \in B_T$ is a transition function, referring to a *(reasoning) agent* $rb: M_T \times t \times N_T \rightarrow M_T \times t' \times N_T$ that takes a triple t of the graph as input, moves to an adjacent node, and depending on its memory, adds a new RDF triple t' to the graph.

There are different type of agents, one for each RDF(S) entailment rule. There are two possibilities with regards to their initialization: either the schema is pre-processed and agents are generated for schema triples of corresponding patterns, or agents initialize themselves whenever they encounter a triple of a certain pattern. Let us regard for example rule rdfs3 from Table 5, which defines range restrictions: whenever a triple with the pattern $\langle p,$ `rdfs:range`, $c \rangle$ is encountered, an agent responsible for this piece of schema information is initialized as a function *rb*3 with the associated memory *rb*3{*p*, *c*}.

Table 6 contains the agents that are needed for RDFS reasoning (in the case where the schema is pre-processed) with their respective inference rules. Reasoning agents *rb*2 and *rb*3 apply the semantics of `rdfs:domain` and `rdfs:range`, while agents *rb*7 and *rb*9 generate the inferences of `rdfs:subPropertyOf` and `rdfs:subClassOf`. They are referred to as domain-agent, range-agent, subproperty-agent and subclass-agent.

TABLE 6 Functioning of reasoning agents.

AGENT	SCHEMA TRIPLE	AGENT MEMORY	IF PATTERN FOUND	THEN ADD
rb2	*p* `rdfs:domain` *c*	rb2 {*p*, *c*}	*s p o*	*s* `rdf:type` *c*
rb3	p `rdfs:range` c	rb3 {*p*, *c*}	*s p o*	*o* `rdf:type` *c*
rb7	p_1 `rdfs:subPropertyOf` p_2	rb7{p_1, p_2}	*s* p_1 *o*	*s* p_2 *o*
rb9	c_1 `rdfs:subClassOf` c_2	rb9{c_1, c_2}	*s* `rdf:type` c_1	*s* `rdf:type` c_2

In our prototype implementation, all schema triples are preprocessed, and the transitive subclass and subproperty closure is calculated before the agents are created. Thus, *rb*5 and *rb*11 are not created. This makes our reasoning safe against ontology hijacking, *i.e.* non-authoritative extensions of ontologies [43]. On the other hand, completeness with respect to the official RDF(S) semantics cannot be guaranteed. The more generic approach (which is shown to converge towards completeness below) is to introduce a subclass-transitivity-agent *rb*11 which searches for connected subclass triples and writes new subclass triples to the graph. These triples can then be used by subclass-agents to update the schema information and to apply it on further triples. The property-transitivity-agent *rb*5 would work accordingly.

a) *Convergence towards the complete closure:* The closure T^* over a data set T contains all triples that follow from the RDF(S) semantics. In our framework, entailment rules are instantiated by the schemata and embodied by agents. Let $b_1, \ldots b_n$ be a swarm with at least one individual per type. The complete closure T^* is derived when the union of the agent outputs $b_1(t_1) \cup \ldots \cup b_n(t_n) \equiv T^*$.

Sketch: to prove that the method converges towards completeness, it has to be shown that all elements of T^* are inferred eventually, *i.e.* that each agent b_m infers the complete closure $b_m(t_m^*)$ of the rule it embodies. Given the agent functions as defined above, an agent infers t_m^* when it visits all triples of the graph that match its inference pattern. This can be achieved by a complete graph traversal, which is possible and can be performed according to different strategies, such as random walk, breadth- or depth-first. It has to be performed repeatedly, as other agents can add relevant triples. T^* is reached when the swarm performed a complete graph traversal without adding a new inferred triples to the graph. Given the possibility of randomly jumping to other nodes within the graph, which also prevents agents from getting stuck in local maxima, the same holds for unconnected (sub-)graphs. When a swarm consists of s members $b_m^1, \ldots b_m^s$ per type, the individuals of one type can infer $b_m(t_m^*)$ collectively.

b) *Example of derivation:* Let us consider the two RDF graphs from our publication example. Figure 7 shows the graph for the first publication. Dashed arrows denote implicit links that are to be derived by reasoning.

We generate one agent per schema triple: a range-agent $rb3_1$ with the memory of `pub:author` and `people:Person` applies the range-triple ⟨`pub:author, rdfs:range, people:Person`⟩, while an agent $rb9_1$ with the memory `people:Person` and `people:Agent` is responsible for the subclass triple ⟨`people:Person, rdfs:subClassOf, people:Agent`⟩. A similar subclass agent is created for the other subclass triples. All agents are distributed randomly over the graph. For example, agent $rb3_1$ starts at node `fvh:SWP` and now has two options. Moving to "SW Primer" leads it to a cul-de-sac, so that it walks back via `cg:ISWC08` towards `cg:Oren`. At node `cg:Oren`, the

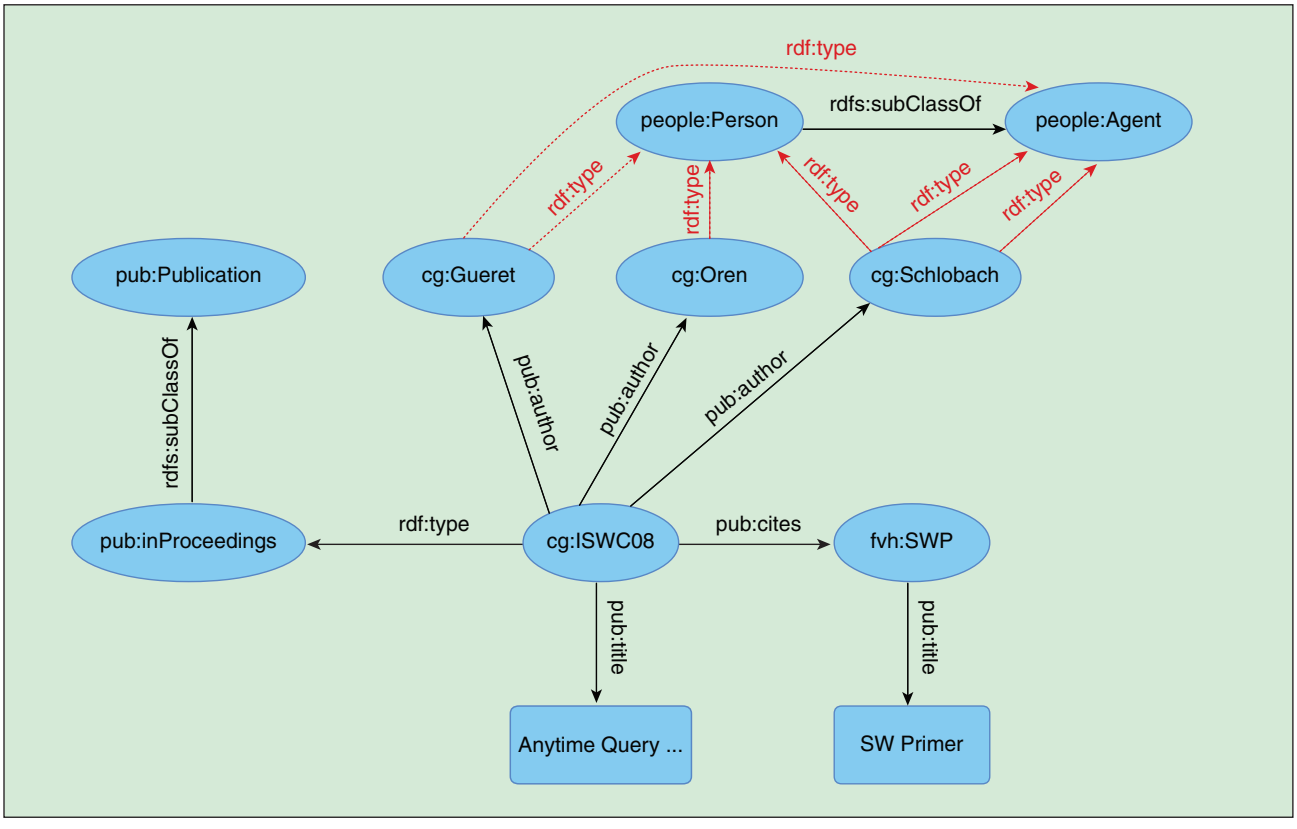


FIGURE 7 An exemplary RDF graph with inferred triples indicated in red.

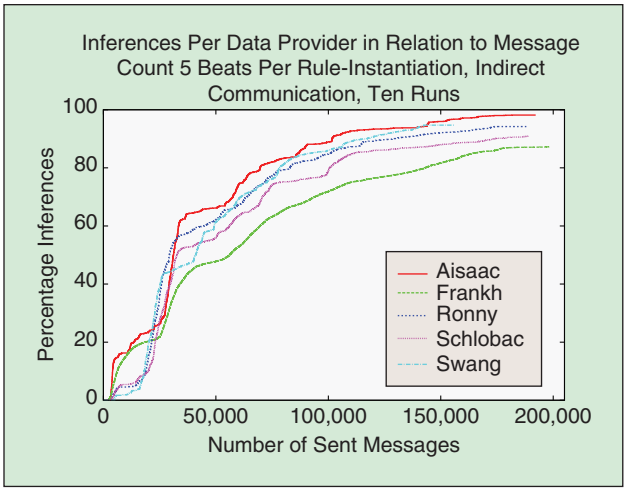


FIGURE 8 The number of inferred triples converge towards closure.

walked path is ⟨`cg:ISWC08, pub:author, cg:Oren`⟩ which means $rb3_1$'s pattern matches the walked triple. It thus adds the triple ⟨`cg:Oren, rdf:type, people:Person`⟩ to the graph. When, after walking other parts of the graph, the subclass agent $rb9_1$ chooses to follow the new `rdf:type` link from `cg:Oren to people:Person`, it finds its memory condition matched, and adds the triple ⟨`cg:Oren, rdf:type, people:Agent`⟩ to the graph, and so forth. This example highlights the challenges faced by the approach: unnecessary paths need to be investigated and the order of visiting agents is important ($rb3_1$ had to be at `cg:Oren` first).

c) *Movement Control:* Our agents operate on an energy metaphor: moving in the graph costs energy and new inferences are rewarding food. This is modeled via a happiness function: the agents are created with an initial happiness value. For every step from RDF node to RDF node made in the graph, the happiness decreases. When an agent finds a new inference, its happiness increases. With this simple mechanism, an agent adapts to its environment: while it is successful and happy, it will probably find even more applications of its rule, whereas an agent that did not find a new inference for a while will either change its rule instantiation, its type, its location or die out of starvation.

In our framework, agents can move to other locations in the network to search for applications of their rules. In our prototype, we apply a simple routing strategy using Bloom filters [44] so that agents are routed to locations that contain elements that they are searching for. Another promising routing strategy would be pheromone based. Distributed graphs can also be bridged by crossing mapping links such as `owl:sameAs`. As per analogy with real ants that find the shortest paths to food sources based on pheromone trails, our swarming agents also leave behind pheromone trails. To choose the next edge to follow, an agent parses all possible ongoing options. If it finds an application of its inference-pattern, it chooses the corresponding path and fires its rule. Otherwise, it categorizes the options into two sets: triples which are promising because they have not been visited before by agents of the same rule instantiation (following other agents is subject to future research) and the triples that already have been visited. If promising options are available, the agent randomly chooses one of them. Otherwise, the ongoing path is chosen with an equation which is inspired by Ant Colony Optimization [45], preferring paths that have been less threaded upon in the past instead of paths that have been frequented.

3) Experiments and conclusion: To prove the concept, we implemented a system based on AgentScape [46], a middle-ware layer that supports large-scale agent systems. It is based on locations, where agents can reside, and active entities that are defined according to the weak notion of agency [47]. Agents can communicate via message passing and migrate from one location to another. In our prototype, each agent is an autonomous agent. Every graph T_i is administered by an agent that is referred to as data provider and linked to other data providers. On each location, there is one data provider which does not migrate. Agents do migrate from location to location and communicate with the local data provider. Reasoning agents migrate to the data, perform local calculations and move on to the next location. Thus, only the agents with their rules and memory are moving in the network. Based on this implementation, we evaluated the feasibility of the approach.

Our experiments have been based on a number of publication files of members of our department. Each of these RDF graphs is a administered by a data provider that resides at a distinct location, e.g. at the computer of the respective colleague, and that is named after the corresponding graph. Based on the employed FOAF and SWRC schemata, we created 195 unique subproperty, subclass, domain and range agents. In the experiments, a swarm that consists of 5 agents per unique rule-instantiation traverses the graph. The swarm operates as described above.

Figure 8 shows how the degree of completeness (*i.e.* the percentage of found inferences in comparison to the output of a standard reasoner) per dataset is rising in relation to the number of sent messages (a message is a request for ongoing options). We can observe a typical anytime behavior: the more messages sent, the more inferences are inferred. In the start-phase, new inferences are found easily, the difficult task is to detect the last remaining inferences. Nevertheless, we can observe that the output converges towards closure.

To conclude, we envisage the Web of Data as an *eternal adaptive anthill* that has decentralized accessible graphs which are constantly reasoned over by restless agents. Because agents can easily deal with added and even deleted triples, we claim that swarm based reasoning is in principle more adaptive and robust than approaches that rely on indices and centralized logical deductions. Also, this reasoning procedure might help in setting up a decentralized publishing model that allows users control over their personal data.

V. Conclusion

The Semantic Web is a complex system made of large-scale, dynamic and potentially incoherent data. State of the art techniques currently employed to deal with this data have not been designed to face such challenges and can not be employed on the actual content of the Semantic Web. Instead, they are applied on curated snapshots of parts of it. In this paper, we discussed the limitations of such an approach and the need to re-consider the basic building blocks of Semantic Web data consumption in light of optimisation problems. In particular, we discussed how the typical tasks of Querying, Storage, Entailment, Consistency checking and Mapping can be rephrased from a logic problem into an optimization problem.

As proven by the work done in this emerging research field, Evolutionary and Swarm computing are particularly suitable solvers for these problems. We sustained this argument by highlighting the current state of the art of the subject and going into details for two approaches we developed to respectively deal with query answering and reasoning. The first addressing the problem of querying with evolutionary computing and the second using swarm computing to compute entailments.

References

[1] M. Dean and G. Schreiber, "Web Ontology Language (OWL): Reference," *W3C Recommendation*, 2004.

[2] D. Fensel and F. v. Harmelen, "Unifying reasoning and search to web scale," *IEEE Internet Comput.*, vol. 11, no. 2, pp. 96–95, 2007.

[3] C. Guéret, S. Wang, P. T. Groth, and S. Schlobach, "Multi-scale analysis of the web of data: A challenge to the complex system's community," *Adv. Complex Syst.*, vol. 14, no. 4, pp. 587–609, 2011.

[4] S. Klarman and V. Gutiérrez-Basulto, "Two-dimensional description logics for context-based semantic interoperability," *in Proc. AAAI*, W. Burgard and D. Roth, Eds. AAAI Press, 2011.

[5] N. Gibbins and N. Shadbolt, *Resource Description Framework (RDF)*, vol. 2004. World Wide Web Consortium, 2009, pp. 4539–4547.

[6] A. Miles and S. Bechhofer, "SKOS," *W3C Recommendation*, 2009.

[7] R. Cyganiak and A. Jentzsch. (2011, Oct.). Linking open data cloud diagram [Online]. Available: http://lod-cloud.net/

[8] J. Bezdek, S. Rajasegarar, M. Moshtaghi, C. Leckie, M. Palaniswami, and T. Havens, "Anomaly detection in environmental monitoring networks [application notes]," *IEEE Comput. Intell. Mag.*, vol. 6, pp. 52–58, May 2011.

[9] F. van Harmelen, A. ten Teije, and H. Wache, "Knowledge engineering rediscovered: Towards reasoning patterns for the semantic web," *in Proc. K-CAP*, Y. Gil and N. F. Noy, Eds. ACM, 2009, pp. 81–88.

[10] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," *W3C Recommendation*, pp. 1–106, Jan. 2008.

[11] J. Broekstra, A. Kampman, and F. Van Harmelen, "Sesame: A generic architecture for storing and querying RDF and RDF schema," *Lect. Notes Comput. Sci.*, vol. 2342, pp. 54–68, 2002.

[12] A. Harth and S. Decker, "Optimized index structures for querying RDF from the web," *in Proc. Web Congr. 2005 LAWEB*, pp. 1–10.

[13] E. Oren, C. Guéret, and S. Schlobach, "Anytime query answering in RDF through evolutionary algorithms," *in Proc. Int. Semantic Web Conf. (ISWC)*. Berlin: Springer-Verlag, 2008, vol. 5318, pp. 98–113.

[14] C. Guéret, E. Oren, S. Schlobach, and M. Schut, "An evolutionary perspective on approximate RDF query answering," *in Proc. Scalable Uncertainty Management*. Berlin: Springer-Verlag, 2008, vol. 5291, pp. 215–228.

[15] C. Guéret, P. Groth, E. Oren, and S. Schlobach. (2010, Oct.). eRDF: A scalable framework for querying the web of data, Vrije Universiteit Amsterdam, Tech. Rep. [Online]. Available: http://dl.dropbox.com/u/2137510/erdf_technicalreport.pdf

[16] H. Stuckenschmidt, R. Vdovjak, G.-J. Houben, and J. Broekstra, "Index structures and algorithms for querying distributed RDF repositories," *in Proc 13th Conf. World Wide Web* (WWW 2004), p. 631.

[17] M. Dorigo and G. D. Caro, "AntNet: Distributed stigmergetic control for communications networks," *J. Artif. Intell. Res.*, vol. 9, pp. 317–365, 1998.

[18] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien, "The dynamics of collective sorting robot-like ants and ant-like robots," in *Proc. 1st Int. Conf. Simulation of Adaptive Behavior on From Animals to Animats*. Cambridge, MA: MIT Press, 1990, pp. 356–363.

[19] R. Tolksdorf and R. Menezes, "Using swarm intelligence in linda systems," *in Proc. 4th Int. Workshop Engineering Societies in the Agents World (ESAW)*. Springer-Verlag, 2003, p. 2004.

[20] H. Mühleisen, A. Augustin, T. Walther, M. Harasic, K. Teymourian, and R. Tolksdorf, "A self-organized semantic storage service," *in Proc. 12th Int. Conf. Information Integration and Web-based Applications and Services*, Paris, France, Nov. 2010, pp. 357–364.

[21] C. Guéret, N. Monmarché, and M. Slimane, "Sharing resources in a p2p network with artificial ants," *J. Math. Model. Algorithms (JMMA)*, vol. 6, pp. 345–360, 2007.

[22] J. Urbani, S. Kotoulas, J. Maassen, F. V. Harmelen, and H. Bal, "Owl reasoning with WebPIE: Calculating the closure of 100 billion triples," *Semantic Web Res. Applicat.*, vol. 6088, pp. 213–227, 2010.

[23] K. Dentler, R. Cornet, A. Ten Teije, and N. De Keizer, "Comparison of reasoners for large ontologies in the OWL 2 EL profile," *Semantic Web J.*, vol. 1, pp. 1–5, 2011.

[24] K. Dentler, C. Guéret, and S. Schlobach, "Semantic web reasoning by swarm intelligence," *in Proc. 5th Int. Workshop Scalable Semantic Web Knowledge Base Systems (SSWS), Collocated at the 8th Int. Semantic Web Conf. (ISWC)*, Oct. 2009.

[25] E. W. Beth, "Semantic entailment and formal derivability," *Proc. Section of Sciences Koninklijke Nederlandse Akademie van Wentenschappen*, vol. 18, no. 13, pp. 309–342, 1955.

[26] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[27] F. Lardeux, F. Saubion, and J.-K. Hao, "GASAT: A genetic local search algorithm for the satisfiability problem." *Evol. Comput.*, vol. 14, no. 2, pp. 223–253, 2006.

[28] H. A. Abbass, "A single queen single worker honey bees approach to 3-SAT," *in Proc. Genetic and Evolutionary Computation Conf. (GECCO2001)*.

[29] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov, "Silk—A link discovery framework for the web of data," *Language*, vol. 538, pp. 1–6, 2009.

[30] S. Wang, G. Englebienne, and S. Schlobach, "Learning concept mappings from instance similarity," *in Proc. 7th Int. Conf. The Semantic Web (ISWC '08)*. Berlin: Springer-Verlag, pp. 339–355.

[31] P. Cudré-Mauroux, "Emergent semantics," Ph.D. dissertation, EPFL, Lausanne, 2006.

[32] P. Cudré-Mauroux and K. Aberer, "A necessary condition for semantic interoperability in the large," *in Proc. Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems* (*ODBASE* 2004).

[33] P. Cudré-Mauroux, M. Jost, and H. D. Meer, "idMesh: Graph-based disambiguation of linked data," *Data Manage.*, pp. 591–600, 2009.

[34] J. Wang, Z. Ding, and C. Jiang, "GAOM: Genetic algorithm based ontology matching," in *Proc. 2006 IEEE Asia-Pacific Conf. Services Computing (APSCC'06)*. Washington, DC: IEEE Computer Society, pp. 617–620.

[35] J. Martinez-Gil, E. Alba, and J. F. A. Montes, "Optimizing ontology alignments by using genetic algorithms," in *Proc. Nature Inspired Reasoning for the Semantic Web (NatuReS), (CEUR Workshop*, vol. 419), C. Guéret, P. Hitzler, and S. Schlobach, Eds., Oct. 2008.

[36] J. Bock and J. Hettenhausen, "Discrete particle swarm optimisation for ontology alignment," *Inform. Sci.,* to be published.

[37] S. Wang, G. Englebienne, C. Guéret, S. Schlobach, A. Isaac, and M. Schut, "Similarity features, and their role in concept alignment learning," in *Proc. 4th Int. Conf. Advances in Semantic Processing (SEMAPRO2010)*. IARIA Press.

[38] O. Hartig, C. Bizer, and J.-C. Freytag, "Executing SPARQL Queries over the Web of Linked Data," in *Proc. Informatikhuberlinde* (Lecture Notes in Computer Science, vol. 5823),A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, Eds. Berlin: Springer-Verlag, 2009, pp. 293–309.

[39] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. New York: Springer-Verlag, 2003.

[40] Y.-S. Ong, M. Lim, and X. Chen, "Memetic computation—Past, present & future [research frontier]," *IEEE Comput. Intell. Mag.*, vol. 5, pp. 24–31, May 2010.

[41] K. Dentler. (2009). Semantic web reasoning by swarm intelligence [Online]. Available: http://beast-reasoning.net/thesis.pdf

[42] P. Hayes and B. McBride, "RDF semantics," *W3C Recommendation*, 2004.

[43] A. Hogan, A. Harth, and A. Polleres, "SAOR: Authoritative reasoning for the web," *Semantic Web*, pp. 76–90, 2008.

[44] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[45] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, pp. 28–39, Nov. 2006.

[46] B. J. Overeinder and F. M. T. Brazier, "Scalable middleware environment for agent-based internet applications," *Lect. Notes Comput. Sci.*, vol. 3732, pp. 675–679, 2006.

[47] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Eng. Rev.*, vol. 10, no. 2, pp. 115–152, 1995.