An Efficient Constraint Planning Algorithm for Distributed Bio-Ontologies*

Ming Fang, Weiling Li, Rajshekhar Sunderraman

Department of Computer Science,
Georgia State University,
Atlanta, GA 30303
{mfang1@student.gsu.edu, wli16@student.gsu.edu, raj@cs.gsu.edu}

Abstract. The data of Semantic Web exist in machine readable format called RDF, in order to promote data exchange on the web based on their semantics. Due to the nature of biological data, bio-ontologies tend to be very large, distributed, and interconnected. Thus, maintaining constraints and enforcing data consistency become very challenging. In previous study, we conducted a pioneer study and presented a framework for checking global constraints and ensuring integrity on data that span multiple ontologies. As an update is issued to a single site, global constraints that can be potentially violated are broken down into sub constraints that only involve a very small subset of ontologies. The checking of sub constraints runs effectively in parallel and returns results about each subset. The collection of these results determines the violation of global constraints. In this work, we present an efficient constraint planning algorithm for distributed bio-ontologies. This algorithm serves as the key part of the global constraint checking framework. This algorithm takes a number of distributed but interconnected bio-ontologies and a set of global constraints expressed in logic programming as inputs, and produces a set of sub-constraints in Semantic Web query language SPARQL for constraint checking. An working example is presented at the end.

Keywords: Semantic Web, OWL, Distributed Bio-Ontology, Integrity
Constraints.

1 Introduction

OWL ([1]) has been widely adopted in areas like science and commerce. One reason for its popularity is its ability to formally describe complex concepts and relationships among concepts. More importantly, OWL provides a way to facilitate automated reasoning at both the conceptual and the instance level ([2]). Although numerous bio-ontologies, such as Cancer Ontology, Gene Ontology, Human Disease Ontology, are available in RDF ([3]) form, there is still a huge demand for developing more OWL ontologies for various purposes.

* This work is supported by Molecular Basis of Disease, GSU
In previous studies, we uncovered the global constraint violation issue. As an initial investigation as well as an evidence to show ontologies are interdependent and dynamic, we explored a group of well-established and well-known ontologies from the biomedical field. We have investigated 81 bio-ontologies from The Open Biological and Biomedical Ontologies [4] and the Ontology Lookup Service [5]. In average, one biomedical ontology references to/depends on three other bio-ontologies. Although some ontologies in the set do not change very often, the majority of these ontologies updates 21.31 times/year, with 7010.57 lines/update in average.

Since there is no formal mechanism available to enforce data consistency among distributed ontologies while they are interconnected and constantly changing, we designed a frame that is faster and of less network traffic than the naïve method. The infrastructure in figure 1 contains:

- **Update Parser**: Parses a user specified update, and return involved ontology objects.
- **Metadata Extractor**: Extract the set of global constraints that could be potentially violated by the update statement.
- **Constraint Planner**: runs an effective algorithm to generate sub constraints that will be dispatched to remote ontology sites.
- **Constraint Optimizer**: reorganizes the order of sub constraints in order to achieve higher efficiency.
- **Constraint Executor**: Execute sub constraints in parallel, and made decisions about the global constraint upon receiving the results of sub constraints.

![Internal Architecture of Constraint Checker](image)

**Fig. 1.** Internal Architecture of Constraint Checker

In section 2, we will introduce the constraint planning algorithm that is the central piece of the Constraint Planner. It is efficient because it breaks a global constraint that requires bringing in multiple larger ontologies into several sub-constraint
queries that are sent to remote ontologies in parallel. In section 3, we will provide a simple working example of this algorithm to conclude this poster.

2 The Constraint Planning Algorithm

The inputs of this algorithm contain an update statement $U$ to ontology site $S$, and a list of global constraints $C$. As we step through the constraint checker to the constraint planning stage, we will also have the following two pieces of information available: (1) Ontology Object List (OOL) that identifies what predicates are to change and their new values; (2) Constraint-Source Table (CST) that specifies the sites involved in this update $U$ for each global constraint in $C$. The output will be a list of sub-constraints for each global constraint in $C$ affected by the update $U$.

**Constraint Planning Algorithm:**

```
For each constraint $c$ in the list of global constraints $C$
    For each site $s$ from CST that is affected by constraint $c$ in the update
        If site $s$ is not where the update happens
            Then generate sub-constraints in the form of SPARQL queries from all the predicates (available in OOL) that reference to site $s$ using appropriate conditions. Include arithmetic queries when necessary.
        Elseif site $s$ is where the update happens
            Then If there are variables whose values are from $s$
                Then generate sub-constraints similar to the above case
                Else If there are variables whose values are from remote sites
                    Then generate queries to retrieve values from those remote sites first, then use those values to generate sub-constraints using similar method above
        End for
    End for
End for
```

3 An Example

In this example, we used the Anatomical Entity Ontology and Cancer Ontology in Protégé environment. The global constraint we used was a useful OWL-style constraint called Specific Individual Type. It requires that the explicitly declared individual of a concept or relationship (property) in the instance data must be the most specific one. This constraint can be expressed in logic programming as:
Ming Fang, Weiling Li, Rajshekhar Sunderraman

\[ \text{inAnatomicalPart}(x, y) \land \text{inAnatomicalPart}(x, z) \land \text{cancer}(x) \land \text{anatomicalPart}(y) \land \text{anatomicalPart}(z) \land \text{isSubClassOf}(y, z). \]

Upon receiving the result of the first two queries, by running the constraint planning algorithm, we will have sub-constraint query 3 targeted at Cancer Ontology and queries 4, 5, 6 targeted at the Anatomical Entity Ontology.

\[
\begin{align*}
C_1 &= \text{SELECT } ?x, ?y \\
&\text{WHERE} \\
&\{ ?x \text{ rdf:inAnatomicalPart } ?y \} \\
C_2 &= \text{SELECT } ?x, ?z \\
&\text{WHERE} \\
&\{ ?x \text{ rdf:inAnatomicalPart } ?z \} \\
C_3 &= \text{SELECT } ?x \\
&\text{WHERE} \\
&\{ ?x \text{ rdf:hasClass } \text{Cancer} \} \\
C_4 &= \text{SELECT } ?y \\
&\text{WHERE} \\
&\{ ?y \text{ rdf:hasClass } \text{AnatomicalPart} \} \\
C_5 &= \text{SELECT } ?z \\
&\text{WHERE} \\
&\{ ?z \text{ rdf:hasClass } \text{AnatomicalPart} \} \\
C_6 &= \text{SELECT } ?y, ?z \\
&\text{WHERE} \\
&\{ ?y \text{ rdf:isSubClassOf } ?z \}
\end{align*}
\]

In this way, we avoid bringing both ontologies onsite for constraint checking.

4 References


