Graph Analysis for the Semantic Web

David Haglin Bob Adolf

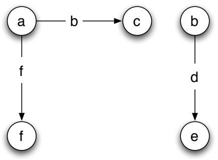
1 The Semantic Web

Over the last decade, the web community has generated a significant interest in computer-driven information exchange across the web, where data is held, accessed, and enriched solely by autonomous, cooperative communication between Internet-attached machines. This vision is "Web 3.0" or, more commonly, the *Semantic Web*. One of the guiding principles for the development of the semantic web is that by aggregating enough data in a common environment, cross-linking relationships will eventually form between heterogeneous datasets, and this fusion will enable automatic knowledge discovery. As a result, the semantic web community has begun to look for large-data analysis techniques which can be applied to this projected data cloud, and graph algorithms are a popular candidate.

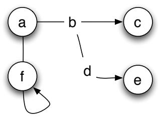
The W3C standard for representing information on the semantic web is the *Resource Description Framework*[1] or RDF. From a high-level, RDF is a set of triples (S, P, O), which can roughly be viewed as a set of directed, labeled edges, implicitly describing a graph. In our opinion, this observation gives rise to the notion of a *semantic graph database* (SGD), a persistent, query-able graph which would form the core of any complex graph analysis done on a corner of the semantic web.

1.1 Graph Definitions

Unfortunately, there are complications to this rough equivalence between a set of ordered triples and a labeled digraph. Foremost amongst these is the issue of value unification on RDF triples. Typically, when creating a graph structure from a list of edges, one creates a table of the unique values found in the source and destination of each edge, and then unifies the sources and destinations into nodes. Since RDF data is formed from triples, however, there is no restriction that a source value (called a subject in RDF) or destination value (an object) cannot be found as an edge label (a predicate). The implication of this is that, assuming one unifies all identical labels, either graph edges can point to other edges or labels found on edges are not considered the same as identical labels found on nodes. Consider these sample triples: $(a, b, c), (b, d, e), (a, f, f)$. Which of the following should we use to represent the corresponding "graph" structure?



(a) subjects and objects unified



(b) completely unified

[1] http://www.w3.org/standards/techs/rdf

1.2 Dirty Data

Typically, structures like the example above arise only in particular situations, specifically *reification* and *ontological triples*. Reification is the name for an approach to allow RDF triples to describe other RDF triples. This occurs with the construction of four individual triples which describe the target triple *by value*. For example, the triple $(a,\ b,\ c)$ would be reified by creating these four triples:

$$(\_{:}x, rdf{:}subject, a), (\_{:}x, rdf{:}predicate, b), (\_{:}x, rdf{:}object, c), (\_{:}x, rdf{:}type, rdf{:}Statement)$$

where $\_{:}x$ represents an anonymous name for the triple (a, b, c). Ontological triples are similar, in that they can explicitly represent relationships between predicates, for example $(a, rdfs{:}subClassOf, b)$.

Nominally, the W3C recommendation describing RDF semantics[2] explicitly mentions the disconnect between reifications of triples and the triples themselves. In practice, however, few in the semantic web community recognize this distinction, and because both reification and ontological triples make their way into the datasets, this calls into question the expected behavior of analysis on graphs generated from RDF input. For instance, should running connected components on an RDF graph honor ontological relationships as "connected"? This semantic web community is also interested in other graph algorithms such as shortest path and community detection, to name a few.

2 Semantic Web Analytics

2.1 SPARQL and Constrained Subgraph Isomorphism

The most common method for asking questions about RDF data is through a query language called *SPARQL*, which is a declarative language which has many similarities to SQL. At its heart, SPARQL is a pattern-matching system which revolves around matching the structure and labels in a query graph to subgraphs in a larger RDF dataset. This problem is effectively an extension of subgraph isomorphism to labeled graphs, and many of the same techniques and algorithms apply. Surprisingly, the semantic web community has not exploited this similarity. Instead, the vast majority of existing semantic web tools utilize relational database methods to compute these relationships. We leave the performance implications of this decision as an exercise for the reader.

2.2 Constrained Reachability

Another topic of interest and research in the semantic web community is path analysis. More specifically, there have been a number of research efforts to extend SPARQL to support arbitrary-length paths in addition to the subgraph patterns which make up a query. Since there is little analysis beyond *identifying* such paths, and the query structure is almost always merely a set of allowable patterns for intermediate nodes along a path, we call this *constrained reachability*. However, due to the tight dependence on SQL in current tools, few scalable implementations for these algorithms exist and support for path queries are just beginning to make an appearance in the mainstream semantic web community. We believe that solid implementations and performance modeling of these queries by the graph community could make a substantial impact.

2.3 Ontological Inference

Finally, the semantic web allows a rather unique capability beyond traditional graph analysis. Using explicit relationships between labels and a pre-defined set of rules (an ontology and inference rules, respectively), a machine can infer new edges in the graph. This has two implications for graph analysis on semantic web data: first, the input representation of a graph does not necessarily explicitly contain all of the information which it encodes, and second, to exploit inferred edges, one must either pre-compute all possible new relationships (and pay the associated cost of materializing those edges) or handle composing the desired analytic on top of a wide range of on-the-fly inferencing rules. Either way, this poses interesting challenges to the current capabilities of a graph library.

[2] `http://www.w3.org/TR/rdf-mt/#Reif`