RinsMatch: a suggestion-based instance matching system in RDF Graphs

Mehmet Aydar and Austin Melton

Kent State University, Department of Computer Science, USA
{maydar,amelton}@kent.edu

Introduction. In this paper, we present RinsMatch (RDF Instance Match), a suggestion-based instance matching tool for RDF graphs. RinsMatch utilizes a graph node similarity algorithm and returns to the user the subject node pairs that have similarities higher than a defined threshold. If the user approves the matching of a node pair, the nodes are merged. Then more instance matching candidate pairs are generated and presented to the user based on the common predicates and neighbors of the already matched nodes. RinsMatch then reruns the similarity algorithm with the merged RDF node pairs. This process continues until there is no more feedback from the user and the similarity algorithm suggests no new matching candidate pairs.

In our previous study [1], we proposed an algorithm for computation of entity similarities of an RDF graph using graph locality, neighborhood similarity, and the Jaccard measure. In the current study we use the proposed RDF entities similarity algorithm for pairing entities which may be merged if approved by the user. We make a similar assumption like the similarity flooding (SF) algorithm proposed in [2], that elements of two graphs are similar when their adjacent elements are similar. Comparing to SF, our technique requires more user interactions and more iterations for computation of entity similarity, but each time the similarity algorithm runs, it produces more accurate results assuming the user provided accurate feedback. Also, merging the RDF nodes reduces the size of the input data graph that the algorithm operates on, yielding less complexity each time.

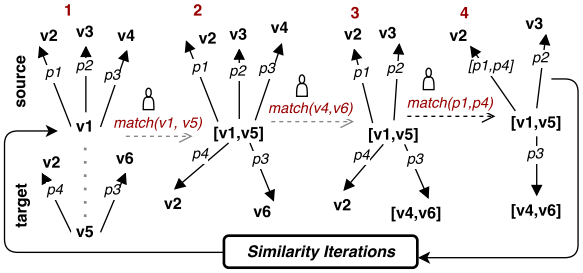

Fig. 1. Instance matching process

User Interaction. RinsMatch presents the subject node pairs that have similarities higher than a defined threshold to the user for possible instance matching. The threshold is a configurable parameter and may be determined by the user. If $s1$ and $s2$ are two subject nodes which have similarity higher than the threshold, then we denote this pair by ($s1$,$s2$). If the user approves the matching of the subject node pair ($s1$,$s2$), then RinsMatch merges the two subjects into a single

subject node which we denote by [$s1$,$s2$], and then all the predicates from both merged subjects are retained by the newly created subject [$s1$,$s2$]. RinsMatch then checks the common neighbors and predicates of $s1$ and $s2$ and generates more instance matching candidate pairs by pairing the predicates $p1$ and $p2$ to get ($p1$,$p2$) if $p1$ and $p2$ are connected to a common object from both $s1$ and $s2$. It also pairs the object nodes ($o1$,$o2$) which are connected with a common predicate by $s1$ and $s2$. RinsMatch then presents the generated matching candidate pairs to the user and merges the pairs to get [$p1$,$p2$] and [$o1$,$o2$] if the user approves that they match. RinsMatch then reruns the RDF node similarity algorithm on the new RDF graph formed by merging matching entities. The steps above are repeated until there is no more feedback from the user and no new matching pairs suggested by the matching algorithm. Figure 1 shows an example of the instance matching and merging process. As shown in the figure, based on the similarities found from the similarity iterations, at phase 1 RinsMatch suggests matching the subject nodes ($v1$,$v5$), and they are merged to get [$v1$,$v5$] with the approval of the user. On phase 2, the algorithm checks the common predicates of the new node [$v1$,$v5$]. Seeing that it connects to the neighbor nodes $v4$ and $v6$ with the common predicate $p3$, RinsMatch merges the nodes $v4$ and $v6$ to get [$v4$,$v6$] once the user approves. On phase 3, the common neighbors of the new node [$v1$,$v5$] are checked. Seeing that [$v1$,$v5$] is connected to a common neighbor $v2$ with the predicates $p1$ and $p4$, then RinsMatch presents the pair ($p1$,$p4$) to the user, and they are merged upon approval by the user to get [$p1$,$p4$]. The output graph of phase 3 is input to phase 1, and the similarity iterations are repeated until the optimum similarities and instance matching pairs are found.

Evaluation. We conducted preliminary experiments based on a subset of DBpedia and a subset of SemanticDB, a Semantic Web content repository for Clinical Research and Quality Reporting. For verification, we duplicated the original dataset and changed the names of the nodes in the duplicated dataset by following a specific naming pattern. We used the original dataset as the source, and the duplicated dataset as the target for the instance matching process, and we leveraged the node naming pattern for verification. To summarize our experiments: for the DBpedia, the source dataset had 90 triples with 60 distinct subject and predicate nodes. 100% of the nodes were matched to a target graph node semi-automatically. The algorithm generated 20 instance matching candidates with 85% accuracy. For SemanticDB, the source dataset had 2500 triples with 520 distinct subject and predicate nodes. 86% of the nodes were matched to a target graph node semi-automatically. The algorithm generated 310 instance matching candidates with 95% accuracy.

References

1. Mehmet Aydar, Serkan Ayvaz, and Austin C Melton. Automatic weight generation and class predicate stability in rdf summary graphs. In *Workshop on Intelligent Exploration of Semantic Data (IESD2015), co-located with ISWC2015*, 2015.
2. Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128. IEEE, 2002.