Implenting Blast: Blocking with Loosely-Aware Schema Techniques

Joanna Polewczyk, Parsa Badiei, Thiago Bell Felix de Oliveira

February 2019

1 Introduction and Problem Definition

Entity resolution is the task of matching entries in datasets corresponding to the same object (e.g. match the entries corresponding to a movie in two datasets). In this report, we focus on Clean-Clean entity resolution (i.e. when we want to merge different datasets without duplicates). This problem is very relevant (e.g. merging corporate databases) and asymptotically difficult: potentially every pair of items in both datasets have to be compared making it extremely difficult for big data applications. Moreover, many techniques like schema-based ER requires domain knowledge or supervised learning. It can be inefficient and excessively complicated.

A common solution for this problem is the use of a blocking step where entities' attribute values are decomposed into tokens. These are used to filter the comparison to be made: entities sharing the same token (belong to the same block) match with higher probability. There are different approaches to blocking, one could add domain knowledge to this procedure, assume nothing about the data sets and to it blindly or, as done in Blast (Blocking with Loosely-Aware Schema Techniques) [1] , the subject of this implementation report, automatically extract features from the data sets to be used for guiding the blocking process.

One important information in data sets is their attribute schema: they can differ but some attributes are expected to refer to the same characteristics of the object as attributes in the other data set. Blast uses this attribute information in the form of the (Shannon) entropy of each attribute and how they are similar to each other. Imagine there are data sets A and B. A contains the attribute *full name*, B attributes *name*, *family name.* if a pair of entries of these databases have similar values for these attributes, then they are a match with high probability. If the value of *full name* is similar to that of an attribute in B called *address* then it is not likely this would be an indication the pair refers to the same person.

2 Approach - Blast

As mentioned previously, Blast extracts features from the attributes' schema in the data sets to improve the blocking step to speed up entity resolution. Blast is composed of three phases:

- 1. Loose schema information extraction
- 2. Loose schema aware Blocking
- 3. Loose Schema aware meta blocking.

2.1 Loose Schema Information Extraction

In this first step, we wish to calculate the similarity of each pair of attributes coming from different data sets using the Jaccardian Distance. Similar pairs are grouped into clusters.

Each attribute goes through a tokenization step where each of their values (from the data set) are tokenized using a *value transformation function* that creates tokens from individual words. This collection of tokens is a representation of each attribute, the type of data they contain and also their "richness", that is, how varied their values are. If they are limited (such as a year field), then a match between pairs due to this attribute may not mean much since it is expected that many records will collide on this attribute.

We use this collection of tokens and their Jaccardian similarity to approximate a similarity metric between attributes. With this similarity measures, we generate a graph where attributes are nodes and there is an edge between a pair of nodes if their similarity is within 90% of that maximum one for the corresponding nodes. From this cluster, the connected components are obtained. These clusters are used in the next step.

2.2 Loose schema aware Blocking

In this phase, an enhanced version of token blocking is implemented. Normally, the Entities are tokenized and the entities in the datasets grouped by these tokens. This process is enhanced by distinguishing the attribute cluster (calculate on the previous phase) where the token occurs. If there is one cluster containing personal name-related attributes and another one for address information and the same word (e.g Brandt) occurs in different clusters for two entities, those entities will not be mapped to the same map for the token 'Brandt'. Furthermore, the source paper implements a heuristic called block purging to remove insignificant blocks. Those that contain more than half of the total number of entity profiles are removed from the blocking set.

2.3 Loose schema aware meta blocking

The purpose of Loose schema aware meta blocking is to obtain the blocking graph

$$G_b\{V_b, E_b, W_b\}$$

with weighted edges from the block collection done in Loose schema aware Blocking. Entities profiles are represented as nodes of the graph whereas relationships between specific entities as the edges whose edges are calculated based on a weighting taking into account the co-occurence of entities and the attributes' entropy. The weights of edges approximate the similarity of two entity profiles based on the blocks they share (and those that they do not).

The next crucial step for the blocking graph is the pruning of the edges. To be able to reduce number of relations we need remove some of the edges whose weights are below a certain threshold. This threshold is edge specific which in turn depends on the nodes. We first calculate for a threshold value for each node

$$\theta = \frac{M}{c}$$

where M is the local maximum weight of edges incident to the entity and c is arbitrary constant (c = 2). From these node-based values, an edge threshold is calculated by taking the average of the threshold values for the incident nodes. This pruning procedure removes those pairs of entities that are less likely of matching.

3 Implementation

The solution was implemented with the use of *Scala* and *Apache Spark*.

3.1 Data sets

The data sets, as well as some helper classes for manipulating them, were obtained from [2]. Due to restricted memory availability, it was not possible to use the larger datasets.

The entity profiles are stored in an RDD of type *EntityProfile.*The EntityProfile class determines the structure of our entities. It contains a Id key (*entityUrl*) and set of all attributes for the particular profile (*Set¡Attribute¿ attributes.* Attributes are structured with attribute name (String) and value (String).

3.2 Loose Schema Information Extraction

3.2.1 Attribute Information and Induction

We load the entity profile objects from a file into an RDD. Transformation operations are applied to it until we obtain an RDD of tuples (attribute_name, tokens) which are then used to compute the Jaccardian distance (all using RDDs). At this point, the Shannon entropy is calculated for each attribute which will be used later.

```
s1.intersect(s2).size.toFloat/set1.union(set2).size.toFloat
```

Next, we determine the maximum similarity of each of the attributes in the data set in order to filter candidates pairs within a constant alpha of maximum (alpha = 0.9). We, then, create a graph with nodes as entity profiles and edges representing similarities. Each maximal connected sub graph corresponds to an attribute cluster. Those entity profiles that match no other, are grouped into a special cluster for singletons which is treated as a normal one in subsequential steps.

3.3 Loose schema aware Blocking

In Token blocking we are mapping each attribute name to their respective cluster id:

```
val attrDS1toCluster : Map[String, Int] =
attrClusters.flatMap{case (clusterID, attrD1, _)
=> attrD1.map{x => (x, clusterID)}}.toMap
```

and adding the list of entities related to particular token

```
val result\_before\_purging = listOfBlocks.aggregateByKey
(List[String]())(Blocker.addEntityIdToList,
(a: List[String], b : List[String]) => a++b)
```

The last step in this phase is to perform the block purging. All blocks containing more than half of the entity profiles are discarded. At the end, we obtain a RDD containing of ((*token*, *clusterId*), *list of entities*) for each block.

```
val blocks : RDD[Tuple2[Tuple2[String, Int], List[String]]]
```

We also compute a index of entities to the blocks they occur on:

```
val flat =blocks.flatMap{ case ((token,cId),list_profiles) =>
list_profiles.map( x =>
(x, Set((token,cId))))
}.reduceByKey((a,b)=> a++b)
```

3.4 Loose schema aware meta blocking

3.4.1 Weight Graph

Having obtained the block collection from Loose schema aware Blocking, we create a weighted graph where entity profiles are nodes and the edges between them represent one or more co-occurrence of these profiles in a block. The weight depends on the on co-occurrence in blocks of the nodes it connects as well as the entropy values of the attributes in such blocks.

3.4.2 Pruning

To prune the graph, the threshold θ_i for every node needs to be determined. Therefore, we find the maximum edge incident to each node:

```
val thetaPerNodeDS1 = edges.aggregateByKey(0.0)
((mv : Double , tup : Tuple2[String, Double]) =>
math.max(mv/c, tup._2),(a:Double, b:Double) => math.max(a,b))

val thetaPerNodeDS2=edges.map{case(profileA ,(profileB , weight))
=> (profileB ,(profileA , weight))}.aggregateByKey(0.0)
((mv : Double , tup : Tuple2[String, Double])
=> math.max(mv/c, tup._2), (a: Double, b : Double)
=> math.max(a,b))

val thetaPerNode = thetaPerNodeDS1 ++ thetaPerNodeDS2
```

To decide whether an edge must be pruned or not, we need the theta of both of its node. Since doing subsequent join operations over the RDD of edges and of thetas would be costly with a lot of network operations and there are relatively a small number of values, we collect the thresholds into a map and then broadcast them to every compute node. Then we can prune those edges quite efficiently and without reshuffling data excessively. Those edges whose weight is smaller than the average of the nodes' thresholds are removed.

4 Evaluation

We used the same fully mappable Clean-Clean data sets as in the original paper (those from [2]). Due to restricted memory, the larger ones were not viable. The evaluation is done with block filtering so it is compared with the corresponding results in the paper. For the (ABT, Buy) data set we obtained different results with comparison to the reference paper. It seems as if our implementation did not provide as many matching pairs as the paper's implementation. For (DBLP, Google Scholar) we had higher precision and for (DBLP, ACM) the results were similar. Experiments were ran on a 4 core i7 processor with 6 GiB of memory.

data set 1	data set 2	precision	recall	run time
DBLP	ACM	2.78%	99.78%	25 min
DBLP	Google Scholar	0.19%	98.44%	2 hours
ABT	Buy	5.09%	92.84%	8 min

5 Project Timeline

Time	Date	Assignments	Person
2 weeks	23.11-07.12	Environment setup	all
2 weeks	07.12-21.12	Data set management	Parsa
6 weeks	21.12-01.02-	Attribute Match Induction	Thiago
6 weeks	21.12-01.02	Token Blocking	Parsa, Thiago
6 weeks	21.12-01.02	Meta blocking	Joanna, Thiago
1 weeks	17.12-23.02	Evaluation	Parsa
1 weeks	15.02-23.02	Report	all

References

[1] G. Simonini, S. Bergamaschi, and H. Jagadish, "Blast: a loosely schema-aware meta-blocking approach for entity resolution," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 1173–1184, 2016.

[2] T. P. George Papadakis, Georgia Koutrika and W. Nejdl, "Meta-blocking: Taking entity resolution to the next level," *In EEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 26, no. 8, pp. 1946–1960, 2014.