Classification of Schema-Agnostic and Schema Based Methods for Progressive Entity Resolution

Bikash Tripathy
st170082@stud.uni-stuttgart.de
University of Stuttgart

ABSTRACT

Entity Resolution[ER]- It refers to find the entity profiles that corresponds to same real world entity. While we look for similar entities in the data, data may be coming from a single source(known schema) or the multiple source(not known schema). These can be based on two ways. The first one is schema based configurations, which relies on the schema information for selecting the blocking keys, which are more distinct and with less noise for design of ER. The second approach is the schema agnostic ER. This method helps in scaling information from multiple sources, plays key role in big data integration. Since schema-agnostic approach involves, heterogeneous data from multiple sources along with high volume data, a progressive approach is far more efficient. Progressive ER methods main purpose to resolve the large data sets when there is limited time and limited computational resources available, and emitting the matches as soon as matches are found rather than waiting for solving the entire data set. To cover these aspects, this papers aims to classify the different approaches for schema-agnostic and schema based method based on progressive entity resolution.

KEYWORDS

Schema-agnostic Entity resolution, similarity based entity resolution, progressive entity resolution, Progressive sorted neighbourhood.

ACM Reference Format:
Bikash Tripathy. 2021. Classification of Schema-Agnostic and Schema Based Methods for Progressive Entity Resolution. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data is the one of the key asset for any company. Sometimes due to the wrong data entry or short forms of names of person used, duplicate might occur in the system, making it tough for find the duplicates. For example the online retail business, offer huge catalogs, which grows constantly as product received from different suppliers arrives. And an independent person change such portfolios in the business, hence duplication of records are common for such scenarios.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Schema play a major role while, deciding the ER methods which can be applied to the data sets. With ever increasing data, we call for various approaches, which can be used for different data sets available. However with emerging big data applications in health care, travel, finance, we also need ER algorithms which can solve the ER problem statements, for various schema. In the present day scenario due to this big data application, we are having heterogeneous data system for the application, making it tough for the existing approaches of schema based ER. And the approaches available tend towards the progressive methods, a progressive ER outputs the results as soon as it finds a match, rather than waiting for resolving entire data set at once. This in turn calls for a bigger picture for a classification of such algorithms, suitable for schema based methods and schema agnostic methods for progressive ER. This will contribute towards having a better understanding while choosing the ER algorithms for complex real time applications.

Progressive ER, resolves most pairs early in the process[2]. Instead of waiting for resolving the whole data set progressive approach try to reduce the overall average time needed to find the duplicates among the records. Which leads to point that, a early termination at any instant of time, progressive ER will have more matches. To achieve these, we need to estimate the similarity of all comparison candidate in order to compare the most promising matching record first. Below presenting two use case where it can be important -

1) A customer having limited time, for data cleaning or to find matching profile, and want to make best use of it. By starting the progressive ER method, and interrupt it when at certain point, will have more matches then a conventional ER. This gives a better understanding of the system early on rather than solving data set completely.

2) A developer looking for the suitable key randomly for initial investigation on the unknown data set. The real time capabilities of Progressive ER helps again in reducing the waiting period.

To present the progressive methods, we proceed by first presenting the schema based method such as Progressive Sorted neighbourhood(PSN) in section 3 and the Hierarchy of Record Partitions(HRP) in section 4. And the section 5 representing the schema agnostic methods such as Schema-Agnostic PSN (SA-PSN) method and a better improved method such as Local Schema-Agnostic PSN (LS-PSN).

2 PREREQUISITE BASICS

The entity resolution, is the method of creating linkage between the disparate data, that represent the same thing in reality. For example, let's say we have a data set of product listed in Amazon, and another data set of product listed in ebay. They might have similar names with slight modification, similar prices, and other

unique identifiers. If human look into these two records, we can identify if it really the same product or not. But surely, this do not scale up, since we have *A times E (A * E)* pairs to look up. Where *A* represents the no of items from Amazon, and *E* represents no of items from ebay. If we have 1000 products in both the data sets, no of comparison scale up to 1 million pairs to check. And in real world data sets this complexity is way higher, as they deal with high volume of data. Also not only we need to take care of only the matches, we also need to consider how much time they take to output the results of matching. And this calls for the progressive ER and advanced methods for ER.

The state-of-art progressive ER method is the *Progressive Sorted Neighbourhood(PSN)*[1]. Which can be thought as a starting point for the progressive methods. The ER method is also termed as *Duplicate detection workflow*[1], which comprises of three steps, first one namely pair-selection. In this we go for presorting using a key from the attributes, filtering out the related entities profile. Second one is the pairwise comparison, where we try to compare the profiles which are presorted. And the last one being clustering, helps in forming groups of related matches for the profile. For the progressive approaches, the first and the last step are important, where our intuition is duplicate entities will be grouped into buckets being in the first step, and then in the last step they will be emitted progressively. Hence the proposed algorithms takes care of the comparison step and independent of quality the similarity using blocking key.An example of the progressive model is the pay-as-you-go[2] approach, which helps in generating the partial result than completely resolving the data set, so that we can get some result faster. The main goal will be is to get as much as overall results as fast as possible.

An ER algorithm E takes R as records which are the entities in the real world, and the ER model outputs the final match as $M = \{r_1, r_4\}, \{r_2\}, \{r_3, r_5, r_6\}$, which shows r_1, r_4 as the same entity r_2 as independent entity. Initially, $\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}, \{r_5\}, \{r_6\}\}$ every record can be seen as individual separate entity. In this progressive approach, the result(G) of a ER algorithm E on the records R, for a given time instant t can be shown by $E(R)[t]$. Suppose we check the algorithms after 3 seconds, we might observe it has resolved into $G = \{\{r_1, r_4\}, \{r_2\}, \{r_3\}, \{r_5\}, \{r_6\}\}$ partitions, showing that r_1, r_4 have merged into the same cluster. Most ER solve these in repetitive process and hence well suited for progressive methodologies described in the further sections. This kind of method will be further applicable in the section 4 for *Hierarchy of Record Partitions*.

Blocking, a key aspect of progressiveness.It's typically a preprocessing step that aims to index together likely match profiles into buckets using a blocking key. All the attributes are sorted using a single key to collect the sorted profiles based on the key value chosen. This set of sorted profiles are aggregated and used for further finding the matching entities.

3 SCHEMA BASED PROGRESSIVE ER

Schema Based methods use the blocking key as the technique for the ER. It sorts the data using the schema-based predefined sorting keys. It pertains to structured data and hence sticks to the specific schema with known semantic and qualitative characteristics for each attribute and value pair. These blocking methods are pre sorted approached for higher precision and getting the more similar probable matches, where overall aim is focus on increasing the efficiency[4]. The progressive ER algorithm produces the results while resolving the data sets. Progressive ER, aims to one hand increases the performance and in the other hand, tries to improve the data quality.This is also well known as pay-as-you-go model[2], and Figure 1[2] represents the a general overview of comparison of matches found for conventional ER and a pay-as-you-go ER. The pay-as-you-go model is just a synonyms for the progressive methods out of so many different name used by different authors. And clearly in the Figure 1 we can see the a general overview of the performance measure for the pay-as-you-go model, where at any instant of time, no if matches found is more than the no of matches found for the conventional ER. The figure 1 presents the number of matches found verses the no of comparison for profiles, showing more no of matches fond for the pay-as-you-go model.

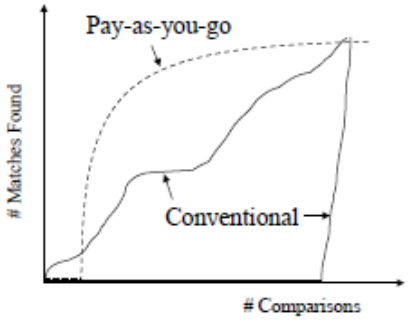

Figure 1: Pay-as-you-go ER

In the following section of 3 and 4, we have presented the schema based progressive ER named Progressive Sorted Neighborhood(PSN)[1] and Hierarchy of Record Partitions (HRP)[2], which uses prior schema information for progressive ER.

3.1 Progressive Sorted Neighborhood(PSN)

The state-of-art progressive method known is Progressive Sorted Neighborhood (PSN)[1]. The sorting is done on the basis of predefined blocking key. This keys helps in collecting the profiles in to a small buckets or groups using the blocking key chosen. After the profiles are grouped using the blocking key in to buckets, we arrange the blocking keys in a list with alphabetically order. And the corresponding profiles which were collected using blocking keys are placed in the list. Then the profiles are sorted in the list containing the profiles. This allows for the comparison of profiles by iterating over them with help of a window size, that defines the number of neighbouring elements to be compared inside the list. The main intuition behind this ides is that the records that are close in the sort order are more probable for matching or duplicates, as they are presorted with the tokens. To formalise this concept, we the pairs based on the chosen window size to emit the matches, which in turn provides an estimate about the matching likelihood of the profiles. This algorithm exploits the variable window size to

Algorithm 1. Progressive Sorted Neighborhood

Require: dataset reference D, sorting key K, window size W, enlargement interval size I, number of records N

```
1:  procedure PSNM(D, K, W, I, N)
2:      pSize ← calcPartitionSize(D)
3:      pNum ← ⌈N/(pSize − W + 1)⌉
4:      array order size N as Integer
5:      array recs size pSize as Record
6:      order ← sortProgressive(D, K, I, pSize, pNum)
7:      for currentI ← 2 to ⌈W/I⌉ do
8:          for currentP ← 1 to pNum do
9:              recs ← loadPartition(D, currentP)
10:             for dist ∈ range(currentI, I, W) do
11:                 for i ← 0 to |recs| − dist do
12:                     pair ← ⟨recs[i], recs[i + dist]⟩
13:                     if compare(pair) then
14:                         emit(pair)
15:                         lookAhead(pair)
```

Figure 2: Progressive Sorted Neighbourhood Algorithm[1]

give the most promising records among the profiles in a progressive manner.

PSN Algorithm - The PSN algorithm takes the 5 parameters as input. In the Fig 2, presents the implementation of the PSN[1], where D represents the data, which is not yet been loaded from the disk. Parameter K defines the attribute used for sorting. And the parameter W sets the window size for comparison along the sorted list. I defines the progressive iteration, and the last parameter N defines the number of records in the set of data.

In the real world scenario, the data volume is really high. This pose a challenge while processing, as the entire data set loading into the memory is not possible. This calls to to solution where we can load the data in the segments or partitions. The PSN algorithm takes care of this by by calculating the appropriate partitions using the function calcPartitionSize(D). This function calculates the size from the data types and match in the main memory for allocation of segments. The line 3 in algorithm focus on the providing the number of partitions pNum, taking into account of *W-1* overlap of profiles in the list. Line 4 depicts the order array, which helps in maintaining the order of records according to the key, hence only providing the ID's. And to keep the current partition order.

The PSN sorting based on the Magpie(Sorting method) explained in the section 3.2. PSN algorithm further iterates by using the window size, which increases the from two to the maximum window size. This helps in identifying the close neighbour first and the and further far neighbours later on. Line 8 and 9, represents segments, where segments are loaded to read the entire data set in the comparison process. For further processing PSN, first iterate over the rank distance *dist* within the current window, and in line 11, we set to compare the neighbouring distance. It is finally followed by a emit function with return true or false for the result found in the comparison. The comparison is executed using the compare function in line 13. If user doesn't terminate these algorithm early, it proceed till all partitions are finished iterating as per the algorithm described.

3.2 MagpieSort- A Progressiveness Technique

Magpie sort is naive sorting algorithm. The sorting is basically works similar to the selection sort.This algorithm repeatedly find the top-z smallest ones, followed by inserting the records in to a buffer having the length z. And if the buffer is full, the new record will displace the largest record from the list. And this process is iterated for next top-z records from the buffer. A important point to mark here is that, a record that has been emitted once will not be emitted again.

Window Interval - As PSN, loads all the records in each progressive iteration and loading partitions from the disk, is expensive. Due to this PSN presents the concept of windows enlargement interval (I) in line 7 and 10. It defines the very logic to load the loaded partitions and how many distant iteration PSN should execute. Hence I is the trade off parameter here for the PSN, that defines progressiveness in the iteration and the overall run time.

4 HIERARCHY OF RECORD PARTITIONS

Hierarchy of Record Partitions[2] is possible way for defining the progressive ER methods. This provides basically the hints for the ER. Hints - A likely matching of the records. These records are in the form of partitions with different level of hierarchies. And each hierarchy represents a possible solutions of ER. And the most resolved level leads to the ER results of our interest. The partitions of the most bottom level are the most fine clustering of the records presenting the results of ER. As we go up in the hierarchy we get the clusters which are coarser and big ones.

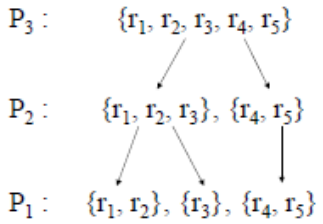

Figure 3: A partition Hierarchy for Resolving Records

Figure 3[2] represents a insight into set of records for HRP. We can see the most fine cluster is the $\{r1, r2\}$ $\{r3\}$ and $\{r4, r5\}$. We present these information in the most resolved cluster as $\{\{r1, r2\}, \{r3\}, \{r4, r5\}\}$. Also, if lets assume for being giving example that, $\{r1, r2\}$ and $\{r3\}$ are more likely to match, than the matching of $\{r4, r5\}$, we go for the coarse bottom level cluster, by merging $\{r1, r2\}$ and $\{r3\}$. From this we can formalize our concepts, where we present that partition hierarchy, which states that a immediate bottom cluster must be finer one from the previous cluster. Figure 3 represents a valid cluster since, P_1 the bottom most cluster is the most finest one. Along with this the HRP clustering also have parent and child relationship. P_2 having $\{r1, r2, r3\}$ is the parent for $\{r1, r2\}$ and $\{r3\}$ in cluster P_1. A remarkable feature for HRP is that, the storage

space is linear for the number of records. Such hints are however application estimates. The generation process is described in the technical report[7], and is beyond the scope of this paper.

4.1 Usage and Algorithm Description

Once we have found the partition hierarchy, we look for this can be helpful for any ER algorithm, and ER algorithm can exploit these information to provide the best matches. In Figure 4, we represent our algorithm[2] for HRP hints, where R represents the set of records, and E represents the an ER algorithm and H represents set of Hints, with a work limit of W (For example, work can be defined as 2 million comparisons at maximum)

1: **Input:** a set of records R, an ER algorithm E, a hint $H = \{P_1, \ldots, P_L\}$, and a work limit W
2: **Output:** an intermediate ER result F of $E(R)$
3: $F \leftarrow \emptyset$, $h \leftarrow \emptyset$
4: **for** $i = 1 \ldots L$ **do**
5: **for** $c \in P_i$ **do**
6: **for** $child \in c.ch$ **do**
7: $F \leftarrow F - \{clus | clus \in F \wedge clus \subseteq child\}$
8: $h(c) \leftarrow Resolve(E, c, h)$
9: $F \leftarrow F \cup h(c)$
10: **if** total work $\geq W$ **then**
11: **return** F
12: **return** F

Figure 4: A partition Hierarchy for Resolving Records

This algorithm is based on a single-link Hierarchical Clustering Algorithm(Termed as HC)[5]. HC helps in the merging the closest clusters by introducing a distance measure, that is, if two clusters having smallest distance, it is merged in to a single cluster. The distance is measured in terms of D, and is non-negative distance between two records, along with this a threshold T is defined for the max lengths between the record to be merged. When measuring the the distance, it takes the smallest distance for forming the cluster. Suppose, $R = \{r1, r2, r3\}$, where the pairwise distances are $D = \{r1, r2\} = 2$, $D = \{r2, r3\} = 4$, $D = \{r1, r3\} = 5$ and the threshold T given as $T = 2$, it merges the records that have a distance smaller or equal to T. This leads to $\{\{r1, r2\}, \{r3\}\}$. For Further resolving, we check all the combination distance between the profiles, as $\{r2, r3\}$, $\{r1, r3\}$, which exceeds the threshold T. And the final ER result is $\{\{r1, r2\}, \{r3\}\}$. Coming back to our algorithm in figure 4, with hints in the partition hierarchy. Suppose $R = \{r1, r2, r3\}$ and given $P_1 = \{\{r1, r2\}, \{r3\}\}$ and $P_2 = \{r1, r2, r3\}$ and W set to max 4 record comparison, we first solve the partition P_1 hints, which resolves r_1 and r_2 shown by Resolve function in line 8. Since r_1 and r_2 match now F becomes $\{\{r1, r2\}, \{r3\}$ at the same time storing the ER results in h. Similarly we start resolving the P_2 cluster, before resolving this we have to subtract the clusters from F, which are subsets of $P_2 = \{r1, r2, r3\}$. This leaves the $F =$ (Presented in line 7). Then we proceed for again the Resolve function in line 8, where r_1 and r_2 match and we take union of F with $\{\{r1, r2\}, \{r3\}$ and our work is just used once, so we satisfy the condition for line 10, and returning the the results for ER as $\{\{r1, r2\}, \{r3\}\}$.

Thus, this algorithm represents the main idea to build hierarchy blocks, in a way that matching likelihood of the profiles is proportional to the level in which they appear together for the first time. The blocks at the bottom contain the which highest likelihood matching, as shown in Figure 3. And hence can be further resolved with help of ER algorithms according to specific application before returning the final result.

5 SCHEMA-AGNOSTIC PROGRESSIVE ER

Modern world, solutions demand for faster match and resolution of the entities, with limited time and limited resources, for this we look beyond the structural data and known schema, as the existing methodologies presented in section 3 and 4 assume schema is already know, so that to avoid comparison between the entities that doesn't belong to same schema. But this doesn't solve the real world problem, where we have web corpus, textual data, semi structured data. This demands for the conceptualisation of schema agnostic Entity resolution methods. It helps in expanding the search space and hence better handling of complex real world problems. Progressive ER is getting more importance due to exponential increase of use cases, such as any retail catalog, which needs to be updated frequently, and hence in such cases detection of duplicates or matching profiles are matter of concern. In the below section, we are going to see two schema agnostic progressive ER techniques, one is known as Schema Agnostic- Progressive Sorted Neighbourhood(SA-PSN)[6], which is naive method, and we present one of advanced method for SA-PSN as Local Schema -Agnostic progressive Sorted Neighbourhood (LS-PSN). This two methods well establishes the idea for schema agnostic methods out of which, Schema Agnostic- Progressive Sorted Neighbourhood(SA-PSN) is being presented in the below section 5.1.

5.1 Schema Agnostic PSN (SA-PSN)

This algorithm uses most of the concept of PSN, including blocking key techniques, window size iteration over the neighbouring list. Neighbouring list are the core part where comparison takes place for ER profiles. The neighbouring list[1,2] is the main data structure for our PSN. It is basically a list of profiles which is generated after sorting all the profiles in a alphabetical manner. The main intuition behind such idea is that, once it is sorted according to the blocking keys(attribute value token), the matching profiles will lie in the proximity, and hence matching can be emitted with higher likelihood. The main idea of this approach is to combine the sliding window concept[2] and neighbouring list, where we define a window size and compare the profiles in that window size, and iterate over the list. This is presented with a help of an example with set of entity profiles.

The figure 5 and 6 represent data sets and base information for analysis of PSN and SA-PSN. The Fig 5(a), represents the set of profiles P, which contains structures, unstructured and semi structure data. And we have considered the $p1 \equiv p2 \equiv p3$ and $p4 \equiv p5$. We also assume that $p1 and p4$ describes all the profiles, even the textual unstructured data will require the preprocessing step. This follows that at this point, we will be defining a schema based blocking that concatenates the surname and the first two letters of the name. The profiles are indexed from the data lake, where in fig 5(a), profile

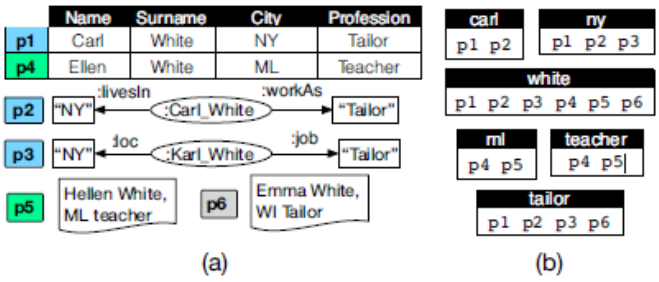

Figure 5: (a) Data from the data lake, i.e a set of profiles P in the different formats: (P1, P4) presenting the structured data,(p2, p3) presenting the semi-structured or the RDF data, (p5, p6) is the unstructured/free-text data. Note- $p1 \equiv p2 \equiv p3$ **and** $p4 \equiv p5$**. (b) Collection of blocks after applying token blocking.**

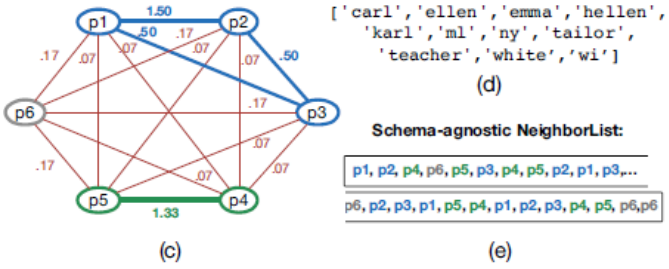

Figure 6: (c) The weighted blocking graph, for each profile comparison. (d) It shows the sorted list of taken values alphabetically ascending order. (e) The final Schema agnostic PSN list.

p1 and p4 represents structured data, p2 and p3 represents graph data, and p5 and p6 represent the unstructured data. The fig 5(b) represents the buckets, where profiles are sorted according to blocking key. The fig 5(d), represents the blocking keys presented in the alphabetical order, and the corresponding profiles of the keys are arranged in the list in fig 5(e) forming the the sorted neighbourhood list. And this sorted neighbourhood list is the key data structure for PSN algorithm. Using this the Figure 7 (a) forms the list for PSN which is the base for our SA-PSN. PSN starts its comparison by the initial window size of 1, and increases its window size for further iterations and comparisons. And if the matching profiles are found in the comparison window, they are emitted. The final comparison take place for the window size of 5 and in the 15th comparison, the last match is emitted. However, SA-PSN uses a single window size to iterate over the whole list for emitting matches.

The Figure 7(b), shows the SA-PSN, it uses the single list of the sorted keys, that holds all the token value entities i.e nothing but all the profiles in to single list as presented in fig 7(b). The entire process of PSN is repeated with slightest modification by merging into single list. In case entities have same key, they are placed randomly in the list. For this we iterate over the specific window size say w = 1, and emit the matches of profiles, and iteration stops

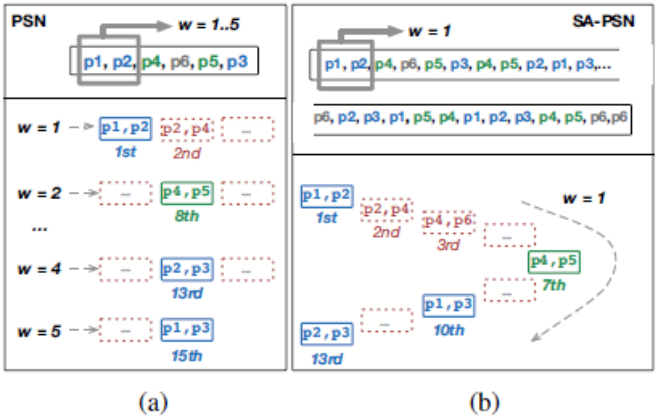

Figure 7: (a)Comparison emission of for PSN using a variable window size. (b)Comparison emission of SA-PSN using a fixed window size.

once we traverse the whole list. We can clearly see the matches produced the final comparison in the 13th comparison set. However, we see some limitation over here, such as, (1) repeated comparisons between, $p1 and p2$ as first and 9th comparison. (2)Since they are placed randomly, we can observe the coincidental proximity.In Figure 7(b), one can observe the profiles for token 'white', are placed randomly at the end of the list. These two problems does not it to work well on the large data sets[6], and presented as the limitations of this approach. This surely calls for a advanced approach.

5.2 Local Schema-Agnostic PSN (LS-PSN)

This method is an improvement to the existing method for SA-PSN described in the section 5.1, by introducing broad spectrum of techniques. This is distinguished by the introducing similarity measure by a weighted Neighbor List(NL). This weighting scheme, allows to assign a numerical value estimation that involves the step for comparison among profiles. This covers the functionality that is both schema and domain agnostic, making it suitable for the variety of web data. This also takes care of the disadvantages listed for the SA-PSN by introducing weighting scheme in the LS-PSN[6].
To measure the weighting *Relative Co-occurrence Frequency(RCF)* is introduced. This assigns a weight for each unique comparisons which takes place in the Neighbour List, thus trading higher computational cost for the comparison phase and the emission phase. RCF presents, how many times a pair profile lies at distance position, normalised with number of position corresponding to each profile. The below equation 1 represents the calculation part for the RCF weighting factor. And to efficiently implement RCF, there is a introduction of *Position Index(PI)*, in core it is an inverted index that associates every profile id with its position in the Neighbor List. And help in the generation of the RCF weighting function. Also, Fig 5(c) represents the weighting graph deduced according to the RCF function for a better understanding. However the values of weighting function is also shown in Figure 9(Step 1 iii) for the RCF function. These weighting function are shown further in details below with an example.

Algorithm 1: Initialization phase for LS-PSN.

Input: (i) Profile collection P, (ii) Weighting scheme, $wScheme$
Output: The overall best comparison

```
1  windowSize = 1;
2  ComparisonList ← ∅;
3  NL[] ← buildNeighborList(P);
4  PI[] ← buildPositionIndex(NL[]);
5  foreach p_i ∈ P do
6      distinctNeighbors ← ∅; // a set containing distinct neighbors
       frequency[] ← ∅;
7      foreach position ∈ PI[i] do
8          p_j ← NL[position+windowSize];
9          if isValidNeighbor(p_j) then
10             frequency[j]++;
11             distinctNeighbors.add(j);
12         p_k ← NL[position-windowSize];
13         if isValidNeighbor(p_k) then
14             frequency[k]++;
15             distinctNeighbors.add(k);
16     foreach j ∈ distinctNeighbors do
17         weight_{i,j} ← wScheme(frequency[j], j, i);
18         ComparisonList.add(getComparison(i, j, weight_{i,j});
19 sortInDescreasingWeight(ComparisonList);
20 return ComparisonList.removeFirst();
```

Figure 8: Algorithm for LS-PSN[6]

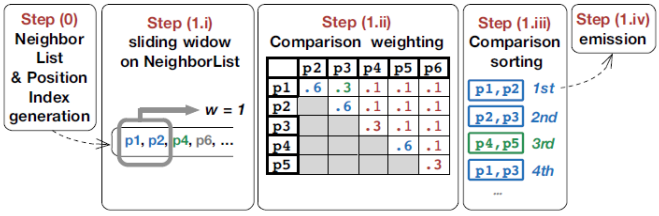

Figure 9: Application of Local Schema-Agnostic method

$$RCF = \frac{frequency[j]}{PI[i].length() + PI[j].length() - frequency[j]} \quad (1)$$

The Local Schema-Agnostic PSN, applies the RCF to selected window size, and thus one single local execution order is applied. It has two data structures, one is the *NL*, which is the Neighbor List which we have seen in the section 5.1, such that NL[i] denoted the profile id that is placed at the i^{th} position of the NL list. And the second one is the Position Index, returns the position associated with the profiles in the NL.

The algorithm for LS-PSN represented in the fig 8, where initially window size is set to 1(Line 1). In (Lines 2-4) it creates the data structure as required for PI and NL. For all the profiles (Line 5), it then iterates over the PI, presented in Line 8. The LS-PSN checks the neighbourhood in both the direction presented by line 8 and 12. Line 11 and 15 takes care of only distinct neighbour, where avoid the repeated comparison. Line 18, takes care of the weighting scheme selected, that is RCF for this algorithm. And finally all comparisons are sorted according to the highest weight to the lowest (Line 20), and the top ones are returned.

This algorithm is demonstrated using Fig 5(a) profiles, and the results is being presented the Fig 9. In the Fig 9, Step (0), using blocking key and alphabetical sorting we generate the NL and in the simultaneous step in the next process we generate the PI once we get the NL. This is followed by step(1.i) which forms the NL and slides a window size of 1 over it, which allows the comparison of profiles. Step(1.ii), provides the weighting for the profile comparison using RCF function for window size equal to 1, this weighting is the most significant step for this algorithm as it is able to eliminate the repeated comparisons from the list and weighting value indicates the probability likelihood of matching of profiles. This making it superior to other methods presented. In step(1.iii), all the comparison are weighted, and sorted from the highest to lowest weight. And then finally the sorted comparisons are emitted one by one in the step(1.iv). This algorithm clearly performs better by reducing the repetitive comparisons due to variable window size presented in the section 5.1, making it a better version of SA-PSN.

6 PROGRESSIVE ER METHOD CLASSIFICATION

As we saw, the approaches for various ER, where schema plays a major for the profiles sorting, in case of the clean data, where we can get a relation among the entities. Thus is more structured and relations are well established for the entity. In this way, a data engineer has really flexible way of choosing the blocking key for further generating profile buckets. Hence this blocking is a well known approach for our ER. These methods clearly exploit the known schema structures for scaling up the quality of ER. However, as the data sources are rising, progressive ER is becoming important[2], which can be seen, by the real time processing capability requirements of the data processing. These things leads to point where we define *Schema Based Progressive ER*. However, for the big data integration and when data sources are coming from multiple stream, calls for a broader approach to present the algorithm where it can be independent of the schema, along with integration of progressiveness. This is modelled as the Pay-as-you-go[2] application. Since web data is indistinguishably diverse[6], and noisy, it is impractical to approximate the ER using schema methods. This definitely has asked for schema agnostic progressive ER methods. Where schema- alignment is highly necessary, and hence algorithms can be modelled for such applications. In Table 1, we have classified the, ER methods as the *Schema Based Progressive ER* and *Schema Agnostic Progressive ER* which perfectly helps in desired outcome for our data available.

Classification	**List Comparison based**	**Cluster Based**
Schema Based Progressive ER	PSN	HRP
Schema-Agnostic Progressive ER	SA-PSN LS-PSN	Not Applicable

Table 1: Progressive ER Classification

We have further classified the Schema and Schema-Agnostic Progressive ER with the methodologies they follow for generating the matching entities via different approach for matching techniques described in the algorithms presented. We see entities being stored in list and compared for further matching as an overall idea in one hand, where basic intuition is the proximity of more probable match because of the blocking key used. And on the other hand, algorithms use hierarchy level to further resolve likelihood of matching. This paves the way for the classification of the methods into two categories, the first one being *List Comparison* and the second one being the *Cluster Based* method. The list comparison takes mainly the sorted list as the main data structure to get the matches. The list generated contains the highest probable matches and hence helps in scaling up the progressiveness or the real time capabilities. The cluster based methods use the level to denote the probability of likely matching, where each cluster profiles are compared among the same cluster to resolve into fine clusters, representing the final match.

(1) The *List Comparison*, starts with the help of list of profiles sorted according to the blocking key, mostly in the descending order of matching likelihood. This forms the base for the profile comparison using the algorithms and later emitting the results by a matching function. Hence the method that require schema as prior information and classified under List comparison is *Progressive Sorted Neighbourhood(PSN)*. This known schema helps in processing the token blocking and the profile indexing steps much easier, making it more efficient for the progressive emission. And for Schema-Agnostic Progressive ER, List comparison based method is *Schema-Agnostic PSN(SA-PSN)*. When the data is coming from the different sources, profile indexing and the pre processing of the data is a key steps for such tasks. To solve such challenges where we see data in different formats and from different sources, schema alignment is the major step in this regard. SA-PSN makes it easier by combining the token blocking in alphabetical order, thus making it possible to compare the profiles from all the sources combined, in a list. The LS-PSN initially uses neighbouring list for building the key data structure for further processing of RCF weighting function, making it suitable to classify LS-PSN into list based methods.

(2) The *Cluster Based* methods produce the cluster levels, that are sorted in the descending level to reach the finer clusters to generate the best matching pairs. In this the likelihood of matching is same for the profiles which are at the same cluster. The method that require schema as prior information and classified under Cluster Based method is *Hierarchy of Record Partition(HRP)*. There are also other methods, such as *Ordered List of Records(OLR)*[2], which further which takes HRP as base. However OLR is beyond the scope of this paper for classifying in to these categories. The HRP solves the problem by resolving the cluster. The profiles that lie in the same cluster have same probability of matching. The cluster are resolved my comparing among the clusters forming more finer cluster if matching is found.

In general HRP is hard to implement in the real world scenario, reason being the distance function. The distance can be naturally calculated only through certain attribute(e.g, product price). Moreover in addition to this we need L layers to be calculated in prior making it less efficient than PSN. And hence been not much in practical focus and are not being scaled to schema agnostic approaches. Hence we do not see HRP being scaled up to schema agnostic algorithms.

7 CONCLUSION AND FUTURE WORK

In this paper we have classified the methodologies for Progressive ER, for both schema based and schema agnostic methods. We have broadly presented this methods into List based comparison and cluster based comparison for the entity matching. Where Progressive Sorted Neighbor-hood(PSN), presenting the list comparison methods to find the matching entity based on schema based keys. And similar approach has been proposed with Schema-Agnostic PSN(SA-PSN) and LS-PSN, for data from multiple sources. We have also seen cluster based approach, which builds on schema based blocking keys.

An interesting direction would be, look how cluster based methods can be incorporated for the schema agnostic progressive ER[2], and come up with some algorithms which can scale up for big data volumes and real time capabilities.

8 REFERENCES

[1] T. Papenbrock, A. Heise, and F. Naumann, "Progressive duplicate detection,"IEEE TKDE, vol. 27, no. 5, pp. 1316–1329, 2015.

[2] S. E. Whang, D. Marmaros, and H. Garcia-Molina, "Pay-as-you-go entity resolution," IEEE TKDE, vol. 25, no. 5, pp. 1111–1124, 2013.

[3] G. Papadakis, G. Alexiou, G. Papastefanatos, and G. Koutrika, "Schemaagnostic vs schema-based configurations for blocking methods on homogeneous data," PVLDB, vol. 9, no. 4, pp. 312–323, 2015.

[4] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. Knowl. Data Eng., 24(9):1537–1555, 2012.

[5] C. D. Manning, P. Raghavan, and H. Sch¨utze, Introduction to Information Retrieval. New York, NY, USA: Cambridge University Press, 2008.

[6] Giovanni Simonini, George Papadakis, Themis Palpanas, Sonia Bergamaschi, Schema-agnostic Progressive Entity Resolution,2018 IEEE 34th International Conference on Data Engineering.

[7] S. E. Whang, D. Marmaros, and H. Garcia-Molina, "Pay-as-yougo entity resolution," Stanford University, Tech. Rep., available at http://ilpubs.stanford.edu:8090/979/.