Edit distance is used for measuring of Detecting the duplicates between XML entities which involves detecting similarity between entities [4].

M. Weis et.al has proposed Dogmatix framework. It consists of three main steps: candidate definition structure, duplicate definition and duplicate detection. Dogmatix method compares XML elements on the similarity of their parents, children and structure [6].

3. Proposed Work

In the previous work , there was Bayesian Network Construction which considers not only information within elements but also the way that how information is Structured . But this structure does not consider ordering of elements.

In this paper we are going to use two algorithms RECONA and ADAMA which considers ordering strategy. It finds the duplicate detection by using ascending order of r and using the relationship between objects , where comparison order is obtained by computing a rank r(v,v') for every candidate pair (v,v'). Whenever there is increase in similarity, RECONA recomputes similarity between two objects. ADAMA avoids re-comparison to increase efficiency. For comparison order ,if we choose ascending order of r ,we always choose to compare objects first that have fewer duplicates. In this case ,the ripple effect to neighbours is low if two objects are found duplicates. So RECONA avoids recomparison if we carefully choose comparison order which improves efficiency. Careful chosen order of ADAMA improves effectiveness.

RECONA Algorithm

The RECONA algorithm is the perfect algorithm for finding duplicate detection. The RECONA algorithm has two phases. The first is initialization phase and the other is comparison phase. Detecting duplicates to an object is based on the assumption that it may affect similarity and duplicate classification on other object.

The initialization phase (lines 2-10) contains all pairs of candidates which is defined in a priority queue *OPEN* which defines ascending order of rank r. DUPS is a set of duplicate pairs which contains duplicates pairs to avoid unnecessary re-comparison

```
1 Procedure ReconA()
2 G: data Graph;
3 OPEN: priority queue of candidate pairs
4 ordered in ascending order of r ;
5 DUPS: set of duplicate pairs;
6 CLOSED: set of possibly re-classified pairs;
7 θ : similarity threshold;
8 Initialize G;
9 Add all candidate pairs to OPEN;
10 while OPEN not empty do
11 begin
12 (v_i , v_j) ← OPEN.popFirst();
13 sim = sim(v_i , v_j);
14 if sim > θ then
15 begin
16 DUPS := DUPS ∪ {(v_i , v_j)};
17 updateOpenReconA(v_i , v_j);
18 end
19 end
```

Listing 1 RECONA Algorithm

```
1 procedure updateOpenReconA(Vertex v, Vertex v')
2 D(v, v') = {(n1, n2)|n1 ∈ D(v) ∧ n2 ∈ D(v') ∧ n1 ≠ n2};
3 for all (n1, n2) ∈ D(v, v') do
4 if (n1, n2) not ∈ DUPS then
5 begin
6 r_update := r(n1, n2);
7 if (n1, n2) ∈ OPEN then
8 OPEN.updateRank((n1,n 2), r_update );
9 else if (n1, n2) ∈ CLOSED then
10 OPEN.Push((n1, n2), r_update );
11 end
```

Listing 2: Updating OPEN in RECONA

ADAMA Algorithm

ADAMA works similar to RECONA ,but the important difference is that , once candidate pair is classified, we do not add it to open regardless of whether they are classified as duplicates or not . So pairwise comparisons are not performed more than once.

We define another set of candidates called NONDUPS in the initialization phase. It avoids re-comparison and compute Rank r in which set of neighbour pairs is indicated as neighbor pairs not in DUPS and not in OPEN. In ADAMA's algorithm , if similarity value of pairs is below threshold , it is added to NONDUPS and is never taken for re-comparisons which is defined in updateOpenAdamA. In updateOpenAdamA procedure we update the ranks of pairs that are present in OPEN. So the complexity of ADAMA algorithm is N because we do not allow re-comparison of pairs.

```
1 procedure AdamA()
2 G, OPEN, t, sim, DUPS as in ReconA;
3 NONDUPS: set of non-duplicate pairs;
4 Initialize G;
5 Add all candidate pairs to OPEN;
6 while OPEN not empty do
7 begin
8 (vi, vj ) ← OPEN.popFirst();
9 sim = sim (vi, vj );
10 if sim > θ then
11 updateOpenAdamA(vi,vj);
12 else
13 NONDUPS := NONDUPS ∪ {(v1, vj )};
14 end
```

Listing 3: ADAMA Algorithm

```
1 procedure updateOpenAdamA(Vertex v, Vertex v' )
2 D(v, v' ) = {(n1, n2)|n1 ∈ D(v) ∧ n2 ∈ D(v' ) ∧
 n1 ≠ n2};
3 forall (n1, n2) ∈ D(v, v' ) do
```