Algo Split ISWC 2022

May 2022

Algorithm 1: Retrieve the list of QID extrema in reverse order

Input : Bucket : B
QID Attributes : QID

$L \leftarrow retrieve\ the\ tuples\ lists\ from\ B$;
$N \leftarrow number\ of\ tuples\ in\ each\ list$;
$minMax \leftarrow stores\ min\ and\ max\ values\ for\ each\ QID$;
$listMinMax \leftarrow Empty\ list$;
for *i = 0; i* $\leq$ *N* **do**
    foreach *list in L* **do**
        $tuple \leftarrow list.get(N - 1 - i)$;
        foreach *qi in QID* **do**
            if *tuple.qi* $>$ *MinMax.qi.max* **then**
                Update MinMax.qi.max;
            if *tuple.QI* $<$ *minMax.qi.min* **then**
                Update minMax.qi.min;
    listMinMax.add(MinMax)

We start by going through the lists L in reverse order so as to create a list $reverse$ allowing us to list the extreme values of each QID: the i-th element of $reverse$ thus contains, for each QID, the maximum and minimum values present in the bucket if it were to contain the last i tuples of the lists L, *i.e.*, the elements $[n - i, ..., n - 1]$. $reverse$ is built incrementally by relying on the extreme QID values which have already been stored this far. If a newly-found tuples happens to have QID value lesser or greater than the current extreme then the extreme is updated.

Now that *reverse* has been created, we pass through the lists L normally in order to gather the extremes for each QID for the i-th first tuples of each list. While doing so, we compute the perimeter sum of the buckets that would be created if we were to choose i as the splitting index. At the end of this pass, we return the value of *index* for which the perimeter sum is the smallest. Pseudo-code for both passes can be found Algo. 1 and 2.

Once the classes have been formed, we use the ARX framework to k-anonymize them individually, *i.e.*, we set k to the size of the equivalence class in order to

Algorithm 2: Find optimal index split

```
Input  : Bucket : B
         QID Attributes : QID
L ← retrieve the tuples lists from B;
N ← number of tuples in each list;
minMax ← stores min and max values for each QID;
listReverse ← Algo1(B, QID);
minPerimeterSum ← ∞;
indexSplit ← 1
for i = 0; i ≤ N do
    foreach list in L do
        tuple ← list.get(N − 1 − i);
        foreach qi in QID do
            if tuple.qi > MinMax.qi.max then
                Update MinMax.qi.max;
            if tuple.qi < MinMax.qi.min then
                Update minMax.QI.min;
        reverse← listReverse.get(N - i - 1);
        ps← perimeterSum(minMax,reverse);
        if ps < PerimeterSum then
            PerimeterSum← ps;
            indexSplit ← i;
```

obtain the same generalization for all the tuples. The reason we rely on ARX to find an efficient generalization scheme is that we want to take advantage of the fast and optimized implementation offered by this tool as well as its support for user-defined attribute hierarchies.