Automatic Generator for Secure Yannakakis

XINYI ZHU, Hong Kong University of Science and Technology

1 BACKGROUND

In this section, the background of two fundamental technologies, join aggregate query[1] and Secure Yannakakis[2] are illustrated with some explanations of other knowledge.

1.1 Join Aggregate Query

Hypergraph and Joins. A natural join takes two relations which have some common attributes as input and generates all tuples with same values on the common attributes. It can be modeled as a hypergraph, where the vertices corresponds to attributes and hyperedge corresponds to a relation.

Join Tree. If the hypergraph of a join can be expressed in the form of a join tree, then it is said to be *acyclic*. Take Fig. 1 as an example, trees in (a) and (b) can express the same acyclic join .

Join Aggregate Query. Generally, a join aggregate query is a specific query that has select, join and aggregate operations.

Free-Connex Join Aggregate Query. According to the definition, a acyclic join has a join tree. A *free-connex* join can be easily explained that all outputs attributes should be placed above the non-outputs attributes in the join tree.

1.2 Secure Yannakakis

To ensure security of the *free-connex* join aggregate query, Secure Yannakakis makes improvement on each relational operators used in Generalized Yannakakis.

(1) *Oblivious Projection-Aggregation.* The first one is $\pi_F^{\oplus}(R)$ and the second one is $\pi_F^{1}(R)$. For $\pi_F^{\oplus}(R)$, Alice locally sorts R by F, so that tuples with the same value on F are consecutive. To make the algorithm oblivious, garbled circuit is used then. For $\pi_F^{1}(R)$, Alice also sorts R by F and permute the shares accordingly. Then Alice uses merge gates and the garbled circuit.

(2) *Oblivious Semijoin.* There are also two types of semijoins. The first type is actually an annotated join $R = R_F \bowtie^{\otimes} R_{F'}$. And the second type is an annotated

Author's address: Xinyi ZHU, Hong Kong University of Science and Technology .

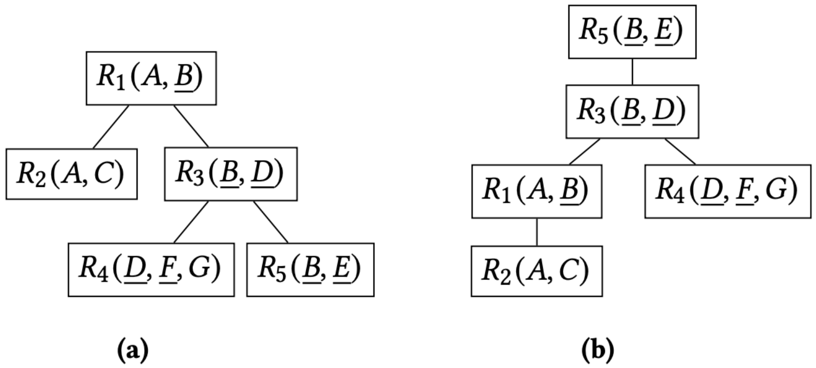

$R_1(A, \underline{B})$
$R_2(A, C)$
$R_3(\underline{B}, \underline{D})$
$R_4(\underline{D}, \underline{F}, G)$
$R_5(\underline{B}, \underline{E})$
(a)
$R_5(\underline{B}, \underline{E})$
$R_3(\underline{B}, \underline{D})$
$R_1(A, \underline{B})$
$R_4(\underline{D}, \underline{F}, G)$
$R_2(A, C)$
(b)

Fig. 1. An acyclic join with different join trees in paper Secure Yannakakis.

semijoin $R = R_F \ltimes^{\otimes} R_{F'}$. The complexity of quadratic time can be decreased by using PSI (private set intersection) to compute the indicators, using OEP (oblivious extended permutation) to get the indicators and using PSI with (shared) protocol to deal with the annotations. The second type is computing $R = R_F \ltimes^{\otimes} R_{F'}$. It first computes $\pi^1_{F \cap F'}(R'_F)$ by oblivious projection-aggregation and then computes $R = R_F \bowtie^{\otimes} \pi^1_{F \cap F'}(R'_F)$ by the first type oblivious semijoin.

(3) *Oblivious Join.* As Alice will always obtain the final results. And the outputs of oblivious semijoin only contain non-dangling tuples. Thus, Alice can compute the result using general join operation.

Finally, Secure Yannakakis algorithm can be assembled and conducted in the following three phases: (a) *Reduce.* It also works in a bottom-up manner over the join tree while computing $R_{F_P} \leftarrow R_{F_P} \bowtie^{\otimes} \pi^{\oplus}_{F'}(R_F)$, with constraint $F^{'} \subseteq F_p$. This phase can be further splitted into two steps: computing $\pi^{\oplus}_{F'}(R_F)$ (by oblivious projection-aggregation) then the oblivious semijoin; (b) *Semijoin.* Different from Generalized Yannakakis, Secure Yannakakis marks dangling tuples as dummy (setting annotations to 0 which can be computed by oblivious semijoin); (c) *Full Join.* As only output attributes remain and dangling tuples are all zero-annotated, Secure Yannakakis invokes oblivious full join and reveals the outputs directly to Alice.

2 ALGORITHMS AND IMPLEMENTATIONS

2.1 Framework

The framework of the automatic generator for Secure Yannakakis is shown in Fig. 2. Our automatic generator treats a join-aggregate query in SQL as the input. When obtained the parsed expressions, our system will determine whether this query can be constructed as a free-connex tree and how to construct it. After the construction, the core expressions will be passed to the linkage section which prepares the templates code for running Secure Yannakakis algorithm.

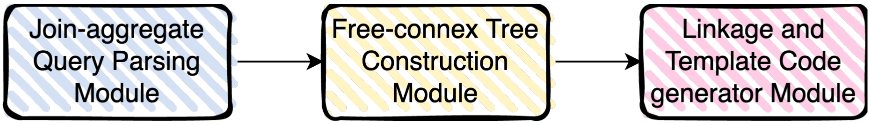

Join-aggregate Query Parsing Module
Free-connex Tree Construction Module
Linkage and Template Code generator Module

Fig. 2. The framework of Automatic Generator for Secure Yannakakis.

We will give a detailed discussion on three parts mentioned above and their principle and design in the following subsections.

2.2 Join-aggregate Query Parsing Module

Firstly, we utilize a developed SQL parser from the C++ SQL Parser GitHub Repository to parse the the input join-aggregate query into different components like "selected statement" and multiple expressions.

Secondly, we process the different components into three parts which will be retained for further free-connex tree construction and template code generating. They are respectively tables in the query, pairs of attributes for join operation and outputs. For tables in the query, we just select from the corresponding data structure from the SQL Parser. For pairs of attributes for join operation from where clause in the query, we extract two expressions from the binary operator "AND" and insert their table IDs into the *Link Graph Map* and attribute names to the *Related Attribute Map* which will be illustrated in the next paragraph. For outputs, there are two types, the output attributes and the output operation. To be specific, output attributes will simply be extracted and saved in memory. The output operation like SUM operation on different attributes from different tables, currently we do not support automatic annotation generating. So, we try to use the original annotations provided by Secure Yannakakis repository as surrogate.

Finally, we will introduce the concrete design for processing and storage.

Link Graph Map. This is a map data structure to record the linkage between different tables represented by their table IDs.

Related Attribute Map. This is a map data structure to record each table's related attributes which are the output attributes or primary key attributes for joining different tables in the query.

Take TPC-H Query 3 as an example.

```
select o_orderkey, o_totalprice, o_orderdate, o_shippriority,
    sum(l_extendedprice*(1-l_discount)) as revenue
from customer, orders, lineitem
where c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and c_mktsegment = 'automobile'
    and o_orderdate < date '1995-03-13'
    and l_shipdate > date '1995-03-13'
group by o_orderkey, o_totalprice, o_orderdate, o_shippriority;
```

The index for table customer, orders and lineitem is respectively 1, 2, 3. The *Link Graph Map* for TPC-H Query 3 is {{3: 2}, {1: 2}, {2: 1, 3}}. The *Related Attributed Map* is {{3: l_orderkey}, {1: o_orderkey, o_totalprice, o_orderdate, o_shippriority, o_custkey}, {2: c_custkey}}.

2.3 Free-connex Tree Construction Module

Overall, we iteratively construct a join tree. And use the algorithm to determining whether a join tree is free-connex. If it is not free-connex, reconstruct another join tree. Else, determine whether it matches our TPC-H query template free-connex tree.

2.3.1 Join Tree. We do the join tree construction in two steps. The first one is how to construct the tree. The second one is how to represent the tree.

Construction. We try to express a join into a hypergraph as a join tree.As in the join tree, each node is a table in the query. We use the classic tree building method. For each node, we store an integer as the index of this table and its children point vector and its parents node point.

Representation. We try to use a string to represent the structure of a join tree. Secure Yannakakis provides the solution of five TPC-H queries. Also, there is an extension query I have realized in the last semester. Try to automatic generate code as much as possible and at same time increase the further matching efficiency. We need to seek a trade-off between the efficiency and effectiveness.

Definition. The representation can be expressed in a string where each number represents a table and the layer in which the table is positioned is separated by the letter "C".

For example, for the join tree in the Fig. 1(a). It can be represented as "1C23C45" according to our definition.

This design is not general to apply for all join tree. It only fails when there are more than one child of a node has their children. However, we would like give a proof about why the representation can be designed as the definition in our scenario.

Lemma 1. *A join query whose longest join path chain is less than* 4 *will not result in the failure situation.*

Proof. A join path exists when two tables join on a primary key. When a new table join one of the table in the join path chain, it will increase the length of join path chain by 1. The shortest length of join path chain in the failure situation happens when the root node has 2 children nodes and both of the children nodes have a child node. The length of join path chain is 4. Because it is the smallest value, for any length less than 4, it will not lead to the failure situation. So, the lemma has been proved. □

The five TPC-H query in the Secure Yannakakis paper and one extension query use part of or all 6 tables. And the longest join path chain is 3, which is table customer join table orders join table lineitem join table partsupp/supplier/part. It will not result in failure situation. Thus, we can use our representation definition here.

2.3.2 Determination of Free-Connex. A free-connex join is that all outputs attributes should be placed above the non-outputs attributes in the join tree. Each node in the join tree as shown in the Fig. 1 includes not only the output attributes but also primary key attributes for joining different tables in the query. We have recorded each table's related attributes including output attributes and primary key attributes as the components for constructing the join tree in the *Related Attributes Map* mentioned above.

The determination algorithm on free-connex can be designed in two parts. Firstly, a data structure *relatedAttrHeight Map* needs to be maintained and updated to record the layer height of all relevant attributes for each table. Initially, set the attributes in each table as the height of the layer the table is positioned. For primary key attributes, they are pairs. And if either one of the pair is outputted, the other one is also should be marked as outputted attribute and stored the information in the *linkageOutputed Map*.

To determine whether the free-connex property is satisfied more efficiently, we assign the attributes which are non-output attributes in the node an infinite value from the root node of the join tree (incremental level height). We maintain two variables to record the maximum value in the previous layer and the minimum value in the current layer. Then, if the maximum value of previous layer is larger than the minimum value of the current layer, the join tree is not free-connex since there is an non-output attributes which is in the higher position than the output attributes. For example, *relatedAttrHeight Map* of a TPC-H Query 3 free-connex tree is {{3: 9999}, {2: 0,0,0,0,9999}, {1: 9999}}. We also provide the pseudo code of the determination algorithm in Alg. 1 and Alg. 2.

2.3.3 Template Tree Matching. After determination of a join tree is a free-connex one, we are supposed to match it with the template tree structure. Our system can now support all join-aggregate queries which have the same join-tree structure as TPC-H Query 3, TPC-H Query 10 and the extension query. These three can be represented as "1C23", "1C2C3" and "1C23C456" in our definition.

Algorithm 1 Maintain Attribute Layer Height Map

```
for attr in RelatedAttr[TableIdx] do
    if AttrIsIn(attr) then
        if attrTail in linkageOutputed then
            relatedAttrHeight[attr] = 9999;
        else
            Add relatedAttrHeight[attr] to linkageHeight;
        end if
    else
        relatedAttrHeight[attr] = 9999;
    end if
end for

function AttrIsIn(attr)
    if attr in output list then
        if attr in linkIsOut then
            return True;
        end if
    end if
    return False;
end function
```

As we support all different queries which have same tree structure as the template query, we only need to check there is the same number of nodes in the same layer. According to the tree representation definition, the layer in which the table is positioned is separated by the letter "C". Thus, for each interval separated by the letter "C", calculate the sum of the nodes in each interval which is the number of nodes in each layer.

2.4 Linkage and Template Code generator Module

After obtaining a free-connex join tree which matches a template query, we should pass all attributes and tables to the template code.

Firstly, in the linkage part, we should pass all attributes and tables that are required to run Secure Yannakakis algorithm, including output attributes, primary key attributes to join, all tables and their join order. Attributes are recorded before. Tables and their join order are stored in the free-connex join tree which is represented in a string format according to our definition. Also, to successfully run Secure Yannakakis for as more queries as possible, we should create many maps, including mapping an attribute name to its data type, mapping an table index to a table name and so on to support more situations. Secondly, the template code's

Algorithm 2 Determine Free Connex Satisfaction

```
Set lastLayerMax = 0, record the maximum value of the attribute layer height in the previous layer;
Set thisLayerMin = 9999, record the minimum value of the attribute layer height in this layer;
Set thisLayerMax = 0, record the maximum value of the attribute layer height in this layer, which is used to assign the lastLayerMax to the next round;
function IsFreeConnex(treeString)
    for item in treeString do
        if item != 'C' then
            for attrHeight in relatedAttrHeight[TableIdx] do
                if thisLayerMin > attrHeight then
                    thisLayerMin = attrHeight;
                end if
                if thisLayerMin < attrHeight then
                    thisLayerMax = attrHeight;
                end if
            end for
            if lastLayerMax > thisLayerMin then
                return False;
            end if
        else
            lastLayerMax = thisLayerMax;
            thisLayerMin = 9999;
            thisLayerMax = 0;
        end if
        lastItemHeight = height;
    end for
    return True;
end function
```

main structure is from Secure Yannakakis realization. However, the variables in the template code is passed by the linkage part.

3 EXPERIMENTS

3.1 Experimental Setup

We have implemented the Secure Yannakakis automatic generator on the cloud server. The server is equipped with two Intel vCPU, 8 GiB RAM, and a 30 GiB SSD(with max throughput 25Mbps). The software runs on Debian 10.0.

We choose to use C++11 to implement all project and use Cmake to automatically build our code repository. The implementation details can be found in AutoGenSECYAN GitHub Repository.

3.2 Dataset Description

Users can use all free-connex join-aggregate queries which have the same join-tree structure as TPC-H Query 3, TPC-H Query 10 and the extension query (shown below). A PDF file named "Data.pdf" containing the expressions of the three sample queries is also provided in our Github repository.

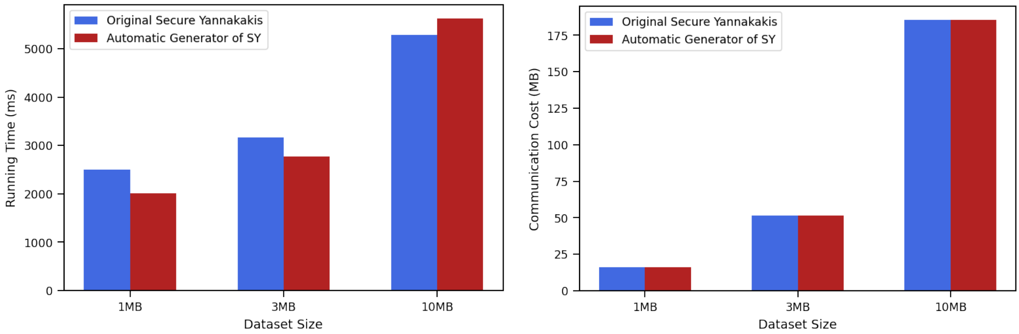
```
select c_name, c_custkey, o_orderkey, l_returnflag,
sum(ps_supplycost*l_quantity)
from PART,SUPPLIER,LINEITEM,PARTSUPP,ORDERS,CUSTOMER
where s_suppkey = l_suppkey and ps_suppkey = l_suppkey
    and c_custkey=o_custkey and ps_partkey = l_partkey
    and p_partkey = l_partkey and o_orderkey = l_orderkey
    and p_name like '%green%' and s_nationkey = 8
group by c_name, c_custkey, o_orderkey, l_returnflag;
```

There are five different size of datasets, which are 1MB, 3MB, 10MB, 33MB and 100MB. Each size of dataset contains six tables, which are customer table, lineitem table, orders table, part table, partsupp table and supplier table. To be specific, each table contains its attributes and annotations for different queries. Also, there are two types of annotations, which are Boolean and numerical respectively. Due to the limitation of time, we do not generate the annotations. So, the annotations will be the same as the one in the original query.

4 EVALUATION

In this section, we will illustrate our experimental results and give some discussions. We test the effectiveness and efficiency on our automatic generator by using original three queries mentioned in Sec. 3. Also, we re-test the original Secure Yannakakis for comparison. We test each of their performance using three different dataset sizes, 1MB, 3MB and 10MB. The effectiveness measures the accuracy of the outcomes processed by our automatic generator is exactly the same as the outcomes from Secure Yannakakis Repository. Ours outcomes can be referred in a PDF file named "results.pdf" and outcomes of Secure Yannakakis can be referred to a PDF file named "Results of Original Secure Yannakakis.pdf" in AutoGenSECYAN GitHub Repository. For efficiency, we would like compare running time and communication cost by our automatic generator with the original Yannakakis using the tables.

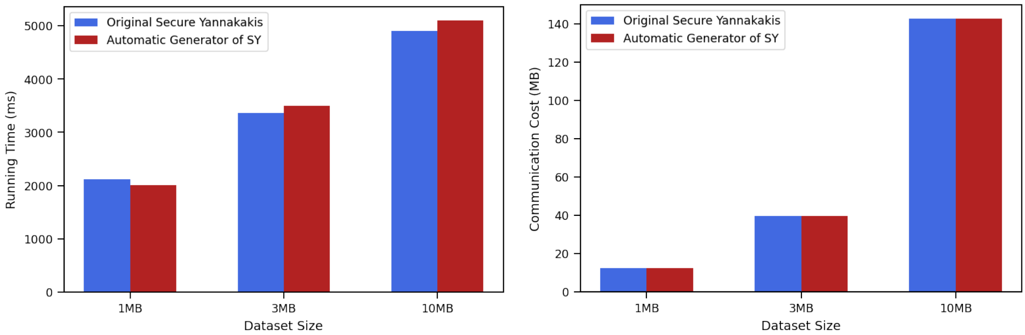
The communication cost between the original Secure Yannakakis and our automatic generator is the same. Also, running time varies with randomness but



(a) Running Time Comparison

(b) Communication Cost Comparison

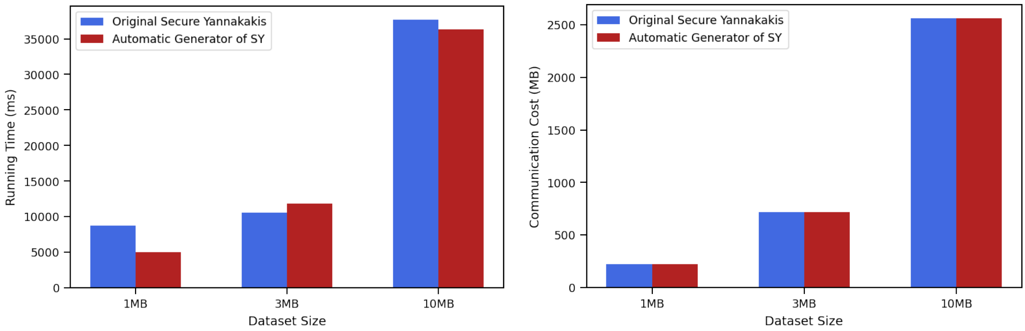
Fig. 3. Evaluation Result on TPC-H Query 3



(a) Running Time Comparison

(b) Communication Cost Comparison

Fig. 4. Evaluation Result on TPC-H Query 10



(a) Running Time Comparison

(b) Communication Cost Comparison

Fig. 5. Evaluation Result on Extension Query

overall are similar. Both the communication cost and running time are increasing when the testing dataset size increases.

5 FUTURE DIRECTIONS

Firstly, our automatic generator currently only supports free-connex join-aggregate queries which have the same join-tree as the TPC-H Query 3, TPC-H Query 10 and the extension query. Thus, in the future, it can be extended to support more queries with different join-tree structure. Secondly, the annotations can not be automatically generated now. We use the annotations in the original TPC-H Query provided by Secure Yannakakis repository. We can also support automatically generating SQL statements to obtain annotations in the future work.

REFERENCES

[1] M. R. Joglekar, R. Puttagunta, and C. Ré. Ajar: Aggregations and joins over annotated relations. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 91--106, 2016.

[2] Y. Wang and K. Yi. Secure yannakakis: Join-aggregate queries over private data. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1969--1981, 2021.