Chapter 3

Complexity of Stream Query Optimization

To ensure a globally optimal query plan which would result in minimal network usage, the query plan would need to be updated with every query arrival and termination. In a practical scenario it would be infeasible to recompute the globally optimal query plan if queries arrive and terminate frequently. However, for the purpose of understanding the challenges in evaluating the globally optimal query plan, we assume that the system is frozen on a query arrival or termination until the plan is computed. In other words, query plan computation is a system-wide atomic step. Although this is not practically feasible, we make this assumption purely for the sake of complexity calculations.

We consider two types of query plans based on the number of sources being used to answer each query. If the all queries have to be answered using a single
data source, we refer to the resultant query plan as a “non-composable” query plan. If queries can be answered using more than one data source then the resultant query plan is known as a “composable” query plan. The reason for differentiating between these two types of query plans is the associated complexity of computing each type of plan.

### 3.1 Non Composable Query Plans

Consider a grid node $x \in X$ processing a query of the form $q_x = \pi_{s_{i1},s_{i2}}(S_i)$. This query can be answered by any node $x' \in X$, that contains either 1) the same or 2) a superset of all the attributes of $S_i$ that is required by $q_x$. This is shown in Figure 3.1 where a query on node $A$ can be answered by either node $B$ or node $C$. We term such operations as “non-composable” projects and selects.
3.2 Composable Query Plans

Alternatively, $q_x$ can also be answered by sourcing streams from two or more grid
nodes even when none of them contain all the attributes of $S_i$ as required by $q_x$. However, as long as the union of all the attributes sourced from the streams covers the set of attributes required by $q_i$, the required projection can be “composed” from smaller projections.

This is shown in Figure 3.2, where node $D$ is the source of the data $S_i$ with attributes $s_{i1}$, $s_{i2}$ and $s_{i3}$. Node $B$ and $C$ have $\pi_{s_{i1}}(S_i)$ and $\pi_{s_{i2}}(S_i)$ respectively and a query $q$ arrives on node $A$ for $\pi_{s_{i1},s_{i2}}(S_i)$. The query on node $A$ is satisfied using subset data available both at node $B$ and $C$.

It should be noted here that such a composition is possible only if the project operation does not remove duplicates in its results – unlike the traditional project operation from relational algebra. Since stream data is continuously generated and data streams are considered to be infinitely long, we assume that a project query does not remove duplicate tuples in the result.

Duplicate removal in stream data itself has been addressed using various techniques like buffering (Garcia-Molina et al. [31]) where there are no false positives. Metwally et al. [55] and Deng and Rafiei [27] use filtering based on Bloom filters [12] along with various windowing techniques to remove duplicates. If such duplicate removal techniques are used, query result composition would not be possible. For many practical applications involving aggregations of data elements (like counting or computing averages over the query result) duplicate removal is
not advisable. These kinds of queries are amenable to be answered by composition in addition to non-composable operations.

An illustrative snapshot of sample data in a window along with data availability at various nodes is given in Table 3.1, where the query at $A$ can be satisfied by combining the data at $B$ and $C$.

The same is true of the selection operator. A given select operator $\sigma$ can be answered by two or more streams $\sigma_1 \ldots \sigma_k$ even when each have a smaller selectivity than $\sigma$ as long as the combined selectivity of $\sigma_1 \cup \cdots \cup \sigma_k$ covers the selectivity of $\sigma$. Composing query results based on selections is similar to computing a query over a larger table from two or more smaller materialized views (Park et al. [57]). In contrast, a query representing a join between two or more streams has to be always composed from the different streams.

It should be noted here that composable query plans are not possible for project operators in traditional relational algebra working on finite data sets. However, it is possible to compose query results for projection operations in streaming data, if duplicates are not removed from the data stream.
Figure 3.2: An Example of Composable Projection

Hence, given a query $q$ comprising a single operation (either select, project or join), a query plan can compute the result in three different ways:

1. Fetch the data from the relevant node hosting the data stream or the primary source.

2. Fetch the data from a secondary source which can satisfy the query. A secondary source is a node which shares data fetched from another node. A secondary source satisfies $q$ if the streams it hosts covers the selectivity or attribute requirements for $q$.

3. Compose the query result using two or more sources.

Composability is shown to result in query plans having lesser network usage than non-composable operations, however they also add an extra layer of complexity. By allowing compositions, determining the optimal query plan not only
involves identifying single sources which can satisfy the query, but also consider all possibilities where a combination of sources could satisfy the query.

We now consider each of the operations (projects, selects and joins) separately for complexity calculation.

3.3 Complexity of Projection

The complexity of projection queries for composable and non-composable query plans are as follows.

3.3.1 Composable Projects

As noted earlier, the absence of duplicate removal in streaming data allows query result composition. To find the optimal query plan for a given query set, we re-write all project queries requesting for multiple attributes of a stream to single attribute queries (SAQ) of the same stream. A SAQ for a query $q$ having multiple projection attributes, is defined as a set of single attribute projection queries, the results of which can be composed to generate the result of $q$. For instance, in Figure 3.2, the SAQ for query $q_A$ at node $A$ is given as, $SAQ(q_A) = \{\pi_{s1}(S_i), \pi_{s12}(S_i)\}$.

In a graph theoretic sense, all nodes requiring a given stream data attribute need to collectively form a directed acyclic graph (DAG) with at least one node connected to the original data source. For instance the minimum spanning tree (MST) overlays for the example in Figure 3.2 are given in Figure 3.3 where, a
MST is the tree incurring minimal network usage that spans across all nodes in the DAG.

To find the DAG with the minimal network usage, we use the following rationale:

1. Ignore the direction of DAG edges and consider all stream connections between all pairs of nodes in the grid

2. Compute a MST for the grid based on the stream connections.

3. The overlay with the minimum network usage between a given source and destination is the path between them that lies on the MST (Theorem 1).

The overall algorithm as shown in Algorithm 1 is explained as follows. Each query \( q \in Q \) incident on the grid is decomposed into individual SAQs required to satisfy \( q \) using the \( SAQ(q) \) operator. These individual SAQs are then added to a set \( S \) which contains all the unique, individual SAQs required to satisfy all the queries \( Q \) incident on the grid. For each \( SAQ_i \in S \), the set of grid nodes \( N_i \in \mathcal{G} \) which require \( SAQ_i \) to satisfy some query incident on it are identified. \( N \) is the set of all \( N_i \)s corresponding to each \( SAQ_i \). All nodes in \( N_i \) and the source of \( SAQ_i \) are connected together to create an overlay of edges \( M_i \) using a minimum spanning tree algorithm \( MST(N_i, \mathcal{G}) \). The network usage for the overlay \( M_i \) is given by \( U(M_i) \) and the network usage for the set of all overlays \( M \) results in the minimum network usage \( U_{\text{min}} \).
Algorithm 1 Minimum Spanning Tree Overlay Algorithm

Require: Grid $G$ and project query set $Q$ incident on $G$
Ensure: Overlay of MST’s $M$ with minimum network usage $U_{\text{min}}$

1: $S \leftarrow \{\}, N \leftarrow \{\}, M \leftarrow \{\}, U_{\text{min}} \leftarrow 0$
2: for all $q \in Q$ do
3:     $S = S \cup SAQ(q)$
4: end for
5: for all $SAQ_i \in S$ do
6:     $N_i = \{x : x \in X \land \exists q_x : SAQ(q_x) \supseteq SAQ_i\}$
7:     $N = N \cup N_i$
8: end for
9: for all $N_i \in N$ do
10:    $M_i = \text{MST}(N_i, G)$
11:    $M = M \cup M_i$
12: end for
13: for all $M_i \in M$ do
14:     $U_{\text{min}} = U_{\text{min}} + U(M_i)$
15: end for

Figure 3.3: Minimum Spanning Tree Overlays
**Theorem 1**  An overlay of minimum spanning trees $M$ as computed by Algorithm 1 gives the minimum network usage query plan $U_{\text{min}}$ for a set of stream projection queries $Q$ if compositions are allowed.

**Proof**  We prove the above theorem by refutation. Consider one of the MST overlays $M_i$, requiring minimal access paths over the set of nodes $N_i$. Suppose there exists another topology $M'_i$ to connect nodes in $N_i$ with a network usage $U(M'_i)$ such that $U(M'_i) < U(M_i)$. This would mean that if we replace the overlay path in $M_i$ with $M'_i$ we would get a spanning tree of smaller weight. This is a contradiction since $M_i$ is the minimum spanning tree.

Thus if compositions are allowed, then the optimal query plan complexity is polynomial time with the optimal query plan being an overlay of minimum spanning trees, as the complexity of computing each minimum spanning tree is polynomial (Prim [61]).

### 3.3.2 Non Composable Projects

If compositions are not allowed, the only way to satisfy a query request is to get it from either the primary source or a secondary source which has a superset of the required data.

To determine the set of possible sources to answer a query, we introduce a term called “satisfiability” of a query, which is checked using the SAQ concept introduced earlier. A query $q_x$ on node $x$ can be satisfied or answered by either
Increasing Satisfiability

Level 0

Level 1

Level 2

Increasing Levels

Figure 3.4: Satisfiability Poset for Figure 3.2

1) the source of the data, or 2) another node \( y \) which answers a query \( q_y \) where \( SAQ(q_x) \subseteq SAQ(q_y) \).

The set of all sources and queries can be represented as a poset of satisfiability hierarchy based on the streams that they possess are require. The poset indicating satisfiability for the example in Figure 3.2 is shown in Figure 3.4. Node \( D \) being the source can satisfy any query and is therefore at the top of the satisfiability poset and at level 0. The query at node \( A \) requiring both \( \pi_{s1i}(S_i) \) and \( \pi_{s2i}(S_i) \) is next and can be satisfied only by source \( D \). Queries at nodes \( B \) and \( C \) can be satisfied by both node \( A \) and source node \( D \) and are hence at the highest level of the poset.

Each poset element is represented as \( e_i \), where \( i \) uniquely identifies the poset element. A function \( L(e_i) \) is defined to denote the level of poset element \( i \) and \( SAQ(e_i) \subseteq SAQ(e_j) \) indicates the satisfiability of poset element \( e_i \) by another poset element \( e_j \).

The hierarchy of the poset ensures if \( SAQ(e_i) \subseteq SAQ(e_j) \), \( L(e_j) \leq L(e_i) \).
This hierarchy ensures that a query represented by a poset can be satisfied by another poset either at the same level or at a lower level as shown in Figure 3.4. A source node \( s \in S \) can answer a query for any attribute related to the source and hence poset elements corresponding to sources are placed at the top most level of the poset with level zero. All poset elements representing queries are hierarchically organized below the source elements. Algorithm 2 explains the poset hierarchy formation process.
Algorithm 2 Poset Hierarchy Formation Algorithm

Require: Grid $G$ and project query set $Q$ incident on $G$
Ensure: Hierarchically ordered poset $P$

1: for all $e_i \in P$ do
2: if $e_i \in S$ then
3: $L(e_i) = 0$
4: else
5: $L(e_i) = 1$
6: end if
7: end for
8: repeat
9: $finish \leftarrow true$
10: for all $e_i \in P$ do
11: for all $e_j \in P$ do
12: if $SAQ(e_i) \subset SAQ(e_j)$ then
13: $L(e_i) = \max[L(e_i), L(e_j) + 1]]$
14: $finish \leftarrow false$
15: end if
16: end for
17: end for
18: until $finish$

Lemma 2 At the end of the poset hierarchy formation process, the $i^{th}$ poset element $e_i$ at level $k = L(e_i)$ can only be satisfied by,

1. poset element $e_j$ at level $k$ if and only if $SAQ(e_i) = SAQ(e_j)$

2. poset element $e_j$ at level $l = L(e_j)$, where $l < k$, if and only if $SAQ(e_i) \subset SAQ(e_j)$

Proof We prove this by refutation. Assuming there exists a poset element $e_j$ at level $l > k$ which can satisfy $e_i$, then either (a) $SAQ(e_i) \subset SAQ(e_j)$, or (b) $SAQ(e_i) = SAQ(e_j)$. **42**
• Refutation for (a): If there exists some \( e_j \) such that \( \text{SAQ}(e_i) \subset \text{SAQ}(e_j) \), then from line 13 of Algorithm 2, \( k \geq l + 1 \). Hence if \( k < l \), then such an \( e_j \) cannot exist.

• Refutation for (b): If \( e_j \) is at level \( l \) there must be some \( e_q \) at level \( l - 1 \) such that \( \text{SAQ}(e_j) \subset \text{SAQ}(e_q) \). If \( \text{SAQ}(e_j) = \text{SAQ}(e_i) \), then \( k = l \) as \( \text{SAQ}(e_i) \subset \text{SAQ}(e_q) \). Hence if \( k < l \), then such an \( e_j \) cannot exist.

Once the poset \( P \) is ordered according to satisfiability, we now create a “minimum network usage graph” \( \text{MinGraph} \). To create the \( \text{MinGraph} \), each poset element is considered as a node in the graph and the set of edges determined ensuring minimum network usage. From Lemma 2 all poset elements requiring the same data are at the same level \( l \) and are grouped into a set \( X \) and referred to as the destination nodes. The poset elements or nodes which can satisfy the poset elements in set \( X \) are in lower levels and are grouped together into set \( Y \) or the source nodes. To determine the edges resulting in the minimum network usage, Prims minimum spanning tree algorithm [61] is used where \( Y \) is considered to be the set of nodes which are already in the tree and \( X \) is the set of nodes still requiring to be connected.

Algorithm 3 determines the set of edges resulting in minimum network usage.

**Theorem 3** The set \( \text{Edges} \) determined using Algorithm 3 results in minimum network usage.

**Proof** We use proof by refutation to prove the above algorithm. The incorrectness can arise from,
Algorithm 3 Minimum Cost Network Usage Graph

Require: Grid $G$ and ordered poset $P$
Ensure: $Edges$ of $MinGraph$

1: $Edges$ ← {} \\
2: for all $e_i \in P$ do \\
3:     $X$ ← {} \{ $X$ is the set of destination nodes \} \\
4:     $Y$ ← {} \{ $Y$ is the set of source nodes \} \\
5:     for all $e_j \in P$ do \\
6:         if $(L(e_j) = L(e_i) \& SAQ(e_i) = SAQ(e_j))$ then \\
7:             $X = X \cup e_j$ \\
8:         end if \\
9:         if $(L(e_j) < L(e_i) \& SAQ(e_i) \subset SAQ(e_j))$ then \\
10:            $Y = Y \cup e_j$ \\
11:       end if \\
12:    end for \\
13:  $Edges = Edges \cup PrimsMST(X, Y, G)$ \\
14: end for \\

- Incorrect selection of source set: Incorrect source selection can occur if any possible source is being not considered while considering the best source to select. Given line 9 of Algorithm 3, if there is such a source present, it must be represented by a poset element with a level greater than the concerned poset element. However this is not possible because of Lemma 2.

- Incorrect selection of destination set: Incorrect destination selection can occur if any destination requiring the same data is being not considered. Line 6 of Algorithm 3 ensures that all equal sources are considered.

- Incorrect selection of edge: If there is an incorrect edge selected, then there exists another edge with lesser weight than the selected edge. This is not possible because of the use of Prim’s algorithm which selects the minimum
3.4 Complexity of Selections

Like project queries, we consider the complexity of composable and non-composable selection query plans.

The main issue in selection queries involving compositions is identifying the set of data sources which would lead to the minimal network usage. The quintessential notion which determines if a source can serve a query is the selection granularity available at the source and the selection granularity required by the query. For instance a query \( q_1 = \sigma_{(b_1=1\& b_2=5)}S_i \) can be answered by composing the result from two secondary data sources having data \( \sigma_{(b_1<3)}S_i \) and \( \sigma_{(b_2>4)}S_i \).

Using selection granularities to determine reuse of data is a well studied in the area of materialized view selection techniques in data-warehouses (Park et al. [57]).

In [57], for a given query \( Q \), there are a set of candidate materialized views \( V(Q) \) to satisfy \( Q \) and a cost function \( cost(MV_i, QR_i) \) which provides the cost for a materialized view \( MV_i \in V(Q) \) with a query region of \( QR_i \). The optimal MV set problem is to find an optimal set \( S \) of pairs \( (MV_i, QR_i) \) which can answer query \( Q \), minimizing the cost of \( S \) or,

\[
\text{arg min}_S \sum_{(MV_i,QR_i) \in S} cost(MV_i, QR_i)
\]  

(3.1)
[57] shows that the minimum set cover decision problem, which is NP-complete can be transformed in polynomial time to this problem thereby rendering this problem as NP-hard.

We map the optimal MV set problem to the problem of identifying the correct set of sources to satisfy a query $q$ incident on the grid. The candidate sources and secondary sources $S(q)$ for answering the query can be considered to be the set of candidate MV’s $V(Q)$. The cost for a materialized view can be considered to be the network usage $U(s_i, r_i)$ for fetching data from the node $s_i \in S(q)$ with selection granularity $r_i$. The optimal network usage problem is to find the optimal set $S$ of pairs $(s_i, r_i)$ to minimize the network usage of $S$ or,

$$\arg \min_S \sum_{(s_i, r_i) \in S} U(s_i, r_i)$$

$$\tag{3.2}$$

Hence the optimal network usage problem is NP-hard as well.

If compositions are not allowed, the problem becomes very similar to the projection without compositions problem. Since queries can be answered from only sources with higher selection granularities, a single query stream is sufficient to answer the query. In such a scenario we need to create a hierarchical poset for a given query set using selection granularities to set the levels. The rest of the algorithm will be the same as in projection queries.
3.5 Join Complexity

In a distributed environment with data being available at different sites, a join query with $n$ relations is formulated as a graph problem (Wang and Chen [89]). A directed graph with $n + 1$ nodes is constructed where one node corresponds to the final destination site $D$ and the remaining $n$ nodes have a one-to-one association with a relation. An edge $(R_i, R_j)$ indicates relation $R_i$ being sent to node with relation $R_j$ to perform a join. An edge $(R_i, D)$ indicates relation $R_i$ being sent to the destination site directly. The objective is to find an inversely directed spanning tree toward $D$ with the minimal transmission cost. Finding the optimal join sequence to minimize the transmission cost is NP hard [89]. By replacing the transmission cost with the network usage associated with shipping relations between nodes, our problem also becomes NP-hard.

Trivially, joins cannot be performed without compositions as no single source will have the joined data. Table 3.2 summarizes the complexities of various kinds of operations.

3.6 Need for Emergent Optimization

Although, polynomial time algorithms exists for computing the globally optimal non-composable query plan for selects and projects, it should be noted that the P-class complexity does not make the open-world problem feasible, since the limiting assumption of freezing the grid for determining the optimal plans has been made. Better query plans involving lesser network usage are possible when selects
Table 3.2: Summary of Query Processing Complexities for Network Usage Optimization

and projects are composed from streams with lower resolution. This however, makes the optimization problem intractable except for project-only queries. Even though the complexity of globally optimal query plans is polynomial with project queries, the continuous arrival and termination of queries makes the computation of such a plan infeasible.

To optimize in an environment with queries arriving and terminating frequently, we propose the notion of “emergent optimization” where, each node in the grid autonomously takes decisions based on some self interest objective and the global query plan emerges as a consequence of the local decisions. The local decisions would involve each node selecting appropriate data sources to answer queries inci-
dent on itself. While emergent query optimization not only allows us to optimize LRC queries in the stream grid, it is shown experimentally, that the emergent query plans, though not optimal, result in network losses not significantly greater than the optimal plans.