Minimizing Communication Cost in Distributed Multi-query Processing

Jian Li, Amol Deshpande, Samir Khuller

Department of Computer Science, University of Maryland, College Park, MD 20742, USA
{lijian, amol, samir}@cs.umd.edu

Abstract— **Increasing prevalence of large-scale distributed monitoring and computing environments such as sensor networks, scientific federations, Grids etc., has led to a renewed interest in the area of distributed query processing and optimization. In this paper we address a general, distributed multi-query processing problem motivated by the need to minimize the communication cost in these environments. Specifically we address the problem of optimally sharing data movement across the communication edges in a distributed communication network given a set of overlapping queries and query plans for them (specifying the operations to be executed). Most of the problem variations of our general problem can be shown to be NP-Hard by a reduction from the Steiner tree problem. However, we show that the problem can be solved optimally if the communication network is a tree, and present a novel algorithm for finding an optimal data movement plan. For general communication networks, we present efficient approximation algorithms for several variations of the problem. Finally, we present an experimental study over synthetic datasets showing both the need for exploiting the sharing of data movement and the effectiveness of our algorithms at finding such plans.**

I. Introduction

Recent years have seen a re-emergence of large-scale distributed query processing in a variety of applications. This has in part been fueled by an increasing number of scientific federations such as SkyServer [1], [2], GridDB [3] etc., where users may issue queries involving a large number of distributed data sources. Many of these datasets tend to be huge, and as the scale of these federations and the number of users issuing queries against them increase, network bandwidth is expected to become the key bottleneck [4]. Similarly in publish-subscribe systems and other distributed stream processing applications, a large number of queries must be executed in a distributed manner across the network [5], [6]. To enable high throughput and low latencies in presence of high-rate data streams, the query processing operators must be placed judiciously across the network to minimize the data movement cost. The emergence of large-scale monitoring infrastructures such as wireless sensor networks poses similar distributed query processing challenges; the queries must be processed inside the network in a distributed fashion so that the lifetime of the typically resource-constrained sensing devices is maximized [7], [8].

Although these applications may appear very different from each other, the query optimization challenges they pose are quite similar to each other. In this paper, we formulate and address a general multi-query optimization problem where the goal is to *minimize the total communication cost* while executing a large number of queries simultaneously in a distributed environment. We note however that the algorithms we develop are centralized. Our main focus is to optimally share the movement of data across multiple queries. We assume that the *query plans* are provided as part of the input, which specify the operations that need to be performed on the data items[2]. We allow the query plans to consist of general n-ary operators and place no restrictions on the types of operators that can be used; however we assume that any node in the communication network is capable of executing any operator. In application domains such as sensor networks, distributed streams and publish-subscribe domains, these operators will typically be aggregate operators, in which case we allow partial aggregation of the results; whereas in distributed databases, the operators will typically be relational operators like joins. We note that in this paper we do not address *join order optimization*, instead adopting a two-phase approach where the join order decisions are made independently of the scheduling and operator placement decisions [9], [10], [11], [12].

Prior analytical results on this problem have been limited to the single-query optimization case (in the distributed query optimization literature [13]) or to specific types of queries and/or specific forms of communication networks [7], [8]. In this paper, we develop a framework to address optimization of general query plans under a flexible communication model. We develop a novel algorithm that finds the optimal solution in polynomial time if the communication network is a tree, extending previous results by Silberstein and Yang [8]. The optimization problem can be shown to be NP-Hard when the communication graph is not a tree by a simple reduction from the Steiner tree problem. In that case, we present a polynomial time algorithm for the problem with an $O(\log n)$ approximation guarantee (this is a worst case bound on the quality of the solution, compared to an optimal solution). We also develop several constant-factor approximation algorithms for special cases of the problem. Finally we present a performance evaluation over synthetic datasets to illustrate the need for sharing data movement when a large number of queries need to be executed simultaneously in a distributed environment.

[1]This research was supported by NSF grants CCF-0430650, CCF-0728839 and IIS-0546136.

[2]We use the terms *relation*, *data source* and *data item* interchangeably in this paper.

A. Outline

We begin with a brief discussion of related work in Section II. We formulate the multi-query optimization problem and summarize our main algorithmic results in Section III. We then present a polynomial-time algorithm to find the optimal plan that minimizes the total communication cost when the communication graph is restricted to be a tree (Sections IV and V). We then consider arbitrary communication graphs, and present several approximation algorithms for the problem (Section VI). We conclude with a preliminary performance study that illustrates the need to share data movement in distributed multi-query processing (Section VII).

II. Related Work

There has been much work on distributed query processing and optimization (see the survey by Kossmann [13]). Most of this work has focused on minimizing the total communication cost for executing a single query by judiciously choosing the join order and possibly adding semi-join operators to the query plan [14], [15], [16], [17], [18], [19]. In contrast to this prior work, we consider *multi*-query optimization which is an inherently harder problem, with few results known even for the centralized case [20], [21]. On the other hand, we don't consider join order optimization in this paper, and assume that a two-phase approach to query optimization is being followed; the two-phase approach, proposed by Hong et al. [9] for parallel query optimization, separates join order optimization from scheduling issues, and is commonly used to mitigate the complexity of query optimization in distributed and parallel settings [10], [22], [11], [12].

Hasan et al. [23] and Chekuri et al. [24] present algorithms for minimizing communication cost in parallel query optimization, again assuming that the query plans are provided as part of the input. Although these algorithms bear superficial similarities to the algorithms we present, the underlying problem they address is fundamentally different from our problem; they assume a uniform communication cost model (the cost of communicating data between any pair of nodes is identical), whereas in distributed systems the underlying communication cost model is non-uniform; this is in fact the chief reason behind the complexity of the problem. Also, this prior work only considered single-query optimization, whereas we focus on multi-query optimization.

Trigoni et al. [7] study the problem of simultaneously optimizing multiple aggregate queries in a sensor network. They use linear algebra techniques to share computation of aggregates, but assume that the communication is along a pre-determined tree. Silberstein et al. [8] study a similar problem under the same restriction (called *many-to-many aggregate queries* problem), and propose a solution based on solving a bipartite vertex cover problem for each edge. Similar to our work, Silberstein et al. don't consider sharing computation between queries either (in other words, only the movement of the original data items is shared between queries). *Both those works assume identical-sized data items and assume that the aggregate size is constant as well. In contrast, we allow arbitrary-sized data items and put no restrictions on the intermediate result sizes either.*

In content delivery networks and publish-subscribe systems, the goal is to transmit the information from a set of sources to a set of sinks as efficiently as possible (see e.g. [25], [26], [27]). Although some of this work has considered the issues in allowing users to subscribe to aggregate functions over the data sources, we are not aware of any work that has considered simultaneous optimization of multiple queries in such a framework. Our results can be directly applied to similar problems in the publish-subscribe domain.

The problems we study in this paper are closely related to several problems that have been extensively studied in the theory literature, specifically, the *Steiner tree* problem [28], [29] and its generalizations like the *Single Sink Rent-or-Buy (SROB)* problem [30] (also called the Connected Facility Location [31]). Both of these problems are NP-hard, and constant-factor approximation algorithms are known for them.

III. Problem Overview and Summary of Results

We begin with a formal definition of the problem, and present an illustrative example that we use as the running example. We use the terminology from multi-query join processing to describe the problem and the algorithms (in particular, we assume that the operations being performed are *joins*); however the results extend naturally to the case when other operations (e.g., aggregates) are being performed instead. We then briefly summarize our main algorithmic results.

A. Formal Problem Definition

Let $\mathcal{X} = \{S_i | i = 1, \ldots, n\}$ denote a set of relations (data sources) stored in a distributed fashion. Without loss of generality, we assume that each relation is stored at a different node (see below). We use an edge-weighted graph $\mathcal{G}_C$ over the nodes to represent the communication network; the weight of an edge indicates the communication cost incurred while sending a unit amount of data from one node to another. Whenever a data item of size $|S|$ is shipped across an edge e of weight $w(e)$, the cost incurred is $|S|w(e)$. In a wireless sensor network, this may be the energy expended during transmission of the data, whereas in a distributed setting this may capture the network utilization [4].

We are also given a set of queries, $Q_1, ..., Q_m$, with the query Q_i requiring access to a subset of relations denoted by $Q_i^R \subseteq \{S_1, \ldots, S_n\}$. For each query, a *query plan* is provided, in the form of a rooted tree, which specifies the operations to be performed on the relations and the order in which to perform them. Finally for each query a destination (called *sink*) is provided where the final result must be shipped.

Given this input, our goal is to find a data movement plan that minimizes the total communication cost incurred while executing the queries.

We note that this metric does not capture the (CPU) cost of operator execution at the nodes. In many cases (e.g., in sensor networks or publish-subscribe domains), these operations (typically aggregates) are not very expensive, whereas

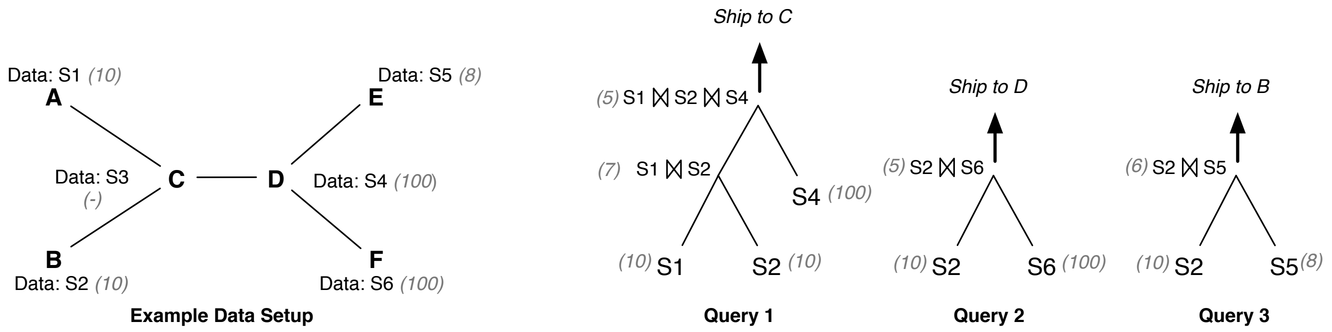


Fig. 1. Running example with 6 data sources, $S_1, \ldots, S_6$ over a tree communication network, and three queries (along with the query plans and the result destinations); for clarity, we assume unit-weight communication edges ($w(e) = 1, \forall e$) and omit the weights from the figures. The italicized numbers in the parentheses indicate the sizes of the data sources and the intermediate results.

in other cases (in scientific federations or other distributed databases) the operator execution cost can be very high. We plan to address load balancing in future work, and we focus on communication cost minimization in this paper.

The above framework is fairly general. In particular we allow the query tree to contain *n-ary* operators and make no assumptions about the operators themselves (except that we know the result sizes). However the *many-to-many aggregate queries* problem [8] cannot be *directly* mapped onto this framework, but can still be indirectly reduced to our problem. We discuss this further in Section V-B.

If multiple (say k) relations are located at the same node, for the purpose of solving the optimization problem, we make $k-1$ copies of the node, and connect it to the original node using zero-length edges. Each of k relations is then assigned to a different copy of the node.

Similarly, if there is a relation such that different queries require access to different parts of the relation (because of selection or projection operations), we subdivide the relation into smaller, disjoint pieces such that each query requires full access to a subset of these pieces. In the worst case, however, this could result in an exponential increase in the number of sources, thus rendering the optimization problem intractable. Understanding and addressing these tradeoffs in a systematic manner is an interesting direction for furthur research.

B. NP-Hardness

The problem is NP-hard for arbitrary communication networks by an easy reduction from the Steiner Tree problem in graphs. In an instance of the Steiner Tree problem, we are given an undirected graph $G = (V, E)$ with non-negative edge weights c_e and a set of *terminals*, $T \subset V$, and we are asked to find the minimum weight tree subgraph of G that connects all the terminals [28], [29].

To reduce the Steiner tree problem to our problem: let $\mathcal{G}_C = G$, and assign a unit-sized relation S_i to node $v_i \in V$. Suppose $T = \{v_0, \ldots, v_k\}$. Define a set of $|T| - 1$ queries: $Q_i^R = \{S_0, S_i\}$ for all $i = 1 \ldots k$, with the result size equal to zero for all queries. It is easy to see that there is a Steiner tree on the terminals of cost B if and only if there is a solution to our problem with cost B – this is because an optimal solution to our problem involves sending S_0 to all remaining nodes in T.

If different nodes can carry the same data item (i.e., data may be replicated), and hence we must also choose which copy of a data item to use for executing the query, then the problem is NP-hard even for tree communication networks (see Appendix for the proof).

C. An Illustrative Example

In Figure 1 we show our running example with six data sources over a tree network, with the data sizes shown in parentheses. We also illustrate a collection of three queries, along with their query plans and the destinations. The three queries have one data source in common, S_2, whereas the rest of the data sources are different for each query. Hence the key optimization challenge here is to share the movement of S_2 across the network while executing the queries.

Figure 2 shows an optimal data movement plan computed by our algorithm to solve these queries.

- **(Query 1)** The data movement plan for Query 1 (which is also optimal for it in isolation) involves (1) joining S_1 and S_2 at C, (2) shipping $S_1 \bowtie S_2$ across the edge (C, D), (3) joining it with S_4 at D, and (4) shipping the result back to C.
- **(Query 2)** S_2 is also shipped across edge (C, D) all the way to F, where a join is performed with S_6 for Query 2 at F, and this result is finally shipped back to D.
- **(Query 3)** Finally note that the optimal plan for Query 3 in isolation (without the other queries) would have shipped S_5 all the way to B where we would have performed a join with S_2. However, since S_2 is shipped from B to F (via D), we can perform a join of S_5 with S_2 at D itself, and then ship the result to B.

D. Summary of Results

We briefly summarize our main algorithmic results, which we elaborate upon in the next two sections.

1) One of our main results is that the optimization problem can be solved optimally in polynomial time when the underlying communication network is a tree. The algorithm involves $|\mathcal{G}_C|$ (number of edges in $\mathcal{G}_C$) hypergraph

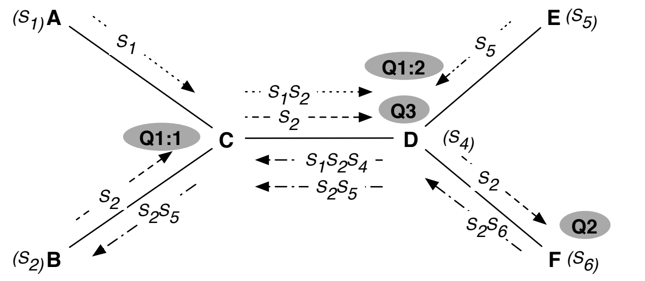


Fig. 2. An optimal Data Movement plan for the above example setup. The figure also indicates where the query operations took place, e.g., the first join for $Q1$ (denoted **Q1:1**) was done at C and the second join was done at D.

cut computations on a graph constructed by "merging" the query trees (Section IV).

2) When the underlying communication network is an arbitrary graph, we get an $O(\log n)$ approximation using the results on embedding arbitrary metrics into trees [32], [33] (Section VI-A).
3) For arbitrary communication networks, we develop constant factor approximation algorithms for several interesting special cases where we restrict the structure of the query-overlap graph (Section VI-C).
4) We show how to reduce the many-to-many aggregate queries problem to a simpler problem called the *pairs* problem, where each query only contains two sources and all result sizes are 0 (equivalently, the results can be computed anywhere and do not need to be shipped to specific destinations). This reduction holds for arbitrary communication networks, allowing us to apply our algorithms to solve this problem. When $\mathcal{G}_C$ is a tree, we use this reduction to derive a considerably simpler proof for the algorithm presented in [8] (Section V).

IV. Case when $\mathcal{G}_C$ is a tree

In this section, we present a polynomial-time algorithm for minimizing the data movement cost when the communication network $\mathcal{G}_C$ is a tree. Our algorithm involves solving a series of min-cut problems on appropriately constructed hypergraphs (one for each edge in $\mathcal{G}_C$). We begin with some background on min-cuts for hypergraphs, and then present our algorithm and the correctness proof. We then present a reduction from many-to-many aggregate queries problem to our problem (the reduction does not require that $\mathcal{G}_C$ be a tree).

A. Background: Hypergraph Min-Cut and Partition Problems

A hypergraph $\mathcal{H}$ is specified by a vertex set V and a set of hyperedges E, where each hyperedge in E is a subset of V (see Figure 3(i)). We are also given two special vertices s and t. The goal is to partition V into S and T, with $s \in S$ and $t \in T$ while minimizing the weight of the hyperedges that include vertices in both S and T (that are *cut*).

This problem generalizes the standard s-t min-cut problem which is usually solved by a max-flow algorithm. In fact, the

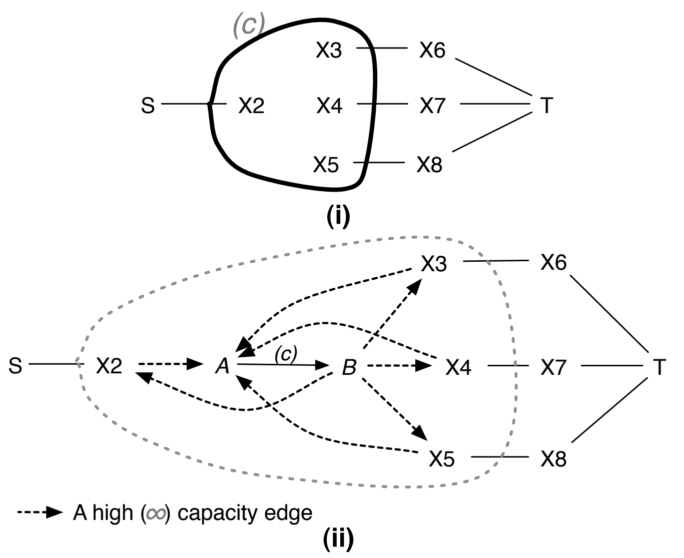


Fig. 3. (i) An example of an edge-weighted hypergraph with one 4-node hyperedge (X_2, X_3, X_4, X_5), of weight c, and several two-node hyperedges (drawn as normal edges, e.g., (S, X_2), weights not shown); (ii) Reduction of the s-t cut problem on the hypergraph to an edge capacitated flow problem requires adding two new nodes per hyperedge, A and B for the hyperedge (X_2, X_3, X_4, X_5) (two-node hyperedges remain unchanged in this construction); solving the max-flow problem on the new graph gives us the min s-t hypergraph cut on the original hypergraph.

hypergraph min cut problem can also be solved by a max-flow computation on a derived graph [34]. For completeness we briefly describe the procedure here. In essence, every hyperedge in the original graph is replaced by a subgraph, containing directed edges, as shown in Figure 3(ii). For every hyperedge, we add two new nodes and a directed edge between them of capacity equal to the weight of the hyperedge. Several high-capacity edges are also added as shown in the figure. Hyperedges containing only two nodes do not need to be changed in the process. Solving the s-t max-flow problem on this new graph (which contains no hyperedges) gives us a min s-t cut on the original hypergraph.

We define and utilize a variant of the above problem called the *hypergraph partition* problem, where instead of two special vertices s and t, we are given two sets of vertices, $L_s \subset V$ and $L_t \subset V$, $L_s \cap L_t = \emptyset$, and we are asked to find a partition of V into (S, T) that separates the vertices in L_s from the vertices in L_t such that $L_s \subseteq S$, and $L_t \subseteq T$. We denote such an instance by $(\mathcal{H}, L_s, L_t)$. It is easy to see that this problem can be reduced to the s-t min-cut problem by: (1) adding two special nodes s and t, and (2) by connecting s (similarly t) to all the nodes in L_s (similarly L_t) by infinite-weight edges.

B. Algorithm

The high level approach behind our algorithm is quite simple. The algorithm follows three main steps:

1) Build a weighted hypergraph, $\mathcal{H}_D$, by combining the query trees for all the queries. This hypergraph explicitly captures all the opportunities for sharing the movement of data sources among the queries.

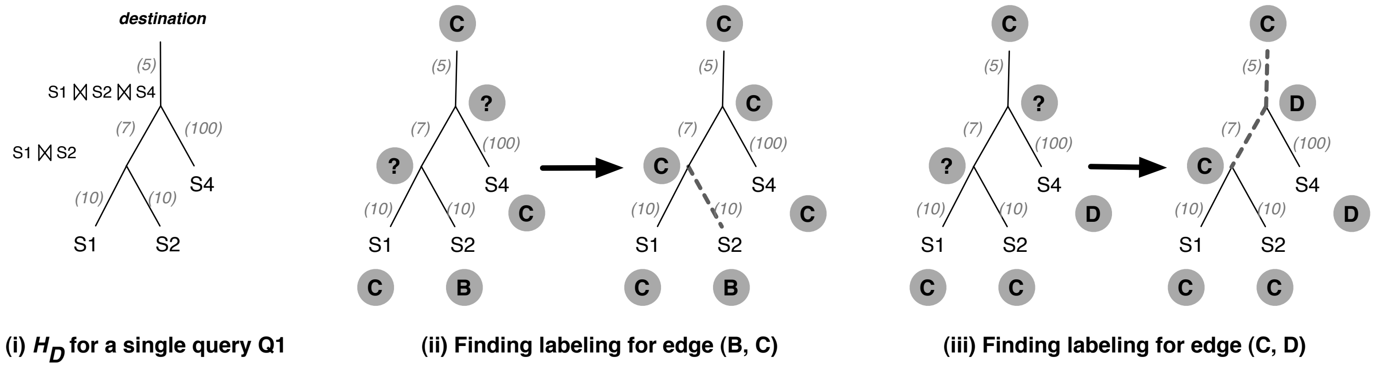

destination
(5)
S1 ⋈ S2 ⋈ S4
(7)
(100)
S1 ⋈ S2
S4
(10)
(10)
S1
S2
C
?
?
C
B
C
C
C
B
?
?
D
C
C
D
D
C
C
(i) H_D for a single query Q1
(ii) Finding labeling for edge (B, C)
(iii) Finding labeling for edge (C, D)

Fig. 4. (i) For a single query (Query 1), the hypergraph construction simply involves adding a new node corresponding to the designation; (ii, iii) Example labelings for two of the edges in the communication network for Query 1. The "?"s indicate the decisions that need to be made, and the dashed edges indicate that the edges that were cut (indicating data movement across the corresponding edge).

2) For each edge $e = (x, y)$ in $\mathcal{G}_C$, decide which data sources and intermediate results move across that edge by solving an instance of the weighted hypergraph partition problem.
3) Combine the local solutions for all the edges into a single global data movement plan.

This approach is quite similar to the approach taken in [8] for solving many-to-many aggregate queries, even though the problem we address is much more general. We discuss this connection in more detail in Section V-B.

Steps 1 and 2 for a Single Query

We begin by describing Steps 1 and 2 for a single query (Figure 4). We additionally assume that each leaf in the query tree is a distinct data source (see the general algorithm for when this assumption does not hold). The hypergraph construction is quite easy in this case. Let T be the rooted query tree for the query. We construct $\mathcal{H}_D$ by adding a new root vertex to T and attaching the original root of T to the new root – the new root denotes the node where the results have to be shipped. The weight of an edge $(b, a) \in \mathcal{H}_D$ is set to be the size of b, where b is the child of a. Figure 4(i) shows an example of this construction for **Query 1** of our running example. No hyperedges with more than 2 nodes need to be created in this case.

Now consider an edge $e = (x, y)$ in $\mathcal{G}_C$. Let $\mathcal{G}_C^x$ and $\mathcal{G}_C^y$ denote the two connected subgraphs (trees) obtained by deleting the edge e (with $x \in \mathcal{G}_C^x$ and $y \in \mathcal{G}_C^y$).

The communication cost incurred in a candidate solution because of the data transmitted over the edge (x, y) is fully determined if we know where each internal node of T (corresponding to a query operator) is evaluated (at a node in $\mathcal{G}_C^x$ *or* at a node in $\mathcal{G}_C^y$); if a node is evaluated in $\mathcal{G}_C^x$ and its parent is evaluated at a node in $\mathcal{G}_C^y$ (or vice versa), then we must ship the result of that node across the edge (x, y).

We capture this using the following partition problem. In the graph $\mathcal{H}_D$, we assign a label to each node based on which connected component it lies in. All the leaf nodes which lie in $\mathcal{G}_C^x$ are labeled x, and the leaf nodes in $\mathcal{G}_C^y$ are labeled y. The new root node (corresponding to the destination) is labeled according as well (depending on whether the destination is in $\mathcal{G}_C^x$ or $\mathcal{G}_C^y$). The partition problem is simply to find a cut in the graph $\mathcal{H}_D$ that separates the nodes labeled x from the nodes labeled y.

It is easy to see that this partition problem exactly captures the communication cost minimization problem for the edge (x, y). Each edge in $\mathcal{H}_D$ that is cut (i.e., has different labels for its endpoints) corresponds to a data movement (of a data source or an intermediate result) across the edge (x, y). We illustrate this with two examples.

Example 1: Continuing with the example shown in Figure 4(i), Figure 4(ii) shows the partition problem instantiated for the edge (B, C) of the communication network. In this case, node S_2 has label B and all the other leaf nodes have label C (including the node corresponding to the new root, i.e., the destination lies in the connected component corresponding to C). By giving both internal nodes of the tree label C, we can see that there is only one edge, $S_2 \rightarrow C$, of cost 10 (shown as a dashed edge) whose endpoints have different labels. This corresponds to shipping data item S_2 across the edge (B, C) in the communication network, and evaluating all operators at the nodes in the connected component $\mathcal{G}_C^C$.

Example 2: Now consider the example shown in Figure 4(iii). This is the same query tree, but the labeling now corresponds to edge (C, D). Note that the only leaf with label D is S_4. With the labeling of the leaves as shown, if the internal labels are set as C and D for the two internal nodes then the cost of the solution corresponds to the two dashed edges shown in the figure since the ends of these edges have different labels. Note that this corresponds to the cost of shipping $S_1 \bowtie S_2$ from C to D and then finally the result $S_1 \bowtie S_2 \bowtie S_4$ back in the other direction.

Steps 1 and 2 for Multiple Queries

Given a set of queries and corresponding query plan trees, we first add new root nodes to the query trees (corresponding to the destinations) as above. We then superimpose all the query trees into a single directed acyclic graph, $\mathcal{H}_Q$, where we merge the leaves carrying the same information together into a single node. The edges in $\mathcal{H}_Q$ are oriented from a node to its parent in a query tree. Let $p(v) = \{v\} \cup \{u|(v,u) \in \mathcal{H}_Q\}$ denote the set of parents of v. We define the hypergraph $\mathcal{H}_D = (V(\mathcal{H}_Q)^3, \{p(v)|v \in V(\mathcal{H}_Q)\})$, i.e., for each vertex v in $V(\mathcal{H}_Q)$, we add a hyperedge containing that vertex and all its parents in $\mathcal{H}_Q$. The weight of hyperedge $p(v)$ is set to be the size of data item v.

Figure 5(i) shows how this construction is done for the three query trees in our running example. All three queries share the data source S_2, and we capture this by using a hyperedge that contains S_2 and three appropriate internal nodes from the query trees. The weight of the hyperedge is set to be the size of data item S_2.

Next consider an edge (x, y) in $\mathcal{G}_C$ and let $\mathcal{G}_C^x$ and $\mathcal{G}_C^y$ be the connected components obtained by deleting (x, y). Once again we assign a label (x or y) to each of the leaf nodes in $\mathcal{H}_D$ depending on which connected component of $\mathcal{G}_C$ it lies in; we then find the minimum cost cut that separates the nodes labeled x from the nodes labeled y.

We denote the minimum weight cut found for edge (x, y) by $C(A_{xy}; \bar{A_{xy}}), A_{xy} \subseteq V(\mathcal{H}_D), \bar{A_{xy}} \subseteq V(\mathcal{H}_D)$ (with A_{xy} denoting the set of vertices labeled x).

Example 3: In Figure 5, we show all three queries superimposed, with the shared source S_2 creating one large hyperedge. All the other hyperedges have size 2 and are shown as regular edges. In Figure 5(ii) we show the labeling for the edge (C, D) of the communication network $\mathcal{G}_C$. Several source and destination nodes are labeled in advance, and the hypergraph min-cut computation labels the remaining nodes optimally. This labeling captures the total communication cost incurred because of the data shipped across edge (C, D).

For Query 1, $S_1 \bowtie S_2$ of size 10 is shipped across the edge (C, D) and the result $(S_1S_2S_4)$ of size 5 is shipped back. In addition, we pay the cost to ship S_2 of size 10 across the edge (this is the cost of the hyperedge having nodes with label C and D) so it is part of the cut. In addition, for Query 3, we pay the cost of shipping S_2S_5 of size 6 across the edge.

Step 3

The above two steps can be used to find the locally optimal solutions for each edge in the communication graph. However these solutions may not be consistent with each other (the locally optimal solutions for two different edges may not agree on where the internal nodes should be evaluated).

Let i denote an internal node in the hypergraph $\mathcal{H}_D$. We will construct a *directed* graph $\mathcal{J}^i$ with vertex set $V(\mathcal{G}_C)$, and edges defined as follows: for $e = (x, y) \in \mathcal{G}_C$, if i is assigned label y in the hypergraph cut found above (i.e., if $i \in \bar{A_{xy}}$), then add a directed edge from x to y in $\mathcal{G}_C$ (and vice versa if the i is assigned label x). For instance, in the example shown in Figure 5, $\mathcal{J}^{S_1S_2S_4}$ will contain a directed edge from C to D (since the node $S_1S_2S_4$ is assigned label D).

Now we consider a vertex v in $\mathcal{J}^i$ with out-degree 0. This implies that the decisions made on all edges incident to v agree to place i on v; then we simply place the query operator corresponding to i at v. The input data items for that operator

[3] $V(\mathcal{H}_Q)$ denotes the vertex set of $\mathcal{H}_Q$.

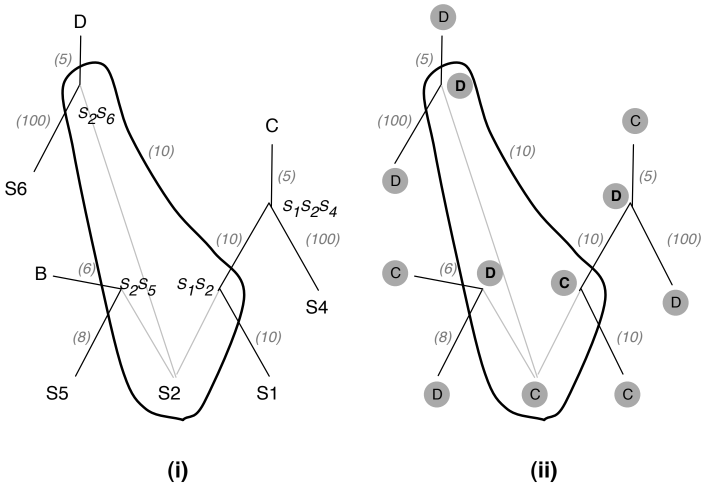


Fig. 5. (i) Hypergraph construction for all three queries in the running example; (ii) A labeling corresponding to the edge (C, D) found after solving the hypergraph partition problem.

are shipped from their respective locations to v. For example, the internal node $S_1S_2S_4$ is evaluated at D in our running example (see Figure 2 for the complete solution), and S_1S_2, which is evaluated at C, is accordingly shipped from C to D (the other input data item S_4 is already at D).

In the next section, we will prove there is exactly one such vertex with out-degree 0 (denoted $\phi(i)$), for every internal node $i \in V(\mathcal{H}_D)$, under the assumption that each hypergraph partition problem has a unique solution (this can be guaranteed by adding small random perturbations to the data item sizes [35]). We call such a solution "globally consistent". We will also prove that the cost of acquiring the data items (original data sources or the intermediate results) required to evaluate the operator corresponding to i at v is accounted for in the costs of the local hypergraph cuts.

C. Proof of Correctness

We begin with some notation. Let $\mathcal{G}_C = (V, E)$ denote the communication graph as before. For an edge $(x, y) \in \mathcal{G}_C$, let L_{xy}^x and L_{xy}^y denote the nodes in $\mathcal{H}_D$ labeled x and labeled y respectively. As above, we denote the minimum cut corresponding to e by $C(A_{xy}; \bar{A_{xy}})$ (so we have that $L_{xy}^x \subseteq A_{xy}$ and $L_{xy}^y \subseteq \bar{A_{xy}}$).

Consider two adjacent edges $e_1(u, v)$ and $e_2(v, w)$ in the communication network $\mathcal{G}_C$. We call $C(A_{uv}; \bar{A_{uv}})$ and $C(A_{vw}; \bar{A_{vw}})$, the minimum weight cuts corresponding to e_1 and e_2, *locally consistent* if $A_{uv} \subseteq A_{vw}$.

The following simple lemma shows that "local consistency" on every pair of adjacent edges implies that $\mathcal{J}_i$ has exactly one vertex with out-degree 0, $\forall i$.

Lemma 1: All cuts form a globally consistent solution if for any two adjacent edges, the two corresponding minimum cuts are locally consistent.

Proof: First we note that $\mathcal{J}^i$ is a tree with all its edges directed (since it is obtained by making each of the edges in

$\mathcal{G}_C$ directed). It can be shown that any such tree has exactly one node with out-degree 0 if and only if no two adjacent edges in $\mathcal{J}^i$ share the same tail.

For $\mathcal{J}^i$ to *not* satisfy the latter property, we must have that, for a pair of adjacent edges $(u, v), (v, w) \in \mathcal{G}_C$:
 in cut $C(A_{uv}; \bar{A}_{uv})$, i is labeled u (i.e., $i \in A_{uv}$), *but*
 in cut $C(A_{vw}; \bar{A}_{vw})$, i is labeled w (i.e., $i \in \bar{A}_{vw}$).
But if $A_{uv} \subseteq A_{vw}$, there is no such node. Therefore, $\mathcal{J}^i$ has exactly one vertex with 0 out-degree for all $i \in V(\mathcal{H}_D)$. ■

The next lemma guarantees that local consistency holds for any pair of adjacent edges.

Lemma 2: We assume the uniqueness of the minimum cut solutions. Let $C(A_{uv}; \bar{A}_{uv})$ and $C(A_{vw}; \bar{A}_{vw})$ be minimum solutions for the instances $(\mathcal{H}_D, L^u_{uv}, L^v_{uv})$ and $(\mathcal{H}_D, L^v_{vw}, L^w_{vw})$ respectively where $L^u_{uv} \subseteq A_{uv}$ and $L^v_{vw} \subseteq A_{vw}$. If $L^u_{uv} \cup L^v_{uv} = L^v_{vw} \cup L^w_{vw} = L$ and $L^u_{uv} \subseteq L^v_{vw}$, then $A_{uv} \subseteq A_{vw}$.

Proof: Suppose the lemma is not true. Let S be the set of vertices s such that $s \notin L$, $s \in A_{uv}$ and $s \notin A_{vw}$. It is not hard to see

$$w(\{e | e \cap S \neq \emptyset \wedge e \cap \bar{A}_{uv} \neq \emptyset \wedge e \cap (A_{uv} - S) = \emptyset\}) \\ < w(\{e | e \cap S \neq \emptyset \wedge e \cap (A_{uv} - S) \neq \emptyset \wedge e \cap \bar{A}_{uv} = \emptyset\})$$

since otherwise $C(A_{uv} - S; \bar{A}_{uv} + S)$ is a better solution than $C(A_{uv}; \bar{A}_{uv})$. But we have

$$C(A_{vw} + S; \bar{A}_{vw} - S) = C(A_{vw}; \bar{A}_{vw}) \\ + w(\{e | e \cap S \neq \emptyset \wedge e \cap (\bar{A}_{vw} - S) \neq \emptyset \wedge e \cap A_{vw} = \emptyset\}) \\ - w(\{e | e \cap S \neq \emptyset \wedge e \cap A_{vw} \neq \emptyset \wedge e \cap (\bar{A}_{vw} - S) = \emptyset\}) \\ < C(A_{vw}; \bar{A}_{vw}).$$

The inequality holds since:
$\{e | e \cap S \neq \emptyset \wedge e \cap (\bar{A}_{vw} - S) \neq \emptyset \wedge e \cap A_{vw} = \emptyset\}$
 $\subseteq \{e | e \cap S \neq \emptyset \wedge e \cap \bar{A}_{uv} \neq \emptyset \wedge e \cap (A_{uv} - S) = \emptyset\}$
which follows from $\bar{A}_{vw} - S \subseteq \bar{A}_{uv}$ and:
$\{e | e \cap S \neq \emptyset \wedge e \cap (A_{uv} - S) \neq \emptyset \wedge e \cap \bar{A}_{uv} = \emptyset\}$
 $\subseteq \{e | e \cap S \neq \emptyset \wedge e \cap A_{vw} \neq \emptyset \wedge e \cap (\bar{A}_{vw} - S) = \emptyset\}$
which follows from $A_{uv} - S \subseteq A_{vw}$. ■

Lemma 3: The cost of moving data items as needed to execute the query operators is equal to the total cost of the hypergraph cut solutions.

Proof: Consider an internal node $i \in V(\mathcal{H}_D)$, and let $\phi(i) \in \mathcal{G}_C$ denote the vertex with out-degree 0 in $\mathcal{J}^i$. Consider an edge $(x, y) \in \mathcal{G}_C$. It is easy to see that if i has label y in $C(A_{xy}; \bar{A}_{xy})$, then $\phi(i) \in \mathcal{G}^y_C$.

Let j denote a child of i in $\mathcal{H}_Q$ (the DAG from which $\mathcal{H}_D$ is derived). Now if $\phi(j) \in \mathcal{G}^x_C$, then we must ship the data generated by j to $\phi(i)$ through (x, y). However since j is labeled x in that case, the edge $(j, i) \in E(\mathcal{H}_D)$ is *cut* in the hypergraph cut $C(A_{xy}; \bar{A}_{xy})$, and the cost of shipping the data across (x, y) is appropriately counted in the weight of the hypergraph cut. On the other hand, if $\phi(j) \in \mathcal{G}^y_C$, then the data generated by j does not have to be communicated across (x, y); this is appropriately captured in the hypergraph cut weight, since the nodes i and j are labeled the same in that case, and the edge (j, i) is not cut. ■

Theorem 1: The algorithm finds a global optimum solution.

Proof: It is easy to see that the minimum cut instances we solve satisfy the condition in Lemma 2. Therefore, we have local consistency for all adjacent edges from which the global consistency follows. Since each of the local solutions is optimal for the corresponding communication edge, the solution obtained by putting those together is also globally optimal. ■

We remark that the above solution cannot be directly extended to share intermediate results. A natural option would appear to be to identify and merge the internal nodes that correspond to the same intermediate result (by adding appropriate hyperedges). However, we then disallow the possibility of *not sharing* the intermediate result (and instead generating it independently at different locations) – it is easy to construct instances where the latter option is optimal.

V. Many-to-many Aggregate Queries and the Pairs Problem

Before moving on to discuss the case when G_C is not a tree, we define a special case of our general problem called the "pairs" problem, and show how it generalizes the Single Sink Rent-or-Buy (SROB) problem. We then show how our algorithm in the previous section can be used to solve the many-to-many aggregate queries problem [8], and present a simple proof of correctness.

A. Pairs Problem

We define the *pairs* problem to be a special case of our general problem where all queries are restricted to be over two nodes each, and furthermore, the query results are all of size 0 (in other words, the query results do not need to be shipped to any sinks). The data items are allowed to be of unequal sizes. With just two data sources in each query, the issue of whether partial aggregation is allowed or not is irrelevant.

Definition 1: A *query-overlap graph* (denoted $\mathcal{H}$) corresponding to an instance of the pairs problem is defined to be a graph where the vertices correspond to the set of data items and each edge corresponds to a pair query.

This problem generalizes the Single Sink Rent-or-Buy (SROB) problem (a generalization of the Steiner tree problem). In an SROB instance, we are given a graph $G = (V, E)$, a parameter $M > 1$, and a set of demands associated with a subset of vertices $D \subset V$. Each demand $j \in D$ has a non-negative weight d_j. A solution consists of a set of facilities $F \subset V$ to be opened, a tree subgraph T of G spanning F, and an assignment, $f()$, of demands to the open facilities (the demand j is assigned to $f(j) \in F$).

The total cost of the solution is $\sum_{j \in D} d_j \cdot \ell(j, f(j)) + M \sum_{e \in T} c_e$, where the function ℓ denotes the shortest path distance using edge lengths c_e. There is no cost for opening facilities but the open facilities have to be connected together. This problem was also called the Connected Facility Location Problem [31]. This problem is NP-hard, and an approximation algorithm with a factor of 2.92 was presented recently [30].

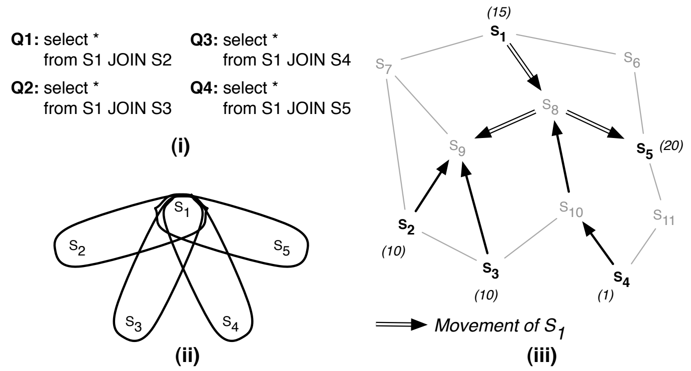
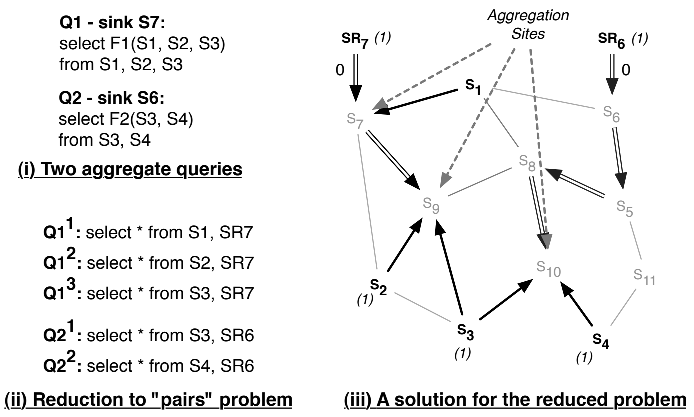

```
Q1: select *
    from S1 JOIN S2
Q2: select *
    from S1 JOIN S3
Q3: select *
    from S1 JOIN S4
Q4: select *
    from S1 JOIN S5
```

Fig. 6. (i) An instance of the pairs problem with 4 queries; (ii) The corresponding query overlap graph, $\mathcal{H}$, is a *star*; (iii) Mapping the case when $\mathcal{H}$ is a *star* to Single Sink Rent-or-Buy (SROB) Problem.


```
Q1 - sink S7:
select F1(S1, S2, S3)
from S1, S2, S3

Q2 - sink S6:
select F2(S3, S4)
from S3, S4
```
```
Q1^1: select * from S1, SR7
Q1^2: select * from S2, SR7
Q1^3: select * from S3, SR7

Q2^1: select * from S3, SR6
Q2^2: select * from S4, SR6
```

(i) Two aggregate queries
(ii) Reduction to "pairs" problem
Aggregation Sites
SR7 (1)
0
SR6 (1)
0
S1
S7
S6
S8
S9
S5
S10
S11
S2
(1)
S3
(1)
S4
(1)
(iii) A solution for the reduced problem

Fig. 7. Reducing a set of many-to-many aggregate queries to the "pairs" problem entails adding a new data source for each query, and replacing each query with a collection of pair queries ("⇒" indicates the movement of a partial or complete aggregate in the opposite direction).

Consider a special case of the pairs problem where $\mathcal{H}$ is a "star" graph. Figure 6(i,ii) shows an example of this. A solution to this problem requires the center of the star (S_1) to be shipped along a tree and the leaves of the star to be shipped to the nearest location where S_1 is shipped. Figure 6(iii) shows a possible solution, where S_1 is shipped to S_8, S_9, and S_5, and it meets with S_2, S_3, S_4, and S_5 at nodes S_9, S_9, S_8, and S_5 respectively so that the queries can be executed.

If all data items have the same size, then the problem is exactly equivalent to the Steiner tree problem (Section III-B). When the data item sizes are unequal, then it reduces to the SROB problem. T corresponds to the tree along with the center of the star (S_1) travels (M = size of S_1). The demands correspond to the remaining data items referenced in the queries, and the weight of a demand is set to be the size of the corresponding data item. Another way to look at the solution is: we "buy" the edges on which S_1 is shipped, and "rent" the other edges since we pay a fixed cost ($M = |S_1|$) for the edges on which S_1 is shipped and a variable cost (depending on the sizes of the other items) for the other edges.

B. Many-to-many Aggregate Queries

In the many-to-many aggregate queries problem [8] each query needs to compute an aggregate function over the values produced by a subset of the data items, and the result needs to be transmitted to a specified sink. The data items are assumed to be identical sizes, and the data may be aggregated along the way to the sink (the size of the partial aggregate is assumed to be a constant, and may be different from the size of a data item). As with our setup, the aggregated values cannot be shared across queries. Figure 7 (i) shows an example instance of this problem with two queries, **Q1** which computes an aggregate function over three sources S_1, S_2, S_3, and **Q2** which computes an aggregate over sources S_3, S_4. All data sources are of unit size, and the size of a partial aggregate for both queries is assumed to be 2 (corresponding to a function like AVERAGE). As mentioned before, this problem cannot be mapped to our general problem directly (because it allows partial aggregation). Next we show how to reduce this problem to the *pairs problem*, and then present an algorithm for it.

Let a query Q be an aggregate query over data sources $S_1, \ldots, S_k$, and let the destination be node S_d. We introduce a new data source, SR_d, with size equal to the size of a partial aggregate for the query, and attach this data source to the node S_d with a zero cost edge. We then create k "pair" queries: (S_1, SR_d), ..., (S_k, SR_d). We then construct an instance of our (pairs) problem by combining the queries generated for each of the aggregate queries.

Figure 7 (ii) shows the resulting set of queries for the example instance, where we introduce the sources SR_7 and SR_6 for **Q1** and **Q2** respectively.

Figure 7 (iii) shows an example solution to the resulting pairs problem (assuming the size of the partial aggregate to be the same as the size of a data item). This solution corresponds to a solution for executing the original two queries. Specifically, the movement of SR_6 or SR_7 (the new nodes added to represent the sinks) across an edge corresponds to a movement of a partial (or full) aggregate in the opposite direction. For example, in Figure 7 (iii), SR_7 is moved across the edge (S_7, S_9). This corresponds to movement of the partial aggregate $F1(S_2, S_3)$ across the edge (S_9, S_7). We formalize this in the following lemma.

Lemma 4: A solution for the resulting pairs problem can be mapped back to a solution for the original many-to-many aggregate queries problem with the same cost.

Silberstein-Yang Construction: Our algorithm, based on solving a hypergraph partition problem for each edge of $\mathcal{G}_C$, reduces to the algorithm presented by Silberstein and Yang [8]. For each edge (x, y) of $\mathcal{G}_C$ (treated as a directed edge from x to y), they construct a bipartite graph where on one side there are nodes corresponding to the queries, and the other side has nodes corresponding to the data items. There is an edge in the bipartite graph between a query node and a data node if the query needs to aggregate the data item, and if the query destination (sink) is in $\mathcal{G}_C^y$ and the data item is in

$\mathcal{G}_C^x$. They then solve a minimum vertex cover (VC) problem over this bipartite graph. The main point is that, for a query Q, either all the data items for it that are in $\mathcal{G}_C^x$ are shipped across the edge (this corresponds to choosing the data items as part of the minimum vertex cover), or the aggregation is done first and the result shipped across the edge (the latter corresponds to the query node being chosen as part of the minimum vertex cover). The main issue to establish is that all the Vertex Cover solutions (corresponding to each edge of $\mathcal{G}_C$) can be put together to create an optimal solution for the entire problem. Next we show a simpler proof of correctness for this algorithm.

Formally, let us consider two adjacent edges $e_1 = (u, v)$ and $e_2 = (v, w)$. Root $\mathcal{G}_C$ at v. Let $T(u)$ be the subtree rooted at node u. Let $A = V(T(u)), C = V(T(w)), B = V - A - C$ and $E_{\mathcal{H}}(X; Y) = \{(x, y) \in E(\mathcal{H}) | x \in X, y \in Y\}$. Let $N_G(v) = \{u | (u, v) \in E(G)\}$ and $N_G(S) = \cup_{v \in S} N_G(v)$. Essentially, for edge e_1, we run minimum vertex cover (VC) algorithm on bipartite graph $G_1(A, B \cup C; E_{\mathcal{H}}(A; B \cup C))$ and for e_2, on $G_2(A \cup B, C; E_{\mathcal{H}}(A \cup B; C))$.

Proof of Correctness: We only need to prove that for any vertex $a \in A$, if $a \in VC(G_2)$, then $a \in VC(G_1)$. The interpretation of a node $a \in VC(G_2)$ is that it is shipped across edge e_2. For the solutions to be consistent, we need to have a also shipped across e_1 so that it can be shipped across e_2. Otherwise if the solution for e_1 corresponds to aggregating a and not shipping it across e_1, but shipping it across e_2 then they are not consistent.

Suppose this is not true, let S be the set of vertices such that $v \in VC(G_2)$ but $v \notin VC(G_1)$. Let $U = N_{G_2}(S) \cap (C - VC(G_2))$. Observe that $w(S) < w(U)$, since if the converse is true we can replace S with U and obtain a vertex cover in G_2 with lower weight (recall that the optimum solution is assumed to be unique). Since $S \cap VC(G_1) = \emptyset$, we get $N_{G_1}(S) \subseteq VC(G_1)$. Thus $U \subseteq VC(G_1)$. We claim that $VC(G_1) - U + S$ is a vertex cover for G_1. The key observation here is

$$N_{G_1}(U) \subseteq N_{G_2}(U) \subseteq VC(G_2) \subseteq VC(G_1) \cup S.$$

Thus, each edge that cannot be covered by $VC(G_1) - U$ has an endpoint in S. So our claim is true, but this violates the optimality of $VC(G_1)$ since $w(VC(G_1) - U + S) = w(VC(G_1)) - w(U) + w(S) < w(VC(G_1))$. ■

VI. Case when $\mathcal{G}_C$ is not a tree

We first present an $O(\log(n))$ approximation obtained by embedding $\mathcal{G}_C$ into a tree (using the result on embedding arbitrary metrics into trees [32], [33]). We are also able to develop an exact dynamic programming-based algorithm for when we have only one query plan. We then present several constant factor approximations for the *pairs* problem by restricting the complexity of the query overlap graph $\mathcal{H}$.

A. An $O(\log n)$ Approximation for General $\mathcal{G}_C$

We use the notation from [33]. Let V be the set of vertices of a graph, and let d, and d' be distance functions over V. The metric (V, d') is said to dominate the metric (V, d) if $d'(u, v) \geq d(u, v)$ for all $u, v \in V$. Let $\mathcal{S}$ be a family of metrics over V, and $\mathcal{D}$ a distribution over $\mathcal{S}$. $(\mathcal{S}, \mathcal{D})$ is called a *α-probabilistic approximation* of (V, d) if every metric in $\mathcal{S}$ dominates (V, d) and $E_{d' \in (\mathcal{S}, \mathcal{D})} d'(u, v) \leq \alpha d(u, v)$. A tree metric is a metric induced by shortest path distances over a tree.

Theorem 2: [33] For any given metric (V, d), we can produce a distribution of tree metrics which is an $O(\log n)$-probabilistic approximation of d in polynomial time.

We sketch our approximation algorithm: suppose $(\mathcal{S}, \mathcal{D})$ is the $O(\log n)$ approximation of (V, d). We randomly pick a tree T from $\mathcal{S}$ according to the distribution $\mathcal{D}$. We solve the problem on T optimally by using the algorithm introduced earlier for trees. Suppose SOL_T is the solution. Now, we map SOL_T back to original graph. Specifically, if an edge $e = (u, v) \in T$ is used for sending i's (i could be a leaf node or internal node of some query plan tree) information in SOL_T for $i \in V(\mathcal{H})$, we use the shortest path from u to v in $\mathcal{G}_C$ for sending i's information.

First, we claim $E_{T \in \mathcal{S}}(OPT_T) \leq O(\log n) OPT$ where OPT_T and OPT are optimal solution in T and $\mathcal{G}_C$, respectively. This can be easily shown by seeing that if there is an optimal solution of cost OPT in $\mathcal{G}_C$, then the expected cost of this solution (expectation taken over the choice of tree) increases by a factor of $O(\log n)$. The cost of an optimal solution OPT_T cannot be more than this cost. Since the tree metric of T dominates the original metric, we can see the (expected) cost of our solution is at most $E_{T \in \mathcal{S}}(OPT_T)$.

B. Dynamic Programming Algorithm for a Single Query

In contrast with some of the other problems in distributed query processing (e.g. the many-to-many aggregate queries problem), our main problem can be solved in polynomial time for a single query (even if the query contains arbitrary n-ary operations). This follows from the observation that the *principle of optimality* holds in this case, and the optimal plan can be computed in a bottom-up fashion using dynamic programming.

For each subtree T in the query tree and for each node v, we compute the optimal cost of computing and transmitting the result of T to v (denoted by $OPT(T, v)$); the final operation (corresponding to the root of the subtree) may or may not be done at v. Now, consider a subtree T_i with c children, $T_i^1, \ldots, T_i^c$. For each node v_k, we can easily compute the optimal cost of computing the result of T_i at v_k using $OPT(T_i^m, v_k), \forall 1 \leq m \leq c, 1 \leq k \leq n$, by considering all possible locations v_k for computing the final operation in T_i. Namely,

$$OPT(T_i, v) = \min_{v_k \in V} \left(\sum_{m=1}^{c} OPT(T_i^m, v_k) + w(T_i) \cdot d(v_k, v) \right)$$

where $w(T_i)$ is size of the result of T_i. This can be done in time $O(nc)$, giving us a $O(n^2 m + n^3)$ algorithm for computing the optimal cost, where m is the number of nodes in the query tree. The second term accounts for the cost to compute shortest paths between all pairs of nodes.

C. Approximation Algorithms for the Pairs Problem

We will make use of approximation algorithms for the Steiner tree problem (when data item sizes are equal) or the SROB problem (when data item sizes are arbitrary) as subroutines. Let ρ denote the approximation ratio for the appropriate problem. As discussed in Section II, the best known values for ρ for the Steiner tree and SROB problems are 1.55 [28] and 2.92 [30] respectively.

Our main result is a constant approximation if $\mathcal{G}_C$ has constant *star arboricity*.

Definition 2: The star arboricity $SN(G)$ of a graph $G = (V, E)$ is the minimum number k such that E can be partitioned into sets $E_1, E_2, \ldots, E_k$ and each connected component of $G_i = (V, E_i)$ is a star for $1 \leq i \leq k$.

Theorem 3: If $SN(\mathcal{H})$ can be computed in polynomial time we can obtain an $\rho SN(\mathcal{H})$-approximation.

Proof: The algorithm simply first decomposes the query-overlap graph $\mathcal{H}$ into star forests $\mathcal{H}_1, \ldots, \mathcal{H}_{SN(\mathcal{H})}$, solves $\mathcal{H}_i$ separately, then glues together the solutions for all $\mathcal{H}_i$ together. It is easy to see that the cost of an optimal solution OPT_i for $\mathcal{H}_i$ for any i is at most the cost of an optimal solution OPT. So, the cost of our approximation is at most $\sum_{i=1}^{SN(\mathcal{H})} \rho \cdot OPT_i \leq \rho \cdot SN(\mathcal{H}) \cdot OPT$. ■

The constant approximations for the following special cases can be obtained by applying the above theorem.

1) $\mathcal{H}$ **is a tree:** It is easy to see that $SN(T) \leq 2$ for any tree T (by defining the centers of stars as alternate levels of the tree). So, we have a 2ρ approximation.
2) $\mathcal{H}$ **is a bounded degree tree:** We can solve this case optimally (in polynomial time) using dynamic programming. The dynamic program is similar to the one we used to solve single query case in Section VI-B. Suppose T_v is the subtree rooted at v and $v_1, \ldots, v_c$ are v's children. $OPT(T_v, u)$ is the optimal cost for the instance where $\mathcal{H} = T_v \cup \{(v, u)\}$ with $w(u) = \infty$ Then, $OPT(T_v, u) = \min_{u_1,\ldots,u_c \in V}(\sum_{i=1}^{c} OPT(T_{v_i}, u_i) + w(v) \cdot MST(u_1, \ldots, u_c, v, u))$ where $MST(.)$ is the cost of minimum Steiner (or SROB) tree connecting all vertices in its argument (Note that minimum steiner or SROB trees can be computed in polynomial time for a constant number of terminals).
3) $\mathcal{H}$ **has arboricity** α: We have a $2\alpha\rho$ approximation[4].
4) $\mathcal{H}$ **is a planar graph:** It is known that the arboricity of any planar graph is at most 3 ([36]). So, we can have a 6ρ approximation.
5) **The maximum degree of** $\mathcal{H}$ **is a bounded constant** Δ**:** We can have a Δ-approximation, since we can decompose $\mathcal{H}$ into at most Δ bounded degree star forests (by repeatedly finding a arbitrary spanning star forest and deleting it).

[4]The arboricity of a graph is defined in a similar way to the star arboricity, except that each connected component is required to be a tree, not a star.

If $\mathcal{H}$ is a tree and all data items are equal-sized, we adopt the following algorithm that performs strictly better than gluing together Steiner trees for alternating levels of stars: Grow a Steiner tree bottom up in the following manner: Let $T(v)$ is the tree grown from v, and $v_1, v_2, \ldots, v_l$ are v's children. $T(v)$ is the (approx) Steiner tree connecting v and all $T(v_i)$s. $T(v)$ can be computed by first shrinking all $T(v_i)$s to single nodes, then run Steiner tree approximation with v and these shrunk nodes as terminals. We prove in next lemma it is 1.5ρ approximation if the height of $\mathcal{H}$ is at most 2. The question whether it achieves a ratio strictly better than 2ρ is left open.

Lemma 5: The above algorithm is a 1.5ρ-approximation when $\mathcal{H}$ has maximum height 2 (see Appendix for the proof).

VII. Experimental Results

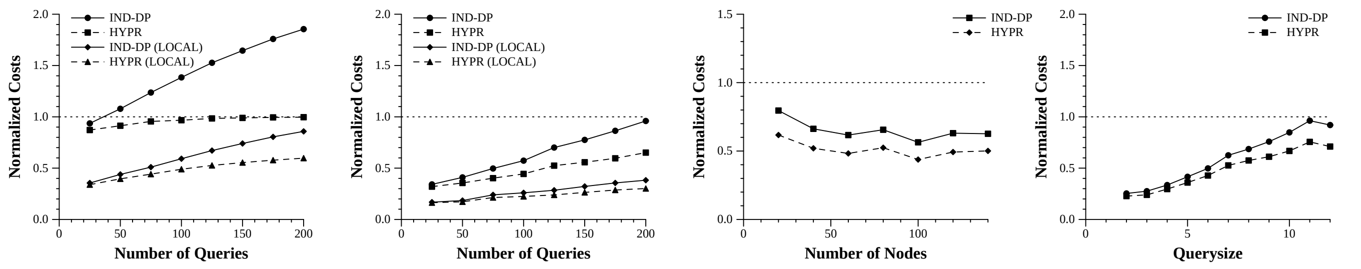
In this section, we present a preliminary performance evaluation of the algorithms presented in this paper over synthetically generated datasets and query workload. The main goals of our evaluation are to illustrate the importance of sharing data movement during multi-query optimization in distributed systems, and to show the effectiveness of our approximation algorithms at finding good sharing plans. We begin with a brief description of the experimental setup.

In all the experiments, we compare the performance of our proposed algorithms with the approach of optimizing each query optimally in isolation using the DP algorithm described in Section VI-B (called *IND-DP*). When using the latter approach, although we don't try to explicitly share data movement, any incidental sharing is accounted for when computing the total communication cost. For each experiment, we also compute the optimal cost of a *NAIVE* approach wherein the data from all data sources that are referenced in the queries, is collected at a single site. We use the cost incurred by this NAIVE approach to normalize the costs of our algorithm and *IND-DP*, and report these normalized costs.

For each of the experiments, we randomly generate a set of data sources, distributed in a 2-dimensional plane, and we add communication edges between pairs of sources that are sufficiently close to each other. If the communication network is required to be a tree, we compute the minimum spanning tree of the communication network and discard the rest of the edges. We report results for two different setups:

- **Dataset 1:** The sizes of all the data sources were set to be identical; this captures application domains such as sensor networks and distributed streams, where the data sources generate equal amounts of data in each time step.
- **Dataset 2:** The data source sizes were randomly chosen from a tri-modal distribution as follows: for 75% of the data sources, the data item sizes were chosen uniformly at random from the interval [100, 200], for 20% of the data sources, the sizes were chosen from [1000, 2000], whereas for the remaining 5% data sources, the sizes were chosen from the interval [10000, 20000].

The query workload is randomly generated by choosing each query to be over a random subset of the sources, with the



(i) Varying No. of Queries - Dataset 1 (ii) Varying No. of Queries - Dataset 2 (iii) Varying No. of Nodes - Dataset 2 (iv) Varying Max Query-Size - Dataset 2

Fig. 8. Results for when the communication network is a tree: the costs of HYPR and IND-DP are normalized using the cost of the NAIVE solution. LOCAL refers to a query workload where the queries are restricted to be over geographically co-located sources.

number of sources in it chosen randomly between 2 and *max-query-size* (an experimental parameter). We also experiment with a query workload where all queries are chosen to be over geographically co-located sources (denoted LOCAL); this is enforced by requiring that all the sources in a query be within a specified distance of each other. Each plotted point in the graphs corresponds to an average over 25 random runs.

A. G_C is a Tree

With the first set of experiments, we compare the performance of our hypergraph-based algorithm (HYPR) with IND-DP. As mentioned above, we restrict the communication network to be a tree by finding the minimum spanning tree and deleting all edges that are not part of the MST.

We ran experiments with several values of the experimental parameters, and report the results from a representative set of experiments in Figure 8. In these experiments, compare the performance of the two algorithms as the number of nodes (default value: 100), number of queries (default value: 50), and the max-query-size (default value: 5) were varied.

Figures 8 (i) and (ii) show the effect of increasing the number of queries on the performance of the two algorithms for the two datasets and for the two query workloads. As we can see, in all four cases, the communication cost incurred by our approach (HYPR) is significantly lower than the costs of the other two approaches (IND-DP or NAIVE); this validates our assertion that sharing of data movement is paramount when executing many queries over distributed data sources. The performance of HYPR and NAIVE illustrates several interesting features. As the number of queries is increased, there is a point at which the optimal solution degenerates to NAIVE (i.e., the optimal solution requires collecting all data at a central location). This is especially true for Dataset 1 (equal-sized data sources) and non-local queries. Since the query may involve data sources that are far from each other, the total amount of data movement required to execute the queries is quite high, and the NAIVE option soon becomes preferable.

Dataset 2 however penalizes the NAIVE approach significantly – it contains several very large data sources (about 5), because of which the optimal solution typically collects the rest of the data sources at those locations and evaluates the queries there; on the other hand, NAIVE is forced to move all but one of those data sources, thus incurring a high penalty. For both datasets, the performance of IND-DP and HYPR is much better than NAIVE for the LOCAL query workload; the NAIVE solution forces a much higher data movement than required to execute such local queries.

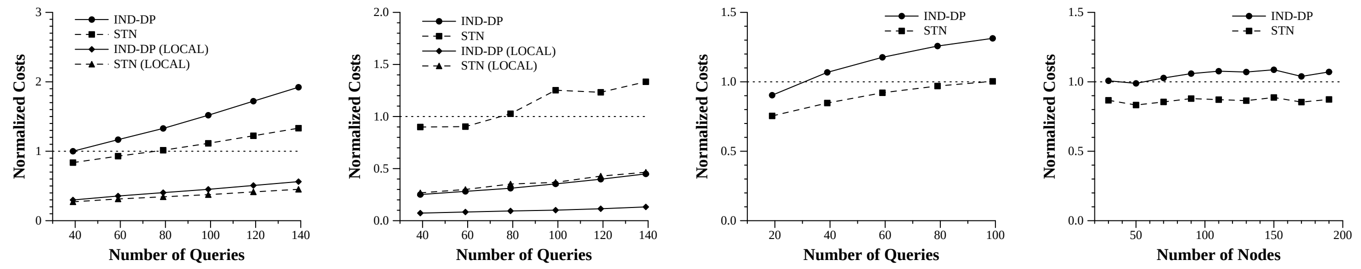
Figures 8 (iii) and (iv) show the results of experiments where the number of nodes in the network, and the max-query-size were varied, for Dataset 2. As we can see, HYPR continues to outperform both NAIVE and IND-DP by large margins across a range of values of the experimental parameters.

B. G_C is a not a Tree

In this case, we restrict ourselves to the case when all queries are of size 2. IND-DP once again optimizes each query optimally but independently from the other queries. We compare it against an approach that greedily chooses the largest *star* in the query overlap graph, and uses the Steiner tree-based algorithm (STN) presented in Section VI. We use the 11/6 approximation by Zelikovsky [29] for computing Steiner trees. We would like to note that this algorithm does not take the data item sizes into consideration, and hence is not expected to perform well for Dataset 2.

In Figures 9 (i) and (ii), we report the results for the two datasets, Dataset 1 and Dataset 2, and for the two query workloads. As we can see, for Dataset 1, STN always performs better than IND-DP for both query workloads, and both of them find much better solutions than NAIVE for the LOCAL query workload. However, both STN and IND-DP perform worse than NAIVE for the non-local query workload for larger numbers of queries. As expected, STN performs much worse than IND-DP and NAIVE for Dataset 2.

In Figure 9 (iii), we compare the performance of the three algorithms for the case when the query overlap graph is restricted to be a tree. Note that this limits the number of queries to 99 (= $number\text{-}of\text{-}data\text{-}sources - 1$). As we can see, STN performs significantly better than IND-DP, but approaches NAIVE as the number of queries approaches its upper limit. Finally in Figure 9 (iv), we report the performance of the algorithms as the number of data sources is varied. The number of queries was set to be half the number of data sources. As we can see, the comparative performance of the algorithms is quite consistent across a range of network sizes. We observed similar behavior for other parameter settings.



(i) Varying No. of Queries - Dataset 1 (ii) Varying No. of Queries - Dataset 2 (iii) Queries with Tree Overlap - Dataset 1 (iv) Varying No. of Nodes - Dataset 1

Fig. 9. Results for the general case - all queries are restricted to be over two sources each

Our experimental evaluation for the general case suggests that the best option might be to run all three algorithms, and take the best solution among those. Development of better algorithms, with guaranteed approximation ratios, is clearly an open and fertile area of further research.

VIII. Conclusions

In recent years we have seen a rise in distributed query processing driven by an increasing number of distributed monitoring and computing infrastructures. In many of these environments the communication cost forms the chief bottleneck. In environments such as sensor networks, the communication cost directly affects the energy consumption of the sensing devices and dictates the lifetime of the network. In Internet-scale environments such as scientific federations, the network bandwidth is the limiting factor. In this paper we addressed the problem of optimizing data movement when executing a large number of queries over distributed data sources. We presented a framework for analyzing this problem by showing the similarities between several variations of the problem. Our main contribution is a new algorithm for finding an optimal sharing plan when the communication is restricted to be along a tree. This algorithm also allows us to develop a $O(\log(n))$ approximation algorithm for general communication graphs. We also develop several approximation algorithms for special cases of the problem. Interestingly, even some very special cases correspond to well studied problems in the literature. Our preliminary experimental analysis shows that sharing of data movement is critical when executing a large number of queries over distributed data sources.

Our work has opened up many avenues for further research. Although we exploit sharing of base data sources, we do not consider sharing of intermediate results. Incorporating intermediate result sharing, load balancing, and join order optimization into our framework for multi-query optimization remains a rich area for further research. Our algorithms assume that the set of queries to be executed is provided as the input; in practice, we expect the queries to arrive one-by-one and we plan to address the issue of developing online algorithms to handle such scenarios.

References

[1] A. S. Szalay, J. Gray, A. R. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg, "The SDSS skyserver: public access to the Sloan digital sky server data," in *SIGMOD*, 2002, pp. 570–581.

[2] T. Malik, A. S. Szalay, T. Budavari, and A. Thakar, "Skyquery: A web service approach to federate databases," in *CIDR*, 2003.

[3] D. T. Liu and M. J. Franklin, "The design of griddb: A data-centric overlay for the scientific grid," in *VLDB*, 2004, pp. 600–611.

[4] X. Wang, R. Burns, A. Terzis, and A. Deshpande, "Network-aware join processing in global-scale database federations," in *ICDE*, 2008.

[5] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *ICDE*, 2006, p. 49.

[6] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable distributed stream processing," in *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[7] N. Trigoni, Y. Yao, A. J. Demers, J. Gehrke, and R. Rajaraman, "Multi-query optimization for sensor networks," in *DCOSS*, 2005, pp. 307–321.

[8] A. Silberstein and J. Yang, "Many-to-many aggregation for sensor networks," in *ICDE*, 2007.

[9] W. Hong and M. Stonebraker, "Optimization of parallel query execution plans in XPRS," *Distributed and Parallel Databases*, vol. 1(1), 1993.

[10] Q. Zhu, "Query optimization in multidatabase systems," in *CASCON '92: Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 1992, pp. 111–127.

[11] A. Deshpande and J. M. Hellerstein, "Decoupled query optimization for federated database systems," in *ICDE*, 2002, pp. 716–732.

[12] M. N. Garofalakis and Y. E. Ioannidis, "Parallel query scheduling and optimization with time- and space-shared resources," in *VLDB*, 1997.

[13] D. Kossman, "The state of the art in distributed query processing," *ACM Computing Surveys*, September 2000.

[14] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. R. Jr., "Query processing in a system for distributed databases (SDD-1)," *TODS*, vol. 6, no. 4, 1981.

[15] R. S. Epstein, M. Stonebraker, and E. Wong, "Distributed query processing in a relational data base system," in *SIGMOD*, 1978.

[16] C. Evrendilek, A. Dogac, S. Nural, and F. Ozcan, "Multidatabase query optimization," *Distributed and Parallel Databases*, 1997.

[17] S. Salza, G. Barone, and T. Morzy, "Distributed query optimization in loosely coupled multidatabase systems," in *ICDT*, 1995.

[18] C. T. Yu, Z. M. Ozsoyoglu, and K. Lam, "Optimization of distributed tree queries," *JCSS*, 1984.

[19] D. Shasha and T.-L. Wang, ""Optimizing Equijoin Queries in Distributed Databases Where Relations Are Hash Partitioned"," *TODS*, vol. 16, no. 2, pp. 279–308, June 1991.

[20] T. K. Sellis, "Multiple-query optimization," *ACM Trans. Database Syst.*, vol. 13, no. 1, pp. 23–52, 1988.

[21] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe, "Efficient and extensible algorithms for multi query optimization," in *SIGMOD*, 2000.

[22] S. Ganguly, W. Hasan, and R. Krishnamurthy, "Query optimization for parallel execution," in *SIGMOD*, 1992, pp. 9–18.

[23] W. Hasan and R. Motwani, "Coloring away communication in parallel query optimization," in *VLDB*, 1995.

[24] C. Chekuri, W. Hasan, and R. Motwani, "Scheduling problems in parallel query optimization," in *PODS*, 1995, pp. 255–265.

[25] Y. Huang and H. Garcia-Molina, "Publish/subscribe tree construction in wireless ad-hoc networks," in *MDM*, 2003.

[26] S. Banerjee, A. Misra, J. Yeo, and A. Agrawala, "Energy-efficient broadcast and multicast trees for reliable wireless communication," *Wireless Communications and Networking (WCNC)*, 2003.

[27] O. Papaemmanouil and U. Cetintemel, "Semcast: Semantic multicast for content-based data dissemination," in *ICDE*, 2005.
[28] G. Robins and A. Zelikovsky, "Improved Steiner tree approximation in graphs," in *SODA*, 2000.
[29] A. Zelikovsky, "An 11/6-approximation algorithm for the network steiner problem," *Algorithmica*, vol. 9, no. 5, pp. 463–470, 1993.
[30] F. Eisenbrand, F. Grandoni, T. Rothvoss, and G. Schafer, "Approximating connected facility loccation problems via random facility sampling and core detoruring," in *SODA*, 2008, pp. 1174–1183.
[31] A. Gupta, A. Kumar, and T. Roughgarden, "Simpler and better approximation algorithms for network design," in *STOC*, 2003, pp. 365–374.
[32] Y. Bartal, "On approximating arbitrary metrices by tree metrics," in *STOC*, 1998, pp. 161–168.
[33] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," in *STOC*, 2003, pp. 448–455.
[34] T. C. Hu and K. Moerder, "Multiterminal Flows in a Hypergraph," *VLSI Circuit Layout: Theory and Design*, pp. 87–93, 1985.
[35] V. Chvatal, *Linear Programming*. W. H. Freeman, 1983.
[36] D. West, *Introduction to Graph Theory (2nd Ed.)*. Prentice Hall, 2000.
[37] M. Garey and D. Johnson, *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W.H. Freeman, 1979.

APPENDIX

NP-HARDNESS FOR REPLICATED DATA SOURCES

The reduction is from the Set Cover problem [37]. In an instance of that problem, we are given a collection C of subsets of a finite set S, and we are asked to find the minimum cardinality set cover, i.e., a subset $C' \subseteq C$ such that every element in S belongs to at least one member of C'.

We construct an instance of our problem as follows. The communication network contains a special node r, and for every set $S_i \in C$, we add a node v_i and connect it to r with a unit-weight edge. Furthermore, for every set $S_i \in C$, for every element $s \in S_i$, we add a node and connect it to v_i using a zero-weight edge. Thus, for each element $s \in S$, we have as many nodes as the number of sets in C that contain s – all these nodes are assigned the same data item, denoted S_s. Finally, the root is assigned a data item S_r. All data items are of unit size.

We are asked to evaluate $|S|$ queries: $\{S_s, S_r\}$ for all $s \in S$.

It is easy to see that the optimal solution to this instance of our problem has the same cost as the optimal solution for the set cover problem.

PROOF OF LEMMA 5

Suppose v is the root of $\mathcal{H}$ and $v_1, v_2, \ldots, v_k$ are its children and v_{ij}s are v_i's children. Let $OPT = \{T^o_v, T^o_{v_1}, T^o_{v_2}, \ldots\}$ be an optimal solution and $SOL = \{T_v, T_{v_1}, T_{v_2}, \ldots\}$. Let the vertex J_i be the meeting point of T^o_v and $T^o_{v_i}$ for all i. Suppose $\sum_i dis(v_i, J_i) = \alpha OPT$ for some constant $0 \leq \alpha \leq 1$. It is easy to see that $T^o_v = OPT - \sum_i T^o_{v_i} \leq OPT - \sum_i dis(v_i, J_i) = (1-\alpha)OPT$. First, we can see our algorithm is a $(1+\alpha)\rho$-approximation algorithm. Let $SP(a, b)$ be the shortest path from a to b. Combining T_v and shortest paths $SP(J_i, v_i)$ gives us a Steiner tree (may have some cycles) spanning v and all v_is.

Let AS$(v, T_{v_1}, T_{v_2}, \ldots)$ be an approximate Steiner tree on the specified set of terminals and MS$(v, T_{v_1}, T_{v_2}, \ldots)$ be an optimal Steiner tree on the same set of terminals.

$$\begin{aligned} T_v = AS(v, T_{v_1}, T_{v_2}, \ldots) &\leq \rho \cdot MS(v, T_{v_1}, T_{v_2}, \ldots) \\ &\leq \rho \cdot MS(v, v_1, v_2, \ldots) \leq \rho \cdot (T^o_v + \sum_i dis(v_i, J_i)) \end{aligned}$$

So

$$\begin{aligned} SOL = T_v + \sum_i T_{v_i} &\leq \rho \cdot (T^o + \sum_i dis(v_i, v_j) + \sum_i T^o_{v_i}) \\ &= (1+\alpha)\rho \cdot OPT. \end{aligned}$$

W.l.o.g we can assume OPT has the following property: all leaves in $T^o_{v_i}$ are terminals (not Steiner points). It is easy to see that there always exists a terminal $s_i \in V(\mathcal{H})$ which is descendant of J_i in $T^o_{v_i}$(rooted at v_i). We also find $dis(s_i, J_i) + dis(J_i, v_i) \leq T^o_{v_i}$, thus $\sum_i dis(s_i, J_i) \leq \sum_i T^o_{v_i} - \sum_i dis(J_i, v_i) \leq (1-\alpha)OPT$. So,

$$MS(v, T_{v_1}, T_{v_2}, \ldots) \leq MS(v, s_1, s_2, \ldots) \leq T^o_v + \sum_i dis(J_i, s_i)$$

where the last inequality holds since T^o_v combined with all $SP(J_i, s_i)$s is a Steiner tree spanning v and all s_is. Now, we can show our final solution:

$$\begin{aligned} SOL = T_v + \sum_i T_{v_i} &= AS(v, T_{v_1}, \ldots) + \sum_i AS(v_i, v_{i1}, v_{i2}, \ldots) \\ &\leq \rho \cdot (MS(v, T_{v_1}, \ldots) + \sum_i MS(v_i, v_{i1}, v_{i2}, \ldots)) \\ &\leq \rho \cdot (T^o_v + (1-\alpha)OPT + \sum_i T^o_{v_i}) = \rho \cdot (2-\alpha) \cdot OPT. \end{aligned}$$

Thus we have:

$$SOL \leq \rho \cdot OPT \cdot \min(1+\alpha, 2-\alpha) \leq 1.5\rho \cdot OPT$$

■