Algorithms for a Topology-aware Massively Parallel Computation Model*

Xiao Hu
xh102@cs.duke.edu
Duke University

Paraschos Koutris
paris@cs.wisc.edu
UW-Madison

Spyros Blanas
blanas.2@osu.edu
The Ohio State University

ABSTRACT

Most of the prior work in massively parallel data processing assumes homogeneity, i.e., every computing unit has the same computational capability and can communicate with every other unit with the same latency and bandwidth. However, this strong assumption of a uniform topology rarely holds in practical settings, where computing units are connected through complex networks. To address this issue, Blanas et al. [9] recently proposed a topology-aware massively parallel computation model that integrates the network structure and heterogeneity in the modeling cost. The network is modeled as a directed graph, where each edge is associated with a cost function that depends on the data transferred between the two endpoints. The computation proceeds in synchronous rounds and the cost of each round is measured as the maximum cost over all the edges in the network.

In this work, we take the first step into investigating three fundamental data processing tasks in this topology-aware parallel model: set intersection, cartesian product, and sorting. We focus on network topologies that are tree topologies, and present both lower bounds as well as (asymptotically) matching upper bounds. Instead of assuming a worst-case distribution as in previous results, the optimality of our algorithms is with respect to the initial data distribution among the network nodes. Apart from the theoretical optimality of our results, our protocols are simple, use a constant number of rounds, and we believe can be implemented in practical settings as well.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; • **Theory of computation** → **Database query processing and optimization (theory)**.

KEYWORDS

query processing, massively parallel computation, topology-aware

*This research has been supported in part by NSF grants IIS-1814493, CCF-2007556, CRII-1850348, III-1910014 and CCF-1816577.

PODS '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8381-3/21/06…$15.00
https://doi.org/10.1145/3452021.3458318

ACM Reference Format:
Xiao Hu, Paraschos Koutris, and Spyros Blanas. 2021. Algorithms for a Topology-aware Massively Parallel Computation Model. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3452021.3458318

1 INTRODUCTION

The popularity of massively parallel data processing systems has led to an increased interest in studying the formal underpinnings of massively parallel models. As a simplification of the Bulk Synchronous Parallel (BSP) model [47], the Massively Parallel Computation (MPC) model [30] has enjoyed much success in studying algorithms for query evaluation [7, 8, 25–29, 48], as well as other fundamental data processing tasks [2–4, 6, 21, 22, 24]. In the MPC model, any pair of compute nodes in a cluster communicates via a point-to-point channel. Computation proceeds in synchronous rounds: at each round, all nodes first exchange messages and then perform computation on their local data.

Algorithms in the MPC model operate on a strong assumption of homogeneity: every compute node has the same data processing capability and communicates with every other node with the same latency and bandwidth. In practice, however, large deployments are heterogeneous in their computing capabilities, often consisting of different generations of CPUs and GPUs. In the cloud, the speed of communication differs based on whether the compute nodes are located within the same rack, across racks, or across datacenters. In addition to static effects from the network topology, a model needs to capture the dynamic effects of different algorithms that may cause network contention. This homogeneity assumption is not confined in the theoretical development of algorithms, but it is also used when deploying algorithms in the real world.

Recent work has started taking into account the impact of network topology for data processing. In the model proposed by Chattopadhyay et al. [11, 12], the underlying network is modeled as a graph, where nodes communicate with their neighbors through the connected edges. Computation proceeds in rounds. In each round, $\tilde{O}(1)$ bits[1] can be exchanged per edge. The complexity of algorithms in such a model is measured by the number of rounds. Using the same model, Langberg et al. [32] prove tight topology-sensitive bounds on the round complexity for computing functional aggregate queries. Although these algorithmic results have appealing theoretical guarantees, they are unrealistic starting points for implementation. As the number of rounds required is usually polynomial in terms of the data size, the synchronization cost would be extremely high in practice. In addition, the size of the data that can be exchanged per edge in each round is too small; the compute

[1]The notation $\tilde{O}$ hides a polylogarithmic factor on the input size.

nodes in today's mainstream parallel data processing systems can process gigabytes of data in each round.

Recently, Blanas et al. [9] proposed a new massively parallel data processing model that is aware of the network topology as well as network bandwidth. The underlying communication network is represented as a directed graph, where each edge is associated with a cost function that depends on the data transferred between the two endpoints. A subset of the nodes in the network consists of *compute nodes*, i.e., nodes that can store data and perform computation—the remaining nodes can only route data to the desired destination. Computation still proceeds in rounds: in each round, each compute node sends data to other compute nodes, receives data, and then performs local computation. There is no limit on the size of the data that can be transmitted per edge; the cost is defined as the sum across all rounds of the maximum cost over all edges in the network at each round. This model is general enough to capture the MPC model as a special case.

In this work, we use the above topology-aware model to prove lower bounds and design algorithms for three fundamental data processing tasks: *set intersection*, *cartesian product*, and *sorting*. These three tasks are the essential building blocks for evaluating any complex analytical query in a data processing system.

In contrast to prior work, which either assumes a worst-case or uniform initial data distribution over the nodes in the network, we study algorithms in a more fine-grained manner by assuming that the cardinality of the initial data placed at each node can be arbitrary and is known in advance. This information allows us to build more optimized algorithms that can take advantage of data placement to discover a more efficient communication pattern.

Our contributions. We summarize our algorithmic results in Table 1. Our results are restricted to network topologies that have two properties. First, they are *symmetric*, i.e., for each link (u, v) there exists a link (v, u) with the same bandwidth. Second, the network graph is a *tree*. Even with these two restrictions, we can capture several widely deployed topologies, such as star topologies and fat trees. All our algorithms are simple to describe and run either in a single round or in a constant number of rounds, hence requiring minimal synchronization. We thus believe that they form a good starting point for an efficient practical implementation. We next present our results for each data processing task in more detail.

- **Set Intersection (Section 3).** In this task, we want to compute the intersection $R \cap S$ of two sets. Our lower bound for set intersection uses classic results from communication complexity on the lopsided set disjointness problem. This lower bound has a rather complicated form (as shown in Section 3.1), since each link has a different data capacity budget depending on the underlying network as well as the initial data distribution. Since set intersection is a computation-light but communication-heavy task, the challenge is how to effectively route the data according to the capacity of each link. We design a single-round randomized routing strategy for set intersection that matches the lower bound with high probability, losing only a polylogarithmic factor (w.r.t. the input size and network size). Surprisingly, the routing depends only on the topology and initial data placement, but not the bandwidth of the links.
- **Cartesian Product (Section 4).** Here we want to compute the cross product $R \times S$ of two sets. This task is fundamental for various join operators, such as natural join, θ-join, similarity join and set containment join. We derive two lower bounds of different flavor. The first lower bound has a similar form as that for set intersection. The second lower bound uses instead a counting argument, which states that each pair in the cartesian product must be enumerated by at least one compute node, and the two elements participating in this result should reside on the same node when it is enumerated. We propose a one-round deterministic routing strategy for computing the cartesian product, which has asymptotically optimal guarantees. Our protocol generalizes the HyperCube algorithm that is used to compute the cartesian product in the MPC model [1].
- **Sorting (Section 5).** We define a valid ordering of compute nodes as any left-to-right traversal of the underlying network tree, after picking an arbitrary node as the root. If the ordering of compute nodes is $v_1, v_2, v_3, \ldots$, at the end of the algorithm all elements on node v_i are in sorted order and no larger than those on node v_j if $i < j$. Our lower bound again has a similar form to the one we derived for set intersection. We present a sampling-based sorting algorithm which runs in a constant number of rounds and matches our lower bound with high probability. The protocol is again independent of the topology and the bandwidth, and depends only on the initial placement of the data.

2 THE COMPUTATIONAL MODEL

In this section, we present the computational model we will use for this work.

Network Model. We model the network topology using a *directed* graph $G = (V, E)$. Each edge $e \in E$ represents a network link with bandwidth $w_e \geq 0$, where the direction of the edge captures the direction of the data flow. We distinguish a subset of nodes in the network, $V_C \subseteq V$, to be *compute* nodes. Compute nodes are the only nodes in the network that can store data and perform computation on their local data. Non-compute nodes can only route data. We only consider connected networks, where every pair of compute nodes is connected through a directed path.

Computation. A parallel algorithm $\mathbb{A}$ proceeds in sequential *rounds* (or *phases*). We denote by $r \in \mathbb{N}$ the number of rounds of the algorithm. In the beginning, each compute node $v \in V_C$ holds part of the input I, denoted as $X_0(v) \subseteq I$. In this work, we assume that $\{X_0(v)\}_{v \in V_C}$ forms a partition of the input I; in other words, there is no initial data duplication across the nodes. The goal of the algorithm is to compute a function over the input I, such that in the end the compute nodes together hold the function output.

We also assume that algorithm $\mathbb{A}$ has knowledge of the following: (i) the topology of the graph, (ii) the bandwidth of each link, and (iii) $|X_0(v)|$ for each compute node $v \in V_C$. In the case of relational data, we further assume that the algorithm knows the cardinality of the local fragment for each relation.

We use $X_i(v)$ to denote the data stored at compute node $v \in V_C$ after the i-th round completes, where $i = 1, 2, \ldots, r$. At every round, the compute nodes first perform some computation on their local data. Then, they communicate by sending data to other compute nodes in the network. We assume that for a data transfer from

Task	Algorithm	Number of Rounds	Lower Bounds	Optimality Guarantee
Set intersection	Randomized	1	Theorem 1	$O(\log \lvert V\rvert \log N)$ with high probability
Cartesian product	Deterministic	1	Theorem 7 and 9	$O(1)$
Sorting	Randomized	$O(1)$	Theorem 15	$O(1)$ with high probability

Table 1: A summary of our results. The graph network is $G = (V, E)$, while the size of the input data is denoted by N.

compute node u to compute node v, the algorithm must explicitly specify the routing path (or a collection of routing paths). We use $Y_i(e)$ to denote the data that is routed through link e during round i, and $|Y_i(e)|$ denote its total size measured in *bits*.

Cost Model. Since the algorithm proceeds in sequential rounds, we can decompose the cost of the algorithm, denoted $\text{cost}(\mathbb{A})$, as the sum of the costs for each round i,

$$\text{cost}(\mathbb{A}) = \sum_{i=1}^{r} \text{cost}_i(\mathbb{A})$$

The model captures the cost of each round by considering only the *cost of communication*. The cost of the i-th round is

$$\text{cost}_i(\mathbb{A}) := \max_{e \in E} |Y_i(e)|/w_e.$$

In other words, the cost of each round is captured by the cost of transferring data through the most bottlenecked link in the network. In some cases, it will be convenient to express the cost using tuples/elements instead of bits, which we will mention explicitly.

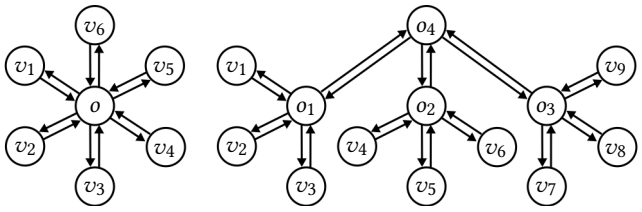
Even though the model does not take into account any computation time in the cost, it is possible to incorporate computation costs in the model by appropriately transforming the underlying graph – for more details, see [10]. We should note here that our model does not capture factors such as congestion on a router node, or communication delays due to large network diameter.

2.1 Network Topologies

Even though the model supports general network topologies, computer networks often have a specific structure. When the underlying topology has some structure, several problems (such as routing [5, 13, 14, 36]) admit more efficient solutions than what is achievable for general topologies. It is therefore natural to consider restrictions on the topology that are of either theoretical or practical interest.

Symmetric Network. Wired networks support full duplex operation that allows simultaneous communication in both directions of a link. Furthermore, datacenter networks allocate the same bandwidth for transmitting and receiving data for each node. These networks are represented in the model using a symmetric network. We say that a network topology is *symmetric* if for every edge $e = (u, v) \in E$, we also have that $e' = (v, u) \in E$ with $w_e = w_{e'}$. In other words, the cost of sending data from u to v is the same as the cost of sending the same data from v to u.

Star Topology. The most common topology for small clusters is the star topology, where all computers are connected to a single switch. A star network with $p + 1$ nodes has p compute nodes $V_C = \{v_1, v_2, \ldots, v_p\}$ that are all connected to a central node w that only does routing. Figure 1a depicts an example of a star network. Within a node, a multi-core CPU also exhibits a star topology: individual CPU cores exchange data through a shared cache and memory hierarchy, which implicitly forms the center of the star.



(a) Star topology. **(b) Tree topology.**

Figure 1: Common computer network topologies have structure, which permits more efficient solutions than what is feasible for arbitrary topologies.

Tree Topology. As the network grows, a single router is no longer sufficient to connect all nodes. A common solution to scale the network further is to arrange r routers $\{o_1, o_2, \cdots, o_r\}$ in a star topology, and connect p compute nodes $V_C = \{v_1, \cdots, v_p\}$ to individual routers. Figure 1b shows an example of a tree topology. A key property in a tree topology is that there exists a unique directed path between any two compute nodes, hence routing is trivial.

In this work, *we will focus on symmetric tree topologies*. We make two observations about such topologies:

- Assume w.l.o.g. that every compute node is a leaf. Indeed, if we have a non-leaf compute node $v \in V_C$, we can transform G to a new graph G' by adding a new compute node v', introduce a new link between v, v' with bandwidth $+\infty$, and make v a non-compute node.
- Assume w.l.o.g. that there are no nodes with degree 2. Indeed, consider a node v with two adjacent edges $e_1 = (v, u_1)$ and $e_2 = (v, u_2)$. We can remove v and replace the two edges with a single edge $e = (u_1, u_2)$ with bandwidth $\min\{w_{e_1}, w_{e_2}\}$.

2.2 Relation to the MPC Model

We discuss here how the topology-aware model can capture the MPC model [7, 31] as a special case.

Recall that in the MPC model we have a collection of p nodes. The MPC model is topology-agnostic: every machine can communicate with any other machine, and the cost of a round is defined as the maximum amount of data that is received during this round across all machines. The MPC model corresponds to an asymmetric star topology with p compute nodes. For every edge $e = (v_i, o)$ that goes from a compute node to the center o the bandwidth is $w_e = +\infty$, while for the inverse edge $e' = (o, v_i)$ the bandwidth is $w_{e'} = 1$.

It should be noted that all previous works using the MPC model assume a uniform data distribution, where each node initially receives N/p data, where N is the input size. This assumption has been used both for lower and upper bounds. In contrast, our algorithms and lower bounds take the sizes of the initial data distribution as parameters.

3 SET INTERSECTION

In the set intersection problem, we are given two sets R and S. Our goal is to enumerate all pairs $(r, s) \in R \cap S$. Note that there is no designated node for each output pair, as long as it is emitted by at least one node. We assume that all elements from both sets are drawn from the same domain.

Given an initial distribution $\mathcal{D}$ of the data across the compute nodes, we denote by $R_v^{\mathcal{D}}, S_v^{\mathcal{D}}$ the elements from R and S respectively in node v. Let $N_v^{\mathcal{D}} = |R_v^{\mathcal{D}}| + |S_v^{\mathcal{D}}|$, and $N^{\mathcal{D}} = \sum_v N_v = |R| + |S|$. Whenever the context is clear, we will drop $\mathcal{D}$ from the notation.

3.1 Lower Bound for Tree Topologies

We present a lower bound on the cost for the case of a symmetric tree topology. To prove the lower bound, we use a reduction from the *lopsided set disjointness* problem in communication complexity. In this problem, Alice holds a set X of n elements and Bob holds a set Y of m elements from some common domain. The goal is to decide whether the intersection $X \cap Y$ is empty by minimizing communication. It is known [19, 44] that for any multi-round randomized communication protocol, either Alice has to send $\Omega(n)$ bits to Bob, or Bob has to send $\Omega(m)$ bits to Alice.

To construct the reduction, we observe that any edge $e = (u, v)$ defines a partitioning of the compute nodes in the tree G into two subsets: V_e^- and V_e^+. Here, V_e^- is the set of compute nodes in the same side as u, and V_e^+ in the same side as v. Hence, any algorithm that computes the set intersection in the tree topology also solves a lopsided set disjointness problem, where Alice holds all data located in V_e^-, Bob holds all data located in V_e^+, and they can only communicate through the edge e. Following this core idea, we can show the following lower bound.

THEOREM 1. *Let $G = (V, E)$ be a symmetric tree topology. Any algorithm computing the intersection $R \cap S$ has cost $\Omega(C_{LB})$, where*

$$C_{LB} = \max_{e \in E} \frac{1}{w_e} \cdot \min\left\{ |R|, |S|, \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \right\}.$$

Observe that the above lower bound holds independent of the number of rounds that the algorithm uses.

PROOF. Consider an edge $e \in E$. Any algorithm that computes the set intersection $R \cap S$ must solve the following problem. Alice holds two sets, $R_A = \bigcup_{v \in V_e^-} R_v$, and $S_A = \bigcup_{v \in V_e^-} S_v$. Similarly, Bob holds two sets, $R_B = \bigcup_{v \in V_e^+} R_v$, and $S_B = \bigcup_{v \in V_e^+} S_v$. Then, Alice and Bob must together compute two set intersections, $R_A \cap S_B$ and $R_B \cap S_A$, communicating only through the link e with bandwidth w_e. The lower bound for lopsided disjointness tells us that in order to compute $R_A \cap S_B$ we need to communicate $\Omega(\min\{|R_A|, |S_B|\})$ bits, and for $R_B \cap S_A$ we need at least $\Omega(\min\{|R_B|, |S_A|\})$ bits. Hence, the cost of any algorithm must be $\Omega(C)$, where:

$$\begin{aligned} C &= \frac{1}{w_e} \max(\min\{|R_A|, |S_B|\}, \min\{|R_B|, |S_A|\}) \\ &\geq \frac{1}{2w_e} \min\{|R_A| + |R_B|, |S_A| + |S_B|, |R_A| + |S_A|, |R_B| + |S_B|\} \\ &= \frac{1}{2w_e} \min\left(|R|, |S|, \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \right) \end{aligned}$$

Applying the above argument to every edge in the tree G, we obtain the desired result. □

Algorithm 1: StarIntersect$(G, \mathcal{D})$

```
V_α ← {v ∈ V_C | min{N_v, N − N_v} < |R|}, V_β ← V_C \ V_α ;
for v ∈ V_C do
    send every a ∈ R_v^D to all nodes in V_β ∪ {h(a)} ;
    if v ∈ V_α then
        send every a ∈ S_v^D to h(a) ;
```

3.2 Warmup on Symmetric Star

We first consider the star topology to present some of the key ideas. W.l.o.g. we assume $|R| \leq |S|$. We present a one-round algorithm based on randomized hashing.

Our algorithm (Algorithm 1) in its core performs a randomized hash join. It first partitions the compute nodes into two subsets, V_α and V_β, depending on the size of the local data. Define $N' = |R| + \sum_{v \in V_\alpha} |S_v|$. Let h be a random hash function that maps independently each a in the domain to node $v \in V_C$ with the following probability:

$$Pr[h(a) = v] = \begin{cases} N_v/N', & v \in V_\alpha \\ |R_v|/N', & v \in V_\beta \end{cases}$$

If $V_\beta = \emptyset$, then the algorithm performs a distributed hash join using the above hash function h. Observe that the algorithm does not hash each value uniformly across the compute nodes, but with probability proportional to the input data N_v that each node holds.

If $V_\beta \neq \emptyset$, we perform hashing only on a subset of the data using a subset of the nodes. In particular, each node $v \in V_\beta$ first gathers all the elements from R (the smaller relation) and locally computes $R \cap S_v$, while hashing is used to compute the remaining set intersection. After the data is communicated, the intersection can be computed locally at each node.

We next show that the above algorithm is optimal within a polylogarithmic factor.

LEMMA 2. *Let $G = (V, E)$ be a symmetric star topology, and consider sets R, S with $N = |R| + |S|$. Then,* StarIntersect *computes the set intersection $R \cap S$ with cost $O(\log N \log |V|)$ away from the optimal solution with high probability.*

PROOF. The correctness of the algorithm is straightforward. We will next bound the cost of the algorithm. We will measure the cost using elements of the set; to translate to bits it suffices to add a $\log(N)$ factor which captures the number of bits necessary to represent each element.

To make the notation simpler, we will use w_v to refer to the bandwidth w_e of edge $e = (v, w)$, where $v \in V_C$ and w is the central node of the star topology. We can now reformulate the lower bound from Theorem 1 as

$$C_{LB} = \max\left\{\max_{v\in V_\alpha} \frac{\min\{N_v, N - N_v\}}{w_v}, \max_{v\in V_\beta} \frac{|R|}{w_v}\right\}$$

We now distinguish two cases, depending on whether the edge is adjacent to a node in V_α or V_β.

Case 1: $v \in V_\beta$**.** Consider the two edges (v, w) and (w, v). The number of tuples that will be sent through edge (v, w) is $|R_v| \le |R|$. As for the tuples received, node v will receive $|R| - |R_v|$ tuples from R, as well as some tuples from S which are in expectation: $\frac{|R_v|}{N'} \cdot \sum_{v\in V_\alpha} |S_v| \le |R_v|$. Thus, the cost incurred by edges adjacent to V_β is: $\max_{v\in V_\beta} \frac{|R|}{w_v} \le C_{LB}$. Even though the above analysis just bounds the expectation, we can use Chernoff bounds to show that with probability polynomially small in the number of compute nodes, the number of tuples will not exceed the expectation by more than an $O(\log |V|)$ factor for any of the edges.

Case 2: $v \in V_\alpha$**.** We will bound separately the number of R-tuples and S-tuples that go through each edge.

The expected number of S-tuples that go through edge (w, v) is

$$\left(\sum_{u\in V_\alpha} |S_u| - |S_v|\right) \cdot \frac{N_v}{N'} \le (N' - R - |S_v|) \cdot \frac{N_v}{N'}$$
$$\le \frac{(N' - N_v)N_v}{N'} \le \min\{N_v, N - N_v\}$$

The third inequality is a direct application of the facts that $\min\{a, b\} \ge \frac{a\cdot b}{a+b}$ for any $a, b \ge 0$ and $N' < N$. Similarly, the expected number of S-tuples that go through edge (v, w) is

$$|S_v| \cdot \frac{N' - N_v}{N'} \le \frac{(N' - N_v)N_v}{N'} \le \min\{N_v, N - N_v\}$$

For R-tuples, we distinguish two cases. If $V_\beta = \emptyset$, then we can bound the expected size using the same argument as above for S-tuples. We now turn to the case where $V_\beta \ne \emptyset$. We first claim that $N_v \le N - N_v$ for each vertex $v \in V_\alpha$. Indeed, if not then we must have that $N - N_v < |R|$, which implies that $N_v > |S|$. However, this is a contradiction since there exists $u \in V_\beta$ with $N_u > |R|$. Hence, it suffices to bound the R-tuples that go through each edge by N_v. Indeed, the number of R-tuples that go through (v, w) for $v \in V_\alpha$ are at most $|R_v| \le N_v$. As for the edge (w, v), the expected number of tuples that use the edge is:

$$(|R| - |R_v|) \cdot \frac{N_v}{N'} \le \frac{|R|}{N'} \cdot N_v \le N_v$$

Combining these two cases above yields the desired claim. Note that all expectation calculations can be extended to high probability statements by losing a factor of $O(\log |V|)$ as mentioned before. □

3.3 Algorithm on General Symmetric Tree

We now generalize the algorithm for the star topology to an arbitrary (symmetric) tree topology. W.l.o.g. we assume $|R| \le |S|$. We

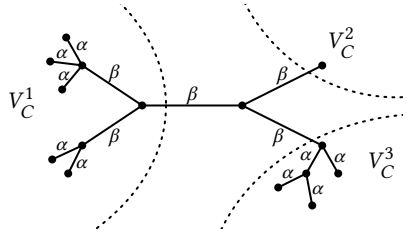


Figure 2: An illustration of a balanced partition.

partition all edges in E into two subsets:

$$E_\alpha = \{e \in E \mid \min\{\sum_{v\in V_e^+} N_v, \sum_{v\in V_e^-} N_v\} < |R|\}$$
$$E_\beta = \{e \in E \mid \min\{\sum_{v\in V_e^+} N_v, \sum_{v\in V_e^-} N_v\} \ge |R|\}$$

An edge $e \in E$ is called α-edge if $e \in E_\alpha$, and β-edge if $e \in E_\beta$. Observe that the definition is symmetric w.r.t. the direction of the edge: if (u, v) is an α-edge, so is (v, u). The intuition behind this partition lies in the lower bound of Theorem 1, where the amount of data that can go through an α-edge is $O(\min\{\sum_{v\in V_e^+} N_v, \sum_{v\in V_e^-} N_v\})$ and through a β-edge is $O(|R|)$. We denote by G_β the edge-induced subgraph of the edge set E_β.

Lemma 3. *The subgraph G_β is a connected tree.*

Proof. For the sake of contradiction, assume there exist vertices $u, v \in V(G_\beta)$ such that u, v are not connected in G_β. Then, there exists an α-edge e on the unique path that connects u and v in G. In turn, e splits G into two connected subtrees: G_e^+ (that contains all nodes in V_e^+), and G_e^- (that contains all nodes in V_e^-). Suppose w.l.o.g. that $u \in V(G_e^+)$ and $v \in V(G_e^-)$.

Since u, v belong in the edge-induced subgraph of E_β, there exists β-edges $e_1 \in G_e^+, e_2 \in G_e^-$. We observe that $V_{e_1}^+ \subseteq V_e^+$ and $V_{e_2}^- \subseteq V_e^-$, which implies $|R| \le \sum_{v\in V_e^+} N_v$ and $|R| \le \sum_{v\in V_e^-} N_v$. In this way, e would be an β-edge, contradicting our assumption. □

On the other hand, the edge-induced subgraph G_α derived from E_α is not necessarily connected and forms a forest.

Balanced Partition. The first step of our algorithm is to compute a partition $\{V_C^1, V_C^2, \cdots, V_C^k\}$ of the compute nodes V_C. In particular, the algorithm seeks a *balanced partition*, as illustrated in Figure 2.

Definition 4. A partition $\{V_C^1, V_C^2, \cdots, V_C^k\}$ of V_C is balanced for data distribution $\mathcal{D}$ if the following properties hold:

(1) If two nodes are connected in G_a, they belong in the same block of the partition ;
(2) Each edge appears in the spanning tree of at most one block of the partition ;
(3) For every block i, $\sum_{v\in V_C^i} N_v^{\mathcal{D}} \ge |R|$;
(4) For every β-edge e in the spanning tree of a block i, we have $\min\{\sum_{v\in V_C^i\cap V_e^+} N_v, \sum_{v\in V_C^i\cap V_e^-} N_v\} \le |R|$.

Before we show how to find a balanced partition, we first discuss how we can use it to compute the set intersection.

Algorithm 2: TreeIntersect$(G, \mathcal{D})$

```
Find a balanced partition {V_C^1, V_C^2, ..., V_C^k};
for v ∈ V_C do
    for i = 1, ..., k do
        send every a ∈ R_v^D to h^i(a);
        if v ∈ V_C^i then
            send every a ∈ S_v^D to h^i(a);
```

The Algorithm. Let $\{V_C^1, V_C^2, \cdots, V_C^k\}$ be a balanced partition of the compute nodes V_C. For every block V_C^i, we define a random hash function h^i that maps independently each value a in the domain to node $v \in V_C^i$ with probability:

$$Pr[h^i(a) = v] = \frac{N_v}{\sum_{u \in V_C^i} N_u}$$

Using the above probabilities, we can now describe the detailed algorithm (Algorithm 2), which works in a single round. Each R-tuple is hashed across all blocks of the partition (hence it may be replicated), while each S-tuple is hashed only in the block that contains the node it belongs in. After all data is communicated, each node locally computes the set intersection.

THEOREM 5. *On a symmetric tree topology $G = (V, E)$, the set intersection $R \cap S$ with $|R| + |S| = N$ can be computed in a single round with cost $O(\log N \log |V|)$ away from the optimal solution with high probability.*

PROOF. The correctness of the algorithm comes from the fact that each subset of nodes V_C^i computes $R \cap \bigcup_{v \in V_C^i} S_v$. Since $S = \bigcup_{i=1}^k \bigcup_{v \in V_C^i} S_v$, the algorithm computes all results in $R \cap S$.

We next analyze the cost. As before, we will measure the cost in number of tuples, and then pay a $O(\log N)$ factor to translate to bits. We first rewrite the lower bound as:

$$C_{LB} = \max\left\{ \max_{e \in E_\alpha} \frac{1}{w_e} \min\{ \sum_{v \in V_e^+} N_v, \sum_{v \in V_e^-} N_v \}, \max_{e \in E_\beta} \frac{|R|}{w_e} \right\}$$

We analyze the cost for the edges in E_α, E_β separately.

Case: $e \in E_\beta$**.** We will bound the amount of data that goes through e by $O(|R|)$. The R-tuples that go through e are at most $|R|$, so it suffices to bound the number of S-tuples that cross edge e. By property (2) of a balanced partition, e is included in at most one spanning tree, say of block V_C^i. Then, w.h.p. the expected amount of S-tuples that goes through e is at most

$$\frac{1}{\sum_{v \in V_C^i} N_v} \cdot \Big(\sum_{v \in V_C^i \cap V_e^-} N_v \Big) \cdot \Big(\sum_{v \in V_C^i \cap V_e^+} N_v \Big)$$
$$\leq \min\{ \sum_{v \in V_C^i \cap V_e^-} N_v, \sum_{v \in V_C^i \cap V_e^+} N_v \} \leq |R|$$

The first inequality comes from the fact that $\frac{a \cdot b}{a+b} \leq \min\{a, b\}$ for any $a, b > 0$. The second inequality is implied directly by property (4) of a balanced partition.

Case: $e \in E_\alpha$**.** We will bound the amount of data that goes through e by $\min\left\{\sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v\right\}$. To bound the number of S-tuples, we again notice that e can belong in the spanning tree of at most one block, say V_C^i. Hence, as in the previous case, w.h.p. the expected amount of S-tuples that goes through e is at most

$$\frac{1}{\sum_{v \in V_C^i} N_v} \cdot \Big(\sum_{v \in V_C^i \cap V_e^-} N_v \Big) \cdot \Big(\sum_{v \in V_C^i \cap V_e^+} N_v \Big)$$
$$\leq \min\{ \sum_{v \in V_C^i \cap V_e^-} N_v, \sum_{v \in V_C^i \cap V_e^+} N_v \} \leq \min\{ \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \}$$

We can bound the number of R-tuples that go through e by distinguishing three cases:

- none of G_e^-, G_e^+ contain β-edges. Then, the partition consists of a single block, and the number of R-tuples can be bounded as we did above with the S-tuples.
- G_e^+ contains β-edges but G_e^- not. Then, all vertices in G_β are in V_e^+. The R-data that goes through e is sent by nodes in V_e^-, so its size is bounded by

$$\sum_{v \in V_e^-} |R_v| \leq \sum_{v \in V_e^-} N_v = \min\left\{ \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \right\}.$$

  Here, the last equality follows from the fact that G_e^+ contains at least one β-edge, which implies $\sum_{v \in V_e^+} N_v \geq |R| > \sum_{v \in V_e^-} N_v$.
- G_e^- contains β-edges but G_e^+ not. Then, all nodes in V_e^+ belong in the same block V_C^i. We can abound the expected amount of S-tuples with:

$$\frac{1}{\sum_{v \in V_C^i} N_v} \cdot \Big(\sum_{v \in V_e^-} |R_v| \Big) \cdot \Big(\sum_{v \in V_C^i \cap V_e^+} N_v \Big)$$
$$\leq \frac{\sum_{v \in V_e^-} |R_v| + \sum_{v \in V_C^i \cap V_e^+} N_v}{\sum_{v \in V_C^i} N_v} \min\{ \sum_{v \in V_e^-} |R_v|, \sum_{v \in V_C^i \cap V_e^+} N_v \}$$
$$\leq \frac{|R| + \sum_{v \in V_C^i} N_v}{\sum_{v \in V_C^i} N_v} \min\{ \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \}$$
$$\leq 2 \min\{ \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \}$$

  where the last inequality is from property (3) of Definition 4.

This completes the proof. □

Finding a Balanced Partition. Finally, we present how we can compute a balanced partition in Algorithm 3. Two vertices in G are α-connected if there exists a path that uses only α-edges that connects them. For the algorithm below, denote $\Gamma(x)$ as the set of nodes that are α-connected with node x in G. Moreover, we use $w(x)$ to denote the quantity $\sum_{x \in \Gamma(x)} N_x$, i.e. the total amount of data in the nodes from $\Gamma(x)$. The algorithm initially creates a group for each set of compute nodes that are connected through α-edges. Then, it starts merging the groups (starting from the leaves of the tree) as long as the total number of the elements in the group is less than $|R|$. We show (in Appendix A.1) that the above algorithm indeed creates the desired balanced partition.

Algorithm 3: BalancedPartition($G, \mathcal{D}$)

```
for x ∈ V(G_β) do
    Γ(x) ← {v ∈ V_C | v, x are α-connected in G} ;
P ← ∅ ;
while |V(G_β)| > 0 do
    pick the leaf vertex x ∈ G_β with the smallest w(x);
    if w(x) ≥ |R| then
        add Γ(x) to P;
    else
        y ← unique neighbor of x in G_β;
        Γ(y) ← Γ(y) ∪ Γ(x);
    G_β ← G_β \ {x};
return P ;
```

Lemma 6. *Algorithm 3 outputs a balanced partition of compute nodes V_C in $O(|V|)$ time.*

Remark. Interestingly, the algorithm we described above does not use the link bandwidths to decide what to send and where to send to. Instead, what matters is the connectivity of the network and how the data is initially partitioned across the compute nodes. This is a significant practical advantage because bandwidth information may be imprecise or have high variability at runtime, such as when sharing a cluster with other users.

4 CARTESIAN PRODUCT

In the cartesian product problem, we are given two sets R, S with $|R| = |S| = N/2$. (We will discuss in the end why the unequal case is challenging, even on the simple symmetric star topology). Our goal is to enumerate all pairs (r, s) for any $r \in R, s \in S$, such that the output pairs are distributed among the compute nodes by the end of the algorithm. Similar to set intersection, there is no designated node for each output pair, as long as it is emitted by at least one node. We assume that all elements are drawn from the same domain, and that initially the input data is partitioned across the compute nodes.

4.1 Lower Bounds on Symmetric Trees

We present two lower bounds on cost for the case of a symmetric tree topology. The first one as stated in Theorem 7 has the same form as the one in Theorem 1 when $|R| = |S| = N/2$, but uses a slightly different argument. Both lower bounds are expressed in terms of elements, and not bits.

Theorem 7. *Let $G = (V, E)$ be a symmetric tree topology. Any algorithm computing $R \times S$ has (tuple) cost $\Omega(C_{LB})$, where*

$$C_{LB} = \max_{e \in E} \frac{1}{w_e} \cdot \min\left\{ \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \right\}.$$

Proof. Let C_{opt} be the cost of any algorithm computing $R \times S$ on the tree topology G. Consider an edge $e \in E$. Suppose that $C_{opt} \cdot w_e \le \sum_{v \in V_e^-} |R_v|$. Then, at least one element in R_u for some $u \in V_e^-$ does not go through e, i.e., entering into any vertex in V_e^+. In this case, in order to guarantee correctness, all data in S must be sent to u, hence $C_{opt} \cdot w_e \ge \sum_{v \in V_e^+} |S_v|$. Thus $C_{opt} \cdot w_e \ge \min\{\sum_{v \in V_e^-} |R_v|, \sum_{v \in V_e^+} |S_v|\}$. Using a symmetric argument, $C_{opt} \cdot w_e \ge \min\{\sum_{v \in V_e^-} |S_v|, \sum_{v \in V_e^+} |R_v|\}$. Summing up the two inequalities, and observing that $\min\{\sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v\} \le |R| (= |S| = N/2)$, we obtain the lower bound on edge e.

Applying the above argument to every edge in the tree G, we obtain the desired result. □

The second lower bound uses a different argument that depends on the underlying tree topology. To state the lower bound, we first define a "directed" version $G^\dagger$ of the symmetric tree G as follows. $G^\dagger$ has the same vertex set as G. Recall that each edge $e = (u, v)$ in G partitions the nodes of V into V_e^+ and V_e^-. Then, if $\sum_{x \in V_e^-} N_x \le \sum_{x \in V_e^+} N_x$, $G^\dagger$ contains only an edge from u to v, otherwise only an edge from v to u. As the next lemma shows, the resulting directed graph $G^\dagger$ has a very specific structure.

Lemma 8. *$G^\dagger$ satisfies the following properties:*

(1) The out-degree of every node is at most one.

(2) There exists exactly one node with out-degree zero.

Proof. By contradiction, assume there exists one node $u \in V$ with at least two out-going edges. Since G has no vertices with degree 2, this means that $G^\dagger$ has three edges $e_1 = (u, v_1)$, $e_2 = (u, v_2)$, $e_3 = (u, v_3)$. For each such edge, we have $\sum_{x \in V_{e_i}^+} N_x \ge \sum_{x \in V_{e_i}^-} N_x$, and thus $\sum_{x \in V_{e_i}^+} N_x \ge N/2$. Observe that because G is a tree, it also holds that the vertex sets $V_{e_i}^+$ are disjoint. Then we come to the contradiction that $N = \sum_{x \in V} N_x \ge \sum_{i=1}^3 \sum_{x \in V_{e_i}^+} N_x \ge \frac{3}{2} N$, thus (1) is proved.

Since $G^\dagger$ is a directed tree, it is easy to see that there must exist at least one node with no outgoing edges; otherwise, there would be a cycle in the graph, a contradiction. Hence, it suffices to show that there is at most one such a node. By contradiction, assume two nodes u, v with out-degree 0. Consider the unique path between u, v: then, there must be a node in the path with out-degree at least two. However, this contradicts (1), thus (2) is proven as well. □

We denote the single node with out-degree zero as r, and call it the *root* of the tree. Every other node in $G^\dagger$ will point towards r, as the example in Figure 3 illustrates. Observe that the root r of the tree could be a compute node. But in this case, the algorithm that simply routes all the data to the root is asymptotically optimal, since the cost matches the lower bound in Theorem 7. Hence, we will focus on the case where the root is not a compute node; in this case, it is easy to observe that all the nodes in $G^\dagger$ with in-degree 0 are exactly the compute nodes.

A *cover* of $G^\dagger$ is a subset $S \subseteq V$ such that every leaf node has some ancestor in S. We will be interested in *minimal covers* of $G^\dagger$. Observe that the singleton set $\{r\}$ is trivially a minimal cover.

Theorem 9. *Let $G = (V, E)$ be a symmetric tree topology. Let U be a minimal cover of $G^\dagger$ such that $U \ne \{r\}$, where r is the root of $G^\dagger$. Then, any algorithm computing the cartesian product $R \times S$ for $|R| = |S| = N/2$ has (tuple) cost $\Omega(C_{LB})$, where*

$$C_{LB} = \frac{N}{\sqrt{\sum_{v \in U} w_v^2}},$$

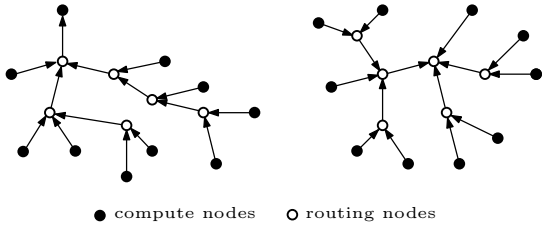


Figure 3: Two examples of a directed graph $G_\dagger$.

where w_v is the capacity of the unique outgoing edge of v in $G^\dagger$.

Proof. Let e_u be the outgoing edge of $u \in U$ in $G^\dagger$, with capacity cost w_u. Let T_u be the subtree rooted at u. From minimality of U, it follows that T_u, T_v have disjoint vertex sets. Moreover, from the definition of a node cover, every compute node belongs in some (unique) subtree. This means that we can bound the output result by at most the union of the outputs in the compute nodes of each subtree. In the following, we will bound the maximum output size of a given subtree T_u.

Assume R'_u, S'_u be the elements of R, S respectively that are in some compute node of T_u. Moreover, assume R''_u, S''_u be the elements of R, S that go through link e_u respectively. Then, the size of the results that can be produced at subtree T_u is at most $|R'_u \cup R''_u| \cdot |S'_u \cup S''_u|$. Observe the following:

- $|R''_u| \le C_{opt} \cdot w_u$ and $|S''_u| \le C_{opt} \cdot w_u$;
- $|R'_u| \le C_{opt} \cdot w_u$ and $|S'_u| \le C_{opt} \cdot w_u$. Indeed, since w_u is an outgoing edge of u in $G^\dagger$, Theorem 7 tells us that $C_{opt} \cdot w_u \ge |R'_u| + |S'_u|$.

Hence, we can bound the number of outputs in T_u as:

$$\begin{aligned} |R'_u \cup R''_u| \cdot |S'_u \cup S''_u| &\le (|R'_u| + |R''_u|)(|S'_u| + |S''_u|) \\ &\le (2 \cdot C_{opt} \cdot w_u)(2 \cdot C_{opt} \cdot w_u) \\ &= 4 \cdot C_{opt}^2 \cdot w_u^2 \end{aligned}$$

To guarantee the correctness, the total size of the output must be at least $|R| \cdot |S|$. Summing over all nodes in the minimal cover U, we obtain $|R| \cdot |S| \le 4 \cdot C_{opt}^2 \cdot \sum_{u \in U} w_u^2$. This concludes the proof. □

4.2 The Weighted HyperCube Algorithm

In this section, we present a deterministic one-round protocol on a symmetric star topology, named *weighted HyperCube* (wHC), which generalizes the HyperCube algorithm [1].

The wHC Algorithm. We assume that the data statistics $|R_v|$, $|S_v|$ are known to all compute nodes. We give a strict ordering $\le$ on the compute nodes in V_C. Each node assigns consecutive numbers to its local data. More specifically, node v labels its data in R_v from $1 + \sum_{u<v} |R_u|$ to $\sum_{u \le v} |R_u|$, and data in S_v from $1 + \sum_{u<v} |S_u|$ to $\sum_{u \le v} |S_u|$. In this way, each element from R is labeled with a unique index, as well as each one from S. In this way, each answer in the cartesian product can be uniquely mapped to a point in the grid $\square = \{1, 2, \ldots, |R|\} \times \{1, 2, \ldots, |S|\}$.

The wHC protocol assigns to each compute node v a square $\square_v$ centered at (x_v, y_v) with dimensions $l_v \times l_v$. Then, a tuple $r_i \in R$ will be sent to v if $x_v - l_v \le i \le x_v + l_v$, and a tuple $s_j \in S$ will be sent to v if $y_v - l_v \le j \le y_v + l_v$. After all tuples are routed, the cartesian product will be computed locally at each compute node. To guarantee correctness, we have to make sure that $\bigcup_v \square_v = \square$, i.e., the squares assigned to each node fully cover the grid.

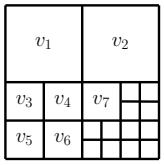


Figure 4: An example of packing squares.

We first compute the dimensions l_v of the square assigned to each node. Intuitively, we want to make sure that l_v is proportional to the capacity of the link. However, to make sure that we can *pack* the resulting set of squares without any overlap, we consider squares that are powers of 2. Specifically,

$$l_v = \arg\min_k \{2^k \ge w_v \cdot L\}, \qquad L = \frac{N}{\sqrt{\sum_u w_u^2}} \tag{1}$$

Second, we need to specify the positions of the squares, i.e. determine how they can be *packed* without any overlap. An example of such a packing is given in Figure 4. To pack the squares, we will make use of the following lemma.

Lemma 10 (Packing Squares). *Let S be a set of squares $d_i \times d_i$, where each d_i is a power of two. Then, we can pack the squares in S such that they fully cover a square of size at least $\sqrt{\sum_i d_i^2}/2$.*

Proof. We provide an algorithm for the packing. We start the following procedure in an increasing order of $i \ge 0$: for each i, if there are 4 squares of size $2^i \times 2^i$ in S, we pack them into a larger square of size $2^{i+1} \times 2^{i+1}$. In this way, we can transform S into a new set of squares S', where for every i, there are at most 3 squares of size $2^i \times 2^i$. It is now easy to see that, by induction starting from the smaller size, all squares of size $\le 2^{i-1}$ can be packed inside a square of size 2^i. Hence, we can pack all squares in S' inside a square of size 2^{i^*+1}, where 2^{i^*} is the dimension of the largest square in S'. To conclude the argument, observe that the square with dimension 2^{i^*} is fully packed. Also, $2^{i^*+1} \ge \sqrt{\sum_i d_i^2}$. Hence, we can fully pack a square of size at least $\sqrt{\sum_i d_i^2}/2$. □

The next lemma bounds the cost of the wHC algorithm.

Lemma 11. *Let G be a symmetric star topology. Then, the wHC algorithm correctly computes the cartesian product $R \times S$ for $|R| = |S| = N/2$ with (tuple) cost $O(C)$, where*

$$C = \max\left\{ \max_v \frac{N_v}{w_v}, \frac{N}{\sqrt{\sum_v w_v^2}} \right\}$$

Proof. We apply Lemma 10 with $S = \{l_v \times l_v \mid v \in V_C\}$ to show the correctness. The squares fully pack a square of area at least

$$\frac{1}{4} \sum_{v \in V_C} (2^{l_v})^2 \ge \frac{1}{4} \sum_{v \in V_C} (L \cdot w_v)^2 = (N/2)^2 = |R| \cdot |S|$$

Algorithm 4: StarCartesianProduct($G, \mathcal{D}$)

1 **if** $\max_u N_u > N/2$ **then**
2 all compute nodes send their data to $\arg\max_u N_u$;
3 **else** run the wHC algorithm ;

Hence, the whole grid can be covered.

Next, we analyze the cost of the algorithm. First, the cost of sending data is $\max_v N_v/w_v$. For the cost of receiving, observe that node v receives at most $2 \cdot (2L \cdot w_v) = 4w_vL$ tuples. Hence, the cost of receiving is bounded by $4L$. Combining these two costs obtains the desired result. □

4.3 Warm-up on Symmetric Star

Before we present the general algorithm for symmetric trees, we warm up by studying the simpler symmetric star case (Algorithm 4).

The algorithm checks whether the maximum data that some node holds exceeds $N/2$. If so, it is easy to observe that the strategy where every compute node sends their data to that node is optimal. If every node holds at most $N/2$ data initially, then in $G^\dagger$ all compute nodes of the star are directed to the central node o, which becomes the root of $G^\dagger$. In this case, running the wHC algorithm on the whole topology can be proven optimal.

Lemma 12. *On a symmetric star topology, Algorithm 4 correctly computes the cartesian product $R \times S$ for $|R| = |S| = \frac{N}{2}$ in a single round deterministically and with cost $O(1)$ away from the optimal.*

Proof. We distinguish the analysis into two cases, depending on whether $\max_u N_u > N/2$ or not. Let C_{opt} be the cost of any algorithm computing $R \times S$ on the tree topology G.

First, suppose that $\max_u N_u > N/2$. Let $u^* = \arg\max_u N_u$. For node u^*, $N - N_{u^*} < N/2 < N_{u^*}$. For every other node $v \neq u^*$, $N_v < N/2$, hence $N_v < N - N_v$. Then, we can write Theorem 7 as:

$$C_{opt} \geq \max_v \frac{\min\{N_v, N - N_v\}}{2w_v} \geq \max\left\{\frac{N - N_{u^*}}{2w_{u^*}}, \max_{v \neq u^*} \frac{N_v}{2w_v}\right\}$$

But this is exactly half the cost of the protocol where all nodes send their data to u^*.

Suppose now that $\max_u N_u \leq N/2$. From Theorem 7, we obtain the lower bound $C_{opt} \geq \max_v \frac{N_v}{2w_v}$. Additionally, observe that in $G^\dagger$ all compute nodes of the star are directed to the central node o, and hence V_C is a minimal cover of $G^\dagger$. Indeed, if we add $\{o\}$ to V_C, the cover is not minimal, since $\{o\}$ is a minimal cover by itself. Plugging this cover in Theorem 9, we obtain $C_{opt} \geq N/\sqrt{\sum_v w_v^2}$. To conclude, notice that these two lower bounds on C_{opt} match the upper bound of wHC in Lemma 11 within a constant factor. □

4.4 Algorithm on Symmetric Tree

We now generalize the techniques for the star topology to an arbitrary tree topology.

The Algorithm. Assume that the data statistics $|R_v|$, $|S_v|$ are known to all compute nodes. Similar to the wHC algorithm, each tuple from R is labeled with a unique index, as well as each one from S. In this way, each answer in the cartesian product can be uniquely mapped to a point in the grid $\square = \{1, \ldots, |R|\} \times \{1, \ldots, |S|\}$. Let

Algorithm 5: BalancedPackingTree(G)

1 **forall** $v \in V \setminus \{r\}$ *in post-order* **do**
2 **if** v *is a leaf* **then** $\tilde{w}_v \leftarrow w_v$;
3 **else** $\tilde{w}_v \leftarrow \min\{w_v, \sqrt{\sum_{u \in \zeta(v)} \tilde{w}_u^2}\}$;
4 $\tilde{w}_r \leftarrow \sqrt{\sum_{u \in \zeta(r)} \tilde{w}_u^2}$, $l_r \leftarrow 1$;
5 **forall** $v \in V \setminus \{r\}$ *in pre-order* **do**
6 $l_v \leftarrow l_{p_v} \cdot \tilde{w}_v / \sqrt{\sum_{u \in \zeta(p_v)} \tilde{w}_u^2}$;
7 **forall** $v \in V_C$ **do**
8 $d_v \leftarrow \arg\min_k \{2^k \geq N \cdot l_v\}$;
9 assign to v a square of sizes $d_v \times d_v$;

r be the root of the directed graph $G^\dagger$. For simplicity, we split the routing phase into two steps.

In the first step, each compute node $v \in V_C$ sends its local data to r. In the second step, we assign to each compute node $v \in V_C$ a square $\square_v$ such that every result $t = (t_r, t_s)$ is computed on some v. To compute t, associated tuples t_r, t_s will be sent to v at least once. In this step, every tuple sent to v will be sent from the root r, which has gathered all necessary data in the first step. Next, we show how to find a balanced assignment on a tree and analyze its capacity cost with respect to the lower bound in Theorem 9.

Balanced Packing on Symmetric Tree. Let $\zeta(u)$ be the set of children nodes of u in $G^\dagger$, and p_u the unique parent of u in $G^\dagger$. To simplify notation, we use w_v to denote the quantity $w(v, p_v)$.

The algorithm is split into two phases. First, it computes a quantity $\tilde{w}_v$ for each node v in $G^\dagger$. For the leaf nodes, we have $\tilde{w}_v = w_v$, while for the internal nodes $\tilde{w}_v$ is computed in a bottom-up fashion (through a post-order traversal). In the second phase, the algorithm computes a quantity l_v for each node, but now in a top-down fashion (through a pre-order traversal). As a final step, each compute node v rounds up $(N/2) \cdot l_v$ to the closest power of 2, and then gets assigned a square of that dimension.

The next lemma shows that Algorithm 5 guarantees certain properties for the computed quantities.

Lemma 13. *The following properties hold:*

(1) For every non-root vertex v, $\tilde{w}_v \leq w_v$.
(2) For every vertex v, $l_v \leq \tilde{w}_v/\tilde{w}_r$.
(3) There is a minimal cover U of $G^\dagger$ such that $\tilde{w}_r = \sqrt{\sum_{u \in U} w_u^2}$.
(4) For every vertex u, $l_u = \sqrt{\sum_{v \in T_u \cap V_C} l_v^2}$ where T_u is the subtree rooted at u.

Proof. Property (1) is straightforward from the algorithm.

We prove property (2) by induction. For the base case, v is the root. In this case, $l_r = 1$, so the inequality holds with equality. Consider now any non-root vertex v with parent p_v. We then have:

$$l_v = \frac{l_{p_v} \cdot \tilde{w}_v}{\sqrt{\sum_{u \in \zeta(p_v)} \tilde{w}_u^2}} \leq \frac{\tilde{w}_v}{\tilde{w}_r} \cdot \frac{\tilde{w}_{p_v}}{\sqrt{\sum_{u \in \zeta(p_v)} \tilde{w}_u^2}} \leq \frac{\tilde{w}_v}{\tilde{w}_r}$$

The first inequality holds from the inductive hypothesis for the parent node p_v. The second inequality comes from line 3 of the

algorithm, which implies that $\tilde{w}_v \le \sqrt{\sum_{u\in\zeta(v)} \tilde{w}_u^2}$ for every non-leaf vertex v.

We also use induction to show property (3). For a subtree rooted at leaf node v, $U = \{v\}$ is a minimal cover. In this case, $\tilde{w}_v = w_v = \sqrt{\sum_{u\in U} w_u^2}$. For the induction step, consider some non-leaf node v. If $\tilde{w}_v = w_v$, then $\tilde{w}_v = \sqrt{\sum_{u\in U} w_u^2}$ holds for the minimal cover $U = \{v\}$. Otherwise, $\tilde{w}_v = \sqrt{\sum_{u\in\zeta(v)} \tilde{w}_u^2}$. From the induction hypothesis, there exists a minimal cover U_u for the subtree rooted at $u \in \zeta(v)$ such that $\tilde{w}_u^2 = \sum_{t\in U_u} w_t^2$. Moreover, it is easy to see that the set $U = \bigcup_{u\in\zeta(v)} U_u$ is a minimal cover for the subtree rooted at v. Hence, we can write:

$$\tilde{w}_v = \sqrt{\sum_{u\in\zeta(v)} \tilde{w}_u^2} = \sqrt{\sum_{u\in\zeta(v)} \sum_{t\in U_u} w_t^2} = \sqrt{\sum_{t\in U} w_t^2}$$

The property (4) directly follows the Algorithm 5. The base case for $u \in V_C$ always holds. Consider any non-leaf node u. By induction, assume $l_x = \sqrt{\sum_{v\in T_x\cap V_C} l_v^2}$ for each node $x \in \zeta(u)$. Implied by line 6 in Algorithm 5, we have

$$l_u = \sqrt{\sum_{x\in\zeta(u)} l_x^2} = \sqrt{\sum_{x\in\zeta(u)} \sum_{v\in T_x\cap V_C} l_v^2} = \sqrt{\sum_{v\in T_u\cap V_C} l_v^2}.$$

This concludes our proof. □

It still remains to specify the position in the grid for each square assigned to a compute node.

Packing squares. In this part, we discuss how we can pack each square of dimension d_v assigned to leaf node v inside □. Our goal is to find an assignment (packing) of each square to compute nodes V_C such that for each vertex u, the number of elements that cross the link (u, p_u) is bounded by $O(N \cdot l_u)$.

We visit all vertices in bottom-up way, starting from the leaves. We recursively assign to each node v a set of squares in the form of $S_v = \{(2^i, c_i) : c_i \in \{0, 1, 2, 3\}\}$, meaning that there are c_i squares of dimensions $2^i \times 2^i$.

For every leaf node $v \in V_C$, only one square is assigned to v by Algorithm 5. Consider some non-leaf node u. Each of its children $v \in \zeta(u)$ is assigned with a set of squares S_v. We start the following procedure in an increasing order of $i \ge 0$: for each i, if there are 4 squares of size $2^i \times 2^i$ in $\bigcup_{v\in\zeta(u)} S_v$, we pack them into a larger square of size $2^{i+1}\times 2^{i+1}$. In this way, we can transform $\bigcup_{v\in\zeta(u)} S_v$ into a new set of squares S_u, where for every i, there are at most $c_i \le 3$ squares of dimensions $2^i \times 2^i$.

Next we bound the number of elements that cross the link (u, p_u) for each node $u \in V$, which is assigned with the set of squares S_u. Let T_u be the subtree rooted u. Let i^* be the largest integer such that $c_{i^*} \ne 0$. Note that each square of dimensions $2^i \times 2^i$ includes 2^i elements from both R and S. Then, the total number of elements for all squares in S_u is $\sum_i c_i \cdot 2 \cdot 2^i \le 2 \cdot (c_{i^*} + 1) \cdot 2^{i^*} \le 8 \cdot 2^{i^*}$, which can be further bounded by

$$\le 8 \cdot \sqrt{\sum_{v\in T_u\cap V_C} d_v^2} \le 16 \cdot N \cdot \sqrt{\sum_{v\in T_u\cap V_C} l_v^2} = 16 \cdot N \cdot l_u$$

The second inequality is implied by Algorithm 5, while the third inequality comes from the fact that $d_v \le 2N \cdot l_v$ for each compute node $v \in V_C$. The last equality is implied by Lemma 13.

THEOREM 14. *On a symmetric tree topology $G = (V, E)$, the cartesian product $R \times S$ for $|R| = |S| = N/2$ can be computed deterministically in a single round optimally.*

PROOF. To prove the correctness of the algorithm, we need to show that the packing of the squares fully covers the $|R| \times |S|$ grid. Indeed, consider the largest square $2^{i^*} \times 2^{i*}$ that occurs in the set of squares S_r assigned to the root node. Observe first that we can pack all squares in S_r inside a $2^{i^*+1} \times 2^{i^*+1}$ square, and thus

$$2^{2(i^*+1)} \ge \sum_v d_v^2 \ge N^2 \sum_{v\in V_C} l_v^2 = N^2$$

Hence, $2^{2i^*} \ge (N/2) \cdot (N/2) = |R| \cdot |S|$, which means that the grid is fully packed by the largest square in S_r.

We next show that the cost is asymptotically close to the lower bounds in Theorem 7 and Theorem 9. It can be easily checked that the number of elements transmitted through any link e at the first step is at most $O\left(\min\{\sum_{v\in V_e^-} N_v, \sum_{v\in V_e^+} N_v\}\right)$, matching the lower bound in Theorem 7. For the second step, we have bounded the number of elements that cross link (u, p_u) by $O(N \cdot l_v)$. Lemma 13 implies that $N \cdot l_v \le N \cdot w_v / \sqrt{\sum_{u\in U} w_u^2}$ for some minimal cover U of $G^\dagger$, hence matching the lower bound in Theorem 9. □

4.5 Discussion on Unequal Case

At last, we discuss the difficulty of computing the cartesian product $R \times S$ with $|R| \ne |S|$ on a symmetric star topology. W.l.o.g., assume $|R| < |S|$. The first lower bound following the same arguement in Theorem 7 is $\Omega(C_{LB})$ where

$$C_{LB} = \max_{v\in V_C} \frac{1}{w_v} \cdot \min\left\{N_v, N - N_v, |R|\right\}$$

We next see how the counting argument yields the second lower bound under the condition $\max_v N_v < \frac{N}{2}$. Let C be the cost of any correct algorithm. Assume R'_u, S'_u are the sets of elements from R, S received by u. Then, the size of results that can be produced at u is $|R_u \cup R'_u| \cdot |S_u \cup S'_u|$. Observe the following:

- $|R'_u| \le C_{opt} \cdot w_u$ and $|S'_u| \le C_{opt} \cdot w_u$;
- If $N_u < |R|$, $|R_u \cup R'_u| \le 2C_{opt} \cdot w_u$ and $|S_u \cup S'_u| \le 2C_{opt} \cdot w_u$;
- If $N_u \ge |R|$, $C_{opt} \cdot w_u \ge |R|$.

Summing over all node, the total size of the output must be at least $|R| \cdot |S|$. We then obtain

$$\begin{aligned} |R| \cdot |S| &\le \sum_{u\in V_C} (|R'_u| + |R''_u|)(|S'_u| + |S''_u|) \\ &\le \sum_{u\in V_C: N_u < |R|} 2\min\{C \cdot w_v, |R|\} \cdot 2\min\{C \cdot w_v, |S|\} \\ &\quad + \sum_{u\in V_C: N_u \ge |R|} |R| \cdot \{C \cdot w_v + S_u, |S|\} \end{aligned}$$

whose minimizer gives the second lower bound, which becomes rather complicated without a clean form as Theorem 9.

This is just an intuition of why the unequal case would make the lower bound hard even on the symmetric star. In Appendix A.3,

we give a more detailed analysis on the lower bound, as well as an optimal algorithm. Extending our current result to the general symmetric tree topology is left as future work.

5 SORTING

In the sorting problem, we are given a set R whose elements are drawn from a totally ordered domain. We first define an ordering of compute nodes in the following way: after picking an arbitrary node as the root, any left-to-right traversal of the underlying network tree is a valid ordering of compute nodes. The goal is to redistribute the elements of R such that on an ordering of compute nodes as $v_1, v_2, \cdots, v_{|V_C|}$, elements on node v_i are always smaller than those on node v_j if $i < j$.

Given an initial distribution $\mathcal{D}$ of the data across the compute nodes, we denote by $N_v^{\mathcal{D}}$ the initial data size in node v. Whenever the context is clear, we drop the superscript $\mathcal{D}$ from the notation.

5.1 Lower Bound

Our lower bound for sorting has the same form as the one for set intersection (Theorem 1), with the only difference that the cost is expressed as tuples, and not bits. Recall that any edge $e = (u, v)$ in the tree topology G defines a partitioning of the compute nodes in the tree G into two subsets: V_e^- and V_e^+. Here, V_e^- is the set of compute nodes in the same side as u, and V_e^+ in the same side as v. Hence, any sorting algorithm in the tree topology also performs some necessary comparisons between data located at V_e^- and V_e^+. Following this core idea, we can show the following lower bound.

THEOREM 15. *Let $G = (V, E)$ be a symmetric tree topology. Any algorithm sorting elements in a set R has (tuple) cost $\Omega(C_{LB})$, where*

$$C_{LB} = \max_{e \in E} \frac{1}{w_e} \cdot \min\left\{ \sum_{v \in V_e^-} N_v, \sum_{v \in V_e^+} N_v \right\}.$$

PROOF. We construct an initial data distribution R as follows. Assume elements in R are ordered as $r_1, r_2, \cdots, r_N$. W.l.o.g., assume N is even. We assign elements to compute nodes in the ordering of $\{r_1, r_3, \cdots, r_{N-1}, r_2, r_4, \cdots, r_N\}$. Moreover, we pick one arbitrary node of G as the root, where all compute nodes are leaves of the tree. All compute nodes in V_C are also labeled as $v_1, v_2, \cdots, v_{|V_C|}$ in an left-to-right traversal ordering. For example, node v_1 with initial data size N_1 will be assigned with $\{r_1, r_3, \cdots, r_{2N_1-1}\}$ if $N_1 \le \frac{N}{2}$, and $\{r_1, r_3, \cdots, r_{N-1}, r_2, r_4, \cdots, r_{2N_1-N}\}$ otherwise. We will argue that any algorithm correctly sorting R must incur a cost of $\Omega(C_{LB})$.

Consider an arbitrary edge $e \in E$ with the partition V_e^-, V_e^+. Denote $R_e^- = \bigcup_{v \in V_e^-} R_v$ and $R_e^+ = \bigcup_{v \in V_e^+} R_v$. It should be noted that R_e^- or R_e^+ is a sub-interval of $\{r_1, r_3, \cdots, r_{N-1}, r_2, r_4, \cdots, r_N\}$, or a sub-interval of $\{r_2, r_4, \cdots, r_N, r_1, r_3, \cdots, r_{N_1}\}$. Every element transmitted between V_e^- and V_e^+ must go through edge e. W.l.o.g., assume $|R_e^-| \le \frac{N}{2} \le |R_e^+|$. It suffices to show that the total number of elements exchanged between V_e^- and V_e^+ is at least $\Omega(|R_e^-|)$.

We start with the case when $|R_e^-| = 1$, say $R_e^- = \{r_i\}$. If r_i is not sent through e, at least one element in R_e^+ must be sent to V_e^-; otherwise, no comparison between r_i and elements in R_e^+ is performed, contradicting to the correctness of algorithms. In general, $|R_e^-| \ge 2$. We further distinguish four cases: (1) $r_2 \notin R_e^-, r_N \notin R_e^-$; (2) $r_1 \notin R_e^-, r_{N-1} \notin R_e^-$; (3) $r_2 \in R_e^-, r_{N-1} \in R_e^-$; (4) $r_1 \in R_e^-, r_N \in R_e^-$. Figure 5 illustrates the data exchange between V_e^- and V_e^+.

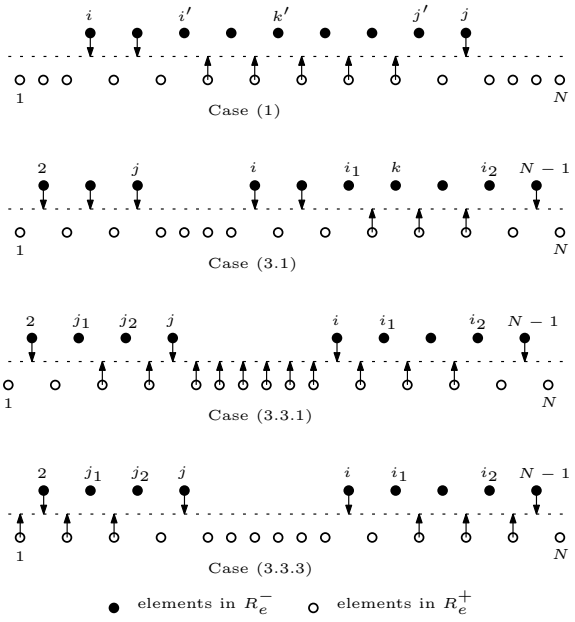


Figure 5: Data exchange between V_e^-, V_e^+.

Case (1): $r_2 \notin R_e^-, r_N \notin R_e^-$. In this case, $R_e^- \subseteq \{r_1, r_3, \cdots, r_{N-1}\}$. Let i, j be the smallest and largest index of elements in R_e^-. If all elements in R_e^- have been sent from V_e^- to V_e^+, then we are done. Otherwise, let i', j' be the smallest and largest of elements in R_e^- which are not sent from V_e^- to V_e^+. Furthermore, if all elements in $R_e^- - \{r_{i'}, r_{j'}\}$ are sent from V_e^- to V_e^+, it can be easily checked that the number of such elements is at least $\frac{|R_e^-|}{2}$. Otherwise, there is $r_{k'} \in R_e^- - \{r_{i'}, r_{j'}\}$ not sent from V_e^- to V_e^+. By the definition, $r_{i'} < r_{k'} < r_{j'}$. Implied by the ordering of compute nodes, all elements in $[r_{i'}, r_{j'}]$ should reside on V_e^- when the algorithm terminates. In this case, each element in $[r_{i'}, r_{j'}] - R_e^-$ should be sent from V_e^+ to V_e^-, and each in $\{r_i, r_{i+2}, \cdots, r_{i'-2}\} \cup \{r_{j'+2}, r_{j'+4}, \cdots, r_j\}$ are sent from V_e^- to V_e^+, as illustrated in Figure 5. So the number of elements transmitted through edge e is at least $\frac{i'-i}{2} + \frac{j-j'}{2} + \frac{j'-i'}{2} = \frac{j-i}{2} \ge |R_e^-| - 1 \ge \frac{|R_e^-|}{2}$. **Case (2)** is symmetric with Case (1).

Case (3): $r_{N-1} \in R_e^-, r_2 \in R_e^-$. Let i be the smallest odd index and j be the largest even index of elements in R_e^-. Note that $j < i$ since $|R_e^-| \le \frac{N}{2}$. We further consider three more cases as below.

Case (3.1): all elements in $\{r_2, r_4, \cdots, r_j\}$ are sent from V_e^- to V_e^+. If all elements in $\{r_i, r_{i+2}, \cdots, r_{N-1}\}$ are also sent from V_e^- to V_e^+, then we are done. Otherwise, let i_1, i_2 be the smallest and largest index of elements in $\{r_i, r_{i+2}, \cdots, r_{N-1}\}$ not sent from V_e^- to V_e^+. Furthermore, if all elements in $\{r_i, r_{i+2}, \cdots, r_{N-1}\} - \{r_{i_1}, r_{i_2}\}$ are sent from V_e^- to V_e^+, it can be easily checked that the number of elements sent from V_e^- to V_e^+ is at least $\frac{|R_e^-|}{2}$. Otherwise, there is $r_{k'} \in \{r_i, r_{i+2}, \cdots, r_{N-1}\} - \{r_{i_1}, r_{i_2}\}$ not sent from V_e^- to V_e^+. By the definition, $r_{i_1} < r_k < r_{i_2}$. Implied by the ordering of compute nodes, all elements in $[r_{i_1}, r_{i_2}]$ should reside on V_e^- when the algorithm terminates. In this case, each element in $[r_{i_1}, r_{i_2}] - R_e^-$ should be sent

from V_e^+ to V_e^-, and each in $\{r_2, r_4, \cdots, r_j\} \cup \{r_i, r_{i+2}, \cdots, r_{i_1-2}\} \cup \{r_{i_2+2}, r_{i_2+4}, \cdots, r_{N-1}\}$ are sent from V_e^- to V_e^+, as illustrated in Figure 5. So the number of elements transmitted through edge e is at least $\frac{j}{2} + \frac{i_1-i}{2} + \frac{N-1-i_2}{2} + \frac{i_2-i_1}{2} = \frac{N-1+j-i}{2} = |R_e^-| - 1 \geq \frac{|R_e^-|}{2}$.

Case (3.2): all elements in $\{r_i, r_{i+2}, \cdots, r_{N-1}\}$ are sent from V_e^- to V_e^+, which is symmetric with Case (3.1).

Case (3.3): at least one element in $\{r_2, r_4, \cdots, r_j\}$ and one element in $\{r_i, r_{i+2}, \cdots, r_{N-1}\}$ are not sent from V_e^- to V_e^+. Let j_1, j_2 be the smallest, largest even index of elements in R_e^- not sent from V_e^- to V_e^+. Let i_1, i_2 be the smallest, largest odd index of elements in R_e^- not sent from V_e^- to V_e^+. Then, each element in $\{r_2, r_4, \cdots, r_{j_1-2}\} \cup \{r_{j_2+2}, r_{j_2+4}, \cdots, r_j\} \cup \{r_i, r_{i+2}, \cdots, r_{i_2-2}\} \cup \{r_{i_2+2}, r_{i_2+4}, \cdots, r_{N-1}\}$ is sent from V_e^- to V_e^+. Implied by the ordering of compute nodes, (3.3.1) elements in $[r_{j_1}, r_{i_2}]$ or (3.3.2) elements in $[r_1, r_{j_2}] \cup [r_{i_1}, r_N]$ should reside on V_e^- when the algorithm terminates. In (3.3.1), each element in $[r_{j_1}, r_{i_2}] - R_e^-$ should be sent from V_e^+ to V_e^-, thus the number of elements transmitted over e is at least $i_2 - j_1 + 1 - \frac{j-j_1}{2} - \frac{i_2-i}{2} + \frac{j_1-2}{2} + \frac{j-j_2}{2} + \frac{i_1-i}{2} + \frac{N-1-i_2}{2} \geq \frac{N-1+i_1-j_2}{2} \geq \frac{N}{2}$. In (3.3.2), each element in $\{r_1, r_3, \cdots, r_{j_2-1}\} \cup \{r_{i_1+1}, r_{i_1+3}, \cdots, r_{N-1}\}$ should be sent from V_e^+ to V_e^-, thus the number of elements transmitted over e is at least $\frac{j_2+1}{2} + \frac{N-i_1+1}{2} + \frac{j_1-2}{2} + \frac{j-j_2}{2} + \frac{i_1-i}{2} + \frac{N-1-i_2}{2} = \frac{N-1-i_2+j_1}{2} \geq \frac{N}{2}$.

Case (4) is symmetric with Case (3). □

5.2 A Sampling-based Algorithm

In the MPC model, the theoretically optimal sorting algorithm inherited from [23] is rather complicated. Instead, sampling-based techniques, such as TeraSort [42], are more amenable to be extended to more complex networks. In this section, we present a randomized communication protocol for a symmetric tree topology, named *weighted TeraSort* (wTS), which generalizes the TeraSort algorithm in three fundamental ways. First, TeraSort is designed for the MapReduce [20] framework, which is an instantiation of the theoretical MPC model (with star topology), and we extend it to the general tree topology. Second, not all nodes participate in the splitting of the data, but only the ones that initially have a substantial amount of data. Third, we do not split the data uniformly, but proportionally to the size of the initial data. Before introducing our algorithm, we revisit the TeraSort algorithm.

TeraSort Algorithm. It first picks an arbitrary node as the coordinator. Set $\rho = 4 \cdot \frac{|V_C|}{N} \ln(|V_C| \cdot N)$.

- **Round 1.** Each node $u \in V_C$ samples each element from its local data with uniform probability ρ, and sends all samples to the coordinator. Let s be the number of samples generated in total.
- **Round 2.** The coordinator sorts all sampled elements received. Let b_i be the $i \cdot \lceil \frac{s}{|V_C|} \rceil$-th smallest object in the sorted samples for $i \in \{1, 2, \cdots, |V_C| - 1\}$, $b_0 = -\infty$ and $b_{|V_C|} = +\infty$. It then broadcasts $|V_C| + 1$ splitters $b_0, b_1, \cdots, b_{|V_C|}$ to all nodes.
- **Round 3.** Upon receiving all splitters, each node scans it own elements. For each element x, the node finds the two consecutive splitters b_i and b_{i+1} such that $b_i \leq x < b_{i+1}$ and then sends x to v_{i+1}. Finally, each node locally sorts all elements received.

Algorithm 6: PROPORTIONAL(V_H, u)

```
Δ ← 0, i ← 1;
while i ≤ k do
    x ← (N_{v_i} / Σ_{v∈V_H} N_v) · N_u;
    if Δ ≥ x − ⌊x⌋ then
        N_u^i ← ⌊x⌋,   Δ ← Δ − (x − ⌊x⌋);
    else
        N_u^i ← ⌊x⌋ + 1,   Δ ← Δ + 1 − (x − ⌊x⌋);
    i ← i + 1;
return N_u^1, N_u^2, ··· , N_u^k;
```

The wTS Algorithm. Now we describe our algorithm. Assume that the data statistics N_v's are known to all compute nodes. A compute node $v \in V_C$ is *heavy* if $N_v \geq |V_C|$ and *light* otherwise. Let $V_H, V_L \subseteq V_C$ be the set of heavy and light compute nodes respectively. For simplicity, we pick an arbitrary non-compute node as the root and label heavy nodes in V_H from left to right as $v_1, v_2, \cdots, v_k$.

- **Round 1.** Each light node $u \in V_L$ sends its local data to heavy nodes proportional to N_{v_i}'s. More specifically, node u sends N_u^i local elements to v_i for each $i \in \{1, 2, \cdots, k\}$, where N_u^i is computed by Algorithm 6. Let M_j be the number of elements residing on heavy node v_j after this round.
- **Round 2.** Each heavy node samples each element from its local storage with the same uniform probability ρ independently and then sends the sampled elements to v_1. Let s be the number of samples generated in total.
- **Round 3.** Node v_1 sorts all samples received. Let t_i be the $i \cdot \lceil \frac{s}{|V_C|} \rceil$-th smallest element among all samples. Let $c_j = \lceil \frac{|V_C|}{N} \cdot M_j \rceil$. It chooses $k + 1$ splitters as follows: (1) $b_0 = -\infty$; (2) $b_i = t_j$ where $j = c_1 + c_2 + \cdots + c_i$; (3) $b_k = +\infty$. Then, v_1 broadcasts $b_0, b_1, \cdots, b_k$ to the remaining heavy compute nodes.
- **Round 4.** Upon receiving all splitters, each heavy node scans its own data. For each element x, the node finds the two consecutive splitters b_i and b_{i+1} such that $b_i \leq x < b_{i+1}$ and then sends x to v_{i+1}. Finally, each node locally sorts all elements received.

We point out some properties of Algorithm 6 in Lemma 16, whose proof is in Appendix A.2. Intuitively, (1) and (2) give bounds on the size of data redistribution for each light node; and (3) guarantees that all data of each light node will be sent to heavy nodes.

LEMMA 16. *The following holds for any light node $u \in V_L$:*

(1) for any $i \in [k]$, $\sum_{j=1}^{i} N_u^j - 1 \leq \frac{\sum_{j=1}^{i} N_{v_j}}{\sum_{j=1}^{k} N_{v_j}} \cdot N_u \leq \sum_{j=1}^{i} N_u^j$;

(2) for any $i_1, i_2 \in [k]$ with $i_1 < i_2$, $\sum_{j=i_1}^{i_2} N_u^j \leq \frac{\sum_{j=i_1}^{i_2} N_{v_j}}{\sum_{j=1}^{k} N_{v_j}} \cdot N_u + 1$.

(3) $\sum_{j=1}^{k} N_u^j \geq N_u$.

THEOREM 17. *Let $G = (V, E)$ be a symmetric tree topology and R be an ordered set of N elements. If $N \geq 4|V_C|^2 \cdot \ln(|V_C| \cdot N)$, with probability $1 - \frac{1}{N}$, the wST algorithm sorts R in 4 rounds with cost $O(1)$ away from the optimal.*

The proof of Theorem 17 is given as below. A possible improvement is that if the maximum data that some node holds exceeds

$N/2$, every node just sends their data to that node. Otherwise, we simply run the wTS algorithm on the whole topology.

Proof. First, at least half the data is distributed across heavy nodes initially, i.e., $\sum_{j=1}^{k} N_{v_j} \geq \frac{N}{2}$. Indeed, the size of initial data distributed across all light node is strictly smaller than $\frac{N}{2|V_C|} \cdot |V_C| = \frac{N}{2}$, so the remaining data with size at least $\frac{N}{2}$ must reside on heavy nodes. We next analyze the cost for each round separately.

Round 1. Consider an arbitrary edge $e \in E$, which defines a partition of compute nodes V_e^-, V_e^+. If $V_H \cap V_e^+ \neq \emptyset$, it holds that $V_H \cap V_e^+ = \{v_i, v_{i+1}, \cdots, v_j\}$ or $\{v_1, v_2, \cdots, v_i\} \cup \{v_j, v_{j+1}, \cdots, v_k\}$ for some $i, j \in [k]$ and $i \leq j$. For any light node $u \in V_L$, the number of data sent to the nodes in $V_H \cap V_e^+$ can then be bounded as follows using Lemma 16(2):

$$\sum_{v \in V_e^+ \cap V_H} N_u^v \leq 2 + \sum_{v \in V_e^+ \cap V_H} \frac{N_v}{\sum_{v' \in V_H} N_{v'}} \cdot N_u$$

In this way, the number of data sent from light nodes in V_e^- to heavy nodes in V_e^+ can be bounded as

$$\begin{aligned}
&\sum_{u \in V_e^- \cap V_L} \left(2 + \sum_{v \in V_e^+ \cap V_H} \frac{N_v}{\sum_{v' \in V_H} N_{v'}} \cdot N_u\right) \\
&\leq \sum_{u \in V_e^- \cap V_L} 2 + \sum_{u \in V_e^- \cap V_L} \sum_{v \in V_e^+ \cap V_H} \frac{2N_v}{N} \cdot N_u \\
&\leq 2\min\{\sum_{u \in V_e^-} N_u, |V_C|\} + \frac{2}{N} \cdot (\sum_{u \in V_e^-} N_u) \cdot (\sum_{v \in V_e^+} N_v) \\
&\leq 4\min\{\sum_{u \in V_e^-} N_u, \sum_{v \in V_e^+} N_v\}
\end{aligned}$$

The rationale behind the third inequality is that $|V_C| \leq \frac{N}{2|V_C|} \leq \sum_{v \in V_e^+ \cap V_H} N_v \leq \sum_{v \in V_e^+} N_v$ and $\frac{a \cdot b}{a+b} \leq \min\{a, b\}$ holds for any $a, b \geq 1$. If $V_H \cap V_e^- \neq \emptyset$, we can make a symmetric argument.

We observe here that the number of data received by any heavy node $v \in V_H$ in round 1 is at most

$$\begin{aligned}
\sum_{u \in V_L} \lceil \frac{N_v}{\sum_{v' \in V_H} N_{v'}} \cdot N_u \rceil &= \sum_{u \in V_L} \frac{N_v}{\sum_{v' \in V_H} N_{v'}} \cdot N_u + \sum_{u \in V_L} 1 \\
&\leq \frac{2N_v}{N} \cdot \sum_{u \in V_L} N_u + |V_C| \leq 3N_v
\end{aligned}$$

where the rationale behind the first inequality is that $\sum_{v' \in V_H} N_{v'} \geq \frac{N}{2}$ and that behind the second inequality is that $|V_C| \leq \frac{N}{2|V_C|} \leq N_v$. Hence, for every heavy node v, $M_v \leq 3N_v + N_v = 4N_v$.

Rounds 2 and 3. During sampling, each element is an independent Bernoulli sample, so we have $E[s] = \rho N$. Applying the Chernoff bound, $\Pr[s \geq 2\rho N] \leq \exp(-\Omega(\rho N))$. In round 2 and round 3, the number of elements received or sent by any node is at most s, which is smaller than $2\rho N$ with probability at least $1 - \exp(-\Omega(\rho N)) \geq 1 - (\frac{1}{|V_C| \cdot N})^{4|V_C|}$. Observe that $2\rho N \leq N/|V_C|$. Since there is a heavy node at each side of an edge that has data getting through, we have $2\rho N \leq \min\{\sum_{u \in V_e^-} N_u, \sum_{v \in V_e^+} N_v\}$.

Round 4. In this round, each heavy node v_i sends out at most M_i elements and receives all the elements falling into the interval $[b_i, b_{i+1})$. Let $t_0 = -\infty$ and $t_{|V_C|} = +\infty$. Under the condition that $s \leq 2\rho N$, we first observe that for any $j \in \{1, 2, \cdots, |V_C|\}$, $|R \cap [t_{j-1}, t_j)| \leq 8 \cdot \frac{N}{|V_C|}$, which holds with probability at least $1 - \frac{1}{N}$, following a similar analysis to [46]. Together, the probability that all these assumptions hold is

$$\left(1 - (\frac{1}{|V_C| \cdot N})^{4|V_C|}\right) \cdot \left(1 - \frac{1}{4N}\right) \geq 1 - \frac{1}{N}$$

Consider any heavy compute node v_j. The number of intervals allocated to v_j is exactly c_j, thus the number of elements received by v_j in the last round is at most

$$\begin{aligned}
\lceil \frac{M_j}{N} \cdot |V_C| \rceil \cdot 8 \cdot \frac{N}{|V_C|} &\leq (\frac{|M_j|}{N} \cdot |V_C| + 1) \cdot 8 \cdot \frac{N}{|V_C|} \\
&\leq M_j + 8 \cdot \frac{N}{|V_C|} \leq 4N_{v_j} + 16N_{v_j} = 20N_{v_j}
\end{aligned}$$

with probability at least $1 - \frac{1}{N}$.

We next bound the size of data transmitted on every edge $e \in E$. W.l.o.g., assume $\sum_{v \in V_e^- \cap V_H} N_v \leq \sum_{v \in V_e^+ \cap V_H} N_v$. The size of data sent from the heavy nodes in V_e^- to V_e^+ is always bounded by the total size of data located in $V_e^- \cap V_H$, i.e., $O(\sum_{v \in V_e^- \cap V_H} N_v) = O(\min\{\sum_{v \in V_e^- \cap V_H} N_v, \sum_{v \in V_e^+ \cap V_H} N_v\})$. The size of data sent from the heavy nodes in V_e^+ to V_e^- is at most the number of elements received by all compute nodes in $V_e^- \cap V_H$, i.e., $O(\sum_{v \in V_e^- \cap V_H} N_v) = O(\min\{\sum_{v \in V_e^- \cap V_H} N_v, \sum_{v \in V_e^+ \cap V_H} N_v\})$. In either way, the size of data transmitted over each edge e is matched by its lower bound, thus completing the whole proof. □

6 RELATED WORK

The fundamental difference of the topology-aware model we use with other parallel models (e.g., BSP [47], MPC [7], LogP [18]) is that the cost depends both on the topology and properties of the network and the nodes. Prior models view the network as a star topology, where each link and each node have exactly the same cost functions. In this sense, our model can be viewed as a generalization, where the topology and the node heterogeneity is taken into account.

There have already been some efforts to introduce topology-aware models, including [11, 32] as mentioned in the introduction.

One line of work in distributed computing on networks are the classical LOCAL and CONGEST models [37, 45], where distributed problems are also considered in networks modeled as an arbitrary graph. These two models differentiate from ours in two important aspects. First, in each round, each node can only communicate with its neighbors; instead, in our model we can send messages to other nodes that may be located several hops away. Second, the target is to design algorithms that minimize the number of rounds. As a combination of both aspects, the diameter of the communication network cannot be avoided as a cost in these models. Moreover, system synchronization after each round is a huge bottleneck of modern massively parallel systems; thus, any algorithm in these two models running in non-constant number of rounds would become hard to implement efficiently in practice.

Network routing has been studied in the context of parallel algorithms (see [33, 34]), distributed computing (see, e.g. [35]), and mobile networks [40]. Several general-purpose optimization methods

for network problems have been proposed [43]. Our proposed research deviates from prior literature by considering a "distribution-aware" setting, and tasks that have not been considered before.

The topology-aware model we use in this paper has been previously used to design algorithms for aggregation [38]. However, only star topologies were considered. Madden et al. [39, 41] also proposed a tiny aggregation service which does topology-aware in-network aggregation in sensor networks. Culhane et al. [16, 17] propose LOOM, a system that builds an aggregation tree with fixed fan-in for all-to-one aggregations, and assigns nodes to different parts of the plan according to the amount of data reduced during aggregation. Chowdhury et al. [15] propose Orchestra, a system to manage network activities in MapReduce systems. Both systems are cognizant of the network topology, but agnostic to the distribution of the input data. They also lack any theoretical guarantees.

7 CONCLUSION

In this paper, we studied three fundamental data processing tasks in a topology-aware massively parallel computational model. We derived lower bounds based on the cardinality of the initial data distribution at each node and we designed provably optimal algorithms for each task with respect to the initial data distribution. Interestingly, these problems have different dependency on the topology structure, the cost functions (bandwidth), as well as the data distribution.

There are several exciting directions for future research. For one, we would like to extend our algorithms and lower bounds to non-symmetric and general (non-tree) topologies. General topologies (e.g., grid, torus) are particularly challenging because there are multiple routing paths between two compute nodes, and thus a topology-aware algorithm needs to consider all nodes in the routing path, instead of just the destination. Looking further ahead, it would be interesting to study more complex tasks that have so far been analyzed only in the context of the MPC model, starting from a simple join between two relations, and continuing to ensembles of tasks in more complex queries.

REFERENCES

[1] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *TKDE*, 23(9):1282–1298, 2011.

[2] P. K. Agarwal, K. Fox, K. Munagala, and A. Nath. Parallel algorithms for constructing range and nearest-neighbor searching data structures. In *PODS*, pages 429–440. ACM, 2016.

[3] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *STOC*, pages 574–583, 2014.

[4] S. Assadi, X. Sun, and O. Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. In *PODC*, pages 461–470, 2019.

[5] N. Bansal, Z. Friggstad, R. Khandekar, and M. R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. *TALG*, 10(1):1:1–1:15, 2014.

[6] R. d. P. Barbosa, A. Ene, H. L. Nguyen, and J. Ward. A new framework for distributed submodular maximization. In *FOCS*, pages 645–654. IEEE, 2016.

[7] P. Beame, P. Koutris, and D. Suciu. Communication Steps for Parallel Query Processing. In *PODS*, 2013.

[8] P. Beame, P. Koutris, and D. Suciu. Skew in Parallel Query Processing. In *PODS*, 2014.

[9] S. Blanas, P. Koutris, and A. Sidiropoulos. Topology-aware parallel data processing: Models, algorithms and systems at scale. In *CIDR*, 2020.

[10] S. Blanas, K. Wu, S. Byna, B. Dong, and A. Shoshani. Parallel data analysis directly on scientific file formats. In *SIGMOD*, pages 385–396, 2014.

[11] A. Chattopadhyay, M. Langberg, S. Li, and A. Rudra. Tight network topology dependent bounds on rounds of communication. In *SODA*, pages 2524–2539, 2017.

[12] A. Chattopadhyay, J. Radhakrishnan, and A. Rudra. Topology matters in communication. In *FOCS*, pages 631–640. IEEE, 2014.

[13] C. Chekuri, A. Ene, and A. Vakilian. Node-weighted network design in planar and minor-closed families of graphs. In *ICALP*, pages 206–217, 2012.

[14] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *SICOMP*, 39(1):281–301, 2009.

[15] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *SIGCOMM*, pages 98–109, 2011.

[16] W. Culhane, K. Kogan, C. Jayalath, and P. Eugster. Loom: Optimal aggregation overlays for in-memory big data processing. In *HotCloud*, pages 13–13. USENIX Association, 2014.

[17] W. Culhane, K. Kogan, C. Jayalath, and P. Eugster. Optimal communication structures for big data aggregation. In *INFOCOM*, pages 1643–1651, 2015.

[18] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *PPOPP*, 1993.

[19] A. Dasgupta, R. Kumar, and D. Sivakumar. Sparse and lopsided set disjointness via information theory. In A. Gupta, K. Jansen, J. Rolim, and R. Servedio, editors, *APPROX/RANDOM*, pages 517–528. Springer Berlin Heidelberg, 2012.

[20] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *CACM*, 51(1):107–113, Jan. 2008.

[21] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrović, and R. Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *PODC*, pages 129–138, 2018.

[22] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrovic, and R. Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *PODC*, pages 129–138, 2018.

[23] M. T. Goodrich. Communication-efficient parallel sorting. *SICOMP*, 29(2):416–432, 1999.

[24] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the mapreduce framework. In *ISAAC*, pages 374–383. Springer, 2011.

[25] X. Hu and K. Yi. Instance and output optimal parallel algorithms for acyclic joins. In *PODS*, pages 450–463, 2019.

[26] X. Hu and K. Yi. Massively parallel join algorithms. *ACM SIGMOD Record*, 49(3):6–17, 2020.

[27] X. Hu, K. Yi, and Y. Tao. Output-optimal massively parallel algorithms for similarity joins. *TODS*, 44(2):6, 2019.

[28] B. Ketsman and D. Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *PODS*, pages 417–428. ACM, 2017.

[29] P. Koutris, P. Beame, and D. Suciu. Worst-case optimal algorithms for parallel query processing. In *ICDT*, 2016.

[30] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234. ACM, 2011.

[31] P. Koutris and D. Suciu. A guide to formal analysis of join processing in massively parallel systems. *SIGMOD Record*, 45(4):18–27, 2016.

[32] M. Langberg, S. Li, S. V. Mani Jayaraman, and A. Rudra. Topology dependent bounds for faqs. In *PODS*, page 432–449, 2019.

[33] F. T. Leighton. *Complexity issues in VLSI: optimal layouts for the shuffle-exchange graph and other networks*. MIT press, 1983.

[34] F. T. Leighton. *Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes*. Elsevier, 2014.

[35] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling ino (congestion+ dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[36] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Computers*, 34(10):892–901, 1985.

[37] N. Linial. Locality in distributed graph algorithms. *SICOMP*, 21(1):193–201, 1992.

[38] F. Liu, A. Salmasi, S. Blanas, and A. Sidiropoulos. Chasing similarity: Distribution-aware aggregation scheduling. *PVLDB*, 12(3):292–306, 2018.

[39] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *OSDI*.

[40] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, 2003.

[41] S. Madden, R. Szewczyk, M. J. Franklin, and D. E. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *WMCSA*, pages 49–58, 2002.

[42] O. O'Malley. Terabyte sort on apache hadoop. 2008.

[43] D. P. Palomar and M. Chiang. A tutorial on decomposition methods for network utility maximization. *J-SAC*, 24(8):1439–1451, 2006.

[44] M. Patraşcu. Unifying the landscape of cell-probe lower bounds. *SICOMP*, 40(3):827–847, June 2011.

[45] D. Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[46] Y. Tao, W. Lin, and X. Xiao. Minimal mapreduce algorithms. In *SIGMOD*, pages 529–540. ACM, 2013.

[47] L. G. Valiant. A bridging model for parallel computation. *CACM*, 33(8):103–111, 1990.

[48] T. Yufei. A simple parallel algorithm for natural joins on binary relations. ICDT, 2020.

A OMITTED PROOFS

A.1 Proof of Lemma 6

Proof. First, we notice that in lines 1-2 each compute node V_C belongs in exactly one $\Gamma(x)$. In the remaining algorithm, every vertex in G_β with $w(x) > 0$ is put into exactly one block, thus $\mathcal{P}$ is a partition of V_C. Indeed, the only issue may occur when we are left with a single vertex x: we claim that in this case we always have $w(x) \geq |R|$. Suppose $w(x) < |R|$, and consider the last vertex u for which $\Gamma(u)$ was added in $\mathcal{P}$ (such a vertex always exists, since every leaf vertex of G_β initially has weight at least $|R|$). But then, the algorithm could not have picked u at this point, since all other leaf vertices have smaller weight, a contradiction.

We now prove that the output partition satisfies all properties of a balanced partition (Definition 4).

(1) The first condition is trivial. From lines 1-2, two compute nodes that are connected in G_α will be in the same initial $\Gamma(x)$, hence they will appear together in a block of the partition.

(2) By contradiction, assume there is an edge $e = (u, v)$ appearing in the spanning trees of V_C^i and V_C^j for $i \neq j$. By the definition of spanning trees, there is one pair of vertices $x, y \in V_C^i$ and one pair of vertices $x', y' \in V_C^j$ such that $x, x' \in G_e^+$ and $y, y' \in G_e^-$. When Algorithm 3 visits e in line 9, w.l.o.g. assume u is visited before v. Since x, x' are placed in different blocks of the partition, it cannot be that both $x, x' \in \Gamma(u)$. W.l.o.g., $x' \notin \Gamma(u)$. This implies that x' has already been put into one block with vertices from G_e^-. Then x', y' won't appear in the same block, contradicting our assumption.

(3) It is easy to see that the algorithm adds a set of nodes to $\mathcal{P}$ only if their total weight is at least $|R|$.

(4) Consider a block V_C^i in the partition. Let $e = (u, v)$ be a β-edge in the spanning tree of V_C^i. Then, Algorithm 3 visits e in line 9: w.l.o.g. assume u is visited before v. At this point, we have $w(u) < |R|$, since $\Gamma(u)$ was merged with $\Gamma(v)$. The key observation is that we have $\Gamma(u) = V_C^i \cap V_e^-$, since no other compute nodes will be added to the "left" of e (since u is a leaf node). Hence,

$$\min\{\sum_{v \in V_C^i \cap V_e^+} N_v, \sum_{v \in V_C^i \cap V_e^-} N_v\} \leq \sum_{v \in V_C^i \cap V_e^-} N_v = w(u) < |R|$$

This completes the proof. □

A.2 Proof of Lemma 16

Proof. We first prove (1) by induction. The base case with $i = 1$ follows since $N_u^1 = \lfloor \frac{N_{v_1}}{\sum_{j=1}^k N_{v_j}} \cdot N_u \rfloor + 1$. Assume the claim holds for i. Let Δ_i be the value of Δ after the i-th iteration of the while loop. Observe that $\Delta_i = \sum_{j=1}^i N_u^j - \frac{\sum_{j=1}^i N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u$ always holds. It can also be checked that $\Delta_i \geq 0$ since $0 \leq x - \lfloor x \rfloor \leq 1$. Consider the $(i+1)$-th iteration of while loop. When it goes into line 4, we have:

$$\begin{aligned}\sum_{j=1}^{i+1} N_u^j &= N_u^{i+1} + \sum_{j=1}^{i} N_u^j \\ &= \lfloor \frac{N_{v_{i+1}}}{\sum_{j=1}^k N_{v_j}} \cdot N_u \rfloor + \Delta_i + \frac{\sum_{j=1}^i N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u \\ &= (\Delta_i - x + \lfloor x \rfloor) + \frac{\sum_{j=1}^{i+1} N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u\end{aligned}$$

In this case, $0 < \Delta_i - x + \lfloor x \rfloor < 1$, so the claim holds. When it goes into line 6,

$$\begin{aligned}\sum_{j=1}^{i+1} N_u^j &= N_u^{i+1} + \sum_{j=1}^{i} N_u^j \\ &= \lfloor \frac{N_{v_{i+1}}}{\sum_{j=1}^k N_{v_j}} \cdot N_u \rfloor + 1 + \Delta_i + \frac{\sum_{j=1}^i N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u \\ &= (\Delta_i + 1 - x + \lfloor x \rfloor) + \frac{\sum_{j=1}^{i+1} N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u\end{aligned}$$

We prove (2) based on (1). Observe that

$$\begin{aligned}\sum_{j=i_1}^{i_2} N_u^j &= \sum_{j=1}^{i_2} N_u^j - \sum_{j=1}^{i_1} N_u^j \\ &\leq \frac{\sum_{j=1}^{i_2} N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u + 1 - \frac{\sum_{j=1}^{i_1} N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u \\ &\leq \frac{\sum_{j=i_1}^{i_2} N_{v_j}}{\sum_{j=1}^k N_{v_j}} \cdot N_u + 1\end{aligned}$$

We can obtain a similar expression for i_2; then the claim holds by adding the two inequalities.

Property (3) follows immediately from (1) by setting $i = k$. □

A.3 Cartesian Product under Unequal Sizes

We consider the general cartesian product $R \times S$ on a symmetric star topology $G = (V, E)$. For simplicity, we divide the compute nodes in V_C into two subsets: $V_\alpha = \{v \in V_C : \min\{N_v, N - N_v\} < |R|\}$ and $V_\beta = V_C - V_\alpha$. The first lower bound can be simplified as follows.

theorem 18. *Any algorithm computing cartesian product $R \times S$ has cost $\Omega(C)$, where*

$$C \geq \max\left\{\max_{v \in V_\alpha} \frac{\min\{N_v, N - N_v\}}{w_v}, \max_{v \in V_\beta} \frac{|R|}{w_v}\right\}.$$

Moreover, we define $L(R, S, V_C)$ as the minimizer for the following formula and give our second lower bound in Theorem 19.

$$\sum_{v \in V_C} \min\{C \cdot w_v, |R|\} \cdot C \cdot w_v \geq |R| \cdot |S| \tag{2}$$

theorem 19. *If $\max_v N_v \leq \frac{N}{2}$, any algorithm computing cartesian product $R \times S$ has cost $\Omega(C)$, where*

$$C \geq \min\left\{\frac{|S|}{\max_v w_v}, \frac{\sum_{u \in V_\alpha} |S_u|}{2 \sum_{u \in V_\beta} w_u}, L(R, \cup_{u \in V_\alpha} S_u, V_\alpha)\right\}.$$

Proof. It suffices to show that if $C \leq |S|/\max_v w_v$, then $C \geq \min\left\{\frac{\sum_{u \in V_\alpha} |S_u|}{2 \sum_{u \in V_\beta} w_u}, L(R, \cup_{u \in V_\alpha} S_u, V_\alpha)\right\}$. We first rewrite the inequality in Section 4.5 as below: $|R| \cdot \sum_{u \in V_\alpha} |S_u| \leq$

$$\sum_{u \in V_\alpha} 4 \min\{C \cdot w_v, |R|\} \cdot C \cdot w_v + \sum_{u \in V_\beta} 2|R| \cdot C \cdot w_v$$

To make this inequality holds, at least one term should be larger than $\frac{1}{2}|R| \cdot \sum_{u \in V_\alpha} |S_u|$, thus yielding the desired result. □

Generalized wHC Algorithm. We extend the wHC algorithm for computing $R \times S$ with $|R| < |S|$ on a symmetric star topology.

Algorithm 7: BalancedPackingUnEqual($G, \mathcal{D}$)

$L^* \leftarrow L(R, S, V_C)$, $w \leftarrow \max_v w_v$;
while $\square$ *is not fully covered* **do**
  $u \leftarrow \arg\max_{v \in V_C} w_v$;
  if $2^{-\ell} w L^* \geq |R|$ **then**
    Assign to u a rectangle of size $|R| \times (w_u \cdot L^*)$;
  else
    $\ell \leftarrow \arg\min_k \{w \geq 2^k \cdot w_u\}$;
    Assign to u a square of size $(2^{-\ell} w L^*) \times (2^{-\ell} w L^*)$;
  $V_C \leftarrow V_C - \{u\}$;

To show the correctness of Algorithm 7, it suffices to show that the grid is fully covered when V_C becomes empty. Indeed, notice that each node v covers an area of size at least $L^* \cdot w_v \cdot \min\{L^* \cdot w_v, |R|\}$. Summing over all compute nodes, the area covered in total is at least

$$\sum_{v \in V_C} L^* \cdot w_v \cdot \min\{L^* \cdot w_v, |R|\} \geq |R| \cdot |S|$$

implied by (2). Hence, the whole area of $\square$ is covered.

Next, we analyze the cost of the algorithm. Observe that each node v receives at most $4L^* \cdot w_v$ tuples. Hence, the cost is bounded by $4L^*$, yielding the following result.

Lemma 20. *The wHC algorithm correctly computes the cartesian product $R \times S$ with (tuple) cost $O(C)$, where*

$$C = \max\left\{\max_v \frac{N_v}{w_v}, L(R, S, V_C)\right\}$$

Putting Everything Together on Symmetric Star. Now we show our algorithm for computing cartesian product on a symmetric star. It can be easily checked that Algorithm 8 has its cost matching the lower bound in Theorem 18 and Theorem 19, thus be optimal.

Algorithm 8: GeneralizedStarCartesianProduct($G, \mathcal{D}$)

if $\max_u N_u > N/2$ **then**
  all compute nodes send their data to $\arg\max_u N_u$;
else
  all compute nodes send their R-tuples to V_β;
  Pick the best of:
    (1) compute nodes send their data to $\arg\max_u w_u$;
    (2) all nodes in V_α send S-tuples proportionally to V_β;
    (3) run wHC algorithm on V_α to compute $R \times \cup_{v \in V_\alpha} S_v$;