A Distributed Algorithm for the Cost-Apportionment Problem on Huge Data Lineage

Anonymous Author(s)

ABSTRACT

Data lineage is an extensively used tool to depict the lifecycle of data, including data's origins and movements over time. It comprises the data entities and data transformations and is always represented as a directed graph, wherein vertices and arcs represent data entities and transformations, respectively. All data entities have their own operating costs, and in particular, the ones without downstream vertices in the data lineage (i.e., leaves) represent the data products that serve for businesses. Obviously, for each data product, only using its operating cost as the charge for business is unreasonable, as its upstream vertices participate in its production as well. Moreover, its upstream vertices may also participate in the production of other data products. Therefore, a reasonable and efficient apportionment of the operating costs from data entities to data products, i.e., calculating the production costs of data products is an essential challenge in the era of big data. Within the investigation, we first consider *acyclic* data lineage and give a distributed synchronous algorithm that apportions the operating costs of data entities to leaves along the arcs according to a specific apportionment rule until they are all assigned to leaves, and theoretically analyze its performance in terms of supersteps. Unfortunately, almost all data lineage with huge scales are *cyclic* in practice, on which the algorithm mentioned above cannot terminate. Thus we dig into the structural properties of cyclic data lineage and give a distributed synchronous algorithm that obtains a feedback arc set of the considered cyclic data lineage, based on which we show that the cost-apportionment process can be simulated as a mathematical expression with matrixes. Finally, we successfully apply the solving approach proposed for the problem to the real data-sets from Alibaba Inc., validating its feasibility and efficiency.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; **Graph algorithms analysis**; • **Mathematics of computing → Graph algorithms**.

KEYWORDS

cost-apportionment, data lineage, distributed algorithm, feedback arc set

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Anonymous Author(s). 2018. A Distributed Algorithm for the Cost-Apportionment Problem on Huge Data Lineage. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In the era of big data, the significance of data's value has gained increasing recognition, with a majority of it now treated as tradable commodities. Data pricing and data valuation have become subjects of substantial attention over the last few decades, with numerous studies dedicated to them, as evidenced by works such as [2, 11, 14, 16, 17, 20, 22, 26, 29, 30] and [12, 19, 23, 24], respectively. However, for the production costs of data products that is a crucial factor within these two subjects, the research on its calculation significantly lags behind. To the best of our knowledge, there is no systematic method proposed for calculating the production costs of data products. Given the unique characteristics of data products compared with physical goods [20], traditional approaches for calculating the production costs of physical goods are inapplicable, particularly when the production process becomes intricate. Therefore, there is an urgent need for a reasonable and efficient approach to calculate the production costs of data products.

A scenario illustrating intricate production process of data products is given below. In the offline data warehouse of a large-scale data company, the data that serves the businesses is always generated by a lengthy computational process known as data production pipeline. This pipeline may consist of hundreds to thousands, or even tens of thousands of individual computational tasks, each of which typically generates a data table. Such a complex architecture is a result of several factors, such as "unified data collection" and "common computation layer".

The data collected for various businesses are usually highly reusable. Thus from the perspectives of cost, data quality and consistency, large-scale data companies build unified data collection platforms to gather the required data. Consequently, the data required by these businesses is naturally centralized in one place, and different business segments have to execute distinct tasks to retrieve the data they require.

Another reason for the complexity of the data production process is the "common computation layer." If a data company's businesses are not very complex, then data developers can generally maintain a set of data metrics to guide the optimization of business decisions. However, as business categories become more complex and critical businesses begin to branch into specialized domains, maintaining metrics becomes exceptionally challenging. This, in turn, gives rise to challenges in ensuring "data accuracy" and "data consistency." To address these challenges, large-scale data companies propose the establishment of a common computation layer. This means that relevant business data needs to be derived from the "unified data tables,"

rather than being recalculated from scratch. This ensures the consistency of data. The essence of such a "common computation layer" is the integration of similar and identical computation tasks into a single one, preventing redundant calculations and data inconsistencies. This integration can occur at any stage of the data production pipeline. Consequently, within the data warehouse, there are many tables that don't directly serve business but are crucial data sources for many businesses.

Additionally, the feedback from downstream data is also crucial for business decision, which leads to a situation that a single table may rely on its own data from yesterday or last year, and the one from multiple downstream layers. Moreover, the data are often used to do the "year-on-year" and "quarter-on-quarter" comparisons. Thus cycles are formed between these data tables. Overall, for large-scale data companies such as Alibaba Inc., the data production process may be very complicated such that calculating the production costs of these data products is not an easy job.

For this case, investigating and analyzing the details of the whole data production process is paramount. Data Lineage [10] is an extensively used tool to depict data lifecycle including their origins and movements over time, which simplifies the jobs such as tracking errors and debugging the data flow process [1, 7, 25]. By utilizing the data lineage information within the analytics process, organizations are able to shorten the decision making process, enhance data loss prevention and enable more efficient and cost-effective compliance and auditing [28]. Within the paper, data lineage is utilized to depict the data production process. Thus we give its related background below, based on which we formulate the calculation of the data production costs as a computational problem.

A data lineage comprises the *data entities* and *data transformations*, where the data entities can be tables stored in data warehouse, the data transformations depict the movements among the data entities, and each data entity has its own operating cost (such as the costs of computation and storage). Within our investigation, a data lineage is simply represented as a directed graph $G = (V(G), A(G), W_v, W_a)$ with a vertex-weight function $W_v : V(G) \Rightarrow \mathbb{R}^{\geq 0}$ defined on the vertex set $V(G)$ and an arc-weight function $W_a : A(G) \Rightarrow \mathbb{R}^{\geq 0}$ defined on the arc set $A(G)$, where the vertices and arcs represent the data entities and transformations, respectively. The vertices with out-degree 0 in data lineage G (called *leaves*) represent the related data products, the ones with in-degree 0 (called *roots*) and the ones with both in-degree and out-degree at least 1 (called *internal vertices*) represent the *intermediate* data entities. Note that any data lineage considered in the paper is assumed to have at least one vertex and be connected (i.e., its undirected underlying graph is connected), thus it has no vertex with both in-degree and out-degree 0. Denote by $L(G)$, $R(G)$ and $I(G)$ the sets containing the leaves, roots and internal vertices in G, respectively. Given two vertices v and v' in G, if there is a (directed) path from v to v' then v is an *upstream vertex* of v', and v' is a *downstream vertex* of v. If there is an arc from v to v', denoted by $v \rightarrow v'$, then v is a *direct upstream vertex* of v', and v' is a *direct downstream vertex* of v. Denote by $U_G(v)$ (resp., $D_G(v)$) the set containing all upstream (resp., downstream) vertices of v in G, and by $U_G^d(v)$ (resp., $D_G^d(v)$) the set containing all direct upstream (resp., direct downstream) vertices of v in G. The weight $W_v(v)$ on each vertex $v \in V(G)$ represents its *operating cost*, and the weight $W_a(v \rightarrow v')$ on each arc $v \rightarrow v' \in A(G)$ represents the *flow rate* of data on it.

For the data products represented by leaves, it is reasonable to calculate their production costs based on the operating costs of the data entities that participate in their production processes. For a leaf $v \in L(G)$, obviously all vertices of $U_G(v)$ participate in its production process, but some of them may participate in the production of other data products. Under this case it is unreasonable to set the production cost of v as the sum over the operating costs of its upstream vertices. A natural idea is to repeatedly *apportion* the operating costs of the vertices in $R(G) \cup I(G)$ along the arcs to their direct downstream vertices in G according to some *cost-apportionment rule*, until they all on the leaves. Remark that within the cost-apportionment process implementing the above idea, each internal vertex $v \in I(G)$ with $U_G(v) \neq \emptyset$ would be apportioned costs from its direct upstream vertices, thus the *cost-apportionment operation* on v considers not only its own operating cost but also the ones obtained from its direct upstream vertices. The cost-apportionment operation on each vertex $v \in R(G) \cup I(G)$ considered in the paper follows the *cost-apportionment rule*: The costs apportioned from v to its direct downstream vertices are in proportion to the flow rates of the arcs from v to its direct downstream vertices, i.e., $W_a(v \rightarrow v')/(\sum_{v'' \in D_G^d(v)} W_a(v \rightarrow v''))$ of the cost on v is apportioned to each vertex $v' \in D_G^d(v)$. [1]

A *cost-apportionment approach* for a data lineage G is a function $C : V(G) \Rightarrow \mathbb{R}^{\geq 0}$ defined on $V(G)$ such that $\sum_{v \in V(G)} C(v) = \sum_{v \in V(G)} W_v(v)$. Two cost-apportionment approaches C_1 and C_2 for G are the *same*, if $C_1(v) = C_2(v)$ for each vertex $v \in V(G)$, denoted by $C_1 = C_2$. A cost-apportionment approach C for G is *complete* if $\sum_{v \in L(G)} C(v) = \sum_{v \in V(G)} W_v(v)$ (i.e., $C(v) = 0$ for each vertex $v \in R(G) \cup I(G)$). Calculating the production costs of the data products as mentioned above is formulated as the *Cost-Apportionment problem* in the paper, whose formulation is given as follows.

Cost-Apportionment problem
Input: a (connected) data lineage G;
Output: a complete cost-apportionment approach for G, where the costs are apportioned by the cost-apportionment rule.

The Cost-Apportionment problem has received lots of attentions in the past few years, but to our best knowledge, there is no published literature on it. The importance of its study is obvious, which not only helps to price the data products but also contributes to data management and data optimization. For example, assume that a data lineage G has two leaves a and b, where they represent the same data product but $U_G(a) \neq U_G(b)$ (i.e., there are two ways to produce the data product). Then comparing the costs of the two production ways is valuable to the product manager to cut back the unnecessary expenses.

Unfortunately, simply using the above idea (i.e., repeatedly applying the cost-apportionment operation on the roots and internal vertices if they are apportioned costs) to solve the Cost-Apportionment problem has two critical issues. (1). The considered data lineage

[1] It is necessary to point out that there are some other cost-apportionment rules (e.g., apportioning the cost of v to its direct downstream vertices evenly), and they may be reasonable in some specific scenarios. However, we only study the one given above within the paper.

may be very huge such that we do not have its complete graph structure, specifically, each vertex in the data lineage only has its neighborhood information. Thus the related algorithms designed for the problem should be in distributed form. (2). The considered data lineage may be cyclic. If there are costs that are apportioned along the arcs in the cycles, then there always exist some costs that remain on the vertices in the cycles no matter how many times the cost-apportionment operation is applied. Under this case, some heuristic ways can be given, e.g., repeatedly applying the cost-apportionment operation on the roots and internal vertices until the sum of the costs apportioned to the leaves is over the total costs time a threshold. However, the way is not only expensive from the aspects of time and resources but also unacceptable from the aspect of accuracy (more discussion will be given in the remaining text).

Contribution. The paper takes the above two issues into consideration and proposes a distributed algorithm named Alg-CA-Imp that can solve the Cost-Apportionment problem efficiently no matter whether the given data lineage G is cyclic or acyclic. Specifically, if G is acyclic, then we directly call the algorithm Alg-CA implementing the idea mentioned above, which is shown to be theoretically efficient in solving the problem on acyclic data lineage. Otherwise (i.e., G is cyclic), we first call the two algorithms Alg-Label and Alg-FAS given in the paper in turn to obtain a feedback arc set S of G and construct an acyclic data lineage G' based on S and G, then call the proposed algorithm Alg-CA1 on G' to get several cost-transition matrixes and transform the Cost-Apportionment problem on G as a mathematical expression with matrixes getting limit.

Remark that all graph algorithms given in the paper are distributed and implemented in a Pregel [18] liked distributed graph computation platform of Alibaba Inc. called MaxCompute, i.e., they are in synchronous form [8, 21]. We show that for each execution of Alg-CA-Imp, the calls of the three algorithms (namely, Alg-Label, Alg-FAS and Alg-CA1) take at most $2n + 3$ supersteps (i.e., synchronous round), where $n = |V(G)|$. Besides the analysis on the theoretical performance of Alg-CA-Imp from the aspect of supersteps, we also implement and successfully apply it in the real scenario of Alibaba Inc. to price their data products. Its feasibility and efficiency are validated.

The paper is organized as follows. Section 2 gives the related notions and a simple distributed synchronous algorithm called Alg-CA for the Cost-Apportionment problem on acyclic data lineage, and Section 3 analyzes the theoretical performance of Alg-CA. Section 4 starts with a discussion about the feedback arc set then presents two distributed synchronous algorithms called Alg-Label and Alg-FAS to construct a feedback arc set of the considered data lineage. Section 5 gives the details of the algorithm Alg-CA-Imp for the Cost-Apportionment problem, and proposes a new algorithm that is a variant of Alg-CA to improve the performance of Alg-CA-Imp. The experimental performance of Alg-CA-Imp is studied in Section 6. Section 7 is used to conclude the paper.

2 PRELIMINARY AND SIMPLE ALGORITHM

W.l.o.g., any data lineage G considered in the paper for the Cost-Apportionment problem is assumed to satisfy the following three properties:

1. **when** $r = 1, 2, \cdots, N$ **do**
2. **begin synchronous round**
   /* Message Receiving Phase */
3. **for** each message $msg(v, x)$ sent to v **do**
4. $C(v) = C(v) + x$;
   /* Message Sending Phase */
5. **if** $C(v) > 0 \wedge D_G^d(v) \neq \emptyset$ **then**
6. **for** each vertex $v_i \in D_G^d(v)$ **do**
7. let $y = C(v) \cdot W_a(v, v_i) / \sum_{v_j \in D_G^d(v)} W_a(v, v_j)$;
8. send message $msg(v_i, y)$ to v_i;
9. $C(v) = 0$.

Figure 1: The pseudo-code of the Pregel implementation of the distributed synchronous algorithm Alg-CA on each worker machine with assigned vertex $v \in V(G)$.

I) Each vertex $v \in R(G) \cup I(G)$ has at least one leaf in $D_G(v)$, i.e., each intermediate data entity contributes to at least one data product;
II) There is no arc with weight 0, as weight 0 indicates that there is no data going through the arc, which is meaningless;
III) There are no *parallel* arcs (in the same direction) between any two vertices, as they can be merged as an arc if exist, whose weight is the sum over the weights on these parallel arcs.

Given a subset $V' \subset V(G)$, $G[V']$ denotes the subgraph of G induced by V' that comprises the vertices in V' and all arcs among them in G. For an arc $v \to w$ of $A(G)$, $G \setminus \{v \to w\}$ denotes the graph obtained by removing the arc from G. If there is a vertex $v \in V(G)$ with in-degree 1 and out-degree 1, then it is *contractable*. *Contradicting* v means removing it and the two incident arcs from G, connecting its direct upstream vertex v_1 and downstream vertex v_2 with a new arc $v_1 \to v_2$, and let $W_v(v_2) = W_v(v_2) + W_v(v)$ and $W_a(v_1 \to v_2) = W_a(v_1 \to v)$.

The data lineage G is acyclic (i.e., it is a *directed acyclic graph*, abbr. DAG) if and only if G has a linear ordering of the vertices, called *topological ordering*, with respect to which the *tail* v locates before the *head* v' for each arc $v \to v'$ of $A(G)$ (under this case, the arc $v \to v'$ is *consistent* with the ordering; otherwise, *inconsistent*). Thus if G is cyclic, then for any linear ordering of the vertices in $V(G)$, there always exist some arcs that are *inconsistent* with it. Note that if G is acyclic then every (directed) path P in G is *simple*, i.e., each vertex and arc of G occurs at most once in P.

A simple distributed synchronous algorithm named Alg-CA for the Cost-Apportionment problem on data lineage G is presented, following the natural idea given before: Repeatedly apportioning the costs on the roots and internal vertices of G to their direct downstream vertices by the cost-apportionment rule, until all costs are apportioned to the leaves. For ease of readability, each worker machine is assumed to be assigned exactly one vertex of G [2]. Thus the Pregel implementation of Alg-CA on each worker machine with assigned vertex $v \in V(G)$ is to repeatedly apportion the cost on it (if any) to its direct downstream vertices if v is not a leaf, whose pseudo-code is given in Fig. 1. Remark that the Pregel implementation of

[2]Note that a worker machine may be assigned more than one vertex of $V(G)$ in real scenarios. Under the case, the machine loops through the active vertices to execute the related instructions in each superstep [18].

Alg-CA on each worker machine with assigned vertex $v \in V(G)$ has the following initial knowledge: $W_v(v)$, $D_G^d(v)$, and the weights of the arcs from v to the vertices in $D_G^d(v)$. Besides, it maintains a variable $C(v)$ to record the current cost on v, which is initialized by its weight $W_v(v)$. That is, G has an initial cost-apportionment approach C with $C(v) = W_v(v)$ for each vertex $v \in V(G)$, which obviously is not a complete cost-apportionment approach.

Each superstep of the Pregel implementation of Alg-CA on the worker machine with assigned vertex $v \in V(G)$ contains two phases: **(I). Message Receiving Phase** and **(II). Message Sending Phase**. In Message Receiving Phase, v adds the costs apportioned from its direct upstream vertices (if any) into its current cost $C(v)$. In Message Sending Phase, if $C(v) > 0 \wedge D_G^d(v) \neq \emptyset$ then v apportions the cost $C(v)$ to its direct downstream vertices by the cost-apportionment rule. Because of the mechanism of Pregel, the messages sent by v to its direct downstream vertices at the t-th superstep ($t \geq 1$ is an integer), will be received by its direct downstream vertices at the $t+1$-th superstep. That is, the cost on v at the t-th superstep, is apportioned to its downstream vertices that are c-hops away from v at the $t+c$-th superstep. Additionally, the Pregel implementations of Alg-CA on the vertices run in parallel, thus the cost-apportionment operations on the vertices are applied independently. Finally, the algorithm Alg-CA as a whole terminates when all vertices are simultaneously inactive and there is no message in transit, i.e., all costs have been apportioned to the leaves, and the cost-apportionment approach C obtained is complete.

3 THEORETICAL PERFORMANCE OF ALG-CA

The section studies the theoretical performance of Alg-CA for the Cost-Apportionment problem.

Theorem 3.1. *Given an acyclic data lineage G with n vertices,* Alg-CA *solves the Cost-Apportionment problem on G with L_p+1 supersteps, where L_p is the length of the longest path in G and $L_p \leq n-1$.*

Proof. As G is acyclic, G has a topological ordering with respect to which all arcs in G are in the same direction. That is, the cost-apportionment process on G specified by Alg-CA is unidirectional. Consequently, once the cost c on a vertex $v \in V(G)$ is apportioned to its direct downstream vertices, then any proportion of the cost c cannot return back to v. Combining the conclusion with the mechanism of Pregel, we have that for any vertex $v \in R(G) \cup I(G)$, the cost-apportionment process to apportion the operating cost of v to the leaves in $D_G(v) \cap L(G)$ requires at most $l+1$ supersteps, where l is the length of the longest one among the paths from v to $D_G(v) \cap L(G)$ in G. Since the Pregel implementations of Alg-CA on the vertices run in parallel, the cost-apportionment process specified by Alg-CA takes L_p+1 supersteps to apportion the operating costs of $R(G) \cup I(G)$ to the leaves. □

Theorem 3.2. *Given a cyclic data lineage G with n vertices,* Alg-CA *can solve the Cost-Apportionment problem on G with infinite supersteps in theory.*

Proof. Observe that the length of the longest path P in G is infinite (note that P is not required to be simple), then the analysis given in Theorem 3.1 shows that Alg-CA cannot solve the problem on G efficiently. Specifically, for a vertex $v \in I(G)$ that is in a cycle O of G, the cost on it would be apportioned along the arcs in O, and a proportion α of that will return back to v. Fortunately, α can be shown to be less than 1 by the following reasoning.

Consider a vertex $v \in R(G) \cup I(G)$ with cost c at the t-th superstep, where $t \geq 1$. Property (I) given in Section 2 for data lineage shows that v has at least one leaf l as its downstream vertex in G. Then the reasoning given in Theorem 3.1 indicates that a proportion of the cost c on v can be apportioned to a leaf by Alg-CA within the next (at most) n supersteps, i.e., $\alpha < 1$. Generalizing the conclusion gives that a proportion of the costs on the roots and internal vertices of G can be apportioned to the leaves of G by Alg-CA with every n supersteps. That is, with every n supersteps, the sum of the costs on the roots and internal vertices of G would be decreased by Alg-CA. As a result, in theory, the sum of the costs on the roots and internal vertices of G would be decreased to 0 by Alg-CA with enough many supersteps. However, as the cost-apportionment operation apportions the costs proportionally, the process is infinitely long due to the existence of cycles. □

Thus directly calling the algorithm Alg-CA to solve the Cost-Apportionment problem on cyclic data lineage is infeasible. A heuristic approach can be given: Set a threshold $0 < \delta < 1$, and call Alg-CA on the considered cyclic data lineage G until the sum of the costs on the leaves is over the total costs on the vertices timing δ. However, this way has two serious disadvantages.

The first one is that the margin of error may be very large. Assume that C is the complete cost-apportionment approach that the Cost-Apportionment problem on G looks for, and C' is the cost-apportionment approach obtained by the above heuristic way. Although we have $\sum_{v \in L(G)} C'(v) \geq \delta \cdot \sum_{v \in L(G)} C(v)$, $C'(v')$ may be less than $\delta \cdot C(v')$ for some vertex $v' \in L(G)$. In particular, the ratio $\frac{\delta \cdot C(v')}{C'(v')}$ between $\delta \cdot C(v')$ and $C'(v')$ may be very large if $C'(v')$ is small, and the margin of error for v' is unaccepted. The second one is that the number of supersteps required by Alg-CA to obtain such a cost-apportionment approach heavily depends on the structure of G (not the number of vertices in G) and may be still unaccepted. Summarizing the above discussion, it is meaningful and necessary to propose an algorithm that can exactly and efficiently solve the Cost-Apportionment problem on cyclic data lineage.

Before designing the algorithm for the Cost-Apportionment problem on cyclic data lineage, we show the feasibility of two modification operations on data lineage that can simplify its structure.

Lemma 3.3. *Given a data lineage G that contains a contractable vertex v, then for the complete cost-apportionment approaches C and C' that the Cost-Apportionment problem on G and G' look for, respectively, $C(v) = C'(v)$ for any vertex $v \in L(G) = L(G')$, where G' is the data lineage obtained by contracting v in G.*

Proof. The operating cost of v would be totally apportioned to its unique direct downstream vertex v' by the cost-apportionment operation, as v has out-degree 1. Thus we can modify the weights on v and v' as: Let $W_v(v') = W_v(v') + W_v(v)$ and $W_v(v) = 0$, obtaining a new data lineage G_1. It is easy to see that the modification on G does change the complete cost-apportionment approach that the Cost-Apportionment problem on G looks for, i.e., for the complete cost-apportionment approaches C and C_1 that the Cost-Apportionment problem on G and G_1 look for, respectively, $C = C_1$.

Observe that the vertex v of G_1 works as a data transfer center transferring the data from its direct upstream vertex v'' to v', without operating cost. Thus we can remove the vertex v of G_1 and its two incident arcs, and connect its direct upstream vertex v'' and downstream vertex v' with a new arc $v'' \to v'$, i.e., directly transferring the data from v'' to v'. For the weight of the new arc $v'' \to v'$, it is set as $W_a(v'' \to v)$, thus any cost apportioned from v'' to v in G_1 would be apportioned to v' directly in the new data lineage obtained by the above modification on G_1. Observe that the new data lineage obtained by the above modification on G_1 is the same as G', and the modification on G_1 also does not change the complete cost-apportionment approach that the Cost-Apportionment problem on G_1 looks for, i.e., for the complete cost-apportionment approaches C_1 and C' that the Cost-Apportionment problem on G_1 and G' look for, respectively, $C_1(v) = C'(v)$ for any vertex $v \in L(G_1) = L(G')$.

Summarizing the above discussion gives that $C(v) = C_1(v) = C'(v)$ for any leaf $v \in L(G) = L(G_1) = L(G')$, and the contraction operation on the contractable vertex v of G is feasible. □

Lemma 3.4. *Given a data lineage G that contains a cycle $v \to v' \to v$, where the out-degrees of v and v' are 1 and at least 2, respectively, then for the complete cost-apportionment approaches C and C' that the Cost-Apportionment problem on G and G' look for, respectively, $C = C'$, where $G' = G \setminus \{v' \to v\}$.*

Proof. Let $D^d_G(v') \setminus \{v\} = \{v_1, \ldots, v_p\}$. Consider the cost c on v at the t-th superstep during the cost-apportionment process, where $t \geq 1$. As the vertex v has a unique direct downstream vertex v' that has v as one of its direct downstream vertices, a fixed proportion (i.e., $\frac{W_a(v' \to v)}{W_a(v' \to v) + \sum_{i=1}^{p} W_a(v' \to v_i)}$) of the cost c would go back to v through the vertex v'. The same reasoning applies to the cost returned back to v. Since the proportion $\frac{W_a(v' \to v)}{W_a(v' \to v) + \sum_{i=1}^{p} W_a(v' \to v_i)}$ is less than 1, with enough many executions of the cost-apportionment operation on v and v', the proportion of the cost c remaining in the cycle $v \to v' \to v$ approaches 0, and the others are all apportioned to the downstream vertices of $D^d_G(v') \setminus \{v\}$ by the cost-apportionment rule. Equivalently, the cost c is apportioned to the vertices of $D^d_G(v') \setminus \{v\}$, specifically, $\frac{W_a(v' \to v_i)}{\sum_{j=1}^{p} W_a(v' \to v_j)}$ of the cost c is apportioned to the vertex $v_i \in D^d_G(v') \setminus \{v\}$ for each $1 \leq i \leq p$.

Remark that the conclusion obtained above applies to the cost on v at any superstep during the cost-apportionment process. Thus it is safe to remove the arc $v' \to v$, obtaining a new data lineage G', and we can derive that the complete cost-apportionment approaches C and C' the Cost-Apportionment problem on G and G' looking for, respectively, are the same. □

4 TWO DISTRIBUTED SYNCHRONOUS ALGORITHMS FOR FEEDBACK ARC SET OF DATA LINEAGE

To solve the Cost-Apportionment problem on cyclic data lineage, we first present two distributed synchronous algorithms that can cooperatively obtain a feedback arc set of the considered cyclic data lineage G in the section.

A *feedback arc set* (abbr. FAS) of G is a set of arcs whose removal from G results in an acyclic data lineage. The *Feedback Arc Set problem* looks for an FAS with the minimum size for a given directed graph, and it is one of the Karp's 21 NP-complete problems [13]. Its research can date back to 1957 [27], including [3, 4, 6, 9, 15]. Fortunately, we do not have to look for a minimum FAS of the considered data lineage G within the paper, but a general FAS of G.

As mentioned in Section 2, if a directed graph is cyclic, then for any linear ordering P of its vertices, there always exists a subset A' of the arc set $A(G)$ of G in which these arcs are inconsistent with P. That is, the arcs of $A(G) \setminus A'$ are consistent with P and $G \setminus A'$ is an DAG, indicating that A' is an FAS of G. As a result, any linear ordering of the vertices in G specifies an FAS of G.

Based on the facts mentioned above, we first give a distributed synchronous algorithm named Alg-Label to label the vertices of G, based on which a linear ordering of the vertices in G can be obtained, where the label of each vertex comprises positive integers and the symbol '.' separating them. The labeling process of Alg-Label for $V(G)$ can be treated as an execution of a variant of the Lexicographic Depth First Search (abbr. LexDFS) [5] on G, and the *lexicographic order* considered in the paper is defined as follows.

Definition 4.1. (**Lexicographic Order**) Given two different labels $A = a_1.a_2.\cdots.a_p$ and $B = b_1.b_2.\cdots.b_q$, where a_i $(1 \leq i \leq p)$ and b_j $(1 \leq j \leq q)$ are positive integers,

1) A is *strictly larger* than B, denoted by $A \gg_{lo} B$, if A is a prefix of B, i.e., $a_i = b_i$ for all $1 \leq i \leq p < q$;
2) A is *larger* than B, denoted by $A >_{lo} B$, if A is strictly larger than B, or there is an index $1 \leq k \leq \min\{p, q\}$ such that $a_i = b_i$ for all $1 \leq i < k$ and $a_k < b_k$.

The pseudo-code of the Pregel implementation of Alg-Label on each worker machine with assigned vertex $v \in V(G)$ is given in Fig. 2, which has the following initial knowledge: $D^d_G(v)$, $U^d_G(v)$, and all the arcs incident to v. Besides, it maintains a variable $L(v)$ to record the current label of v and a boolean one $update$ to record whether or not the label of v is updated in the considered superstep. Initially, each root of $R(G)$ has a unique integer i ranging from 1 to $|R(G)|$ as its label (note that the assignment of labels to the roots is random), and each vertex of $V(G) \setminus R(G)$ has an empty label.

Each superstep of the Pregel implementation of Alg-Label on the worker machine with assigned vertex $v \in V(G)$ contains two phases: **(I). Message Receiving Phase** and **(II). Message Sending Phase**. In Message Receiving Phase, if $v \in V(G)$ receives a label L from one of its direct upstream neighbors that is larger than its current label $L(v)$ according to the lexicographic order, i.e., $L >_{lo} L(v)$, then $L(v)$ is updated as L and the boolean variable $update$ is set as 1. In Message Sending Phase, if $update = 1 \wedge D^d_G(v) \neq \emptyset$ then v sends its new label with an extra suffix $.j$ to its direct downstream neighbors one by one, where j increases from 1 to $|D^d_G(v_i)|$ with the number of messages sent to ensure that its direct downstream neighbors obtain different labels.

A label has *length* $x+1$ ($x \geq 0$ is an integer) if it contains x many occurrences of the symbol '.'. For an arc $v \to v'$ in G, if $L(v)$ is strictly larger than $L(v')$, and the length of $L(v)$ is exactly one less than that of $L(v')$, then $v \to v'$ is a *tree-arc* (i.e., if $L(v') = a_1.\cdots.a_i$ $(i \geq 2)$ then $L(v) = a_1.\cdots.a_{i-1}$).

For the final labels of the vertices in G specified by Alg-Label, we have the following several propositions by the generating rule and updating rule of the labels given in Alg-Label.

1. **when** $r = 1, 2, \cdots, N$ **do**
2. **begin synchronous round**
3. $update = 0$; /* recording whether the label is updated */
   /* Message Receiving Phase */
4. **for** each message $msg(v, L)$ sent to v **do**
5. **if** $L >_{lo} L(v)$ **then**
6. $L(v) = L$;
7. $update = 1$;
   /* Message Sending Phase */
8. **if** $update = 1 \wedge D_G^d(v) \neq \emptyset$ **then**
9. let $i = 1$;
10. **for** each vertex $v_j \in D_G^d(v)$ **do**
11. send message $msg(v_j, L(v) + ".i")$ to v_j;
12. $i = i + 1$.

Figure 2: The pseudo-code of the Pregel implementation of the distributed synchronous algorithm Alg-Label **on each machine with assigned vertex** $v \in V(G)$**.**

Proposition 4.2. *For any vertex v in G with in-degree greater than 0, it has a direct upstream vertex v_p such that $v_p \rightarrow v$ is a tree-arc.*

Proof. Recall that each vertex $v \in I(G) \cup L(G)$ has an empty lable initially, and its final label is specified by a message from one of its direct upstream vertices (say v_p). By the generating rule of the labels, we have that the label of v_p is a prefix of that of v, moreover, the length of the label of v_p is exactly one less than that of v. That is, the arc $v_p \rightarrow v$ is a tree-arc. □

Proposition 4.3. *The final label of each vertex specified by* Alg-Label *is unique.*

Proof. Assume that there are two different vertices v and v' that have the same final label. If the length of the final label of v and v' is 1, then v and v' are roots of G. Recall that the roots of G have different labels initially, and their labels cannot be updated as they have no upstream vertices. Thus there is a contradiction, and in the following discussion we consider the case that the length of the final label of v and v' is greater than 1.

By Proposition 4.2, there are tree-arcs $v_1 \rightarrow v$ and $v_1' \rightarrow v'$ leading to v and v', respectively. Observe that v_1 and v_1' have the same final label by the definition of tree-arc.

Case (1). Any tree-arc $v_1 \rightarrow v$ leading to v and any tree-arc $v_1' \rightarrow v'$ leading to v' share the same tail (i.e., there is only one tree-arc $v_1 \rightarrow v$ leading to v and one tree-arc $v_1' \rightarrow v'$ leading to v', and $v_1 = v_1'$). Under this case, the final labels of v and v' are specified by the messages from $v_1 = v_1'$. However, we have that the messages sent from $v_1 = v_1'$ to different vertices have different labels, implying that v and v' obtain different final labels, a contradiction.

Case (2). There is a tree-arc $v_1 \rightarrow v$ leading to v and a tree-arc $v_1' \rightarrow v'$ leading to v' that have different tails (i.e., $v_1 \neq v'$). Under this case, we have that v_1 and v_1' have the same final label. Then we apply the above reasoning to v_1 and v_1'.

Summarizing the above reasoning gives that the final label of each vertex is unique. □

Proposition 4.4. *For any vertex v in G with in-degree greater than 0, there exists exactly one tree-arc $v_p \rightarrow v$ heading to v. Moreover, the label of v_p is larger than that of v_p', for any other direct upstream vertex $v_p' \in (U_G^d(v) \setminus \{v_p\})$.*

Proof. By Propositions 4.2 and 4.3, there exists exactly one tree-arc $v_p \rightarrow v$ leading to v; otherwise, there exist two vertices that have the same final label. Assume that there is another direct upstream vertex $v_p' \in U_G^d(v) \setminus \{v_p\}$ satisfying that $L(v_p') >_{lo} L(v_p)$. W.l.o.g., assume that v_p and v_p' obtain their final labels at the t-th and t'-th supersteps, respectively.

Case (1). $t' < t$. Then by the updating rule of the labels given in Alg-Label, we have that once the label of v is specified by the message from v_p' at the $t'+1$-th superstep, then it cannot be updated by any message from v_p by the lexicographic order, contradicting to the fact that the final label of v is specified by a message from v_p.

Case (2). $t' > t$. Similarly, by the updating rule of the labels given in Alg-Label, we have that if the label of v is specified by the message from v_p, then it can be updated by the message from v_p' by the lexicographic order, contradicting to the fact that the final label of v is specified by a message from v_p.

Case (3). $t' = t$. During the Message Receiving Phase of the $t+1$-th superstep, v would get the messages from v_p and v_p'. However, no matter what the order of the messages that the vertex v receives is, by the above reasoning, we always that the final label of v is specified by the message from v_p', contradicting to the fact that the final label of v is specified by a message from v_p.

Summarizing the above discussion gives that the assumption is incorrect, and there is no other direct upstream vertex $v_p' \in U_G^d(v) \setminus \{v_p\}$ satisfying that $L(v_p') >_{lo} L(v_p)$. □

Proposition 4.5. *For any two vertices v_1 and v_2 of G such that $L(v_1) >_{lo} L(v_2)$ but $L(v_1)$ is not strictly larger than $L(v_2)$, there is no path from v_1 to v_2 in G, and any vertex $v \in D_G(v_1)$ has $L(v) >_{lo} L(v_2)$.*

Proof. The condition that $L(v_1) >_{lo} L(v_2)$ but $L(v_1)$ is not strictly larger than $L(v_2)$ indicates that for any downstream vertex v_1' of v_1, $L(v_1') >_{lo} L(v_2)$. That is, if v_2 is a downstream vertex of v_1, then the label of v_2 would be updated such that $L(v_1)$ is strictly larger than it. □

Before analyzing the theoretical performance of Alg-Label, we give some related notions. Let the root-set of G be $\{r_1, r_2, \ldots, r_k\}$, in which each root r_i has label i ($1 \leq i \leq k$), $G_i = G[D_G(r_i)] \setminus \cup_{j=1}^{i-1} G_j$, and l_i be the length of the longest simple path in G_i. Observe that each vertex of G_i (for any $1 \leq i \leq k$) has a final label with prefix i.

Theorem 4.6. *Each vertex of G obtains its final label within at most $L+1$ supersteps, where $L = \max\{l_1, l_2, \ldots, l_k\} \leq n-1$, where n is the number of vertices in G. That is,* Alg-Label *takes at most $L+1 \leq n$ supersteps.*

Proof. By Proposition 4.4, for any vertex v in G with in-degree greater than 0, there is a simple path in G from a root to it that consists of tree-arcs. That is, for each $v \in G_i$, it can get its final label within $l_i + 1$ supersteps, where $l_i \leq n-1$. □

Lemma 4.7. *For each arc $v \rightarrow v'$ in G, the final labels of v and v' specified by* Alg-Label *satisfy one of the following four properties:*

(1) $L(v) \gg_{lo} L(v')$ *and the length of* $L(v)$ *is less than that of* $L(v')$ *by exactly one (i.e.,* $v \to v'$ *is a tree-arc);*
(2) $L(v) \gg_{lo} L(v')$ *and the length of* $L(v)$ *is less than that of* $L(v')$ *by at least two;*
(3) $L(v') \gg_{lo} L(v)$*;*
(4) $L(v') >_{lo} L(v)$ *but* $L(v')$ *is not strictly larger than* $L(v)$*.*

Proof. Let $v \to v'$ be an arc in G with $L(v) >_{lo} L(v')$. If $L(v)$ is not strictly larger than $L(v')$, then Proposition 4.5 gives that there is no path from v to v', contradicting the fact that there is an arc from v to v'. Thus $L(v)$ is strictly larger than $L(v')$. The reasoning for Property (1) and (2) is done.

For Property (3), some arcs in the cycles of G may satisfy it. If v and v' are downstream vertices of roots r and r', respectively, where the final label of r' is larger than that of r, and v is not a downstream vertex of r', then we can get that the arc $v \to v'$ satisfies Property (4). □

Denote by A_1, A_2, A_3, and A_4 the sets containing all the arcs in G satisfying Property (1), (2), (3) and (4) given in Lemma 4.7, respectively (i.e., A_1 comprises the tree-arcs). For each integer $1 \leq i \leq |R(G)|$, denote by V_i the subset of $V(G)$ in which each vertex has a label with prefix "i". Denote by A_1^i, A_2^i, A_3^i and A_4^i the subsets of A_1, A_2, A_3 and A_4 in which each arc is between the vertices of V_i, respectively, and denote by G_1^i the graph comprising the vertices of V_i and the tree-arcs of A_1^i.

Lemma 4.8. *For any* $1 \leq i \leq |R(G)|$, G_1^i *is a rooted spanning tree of* $G[V_i]$ *that is rooted at the root with label* i*. Moreover, for any two vertices* v *and* v' *in* G_1^i*, there is a path from* v *to* v' *in* G_1^i *if and only if* $L(v)$ *is strictly larger than* $L(v')$*.*

Proof. Denote by r_i the root in G with label "i". Proposition 4.4 shows that each vertex $v \in V_i \setminus \{r_i\}$ has exactly one tree-arc $v' \to v$ leading to it, and the vertex v' is also of V_i. Thus we can derive that G_1^i is connected and contains exactly $|V_i| - 1$ arcs, implying that G_1^i is a rooted spanning tree of $G[V_i]$. □

Lemma 4.9. A_3 *is an FAS of* G*.*

Proof. By Proposition 4.5, we can derive that G has no cycle that contains two vertices v and v' such that $v \in V_i$ and $v' \in V_j$ with $1 \leq i < j \leq |R(G)|$; otherwise, the label of v' can be updated. Thus A_3 is an FAS of G if and only if the graph G_{124}^i comprising the vertex-set V_i and arc set $A_1^i \cup A_2^i \cup A_4^i$ is acyclic for any $1 \leq i \leq |R(G)|$. Fix an arbitrary integer $1 \leq i \leq |R(G)|$. In the following discussion, we first show that G_{12}^i comprising the vertex-set V_i and arc set $A_1^i \cup A_2^i$ is acyclic by induction on the arcs in A_2^i. Note that Proposition 4.5 holds for any subgraph of G as it holds for G.

Recall that G_1^i is acyclic by Lemma 4.8. Assume that $G_1^i \cup A'$ is acyclic, where A' is a subset of A_2^i. Let $v \to v'$ be an arbitrary arc in $A_2^i \setminus A'$. Property (2) given in Lemma 4.7 shows that there is a path starting from v to v' in $G_1^i \cup A'$. By the assumption that $G_1^i \cup A'$ is acyclic, there is no path starting from v' to v in $G_1^i \cup A'$. Thus the addition of the arc $v \to v'$ into $G_1^i \cup A'$ does not result in a cycle. Therefore, $G_1^i \cup A' \cup \{v \to v'\}$ is still acyclic. Summarizing the above analysis gives that G_{12}^i is acyclic.

1. **when** $r = 1, 2, \cdots, N$ **do**
2. **begin synchronous round**
   /* Message Receiving Phase */
3. **for** each message $msg(v', v, L(v'))$ sent from v' to v **do**
4. **if** $L(v) \gg_{lo} L(v')$ **then**
5. send message $msg(v, master, < v', v >)$;
   /* Message Sending Phase */
6. **for** each vertex $v_j \in D_G^d(v)$ **do**
7. send message $msg(v, v_j, L(v))$ to v_j.

Figure 3: The pseudo-code of the Pregel implementation of the distributed synchronous algorithm Alg-FAS **on each machine with assigned vertex** $v \in V(G)$**.**

Now we show that G_{124}^i is acyclic by induction on the arcs in A_4^i. Assume that $G_{12}^i \cup A'$ is acyclic, where A' is an arbitrary subset of A_4^i. Let $v \to v'$ be an arbitrary arc in $A_4^i \setminus A'$. Recall the fact that the labels of v and v' satisfy Property (4) given in Lemma 4.7. Then combining the fact with that Proposition 4.5 holds for $G_{12}^i \cup A'$ gives that there is no directed path from v' to v in $G_{12}^i \cup A'$. Thus $G_{12}^i \cup A' \cup \{v \to v'\}$ is still acyclic. Summarizing the above analysis gives that G_{124}^i is acyclic.

Overall, A_3 is an FAS of G. □

Now we are ready to give the algorithm named Alg-FAS to find the FAS specified by A_3, please refer to Fig. 3. It is worthy to point out that for each message considered in Alg-FAS, except the receiver identifier and the message content, it also contains the sender identifier. Additionally, the information about the arcs of A_3 are directly sent to the master (for more details about the master, please refer to [18]). It is easy to get the following theorem.

Theorem 4.10. *Given a cyclic data lineage* G *with* n *vertices, in which each vertex has a label specified by* Alg-Label*,* Alg-FAS *takes 3 supersteps to find the arcs of* A_3*, which comprise an FAS of* G*.*

Proof. By the mechanism of the Pregel, we can get that the master gets the information about the arcs of A_3 at the 3-rd superstep. Then by Lemma 4.9 we have the correctness of the theorem. □

5 AN IMPROVED DISTRIBUTED SYNCHRONOUS ALGORITHM FOR THE COST-APPORTIONMENT PROBLEM

The section starts with an introductory example to show the main idea of our method to solve the Cost-Apportionment problem on cyclic data lineage then formally gives the details of the method.

5.1 Introductory Example

As an introductory example, let G_1 be a simple data lineage with exactly one cycle, which contains an arc $v \to w$. Firstly, we modify the structure of G_1 by the following step and obtain a new data lineage G_2.

Modification Step. Create a *sink* vertex $SK(v, w)$ and a *source* vertex $SC(v, w)$ for $v \to w$, and replace the arc $v \to w$ with a 3-arc path $v \to SK(v, w) \to SC(v, w) \to w$; let $W_a(v \to SK(v, w)) = W_a(v \to w)$, $W_a(SK(v, w) \to SC(v, w)) = W_a(SC(v, w) \to w) = 1$, and $W_v(SK(v, w)) = W_v(SC(v, w)) = 0$ (note that $W_a(SK(v, w) \to$

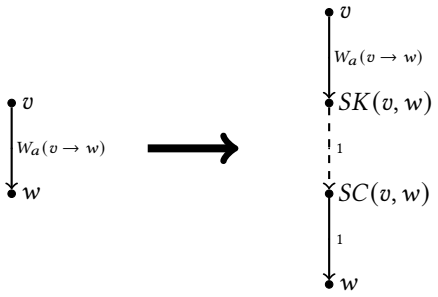


Figure 4: An illustration of Modification Step.

$SC(v, w)$) and $W_a(SC(v, w) \to w)$ can be any value as each of $SK(v, w)$ and $SC(v, w)$ has exactly one direct downstream vertex).

An illustration of Modification Step is given in Fig. 4. Let C_{G_1} and C_{G_2} be the complete cost-apportionment approaches obtained by Alg-CA on G_1 and G_2 with infinity supersteps, respectively (i.e, C_{G_1} and C_{G_2} are the complete cost-apportionment approaches that the Cost-Apportionment problem on G_1 and G_2 look for, respectively). By Lemma 3.3 we have the following corollary for C_{G_1} and C_{G_2}.

Corollary 5.1. *$C_{G_1}(v) = C_{G_2}(v)$ for any leaf $v \in L(G_1)$, where $L(G_1) = L(G_2)$.*

By Corollary 5.1, now we can move our attention from G_1 to G_2. To analyze the cost-apportionment process specified by Alg-CA on G_2 from a new perspective, the arcs in G_2 are *artificially* divided into two types: **(Type-I).** the arcs in $A(G_2) \setminus \{SK(v, w) \to SC(v, w)\}$; **(Type-II).** the arc $SK(v, w) \to SC(v, w)$. Then the cost-apportionment process specified by Alg-CA on G_2 can be regarded as a series of iterations, each of which contains two phases: **Phase (I).** the cost-apportionment along the arcs of Type-I; **Phase (II).** the cost-apportionment along the arc $SK(v, w) \to SC(v, w)$ of Type-II. For ease of notation, denote by G_2' the data lineage obtained by removing the arc $SK(v, w) \to SC(v, w)$ from G_2. Note that G_2' is an DAG with $L(G_2') = L(G_2) \cup \{SK(v, w)\}$, thus Phase (I) specified by Alg-CA on G_2' can be realized efficiently.

By the above discussion, the cost-apportionment process on G_2 with infinity supersteps can be depicted as follows, where the function C_t defined on the vertices of G_2 represent the costs on them after the execution of the t-th iteration ($t \geq 1$).

1. initialize the cost on each vertex in G_2 by its weight;
2. when $t = 1, 2, \cdots, N = +\infty$
  2.1. call Alg-CA on G_2'; // Phase (I)
  2.2. apportion the cost on $SK(v, w)$ to $SC(v, w)$; // Phase (II)

Within the 1-st iteration, Step 2.1 apportions the initial costs on the vertices of G_2' to the leaves, and Step 2.2 apportions the cost on $SK(v, w)$ to $SC(v, w)$. Thus we have that all internal vertices and roots of G_2' except $SC(v, w)$ have no cost after the execution of the 1-st iteration, i.e., $C_1(v) = 0$ for any $v \in R(G_2') \cup I(G_2') \setminus \{SC(v, w)\}$. Moreover, it is easy to see that the conclusion applies for any t-th iteration with $t \geq 1$, i.e., $C_t(v) = 0$ for any $v \in R(G_2') \cup I(G_2') \setminus \{SC(v, w)\}$ and any $t \geq 1$.

Now we consider the t-th iteration with $t \geq 2$. As mentioned above, all internal vertices and roots except $SC(v, w)$ have no cost before the execution of Step 2.1 (specifically, the cost on $SC(v, w)$ is $C_{t-1}(SC(v, w))$). By Step 2.1 and 2.2, part of the cost $C_{t-1}(SC(v, w))$ (i.e., $C_t(SC(v, w))$) returns back to $SC(v, w)$, and the others are all apportioned to the leaves of $L(G_2)$. Because of Property (I) given in Section 2, $C_t(SC(v, w)) < C_{t-1}(SC(v, w))$. Thus with the value of t increasing, the costs remained on the internal vertices and roots of G_2' decrease, approaching 0. Fortunately, as (the structure of) G_2' is fixed, the proportions of the costs apportioned from $SC(v, w)$ to the leaves of G_2' are fixed within the t-th iteration for any $t \geq 2$. Moreover, the cost-apportionment process specified by Step 2.1 within the t-th iteration can be depicted with several mathematical expressions given below.

Let $L(G_2) = \{l_1, l_2, \ldots, l_q\}$, and vector $F = [\alpha_1, \ldots, \alpha_q]^T$ be the *cost-transition vector* of $SC(v, w)$ for $L(G_2) = L(G_2') \setminus \{SK(v, w)\}$, in which each $0 \leq \alpha_i < 1$ is the proportion of the cost on $SC(v, w)$ apportioned to $l_i \in L(G_2)$ within the t-th iteration. Denote by $0 < \alpha < 1$ the proportion of the cost apportioned from $SC(v, w)$ to $SK(v, w)$ within the t-th iteration. Obviously, $\alpha + \sum_{i=1}^{q} \alpha_i = 1$. Note that $0 < \alpha < 1$, $\sum_{i=1}^{q} \alpha_i > 0$ and each $0 \leq \alpha_i < 1$ because G_1 is cyclic and assumed to meet Property (I) given in Section 2. By the notations given above, we have the following expressions.

$$C_t(SC(v, w)) = \alpha \cdot C_{t-1}(SC(v, w)) = \alpha^{t-1} \cdot C_1(SC(v, w)) \quad (1)$$

$$\text{and } C_t(L(G_2)) = C_{t-1}(SC(v, w)) \cdot F + C_{t-1}(L(G_2)), \quad (2)$$

where $C_t(L(G_2)) = [C_t(l_1), \cdots, C_t(l_q)]^T$. Then combining Expressions (1) and (2), we have

$$\begin{aligned}
&C_t(L(G_2)) \\
&= C_{t-1}(SC(v, w)) \cdot F + C_{t-1}(L(G_2)) \\
&= \alpha^{t-2} \cdot C_1(SC(v, w)) \cdot F + C_{t-1}(L(G_2)) \\
&= \alpha^{t-2} \cdot C_1(SC(v, w)) \cdot F + [C_{t-2}(SC(v, w)) \cdot F + C_{t-2}(L(G_2))] \\
&= (\alpha^{t-2} + \alpha^{t-3}) \cdot C_1(SC(v, w)) \cdot F + C_{t-2}(L(G_2)) \\
&\vdots \\
&= (\alpha^{t-2} + \alpha^{t-3} + \cdots + 1) \cdot C_1(SC(v, w)) \cdot F + C_1(L(G_2)) \\
&= \frac{1 - \alpha^{t-1}}{1 - \alpha} \cdot C_1(SC(v, w)) \cdot F + C_1(L(G_2)).
\end{aligned}$$

Obviously, if the iteration number t approaches $+\infty$ (recall that $0 < \alpha < 1$), then

$$C_{+\infty}(L(G_2)) = \frac{C_1(SC(v, w))}{1 - \alpha} \cdot F + C_1(L(G_2)). \quad (3)$$

With respect to α and F, they can be obtained by the following way: Assign cost 1 to $SC(v, w)$ and 0 to the other vertices in G_2', and call Alg-CA on G_2'. Then it is easy to see that the value of the cost apportioned to each leaf $l \in L(G_2')$ is exactly the proportion of the cost on $SC(v, w)$ apportioned to l.

Summarizing the above analysis, the values of α and F are critical to *calculate* the complete cost-apportionment approach that the Cost-Apportionment problem on G_2 or G_1 looks for.

5.2 Generalized Method

Now we generalize the above method to obtain the complete cost-apportionment approach that the Cost-Apportionment problem on the data lineage G looks for, where G may contain more than one

cycle. First of all, we construct a new data lineage G_1 based on G and an FAS A_3 obtained by Alg-Label and Alg-FAS for G, using the operation given below.

Generalized Modification Operation: For each arc $v \to w$ in A_3, replacing it with a 3-arc path $v \to SK(v,w) \to SC(v,w) \to w$, where $SK(v,w)$ and $SC(v,w)$ are two new vertices created for the arc $v \to w$ that are called *sink* vertex and *source* vertex, respectively, and the arc $SK(v,w) \to SC(v,w)$ is called *sink-source arc*. Additionally, let $W_v(SK(v,w)) = W_v(SC(v,w)) = 0$, $W_a(v \to SK(v,w)) = W_a(v \to w)$ and $W_a(SK(v,w) \to SC(v,w)) = W_a(SC(v,w) \to w) = 1$.

Denote by V_{sk} and V_{sc} the sets containing all sink vertices and source vertices in G_1, respectively, and by A_{SS} the set containing all sink-source arcs in G_1. Additionally, let ϕ be the bijectively mapping relation from V_{sc} to V_{sk}, i.e., for each vertex $v \in V_{sc}$, $\phi(v)$ represents the sink vertex such that $\phi(v) \to v$ is the corresponding sink-source arc in A_{SS}.

Let C_G and C_{G_1} be the complete cost-apportionment approaches obtained by Alg-CA on G and G_1 with infinity supersteps, respectively (i.e, C_G and C_{G_1} are the complete cost-apportionment approaches that the Cost-Apportionment problem on G and G_1 look for, respectively). By Lemma 3.3, we have the following corollary.

Corollary 5.2. *$C_G(v) = C_{G_1}(v)$ for any vertex $v \in L(G)$, where $L(G) = L(G_1)$.*

By Corollary 5.2, now we move our attention from G to G_1. Similarly, the arcs in G_1 are *artificially* divided into two types: **(Type-I).** the arcs in $A(G_1) \backslash A_{SS}$; **(Type-II).** the sink-source arcs in A_{SS}. Then the cost-apportionment process specified by the algorithm Alg-CA on G_1 can be regarded as a series of iterations, each of which contains two phases: **Phase (I).** the cost-apportionment along the arcs of Type-I; **Phase (II).** the cost-apportionment along the arcs of Type-II.

For ease of notation, denote by G_1' the data lineage obtained by removing the arcs of A_{SS} from G_1. Note that $L(G_1') = L(G_1) \cup V_{sk}$, and G_1' is an DAG as A_3 is an FAS of G. The cost-apportionment process specified by the algorithm Alg-CA on G_1 with infinity supersteps can be depicted as follows, where the function C_t defined on the vertices of G_1 represent the costs on them after the execution of the t-th iteration.

1. initialize the cost on each vertex in G_1 by its weight;
2. when $t = 1, 2, \cdots, N = +\infty$
  2.1. call Alg-CA on G_1'; // Phase (I)
  2.2. for each source vertex $v \in V_{sc}$ do // Phase (II)
    apportion the cost on $\phi(v)$ to v;

Observe that within the t-th iteration for any $t \geq 2$, the execution of the algorithm Alg-CA on G_1' specified by Step 2.1 only considers the costs on the source vertices and apportions them to the leaves in G_1' as all internal vertices in G_1' have costs 0 at that moment. Fortunately, as the graph structure of G_1' is fixed, the proportions of the costs apportioned from each source vertex to the leaves of G_1' are fixed, and the cost-apportionment process can be depicted by several mathematical expressions given below.

Let $V_{sc} = \{v_1, \ldots, v_p\}$, $V_{sk} = \{\phi(v_1), \ldots, \phi(v_p)\}$ and $L(G_1) = L(G_1') \setminus V_{sk} = \{l_1, \ldots, l_q\}$. Let matrix A be the *cost-transition matrix* of V_{sc} for V_{sk}, in which each element $A[i,j]$ represents the proportion of the cost on the source vertex v_j apportioned to the sink vertex $\phi(v_i) \in V_{sk}$ with $1 \leq i, j \leq p$. Similarly, let matrix B be the *cost-transition matrix* of V_{sc} for $L(G_1)$, in which each element $B[i,j]$ represents the proportion of the cost on the source vertex v_j apportioned to the leaf $l_i \in L(G_1)$ with $1 \leq i \leq q$ and $1 \leq j \leq p$.

Thus each column vector $A[-,j]$ $(1 \leq j \leq p)$ denotes the *cost-transition vector* of the source vertex v_j for V_{sk}, and each column vector $B[-,j]$ $(1 \leq j \leq p)$ denotes the *cost-transition vector* of the source vertex v_j for $L(G_1)$. It is easy to see that $\sum_{i=1}^{p} A[i,j] + \sum_{i=1}^{q} B[i,j] = 1$ for any $1 \leq j \leq p$. Due to Property (I) of G (given in Section 2), the following lemma obviously holds.

$$\begin{array}{c} \\ \phi(v_1) \\ \vdots \\ \phi(v_i) \\ \vdots \\ \phi(v_p) \end{array} \begin{array}{c} \begin{array}{ccccc} A[-,1] & \cdots & A[-,j] & \cdots & A[-,p] \end{array} \\ \begin{pmatrix} A[1,1] & \cdots & A[1,j] & \cdots & A[1,p] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ A[i,1] & \cdots & A[i,j] & \cdots & A[i,p] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ A[p,1] & \cdots & A[p,j] & \cdots & A[p,p] \end{pmatrix} \end{array} = A$$

$$\begin{array}{c} \\ l_1 \\ \vdots \\ l_i \\ \vdots \\ l_q \end{array} \begin{array}{c} \begin{array}{ccccc} B[-,1] & \cdots & B[-,j] & \cdots & B[-,p] \end{array} \\ \begin{pmatrix} B[1,1] & \cdots & B[1,j] & \cdots & B[1,p] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ B[i,1] & \cdots & B[i,j] & \cdots & B[i,p] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ B[q,1] & \cdots & B[q,j] & \cdots & B[q,p] \end{pmatrix} \end{array} = B$$

Lemma 5.3. *Any element $A[i,j]$ at the i-th row and j-th column of A has $0 \leq A[i,j] < 1$ for any $1 \leq i, j \leq p$, and $0 \leq \sum_{i=1}^{p} A[i,j] < 1$ for any $1 \leq j \leq p$.*

Let $C_t(S)$ be the vector with size $|S| \times 1$ consisting of the values of $C_t(v)$ for all vertices $v \in S$. That is, $C_1(V_{sc}) = [C_1(v_1), \cdots, C_1(v_p)]^T$ and $C_1(L(G_1)) = [C_1(l_1), \cdots, C_1(l_q)]^T$. Then we can derive that

$$C_t(V_{sc}) = A \cdot C_{t-1}(V_{sc}) = \cdots = A^{t-1} \cdot C_1(V_{sc}) \tag{4}$$

and

$$\begin{aligned} C_t(L(G_1)) &= B \cdot C_{t-1}(V_{sc}) + C_{t-1}(L(G_1)) \\ &= B \cdot A^{t-2} \cdot C_1(V_{sc}) + C_{t-1}(L(G_1)) \\ &= B \cdot A^{t-2} \cdot C_1(V_{sc}) + B \cdot C_{t-2}(V_{sc}) + C_{t-2}(L(G_1)) \\ &= B \cdot (A^{t-2} + A^{t-3}) \cdot C_1(V_{sc}) + C_{t-2}(L(G_1)) \\ &\vdots \\ &= B \cdot (A^{t-2} + A^{t-3} + \ldots + I_p) \cdot C_1(V_{sc}) + C_1(L(G_1)), \end{aligned} \tag{5}$$

where I_p is the unit matrix with size $p \times p$. It is easy to see that the key point to calculate Expression (5) is to get the value of

$$A^{t-2} + A^{t-3} + \ldots + I_p. \tag{6}$$

If $I_p - A$ is *invertible* then Expression (6) equals $\frac{I_p - A^{t-1}}{I_p - A}$. Moreover, if t approaches $+\infty$ then Expression (6) equals $\frac{I_p}{I_p - A}$ as all elements in A are at least 0 but less than 1 due to Lemma 5.3. Unfortunately,

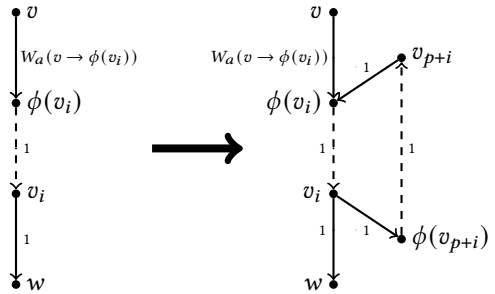


Figure 5: An illustration of Additional Step. The dashed arcs are sink-source ones.

it is unclear whether or not $I_p - A$ is *invertible*. Thus we introduce an additional step below to further modify the structure of G_1.

Additional Step. For each sink-source arc $\phi(v_i) \to v_i$ in G_1, create a new source vertex v_{p+i} and a new sink vertex $\phi(v_{p+i})$, and add a 3-arc path $v_i \to \phi(v_{p+i}) \to v_{p+i} \to \phi(v_i)$. Additionally, let $W_v(v_{p+i}) = W_v(\phi(v_{p+i})) = 0$ and $W_a(v_i \to \phi(v_{p+i})) = W_a(\phi(v_{p+i}) \to v_{p+i}) = W_a(v_{p+i} \to \phi(v_i)) = 1$.

An illustration of Additional Step is given in Fig. 5. For ease of notation, denote by G_2 the data lineage obtained by Additional Operation on G_1, and by $V'_{sc} = \{v_{p+1}, \ldots, v_{2p}\}$ and $V'_{sk} = \{\phi(v_{p+1}), \ldots, \phi(v_{2p})\}$ the sets containing the newly created source vertices and sink vertices in G_2, respectively (assume that the mapping relation ϕ also holds from V'_{sc} to V'_{sk}).

Let C_{G_2} be the complete cost-apportionment approach obtained by Alg-CA on G_2 with infinity supersteps (i.e, C_{G_2} is the complete cost-apportionment approach that the Cost-Apportionment problem on G_2 looks for). Then by Lemmata 3.3 and 3.4, and Corollary 5.2, we have the following corollary.

Corollary 5.4. *$C_G(v) = C_{G_1}(v) = C_{G_2}(v)$ for any vertex $v \in L(G)$, where $L(G) = L(G_1) = L(G_2)$.*

Similarly, the cost-apportionment process of the algorithm Alg-CA on G_2 can be depicted as follows, where G'_2 is the data lineage obtained by removing the sink-source arcs in G_2.

1. initialize the cost on each vertex in G_2 by its weight;
2. when $t = 1, 2, \cdots, N = +\infty$
  2.1. call Alg-CA on G'_2; // Phase (I)
  2.2. for each source vertex $v \in V_{sc} \cup V'_{sc}$ do // Phase (II)
    apportion the cost on $\phi(v)$ to v;

Let $\mathcal{A}$ be the cost-transition matrix of $V_{sc} \cup V'_{sc}$ for $V_{sk} \cup V'_{sk}$, in which each element $\mathcal{A}[i, j]$ $(1 \le i, j \le 2p)$ is the proportion of the cost on v_j apportioned to $\phi(v_i)$ by Alg-CA on G'_2. Let $\mathcal{B}$ be the cost-transition matrix of $V_{sc} \cup V'_{sc}$ for $L(G_1) = L(G'_2) \setminus (V_{sk} \cup V'_{sk})$, in which each element $\mathcal{B}[i, j]$ $(1 \le i \le q$ and $1 \le j \le 2p)$ is the proportion of the cost on v_j apportioned to l_i by Alg-CA on G'_2.

Lemma 5.5.

$$\mathcal{A} = \left[\begin{array}{c|c} \frac{A}{2} & I_p \\ \hline \frac{I_p}{2} & 0_{p,p} \end{array}\right],$$

and

$$\mathcal{B} = \left[\begin{array}{c|c} \frac{B}{2} & 0_{q,p} \end{array}\right],$$

where I_p denotes the identity matrix with size $p \times p$, and $0_{p,p}$ and $0_{q,p}$ denote the zero matrixes with size $p \times p$ and $q \times p$, respectively.

Proof. Comparing the structures of G'_1 and G'_2, we find that each source vertex $v_i \in V_{sc}$ $(1 \le i \le p)$ has exactly one direct downstream vertex w in G'_1 (see Fig. 5 for illustration), but has an extra direct downstream vertex $\phi(v_{p+i})$ in G'_2. As the weights on the arc $v_i \to w$ and $v_i \to \phi(v_{p+i})$ are both 1, one half of the cost on v_i is apportioned to $\phi(v_{p+i})$, and the other half is apportioned to w and then to the its downstream vertices.

In the following, we show that the subgraph $G'_1[D_{G'_1}(w)]$ of G'_1 induced by the vertex-set $D_{G'_1}(w)$ is the same as the subgraph $G'_2[D_{G'_2}(w)]$ of G'_2 induced by the vertex-set $D_{G'_2}(w)$. Observe that for any vertex $v_j \in V_{sc}$ $(1 \le j \le p)$, there is no arc leading to v_j in G'_2, thus $v_j \notin D_{G'_2}(w)$. Consequently, $\phi(v_{p+j}) \notin D_{G'_2}(w)$ and $v_{p+j} \notin D_{G'_2}(w)$ for any $1 \le j \le p$, i.e., $(V'_{sc} \cup V'_{sk}) \cap D_{G'_2}(w) = \emptyset$. As a result, we have that $D_{G'_1}(w) = D_{G'_2}(w)$. Combining the conclusion with the structural difference between G'_1 and G'_2, we can derive the claim that the subgraph $G'_1[D_{G'_1}(w)]$ of G'_1 induced by the vertex-set $D_{G'_1}(w)$ is the same as the subgraph $G'_2[D_{G'_2}(w)]$ of G'_2 induced by the vertex-set $D_{G'_2}(w)$.

Thus the proportions of the costs apportioned from w to the leaves of $D_{G'_2}(w) \cap L(G'_2)$ in G'_2 are the same as that of the costs apportioned from w to the leaves of $D_{G'_1}(w) \cap L(G'_1)$ in G'_1. Summarizing the above analysis, the cost-transition matrixes of V_{sc} for V_{sk} and $L(G_1) = L(G_2)$ in G'_2 are $\frac{A}{2}$ and $\frac{B}{2}$, respectively.

As mentioned above, each source vertex $v_i \in V_{sc}$ $(1 \le i \le p)$ apportions one half of its cost to $\phi(v_{p+i})$. Combining the conclusion with the fact that each sink vertex $\phi(v_{p+i}) \in V'_{sk}$ has exactly one direct upstream vertex v_i, we can derive that the cost-transition matrix of V_{sc} for V'_{sk} in G'_2 is $\frac{I_p}{2}$.

By the structure of G'_2, each source vertex $v_i \in V'_{sc}$ $(p + 1 \le i \le 2p)$ apportions its cost to $\phi(v_{i-p})$ totally. Thus the cost-transition matrixes of V'_{sc} for V_{sk}, V'_{sk} and $L(G_1) = L(G_2)$ in G'_2 are I_p, $0_{p,p}$ and $0_{q,p}$, respectively. □

Lemma 5.6. *Each element of $\mathcal{A}^2$ is less than 1 but not less than 0.*

Proof. Each element of A is not greater than 1 by Lemma 5.3. For ease of notation, denote by $\mathcal{A}[i, -]$ and $\mathcal{A}[-, i]$ the i-th row vector and column vector in $\mathcal{A}$ $(1 \le i \le 2p)$, respectively. Consider the element $\mathcal{A}^2[i, j]$ at the i-th row and j-th column in $\mathcal{A}^2$, with $1 \le i, j \le 2p$.

Case (I). $1 \le i, j \le p$. For the column vector $\mathcal{A}[-, j]$, it satisfies two properties: (1). $\sum_{s=1}^{p} \mathcal{A}[s, j] < 1/2$; (2). $\mathcal{A}[s, j] = 0$ for all $p + 1 \le s < p + j$ and $p + j < s \le 2p$ and $\mathcal{A}[p + j, j] = 1/2$. For the row vector $\mathcal{A}[i, -]$, it satisfies two properties: (1). all element $\mathcal{A}[i, s]$ are less than $1/2$ for any $1 \le s \le p$; (2). $\mathcal{A}[i, s] = 0$ for all $p + 1 \le s < p + i$ and $p + i < s \le 2p$ and $\mathcal{A}[i, p + i] = 1$.

Combining the above properties, we can derive that

$$\begin{aligned} \mathcal{A}[i, -] \times \mathcal{A}[-, j] &= \sum_{s=1}^{p} \mathcal{A}[i, s] \times \mathcal{A}[s, j] + \frac{1}{2} \times \mathcal{A}[i, p + j] \\ &\le \frac{1}{2} \sum_{s=1}^{p} \mathcal{A}[s, j] + \frac{1}{2} \le \frac{3}{4}. \end{aligned} \tag{7}$$

Case (II). $1 \le i \le p$ and $p+1 \le j \le 2p$. All elements in the column vector $\mathcal{A}[-, j]$ are 0, except the element $\mathcal{A}[j-p, j]$ whose value is 1. Thus

$$\mathcal{A}[i, -] \times \mathcal{A}[-, j] = \mathcal{A}[i, j-p] \times \mathcal{A}[j-p, j] < 1/2.$$

Case (III). $p+1 \le i \le 2p$ and $1 \le j \le p$. All elements in the row vector $\mathcal{A}[i, -]$ are 0, except the element $\mathcal{A}[i, i-p]$ whose value is 1/2. Thus

$$\mathcal{A}[i, -] \times \mathcal{A}[-, j] = \frac{1}{2} \times \mathcal{A}[i-p, j] < 1/4.$$

Case (IV). $p+1 \le i, j \le 2p$. Using the reasoning similar to that given for Case (II), we derive that

$$\mathcal{A}[i, -] \times \mathcal{A}[-, j] = \frac{1}{2} \times I_p[i-p, j-p] < 1/2.$$

Summarizing the above analysis, we have the claim. □

Lemma 5.7. *The matrix $I_{2p} - \mathcal{A} = \mathcal{M}$ with size $2p \times 2p$ is invertible.*

Proof. As before, for any $1 \le i \le 2p$, the column vector $\mathcal{A}[-, i]$ is the cost-transition vector of the source vertex v_i for $V_{sk} \cup V'_{sk}$.

By Lemma 5.5, for any $1 \le i \le p$, $\mathcal{M}[i, i] > 1/2$, $\mathcal{M}[p+i, i] = -1/2$, $\mathcal{M}[p+j, i] = 0$ for any $1 \le j < i$ and $i < j \le p$, and $\mathcal{M}[j, i] = -A[j, i]/2 > -1/2$ for any $1 \le j < i$ and $i < j \le p$. Moreover, for any $p \le i \le 2p$, $\mathcal{M}[i-p, i] = -1$, $\mathcal{M}[i, i] = 1$, and all the other elements in the column vector $\mathcal{M}[-, i]$ are 0.

Now we are ready to show that the column vectors in the matrix $\mathcal{M}$ are linear independence based on the above two claims.

Consider a column vector $\mathcal{M}[-, i]$ $(1 \le i \le p)$, and assume that it is linear dependent with the column vectors $\mathcal{M}[-, j]$ with $j \in S$, where S is a subset of $[1, i-1] \cup [i+1, 2p]$ (i.e., $\mathcal{M}[-, i] = \sum_{j \in S} \gamma_j \cdot \mathcal{M}[-, j]$, and γ_j are constants for all $j \in S$). Based on the above two claims, we have that $S = \{p+i\}$ and $\gamma_{p+i} = -1/2$. However, $\mathcal{M}[i, i] > 1/2 \ne \gamma_{p+i} \cdot \mathcal{M}[i, p+i]$, where $\gamma_{p+i} \cdot \mathcal{M}[i, p+i] = 1/2$. Thus the assumption is incorrect.

For any column vector $\mathcal{M}[-, i]$ $(p+1 \le i \le 2p)$, using the reasoning similar to that given above, we can also derive that it is not linear dependent with the other column vectors. □

Let C'_t be a function defined on the vertices of G'_2 that represent the costs on them after the execution of the t-th iteration. By generalizing Expression (5), we have

$$\begin{aligned} & C'_t(L(G_2)) \\ &= \mathcal{B} \cdot (\mathcal{A}^{t-2} + \mathcal{A}^{t-3} + \ldots + I_{2p}) \cdot C'_1(V_{sc} \cup V'_{sc}) + C'_1(L(G_2)) \\ &= \mathcal{B} \cdot (\frac{I_{2p} - \mathcal{A}^{t-1}}{I_{2p} - \mathcal{A}}) \cdot C'_1(V_{sc} \cup V'_{sc}) + C'_1(L(G_2)). \end{aligned} \tag{8}$$

When t approaches $+\infty$, by Lemmata 5.6 and 5.7, we have

$$C'_{+\infty}(L(G_2)) = \mathcal{B} \cdot (\frac{I_{2p}}{I_{2p} - \mathcal{A}}) \cdot C'_1(V_{sc} \cup V'_{sc}) + C'_1(L(G_2)). \tag{9}$$

Furthermore, we have some observations for Expression (9) that are formulated in the following two lemmata.

Lemma 5.8. *For any $1 \le i \le p$, $C'_1(v_i) = C_1(v_i)$ and $C'_1(v_{p+i}) = 0$ (note that $v_i \in V_{sc}$ and $v_{p+i} \in V'_{sc}$).*

Proof. We first analyze the value of $C'_1(v_{p+i})$. As the weight of v_i is 0, $\phi(v_{p+i})$ gets no cost by the execution of Step 2.1 within the 1-st iteration. Consequently, v_{p+i} gets no cost by the execution of Step 2.2 within the 1-st iteration. Therefore, $C'_1(\phi(v_{p+i})) = 0$ and $C'_1(v_{p+i}) = 0$.

Now we consider the value of $C'_1(v_i)$. As all vertices of V'_{sc} have weight 0, the costs apportioned to $\phi(v_i)$ by the application of Step 2.1 within the 1-st iteration are from $U_{G'_2}(\phi(v_i)) \setminus V'_{sc}$, where $U_{G'_2}(\phi(v_i)) \setminus V'_{sc} = U_{G'_1}(\phi(v_i))$. Furthermore, it is easy to see that the subgraph of G'_2 induced by $U_{G'_2}(\phi(v_i)) \setminus V'_{sc}$ is the same as the one of G'_1 induced by $U_{G'_1}(\phi(v_i))$. Combining the above conclusions gives that $C'_1(v_i) = C_1(v_i)$. □

Lemma 5.9. $C'_1(L(G_2)) = C_1(L(G_1))$.

Proof. For each vertex of $v_{p+i} \in V'_{sc}$ $(1 \le i \le p)$, its weight is 0 and has no direct upstream vertex in G'_2, thus it has no contribution to the cost-apportionment process specified by the execution of Alg-CA within the 1-st iteration. Similarly, for each vertex $\phi(v_{p+i}) \in V_{sk}$ $(1 \le i \le p)$, itself and its unique direct upstream vertex have weight 0 and it has no direct downstream vertex in G'_2, thus it also has no contribution to the cost-apportionment process specified by the execution of Alg-CA within the 1-st iteration. That is, all vertices of $V'_{sc} \cup V'_{sk}$ actually do not participate the cost-apportionment process specified by the execution of Alg-CA within the 1-st iteration.

Combining the above conclusion with the structural difference between G'_1 and G'_2, we can derive that $C'_1(L(G_2)) = C_1(L(G_1))$. □

By Expression (9) and Lemmata 5.5, 5.8 and 5.9, we find that to get the complete cost-apportionment approach of G we only need to construct the data lineage G_1 and G'_1, and obtain the cost-transition matrixes A and B, and $C_1(V_{sc})$ and $C_1(L(G_1))$.

5.3 Theoretical Performance of Alg-CA-Imp

Before giving the algorithm for the Cost-Apportionment problem on cyclic data lineage, we first propose an efficient way to obtain the cost-transition matrixes A and B, and $C_1(V_{sc})$ and $C_1(L(G_1))$. A trivial way to get A and B is: For the source vertex $v_j \in V_{sc}$, let the cost on v_j be 1, and the costs on the vertices of $V(G'_1) \setminus \{v_j\}$ be 0; then call Alg-CA on G'_1 to obtain $A_{-,j}$ and $B_{-,j}$. That is, the trivial way calls the algorithm Alg-CA on G'_1 $p = |V_{sc}|$ many times to get A and B, which is very time-consuming. Therefore, it is worthy to give an efficient algorithm, which is a variant of the algorithm Alg-CA, called Alg-CA1, whose aim is to get A, B, $C_1(V_{sc})$ and $C_1(L(G_1))$ by exactly one call on G'_1.

W.l.o.g., assume that each vertex $v \in V$ has a unique id i ranging from 1 to n, in particular, each vertex $v_i \in V_{sc}$ has id i ranging from 1 to p. The Pregel implementation of Alg-CA1 on each worker machine with assigned vertex $v \in V(G)$ with id $i \in [1, n]$ has the following initial knowledge: $W_v(v)$, $U^d_G(v)$, $D^d_G(v)$, and the weights of the arcs from v to the vertices in $D^d_G(v)$. Besides, it maintains a variable m with initial value 0 to record the number of messages it received and a *vector variable* $C(v)$ with size $1 \times n$ to record the current cost on v_i, which is initialized as follows: If $v \notin V_{sc}$ then $C(v) = (0, \ldots, W_v(v), \ldots, 0)$; otherwise, $C(v) = (0, \ldots, 1, \ldots, 0)$, in which all elements are 0, except the i-th element with value $W_v(v)$ or 1.

1. **when** $r = 1, 2, \cdots, N$ **do**
2. **begin synchronous round**
   /* Message Receiving Phase */
3. **for** each message $msg(v, x)$ sent to v **do**
4. $C(v) = C(v) + x$ and $m++$;
   /* Message Sending Phase */
5. **if** $m = |U_G^d(v)| \wedge C(v) \neq 0_{1,p} \wedge D_G^d(v) \neq \emptyset$ **then**
6. **for** each vertex $v' \in D_G^d(v)$ **do**
7. let $y = C(v) \cdot W_a(v, v') / \sum_{v'' \in D_G^d(v)} W_a(v, v'')$;
8. send message $msg(v, y)$ to v';
9. $C(v) = 0_{1,n}$.

Figure 6: The pseudo-code of the Pregel implementation of the distributed synchronous algorithm Alg-CA1 **on each worker machine with assigned vertex** $v \in V(G_1')$**.**

It is worthy to point out that the vector variable is very long and consumes memory due the huge scale of the considered data lineage. If each worker machine sends messages to its direct downstream vertices once it gets a message from its direct upstream vertices, then the memory consumption is very huge, resulting bad performance of the algorithm. Thus for Message Sending Phase, we give the condition "$m = |U_G^d(v_i)| \wedge C(v) \neq 0_{1,p} \wedge D_G^d(v) \neq \emptyset$", i.e., if the worker machine has got the messages from its all directed upstream vertices then it can send a message. Note that as the considered data lineage G_1' is acyclic, each work machine sends exactly one message during the execution of the algorithm Alg-CA1.

It is easy to see that by an execution of Alg-CA1 on G_1', where the costs on the vertices are initiallized as above, each vertex of $v \in V_{sk} \cup L(G_1)$ get the cost that is the sum of the last $n-p$ elements in the vector variable $C(v)$. For the sub-vector that consists of the first p elements in the vector maintained by the vertex $\phi(v_i)$ $(1 \leq i \leq p)$ of V_{sk}, it is the same as the i-th row vector $A[i, -]$ of matrix A. Similarly, the sub-vector that consists of the first p elements in the vector maintained by the leaf l_i $(1 \leq i \leq q)$, is the same as the i-th row vector $B[i, -]$ of matrix B.

Theorem 5.10. *For the acyclic data lineage G_1',* Alg-CA1 *solves the Cost-Apportionment problem on G_1' with $L_p + 1$ supersteps, where $L_p \leq n-1$ is the length of the longest path in G_1', and n is the number of vertices in G.*

Proof. The reasoning runs in a similar way to that for Theorem 3.1. As G_1' is acyclic, the apportionment process is unidirectional. For any vertex v in G_1', by the mechanism of Pregel, we have that the worker machine with v assigned gets the messages from its all direct upstream vertices and sends the message at the $l+1$-th superstep, where $l+1$ is the length of the longest path P connecting a root and v in G_1'. Since the internal vertices of P are also internal ones of G, the length l of P is at most $n-1$ (as G contains at least one root and one leaf). □

Now we are ready to present the algorithm Alg-CA-Imp (given in Fig. 7) that can efficiently solve the Cost-Apportionment problem on cyclic data lineage, whose theoretical performance is formulated in the following theorem.

Theorem 5.11. *The algorithm* Alg-CA-Imp *can solve the Cost-Apportionment problem on data lineage G. Specifically, the calls of the algorithms including* Alg-Label, Alg-FAS, Alg-CA1 *take at most $2n+3$ supersteps, where $n = |V(G)|$.*

1. call Alg-Label and Alg-FAS on G to get an FAS of G;
2. **if** the obtained FAS is empty **then**
3. call Alg-CA on G and return the obtained complete cost-apportionment approach;
4. **else**
5. construct data lineage G_1 by the generalized modification operation;
6. initialize the cost of each vertex in G_1 by its weight, and let the costs on the vertices of V_{sc} be 1;
7. call Alg-CA on G_1', obtaining A, B, $C_1(V_{sc})$ and $C_1(L(G_1))$;
8. construct $\mathcal{A}$ and $\mathcal{B}$ based on A and B;
9. calculate the complete cost-apportionment approach based on $\mathcal{A}$, $\mathcal{B}$, $C_1(V_{sc})$ and $C_1(L(G_1))$.

Figure 7: The algorithm Alg-CA-Imp**.**

Proof. For the correctness of the algorithm Alg-CA-Imp for the Cost-Apportionment problem, we can get it by the above discussion for Expression (9) (including Lemmata 5.5, 5.6, 5.7, 5.8 and 5.9).

By Theorems 4.6 and 4.10, we have that the execution of Step 1 takes at most $n+3$ supersteps. By Theorem 5.10, the execution of Step 7 takes at most n supersteps. Therefore, Alg-CA-Imp takes at most $2n+3$ supersteps totally. □

6 EXPERIMENT

In the section, we study the real performance of the algorithm Alg-CA-Imp on the data lineage from Alibaba Inc. As the aim of the experiment is to show the feasibility and efficiency of the algorithm, we just list 12 data lineage with different scales (i.e., the numbers of arcs and vertices), whose details are given in Table 1 (due to privacy, we do not introduce any background of the data lineage). They are separately called $G_1, \ldots, G_{12}$ in order in the following discussion. Every three consecutive data lineage of them with almost the same scale are divided into a group, and the dense subgraphs in them have almost the same density. Note that for the data lineage $G_{10} - G_{12}$ in the 4-th group, although their scales are smaller than that of $G_1 - G_9$, their dense subgraphs have larger densities than that of $G_1 - G_9$. Moreover, the arcs in the 12 data lineage all have weights 1, i.e., the operating cost of each internal vertex is apportioned to its direct downstream vertices evenly. The special setting does not change the behavior of the algorithm. For experimental configuration, the distributed algorithms called by Alg-CA-Imp (including Alg-Label, Alg-FAS, and Alg-CA1) are tested on MaxCompute that is a Pregel liked distributed graph computation platform of Alibaba Inc., and the related matrix calculation is executed on a PC with a 8-core CPU and 12G RAM.

The items "#arcs", "#vertices" and "#leaves" shown in Table 1 represent the numbers of arcs, vertices and leaves in the considered data lineage, respectively. For each execution of Alg-CA-Imp on the 12 data lineage, we separately record the runtime to get an FAS (i.e., the runtime taken by the calls of Alg-Label and Alg-FAS, represented by the item "FAS T(s)"), that to do the cost-apportionment on the DAG (i.e., the runtime taken by the call of Alg-CA1, represented by the item "DAG Apportion T(s)"), that to get the inverse matrix $(I_{2p} - \mathcal{A})^{-1}$ with $\mathcal{A}$ given (represented by the item "Inversion T(s)"), and that to calculate $C'_{+\infty}(L(G_2)$ according to Expression (9) with all

Table 1: The sizes of the data lineage from Alibaba Inc.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
#vertices	28181925	28225292	28267870	712769	712767	712772	581853	581760	581681	122485	122458	121037
#arcs	33611552	33654363	33698118	1921756	1921956	1921667	1678909	1678828	1678582	260502	267700	262561
#leaves	694471	695117	695558	35956	36021	36048	16250	16299	16296	40230	38590	38602

variables given (represented by the item "Exp (9) Cal T(s)") [3]. Note that the runtime given in the four items are all in seconds, and the sum of the corresponding runtime almost equals the total runtime of an execution of Alg-CA-Imp to solve the Cost-Apportionment problem on the considered data lineage.

Besides, it is theoretically meaningful to record the numbers of supersteps taken by the executions of these distributed algorithms. Thus we also record the following information: The item "FAS #SS" represents the number of supersteps taken by the calls of Alg-Label and Alg-FAS to get an FAS of the data lineage, and the item "DAG Apportion #SS" represents the number of supersteps taken by the call of Alg-CA1 to do the cost-apportionment on the data lineage with the arcs in the obtained FAS removed.

Recall that during each execution of Alg-Label, the roots of the considered data lineage are labeled randomly. Thus different executions of Alg-Label may assign different labels to these roots, such that the numbers of supersteps required to find FASs and the sizes of the obtained FASs may be different (the item "|FAS|" represents the size of the obtained FAS). Moreover, different FASs lead to different dimensions of the related matrixes and different runtime to do the matrix calculation. Consequently, we run Alg-CA-Imp on each data lineage 10 times to get its average performance.

Now we analyze Figure 8. For each combination of item and data lineage, there is a vertical line segment with three same symbols (e.g., short segment, solid circle, triangle and rectangle) in the figure. The highest and lowest symbols correspond to the maximum and minimum values for the item obtained by the 10 runs of the algorithm on the considered data lineage. The middle symbol corresponds to the average of remaining 8 values excluding the maximum and minimum ones.

For data lineage $G_{10} - G_{12}$, we find that although their scales are smaller than that of $G_4 - G_9$, the sizes of their FASs obtained are greater than that of $G_4 - G_9$. Moreover, for data lineage $G_1 - G_{11}$, we find that the fluctuating intervals of the sizes of the FASs obtained are very small, specifically, $[-25, +25]$ around the average value. But for data lineage G_{12}, the gap between the sizes of the minimum and maximum FASs found is as high as 204. The essential reason for the two phenomena is that the densities of the dense subgraphs in $G_{10} - G_{12}$ are larger than that of the remaining data lineage, such that the numbers of cycles in $G_{10} - G_{12}$ are greater than that of $G_4 - G_9$, and the FASs and DASs obtained for $G_{10} - G_{12}$ may have great structural difference. Additionally, we find that the values given in the item "DAG Apportionment #SS" are relatively stable, thus the values of "|FAS|" have no direct connection with that of "DAG Apportionment #SS" (even for the results of G_{12}), which violates the intuition that given a data lineage, if the FAS found has a larger size, then the longest simple path in the resulting DAG has a shorter length. Besides, we find that the average time take by each superstep is changeable, which theoretically depends not only on the in-degree and out-degree of the vertex assigned to the worker machine but also on the state of the vertex (e.g., the worker machine does not send any message if it does not receive any message).

For items "FAS #SS" and "DAG Apportion #SS", by Theorems 4.6 and 5.10, we get that $L \leq L_p$ (the definitions of L and L_p can be found at Theorems 4.6 and 5.10). Thus the values of "DAG Apportion #SS" is always larger than that of "FAS #SS" in Fig 8.

For $G_1 - G_3$, we find that the time given in the item "Inversion T(s)" dominates the runtime of the execution of Alg-CA-Imp, as the obtained FAS is very large. For the item "Exp(9) Cal T(s)", the time not only depends on the sizes of the matrixes $\mathcal{A}$ and $\mathcal{B}$, but also depends on the number of non-zero elements in the matrix $\mathcal{B} \cdot (\frac{I_{2p}}{I_{2p} - \mathcal{A}})$, which is given in the item "Non-zero Ele Number". That is why the values of "Exp(9) Cal T(s)" for $G_{10} - G_{12}$ are much larger than that of $G_4 - G_9$.

Overall, our proposed solving approach takes no more than one hour to solve the Cost-Apportionment problem on each data lineage given in Fig 8, which is acceptable considering the huge scales of these data lineage.

7 CONCLUSION

To price the producing cost of data, the paper formulated the Cost-Apportionment problem and proposed a distributed algorithm for it. The theoretical feasibility and correctness of the proposed algorithm were studied in the paper. Besides, the practical efficiency of the algorithm was validated on the data lineage from Alibaba inc.

To our best knowledge, there is no distributed algorithm that can solve the Feedback Arc Set problem efficiently with theoretical guarantee. Moreover, the number of supersteps taken by our proposed algorithm Alg-CA for the Cost-Apportionment problem on an *acyclic* data linege is already linear with the length of the longest simple path in it. Thus Alg-CA cannot be improved anymore from the theoretical perspective, and our solving approach Alg-CA-Imp for the Cost-Apportionment problem proposed in the paper has an almost perfect theoretical performance. The future study on the algorithm is suggested to focus on the specific implementation. Besides, there may be many other application scenarios for the algorithm Alg-CA-Imp, which needs further study.

REFERENCES

[1] Michael Backes, Niklas Grimm, and Aniket Kate. 2015. Data lineage in malicious environments. *IEEE Transactions on Dependable and Secure Computing* 13, 2 (2015), 178–191.

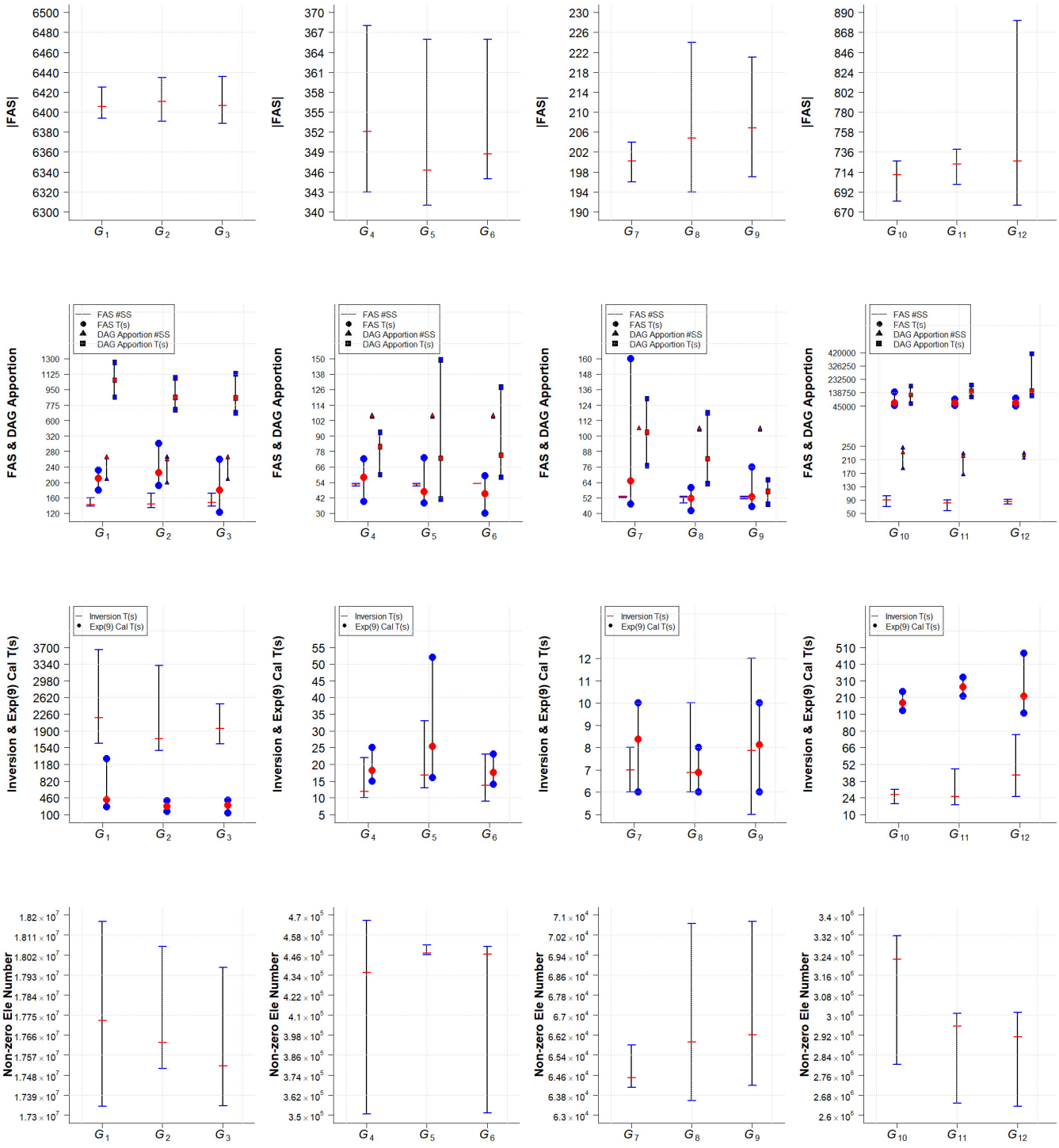
[2] Magdalena Balazinska, Bill Howe, and Dan Suciu. 2011. Data markets in the cloud: An opportunity for the database community. *Proceedings of the VLDB Endowment* 4, 12 (2011), 1482–1485.

[3] Hans L Bodlaender, Fedor V Fomin, Arie MCA Koster, Dieter Kratsch, and Dimitrios M Thilikos. 2012. A note on exact algorithms for vertex ordering problems on graphs. *Theory of Computing Systems* 50, 3 (2012), 420–432.

[4] Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. 2008. A fixed-parameter algorithm for the directed feedback vertex set problem. In

[3] We use the package UJMP to do the matrix inversion and calculation.

Figure 8: The real performance of ALG-CA-IMP on the 12 data lineage from Alibaba Inc.



Proceedings of the fortieth annual ACM symposium on Theory of computing. 177–186.

[5] Derek G Corneil and Richard M Krueger. 2008. A unified view of graph searching. *SIAM Journal on Discrete Mathematics* 22, 4 (2008), 1259–1276.

[6] Pierluigi Crescenzi, Viggo Kann, and M Halldórsson. 1995. A compendium of NP optimization problems.

[7] Yingwei Cui and Jennifer Widom. 2003. Lineage tracing for general data warehouse transformations. *the VLDB Journal* 12, 1 (2003), 41–58.

[8] Kayhan Erciyes. 2013. Distributed graph algorithms for computer networks. (2013).

[9] Guy Even. 1995. Joseph (Seffi) Naor, Baruch Schieber, and Madhu Sudan. *Approximating minimum feedback sets and multi-cuts in directed graphs. Integer Programming and Combinatorial Optimization* (1995), 14–28.

[10] Robert Ikeda and Jennifer Widom. 2009. Data lineage: A survey. *Stanford University Publications. http://ilpubs. stanford. edu* 8090, 918 (2009), 1.

[11] Sanjay Jain and PK Kannan. 2002. Pricing of information products on online servers: Issues, models, and analysis. *Management Science* 48, 9 (2002), 1123–1142.

[12] Kalapriya Kannan, Rema Ananthanarayanan, and Sameep Mehta. 2018. What is my data worth? From data properties to data value. *arXiv preprint arXiv:1811.04665* (2018).

[13] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[14] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. 2015. Query-based data pricing. *Journal of the ACM (JACM)* 62, 5 (2015), 1–44.

[15] E Lawler. 1964. A comment on minimum feedback arc sets. *IEEE Transactions on Circuit Theory* 11, 2 (1964), 296–297.

[16] Fan Liang, Wei Yu, Dou An, Qingyu Yang, Xinwen Fu, and Wei Zhao. 2018. A survey on big data market: Pricing, trading and protection. *Ieee Access* 6 (2018), 15132–15154.

[17] Claudia Loebbecke. 2002. Digital Goods: An Economic Perspective. *Encyclopedia of information systems* 1 (2002), 635–647.

[18] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 135–146.

[19] H Gilbert Miller and Peter Mork. 2013. From data to decisions: a value chain for big data. *It Professional* 15, 1 (2013), 57–59.

[20] Jian Pei. 2020. A survey on data pricing: from economics to data science. *IEEE Transactions on knowledge and Data Engineering* 34, 10 (2020), 4586–4608.

[21] Michel Raynal. 2013. *Distributed algorithms for message-passing systems*. Vol. 500. Springer.

[22] Calvin Rix, Jana Frank, Volker Stich, and Dennis Urban. 2021. Pricing models for data products in the industrial food production. In *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems: IFIP WG 5.7 International Conference, APMS 2021, Nantes, France, September 5–9, 2021, Proceedings, Part V*. Springer, 553–563.

[23] Suthamathy Sathananthan. 2018. Data valuation considering knowledge transformation, process models and data models. In *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 1–5.

[24] James Short and Steve Todd. 2017. What's your data worth? *MIT Sloan Management Review* 58, 3 (2017), 17.

[25] MingJie Tang, Saisai Shao, Weiqing Yang, Yanbo Liang, Yongyang Yu, Bikas Saha, and Dongjoon Hyun. 2019. Sac: A system for big data lineage tracking. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1964–1967.

[26] Ruiming Tang, Huayu Wu, Zhifeng Bao, Stéphane Bressan, and Patrick Valduriez. 2013. The price is right: Models and algorithms for pricing data. In *Database and Expert Systems Applications: 24th International Conference, DEXA 2013, Prague, Czech Republic, August 26-29, 2013. Proceedings, Part II 24*. Springer, 380–394.

[27] Stephen H Unger. 1957. A study of asynchronous logical feedback networks. (1957).

[28] Allison Woodruff and Michael Stonebraker. 1997. Supporting fine-grained data lineage in a database visualization environment. In *Proceedings 13th International Conference on Data Engineering*. IEEE, 91–102.

[29] Haifei Yu and Mengxiao Zhang. 2017. Data pricing strategy based on data quality. *Computers & Industrial Engineering* 112 (2017), 1–10.

[30] Mengxiao Zhang, Fernando Beltrán, and Jiamou Liu. 2023. A Survey of Data Pricing for Data Marketplaces. *IEEE Transactions on Big Data* (2023).