University of Tartu
Faculty of Mathematics and Computer Science
Institute of Computer Science
Cyber Security

Ivo Kubjas

Set Reconciliation

Master's Thesis (30 ETCS)

Supervisor: Vitaly Skachek, Ph.D.

Tartu 2014

Acknowledgments

This Thesis is supported by the Norwegian-Estonian Research Cooperation Programme through the grant EMP133 and by the Estonian Research Council through the grant IUT2-1.

The author has received a scholarship through the IT Academy scholarship programme.

Hulkade ühendamine

Lühikokkuvõte: Olgu erinevad seadmed, mis omavad failide hulga erinevaid alamhulki. Talletades neid hulki pilves, kaasneb sellega failide sünkroniseerimise probleem. Eesmärgiks on igas seadmes leida nende hulkade ühend.

Naiivne lahendus sellele probleemile on kõikide hulkade edastamine kõikide osapoolte poolt. Selline lähenemine toob kaasa suure andmeedastuskeerukuse. Soovitav oleks leida algoritm, mille andmeedastuskeerukus oleks proportsionaalne hulkade sümmeetrilise vahe suurusega, mis on tüüpiliselt väike võrreldes kõikide failide arvuga.

Me defineerime mitmeid erinevaid andmete edastamise võrkude mudeleid. Efektiivsed algoritmid hulkade ühendamiseks on teada kahe osapoolega võrkude jaoks, kuid sarnased algoritmid üldiste võrkude jaoks on veel teadmata.

Me uurime võrgu topoloogiate ja hulkade ühendamise algoritmide vahelist seost. Me paneme tähele, et andmeedastuskeerukust on võimalik vähendada spetsiifilise topoloogiaga võrkudes.

Me samuti uurime juhtmega võrkudes iteratsioonide minimiseerimise ülesannet. Me defineerime ühenduste kaaludena ühenduste otspunktideks olevates seadmetes asuvate erinevate failide arvu. Me uurime katseliselt maksimaalsete kaalude valikumeetodi efektiivsust suhtlevate seadmepaaride valikuks. Tulemused viitavad, et see algoritm annab paremaid tulemusi võrreldes suhtlevate seadmepaaride juhusliku valikuga.

Selle magistritöö põhitulemuseks on algebraline analüütiline raamistik hulkade ühendamise algoritmide uurimiseks juhtmeta võrkudes. Raamistiku abil on võimalik optimeerida hulkade ühendamise algoritme, mis kasutavad lineaarselt kodeeritud teateid. See lähenemine üritab minimiseerida iteratsioonide ja edastavate teadete arvu.

Märksõnad: hulkade ühendamine, leviedastusega võrgud, maksimaalsete kaalude valikumeetod, järgu optimiseerimise ülesanne, andmete vahetamise ülesanne.

Set reconciliation

Abstract: Assume that we have several devices with different subsets of a set of files. The problem of file synchronization arises in cloud storage. The goal is to find a union of the sets at each of these devices.

The naive solution to the problem is to transmit the whole sets by all parties in the network. This, however, results in high communicational complexity. It would be desirable to find an algorithm with communicational complexity that is proportional to the size of the symmetric difference of the sets, which is typically small when compared to the total number of files.

We define a number of communication network models. Several efficient algorithms for set reconciliation over a network with two devices have been described in the literature, but similar algorithms for general networks are still unknown.

We study the connection between the network topologies and the communicational cost in the set reconciliation algorithms. We observe that it is possible to reduce the communicational cost in networks of specific topology.

We also study a problem of minimization of a number of communication rounds in a wired network. We define weights on the edges of the graph according to the number of different files in the communicating devices. Then, we experimentally test the efficiency of choosing the communicating pairs using maximum weight matching. The results imply that this algorithm provides better results than its counterpart which chooses the communicating pairs randomly.

The main result of this Thesis is an algebraic analytical framework for studying the set reconciliation algorithms in wireless networks. This frameworks allows for optimizing set reconciliation protocols which use linearly coded messages. This approach aims at minimizing the number of iterations and the number of transmissions during each iteration.

Keywords: set reconciliation, broadcast networks, maximum weight matching, rank-optimization problem, data exchange problem.

Contents

Notation

$[n]$	integer sequence $\{1, \ldots, n\}$
$\mathcal{G}$	directed graph
$\mathcal{H}$	undirected graph
$\mathcal{V}$	set of all nodes in a graph
k	the number of all nodes in $\mathcal{V}$
v_i	specific node in $\mathcal{V}$
$S(v_i)$	set of neighbours of the node v_i
$S_r(v_i)$	set of neighbours at distance r of the node v_i
$S_{\leq r}(v_i)$	set of neighbours at distance up to r of the node v_i
$\mathcal{E}$	set of all edges in a graph
(v_i, v_j)	specific edge in $\mathcal{E}$
$w((v_i, v_j))$	weight of edge (v_i, v_j)
M	matching in a graph
x	single object
b	number of bits to represent an object
X	set of objects
X_i	set of objects associated with node v_i
$\overline{X}_i$	complement of the set X_i: $\overline{X}_i = X_{\mathsf{all}} \setminus X_i$
$X_{i,j}$	union $X_i \cup X_j$
n_i	number of objects in X_i
$\mathbf{X}$	universe of objects
X_{all}	set of all objects
n	number of objects in X_{all}
n_{max}	number of objects in the largest set X_i, $i = 1, \ldots, k$
n_{min}	number of objects in the smallest set X_i, $i = 1, \ldots, k$
$\Delta_{i,j}$	set $X_i \setminus X_j$
$d(i, j)$	size of the set $\Delta_{i,j}$
$\mathrm{dist}(i, j)$	$\mathrm{dist}(i, j) = d(i, j) + d(j, i)$
$\mathfrak{A}$	set reconciliation algorithm

Communication($\mathfrak{A}$)	communication cost of the algorithm $\mathfrak{A}$
Computation($\mathfrak{A}$)	computation cost of the algorithm $\mathfrak{A}$
Time($\mathfrak{A}$)	time cost of the algorithm $\mathfrak{A}$
Rounds($\mathfrak{A}$)	round count of the algorithm $\mathfrak{A}$
$\mathbb{F}$	a field
$\mathbb{F}_q$	a finite field of order q
$\chi_X(Z)$	characteristic polynomial $\chi_X(Z) = (Z - x_1) \dots (Z - x_n)$
Ev	set of evaluation points $\mathsf{Ev} = \{\mathsf{ev}_1, \dots, \mathsf{ev}_n\}$
$\mathcal{F}$	Bloom filter
h_i	i-th hash function used in Bloom filter
H	number of hash functions used in Bloom filter
N	number of cells in Bloom filter
Γ_y	vector of linear coefficients such that $y = \Gamma_y \cdot (x_1, \dots, x_n)$
$\mathbb{A}$	possession matrix of the graph
max-rank($\mathbb{A}$)	maximum rank over all $A \in \mathbb{A}$
$\mathbb{A}_j$	j-th sub-matrix of $\mathbb{A}$
$\mathbb{A}^{(i)}$	possession matrix after i-th round of algorithm iteration
$A^{(i)}$	a transmission matrix from matrix family $\mathbb{A}^{(i-1)}$
$A^{(i)}_j$	j-th sub-matrix of $A^{(i)}$
D	adjacency matrix
$B_j(\mathbb{A})$	operator which returns a matrix with minimal rank from $\mathbb{A}$
$M_{j,\star}$	j-th row of the matrix M
I	diagonal matrix
E	ones matrix
$Y \otimes Z$	tensor product of matrices Y and Z
rowspace(A)	vector space spanned by the row vectors of A

Introduction

With the advances in cloud storage, increasing amount of information is being stored off-site and used in several computer terminals. If the files are used concurrently then inherently the problem of file synchronization arises. Synchronization can be viewed as a problem of set reconciliation, which is defined as finding a union of sets of data with the smallest communication complexity.

Few problems can be addressed in regard to current technology. As an example, RSYNC protocol [Tri99] compares the modification dates of all files in the directories being synchronized. Even though this guarantees the total synchronization between the terminals, the amount of communicated data is proportional to the number of files. In the case, where only few files were modified (compared to the total number of files), redundant information is sent between the terminals.

Current protocols usually require point-to-point connection between the terminals as the differences are calculated pairwise. This leads to communication complexity, which is of the order of the square of the number of terminals. The decrease in the total number of messages would allow for a faster synchronization and for increase in the system efficiency.

The point-to-point connection model does not cover wireless networks and leads to congestion of scarce transmission medium. If several terminals transmit packets simultaneously, then the interference creates errors for the receiver. Instead of allowing one terminal to transmit at a time, the model could account for parallel transmissions.

In the Thesis, we study different problems of set reconciliation and give several results.

In Chapter 1, we define the notation used in the Thesis. As the context in the literature is heterogeneous then we adapt a uniform approach to the definitions and models.

In Chapter 2, we give an overview of the related literature.

In Chapter 3, we introduce our results.

In Section 3.1, we specify a bound introduced in [MTZ03] and generalize the approach to a certain type of graphs.

In Section 3.2, we test experimentally the efficiency of choosing the transmitting nodes based on a metric. Based on our experiments, maximum weight matching gives satisfying results if lacking other metrics or a priori knowledge of the used network.

In Section 3.3, we generalize the approach for solving the data exchange problem introduced in [RSS10] to general network topologies.

We finish with a conclusion and a set of open problems in Chapter 4.

Chapter 1

Models, settings

Definition 1. The **set reconciliation problem** is defined as finding the union of sets with the smallest communication complexity.

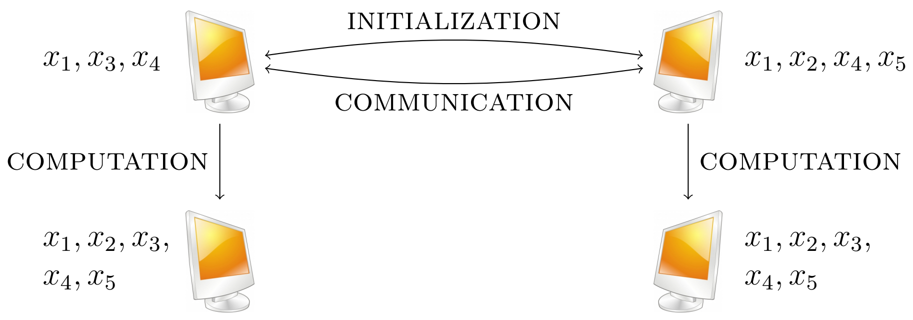


Figure 1.1: Set reconciliation process

1.1 Notation

In this Thesis we use a unified notation for describing the models. As the set reconciliation problem is related to several fields then the notation is heterogeneous. This is due to the fact that similar problems have arisen from different problems and the authors have used familiar concepts in each of the cases.

For example, if we speak about set reconciliation, then in practice finding a union of a set of files is preferred. However, in network coding one may refer to finding a vector space from received messages [KK08]. If some differences can be resolved through wording the problem differently, others raise a more challenging issue.

To overcome the differences, no assumptions on the types should be enforced. We say that the items which are being reconciled, are *objects* and we denote them by small letters. We denote the *set of objects* by X_i. We use the usual set-theoretic notations to denote the inclusion of a object in a set of objects by $x \in X$ and exclusion by $x \notin X$. We denote the distinct objects by different indices. Thus two objects x_i and x_j are different if $i \neq j$. If it is not mentioned explicitly, then we assume that the indices are integers from a set $[n] = \{1, \ldots, n\}$, where n is the number of objects. The objects are part of a *universe of all objects*, which we denote by $\mathbf{X}$.

As the underlying network can be viewed as graph, we call the participants of the protocol *nodes*. We denote the nodes as v_i for $i \in [k]$. The set of all nodes is denoted as $\mathcal{V}$. The number of nodes in the graph is k. If we have associated a set of objects to the node v_i, then we denote this set of objects as X_i. The *set of all objects* is the set $X_{\text{all}} = \cup_{i \in [k]} X_i$.

We call a connection between two nodes an *edge*. If there is a directed edge between the nodes v_i and v_j, then we denote it as an ordered pair $(v_i, v_j) \in \mathcal{V} \times \mathcal{V}$. If there exists an edge (v_i, v_j) then this implies that node v_i can transmit to v_j but not vice versa. The set of all edges is denoted as $\mathcal{E}$. A path between the nodes v_i and v_j is a sequence of edges such that the second node in the first edge collides with the first node in the second edge for any two consecutive edges in the sequence.

In undirected graphs, the edges are unordered sets of two nodes $\{v_i, v_j\} \subset \mathcal{V}$ and messages can be transmitted either to the node v_i or to the node v_j. The set of all undirected edges is also denoted as $\mathcal{E}$. The path in an undirected graph is defined similarly as in a directed graph, but the requirement of the position of the node in the edges is omitted.

The set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$ make up a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The undirected graph is denoted as $\mathcal{H} = (\mathcal{V}, \mathcal{E})$.

The *adjacency matrix* of a directed graph $\mathcal{G}$ is an integer matrix where the element in i-th row and j-th column is 1 if there is an edge $(v_i, v_j) \in \mathcal{E}$ and 0 otherwise. In an undirected graph, the element in i-th row and j-th column is 1 if there exists an edge $\{v_i, v_j\} \in \mathcal{E}$ and 0 otherwise. We denote the adjacency matrix of the graph by D. If we construct an adjacency matrix of the graph $\mathcal{G}$ then we set all diagonal elements to 1.

Example 1. *Take the graph which is shown in Figure 1.2.*

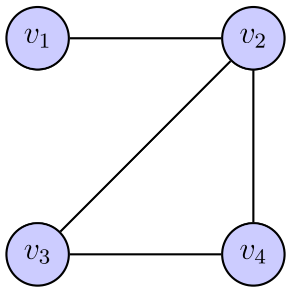


Figure 1.2: Graph used in Example 1

The (4×4) *integer-valued adjacency matrix* D *for this graph is*

$$D = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

A nodes *neighbours* are nodes to whom the node has edges. For a node v_i, we denote the set of the neighbours as $S(v_i)$. Using similar notation, for the node v_i, we denote the set of the nodes to whom the node has a path of r edges as $S_r(v_i)$. We define also $S_{\leq r}(v_i) = \sum_{j=1}^{r} S_j(v_i)$.

If two nodes v_i and v_j have associated sets of objects X_i and X_j, then we call the set $\Delta_{i,j} = X_i \setminus X_j$ as set of objects difference of v_i and v_j. We call the size of the set $\Delta_{i,j}$ as the difference of v_i and v_j and denote it by $\mathrm{d}(i,j) = |\Delta_{i,j}|$. We call the size of the symmetric difference as the distance and denote it by $\mathrm{dist}(i,j) = \mathrm{d}(i,j) + \mathrm{d}(j,i)$.

Lemma 1. *The distance* $\mathrm{dist}(\cdot,\cdot)$ *satisfies non-negativity,* $\mathrm{dist}(i,i) = 0$*, symmetry and triangle inequality, thus being a pseudometric.*

Proof. Because the size of the set is non-negative, then the distance is non-negative.

The distance to itself is zero:

$$\mathrm{dist}(i,i) = |X_i \setminus X_i| + |X_i \setminus X_i| = 0.$$

The distance is symmetric:

$$\mathrm{dist}(i,j) = \mathrm{d}(i,j) + \mathrm{d}(j,i) = \mathrm{d}(j,i) + \mathrm{d}(i,j) = \mathrm{dist}(j,i).$$

For an element in $X_i \setminus X_j$, if it is in X_l, then it is in $X_l \setminus X_j$. Otherwise, it is in $X_i \setminus X_l$. Similarly, for an element in $X_j \setminus X_i$, if it is in X_l, then it is in $X_l \setminus X_i$.

Otherwise, it is in $X_j \setminus X_l$. Thus

$$|X_i \setminus X_j| \leq |X_l \setminus X_j| + |X_i \setminus X_l|$$

and

$$|X_j \setminus X_i| \leq |X_l \setminus X_i| + |X_j \setminus X_l|.$$

From the definition of the distance we see that the triangle inequality $\text{dist}(i, j) \leq \text{dist}(i, l) + \text{dist}(l, j)$ holds. □

Observe that if $X_i = X_j$ for $i \neq j$, then $\text{dist}(i, j) = 0$, and so $\text{dist}(\cdot, \cdot)$ is not a metric.

We define E as a square matrix where all elements are ones. We call this matrix as $(n \times m)$-dimensional ones matrix. We use the notation I for an identity matrix. The dimensions of these matrices are given in the context if it is not evident.

Given a vector $\boldsymbol{x}$, we denote the diagonal matrix where the elements on the diagonal are the elements of $\boldsymbol{x}$ as $\text{diag}(\boldsymbol{x})$.

We denote the i-th row vector of L as $L_{i,\star}$ and the j-th column vector as $L_{\star,j}$.

For a $(m \times n)$-dimensional matrix

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix},$$

we write $A = (a_{i,j})_{i=1,\ldots,m}^{j=1,\ldots,n}$.

Let $A = (a_{i,j})$ be a $(m \times n)$-dimensional matrix and B be a $(k \times l)$-dimensional matrix. The tensor product $A \otimes B$ is a $(mk \times nl)$-dimensional matrix defined through $(k \times l)$-dimensional submatrices $A_{i,j}$ where $A_{i,j} = a_{i,j}B$.

The row space of the matrix A is a vector space which is spanned by the row vectors $A_{i,\star}, i \in [k]$. The row space of the matrix A is denoted as $\text{rowspace}(A)$.

Using the notation, it is possible to give a refined definition of the set reconciliation problem.

Definition 2. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph describing the network of k nodes. Let there be a universe $\mathbf{X}$ of objects. Let each node $v_i \in \mathcal{V}$, $i \in [k]$, have an associated set of objects $X_i \subset \mathbf{X}^*$. The set reconciliation problem is finding an algorithm which returns a union $X_{\text{all}} = \cup_{i \in [k]} X_i$ for each of the nodes with smallest communication complexity.

1.2 Models

1.2.1 Pairwise set reconciliation

Consider a situation where there are only two nodes with corresponding sets of objects.

In this case, the nodes are

$$\mathcal{V} = \{v_1, v_2\}$$

with a single edge

$$\{v_1, v_2\}$$

The associated sets of objects are X_1 and X_2. Let $n = |X_1 \cup X_2|$ and $m = \mathrm{dist}(1, 2)$. We denote the algorithm which reconciles the sets of objects X_1 and X_2, as $\mathfrak{A}$.

We denote the amount of communication in bits as Communication($\mathfrak{A}$). The amount of communication is counted over all messages back and forth between the nodes. The communication takes into account the size to represent elements of algebraic structure. If the exact amount of communication is not needed then we may also give an asymptotic amount.

The total amount of computation operations needed for running $\mathfrak{A}$ is denoted Computation($\mathfrak{A}$). We usually count all the required computation steps, even if some of the operations are cheaper to perform (eg. exponentiation is faster than addition on most of the computers). Similarly to communication, we can also give only an asymptotic amount.

The number of dependent messages is denoted as Time($\mathfrak{A}$). If the algorithm could finish with some of the messages concatenated and sent as one, then we count these messages as one message. This implies that time is the number of rounds in the algorithm. Here, we can also give only the asymptotic amount.

Depending on the requirements, the goals of the algorithms may differ and the notation of different amounts should describe these requirements.

For example, if we consider wireless mobile devices, then initializing the transmission to the base station requires temporary power increase and the wish is to reduce Time($\mathfrak{A}$) while loosening the requirements on Communication($\mathfrak{A}$). Or, for sensors with constrained computing capabilities, one hopes for a small Computation($\mathfrak{A}$).

It can be easily seen that we can find the optimal communication and time cost.

Lemma 2. *There exists no set reconciliation algorithm $\mathfrak{A}$ which can reconcile all instances of sets of objects in less than $m \log |\mathbf{X}|$ bits.*

Proof. We assume that exists an algorithm $\mathfrak{A}$, which can reconcile any instances $X_1, X_2 \subset \mathbf{X}$ in fewer than $m \log |\mathbf{X}|$ bits. For any $X \subset \mathbf{X}$ there exists $X_1, X_2 \subset \mathbf{X}$ such that $X = X_1 \setminus X_2 \cup X_2 \setminus X_1$. As $|X| = m$, then using the algorithm $\mathfrak{A}$, this set can be transmitted with less than $|X| \log |\mathbf{X}|$ bits.

We have constructed an algorithm to transmit all sets $X \subset \mathbf{X}$ with less than $|X| \log |\mathbf{X}|$ bits. This is a contradiction as it takes at least $\log |\mathbf{X}|$ bits to represent random object in $\mathbf{X}$ and $|X| \log |\mathbf{X}|$ bits to represent a random subset X in $\mathbf{X}$. $\square$

Rephrasing Lemma 2, we obtain a lower bound on communicational cost for any set reconciliation algorithm.

Corollary 1. *For any set reconciliation algorithm $\mathfrak{A}$ and for any sets of objects, the communicational cost is bounded*

$$\mathrm{Computation}(\mathfrak{A}) \geq m \log |\mathbf{X}|.$$

Lemma 3. *Optimal time cost for any set reconciliation algorithm $\mathfrak{A}$ is* $\mathrm{Time}(\mathfrak{A}) = 2$.

Proof. It takes at least one message to transmit $\Delta_{1,2}$ and one message to transmit $\Delta_{2,1}$. $\square$

1.2.2 Pairwise sequence reconciliation

We diverge shortly from the initial setup and notation to see an alternative approach to solving the set reconciliation problem using the methods from coding theory.

Let $\boldsymbol{x}$ be a sequence of symbols from $\mathbf{X}$. Let the nodes v_1 and v_2 have a sequence which is $\boldsymbol{x}$ but with some symbols erased. The sequence reconciliation problem is finding $\boldsymbol{x}$ from the nodes sequences.

Now, if we consider the methods from coding theory, then the nodes need to jointly decode their sequences to recover initial $\boldsymbol{x}$. If they have recovered $\boldsymbol{x}$, then they have reconciled their sequences.

If instead of the sequence $\boldsymbol{x} = (x_1 x_2 \dots x_n)$ we consider the set $\{(x_1, 1), \dots, (x_n, n)\}$, then both nodes have a subset of this set and they need to find the union.

Even though there are well-studied methods to solve the coding problem, then they may not be practical in the set reconciliation context. Firstly, the codes may be inefficient for large symbol spaces $\mathbf{X}$. Secondly, the coding problem covers also errors, which are not allowed in the usual set reconciliation model. However, finding an optimal solution to a set reconciliation problem can lead to coding algorithm.

1.2.3 Broadcast set reconciliation

In broadcast set reconciliation, the number of the nodes is arbitrary. Let $\mathcal{V}$ be the set of all nodes and $\mathcal{E}$ the edges between the nodes.

We require that the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is strongly connected, i.e. there exists a directed path between any pair of nodes. Otherwise, there exists no set reconciliation algorithm.

In this model, if a node v_i transmits a message, then it is received by all its neighbours $S(v_i)$. We assume that the messages can be sent in parallel and each node can transmit and receive a message at a single step. As with the pairwise set reconciliation, if messages sent by a single node are independent, then they can be concatenated and viewed as a single message. A single step of nodes transmitting messages in parallel is called a *round*. The total number of rounds for an algorithm $\mathfrak{A}$ to reconcile the sets is denoted as $\text{Rounds}(\mathfrak{A})$.

Broadcast set reconciliation models the wireless networks. If a node transmits a message, then it is received by all nodes which can decode the message successfully. The messages can be transmitted in parallel using a time-slicing method. Using the time-slicing method, the channel is divided into short time-slots and each of the nodes transmits during a time slot assigned to it. An example of time-slicing method is illustrated in Figure 1.3.

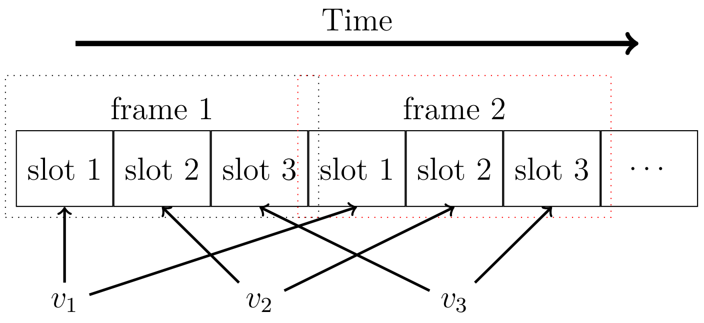


Figure 1.3: Time slicing method used in broadcast network model with three nodes v_1, v_2, v_2

1.2.4 Unicast set reconciliation

The setting for unicast set reconciliation is similar to broadcast set reconciliation with the exception that a node can communicate with one node at a time.

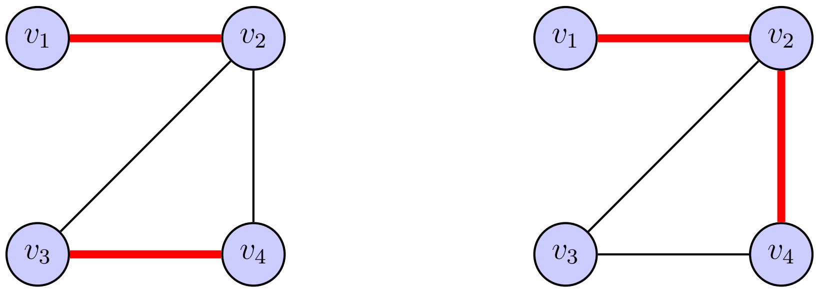


Figure 1.4: Red edges on the left figure make up a matching. On the right figure, the node v_2 is an endpoint for two red edges, thus the red edges do not make up a matching

In a single round, we allow two neighbouring nodes to transmit more than one message. Thus, the round ends if all the transmitting nodes have finished transmitting their messages and are ready to choose a new neighbour.

For analysis of unicast set reconciliation, the following definition would be useful.

Definition 3. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph. A matching M is a subset of $\mathcal{E}$ such that each node in $\mathcal{V}$ is an endpoint to only one edge in the matching.

An illustration of a matching is shown in Figure 1.4.

It is possible to construct a set reconciliation algorithm $\mathfrak{A}$, which chooses the optimal matching and performs pairwise set reconciliation between pairs of nodes iteratively. It stops when the sets of objects in all nodes are reconciled.

There can also be an algorithm which does not perform full pairwise set reconciliation algorithm for some neighbouring nodes but rather decreases some metric which measures the difference of the sets of objects. The study of these algorithms is not in the scope of this Thesis.

1.2.5 Threshold reconciliation

In all previous models, the goal was to reconcile the sets of objects for all nodes. In this model, the goal is to have at least some fixed number of copies of each of the object. The required minimum number is called a *threshold*. The transmission model of threshold reconciliation can either be unicast or broadcast.

The threshold reconciliation model covers some storage systems where the requirement is to provide redundancy while saving the storage space. Threshold reconciliation could also be used in error correction. If the count of any object in the graph is smaller than the threshold, then this object can be considered as an error. Otherwise, if the count of any object exceeds the threshold, then the deficiency of this object can be considered as an erasure.

Chapter 2

Related work

2.1 Characteristic polynomials

In [MTZ03], an algorithm for pairwise set reconciliation model was proposed. Even if the computational cost of the algorithm is large, then it can be use for set reconciliation if the differences are small as the communicational overhead of the algorithm is small and the algorithm is very easy to implement.

A sample implementation of this algorithm by the author is available at [Kub13]. An example using this implementation is given in Appendix A.

2.1.1 Description of the algorithm

This approach uses the pairwise set reconciliation model.

The objects in this algorithm are elements from a field $\mathbb{F}_q$. Here we consider the case where q is a prime. Due to Bertrand's postulate [MM13], for every $b \in \mathbb{N}$ there is a prime p such that $2^b \leq p \leq 2^{b+1}$. Thus, we can represent arbitrary bitstrings of length b as bitstrings of length $b+1$ in the finite field $\mathbb{F}_q$.

For a set of objects $X = \{x_1, \ldots, x_n\}$ we define a characteristic polynomial $\chi_X(Z)$ of the set X as:

$$\chi_X(Z) = (Z - x_1)(Z - x_2) \ldots (Z - x_n).$$

We see that the roots of $\chi_X(Z)$ correspond to the elements of X. If we are able to factor $\chi_X(Z)$ then we have found all the objects of a node. As we need only the differences, then we can instead consider the rational polynomial

$$\frac{\chi_{X_1}(Z)}{\chi_{X_2}(Z)} = \frac{\chi_{X_1 \cap X_2}(Z) \cdot \chi_{\Delta_{1,2}}(Z)}{\chi_{X_1 \cap X_2}(Z) \cdot \chi_{\Delta_{2,1}}(Z)} = \frac{\chi_{\Delta_{1,2}}(Z)}{\chi_{\Delta_{2,1}}(Z)}.$$

Instead of sending the polynomials and finding a ratio, the polynomials can be evaluated at predefined points. The values are divided and a rational polynomial is interpolated using the divided values. From the interpolated polynomial, it is possible to recover $\chi_{\Delta_{1,2}}(Z)$ and $\chi_{\Delta_{2,1}}(Z)$. The roots of these polynomials correspond to the differences $\Delta_{1,2}$ and $\Delta_{2,1}$, respectively. Each of the nodes now adds the objects from the differences to its set of objects and thus recovers X_{all}.

2.1.2 Evaluation and interpolation of polynomials

If the total number of differences is m, then we denote by $\overline{m}$ an upper bound $m \leq \overline{m}$. The case where there is no known upper bound is considered in Section 2.1.5.

The polynomials are evaluated at $\overline{m}$ predefined points $\mathsf{Ev} = \{\mathsf{ev}_1, \dots, \mathsf{ev}_{\overline{m}}\}$. If some of the points in Ev is an object in X_{all}, then either of the characteristic polynomials evaluates to zero and the algorithm is halted. A new evaluation point should then be chosen or a pseudorandom generator should be used to generate a new evaluation point.

Interpolating a rational function depends on evaluation points and values. Specifically, if we have polynomials $P(Z) = \sum_i p_i Z^i$ and $Q(Z) = \sum_i q_i Z^i$ with degrees bounded with d_1 and d_2, then it is necessary to have $d_1 + d_2 + 1$ pairs $(k_i, f_i) \in \mathbb{F}^2$, which we denote as $\mathcal{P}$, to find a unique rational polynomial f such that $f(k_i) = f_i$ [Zip93]. The pairs in $\mathcal{P}$ establish linear constraints on f:

$$k_i^{d_1} + p_{d_1-1}k_i^{d_1-1} + \dots + p_0 = f_i \cdot (k_i^{d_2} + q_{d_2-1}k_i^{d_2-1} + \dots + q_0).$$

Because bounds on the degrees of polynomials are not known, we can find them through the set sizes. Take $\delta = d(1,2) - d(2,1)$. Then:

$$d(1,2) \leq \left\lfloor \frac{\overline{m} + \delta}{2} \right\rfloor = \overline{d(1,2)},$$
$$d(2,1) \leq \left\lfloor \frac{\overline{m} - \delta}{2} \right\rfloor = \overline{d(2,1)}.$$

Because the numerator and denominator polynomials are monic in rational polynomial being recovered and if δ and $\overline{m}$ have same parity, then $\overline{d(1,2)} + \overline{d(2,1)} = \overline{m}$. Thus $\overline{m}$ evaluation values are enough to interpolate f. The following proposition gives sufficient conditions on uniqueness.

Theorem 1 ([SB93], Theorem 2.2.1.4)**.** *Let $\mathcal{P}$ be a support set with $\overline{m}$ elements over the field $\mathbb{F}$. Assume there exists two monic rational functions f and g that satisfy $\mathcal{P}$,* *and that the numerator and denominator of f (respectively g) have degrees summing to at most $\overline{m}$. If the difference in degrees between numerator and denominator of f is the same as for g, then f and g are equivalent.*

2.1.3 Computational complexity

Evaluation of characteristic polynomials of sets $\Delta_{1,2}$ and $\Delta_{2,1}$ takes up to $2|X_{\text{all}}|\overline{m}$ additions and $2|X_{\text{all}}|\overline{m}$ multiplications.

Solving system of $\overline{m}$ linear equations with Gaussian elimination has a complexity of $O(\overline{m}^3 \log\log|\mathbf{X}|)$. Using other methods for solving linear equations could reduce the complexity.

There is additional computation during the factorisation of interpolated polynomials. In [KS98], an algorithm with computational complexity $O(\overline{m}^{1.815} \log\log|\mathbf{X}|)$ was given.

We see that the computational complexity is

$$\text{Computation}(\mathfrak{A}) = O(\overline{m}^3 \log\log|\mathbf{X}|).$$

2.1.4 Communicational complexity

In this algorithm, the first node has to send $\overline{m}$ evaluation values of its characteristic polynomial. Each of the evaluation value can be represented with $b+1$ bits. In addition, it needs to send the number objects in its set. The total number of bits required to send by the first node is

$$(b+1)\overline{m} + b.$$

The second node can recover the sets $\Delta_{1,2}$ and $\Delta_{2,1}$ from the received bits. It has to send $\Delta_{2,1}$ to the first node for set reconciliation. Each of the objects can be represented with $b+1$ bits. The total number of bits sent by the second node is $d(2,1)(b+1) \leq \overline{m}(b+1)$.

The communication cost for this algorithm is bounded by

$$\text{Communication}(\mathfrak{A}) \leq 2\overline{m}(b+1) + b.$$

As the size of the universe is smaller than 2^{b+1}, then the communication complexity is $\text{Communication}(\mathfrak{A}) = O(\overline{m}\log|\mathbf{X}|)$. If the upper bound $\overline{m}$ is chosen close to m, then the algorithm reaches optimal communication cost.

As there is only one message sent at each direction, then the time cost is $\text{Time}(\mathfrak{A}) = 2$.

2.1.5 Algorithm without known bound on the size of the symmetric difference

If there is no known bound on m, then the algorithm needs to be modified. The idea of the modification is to test that the interpolated polynomial $g(Z)$ equals to $f(Z) = \chi_{\Delta_{1,2}(Z)}/\chi_{\Delta_{1,2}(Z)}$. The equivalence testing can be done by evaluating the polynomials at random points and see if the values are equal.

Let σ be the upper bound on degrees of $g(Z)$ and $f(Z)$. From Theorem 1 we know that σ evaluations are enough for polynomials equivalence. In a worst case scenario, we can choose $\sigma - 1$ evaluation points such that different polynomials $g(Z)$ and $f(Z)$ agree on these points. So, the probability that test succeeds for different polynomials is $\rho = (\sigma - 1)/|\mathsf{Ev}|$.

Choosing q such that sets $X_1 \subset \mathbb{F}_q$ and $X_2 \subset \mathbb{F}_q$ are sparse an taking σ as $\sigma = |X_1| + |X_2|$ and $\mathsf{Ev} \approx \mathbb{F}_q$, then ρ is small and repeating test l times, the probability for false positive is $\rho^l \xrightarrow{l} 0$.

Using this approach, evaluations can be sent one at a time. Then $g(Z)$ should be recalculated only then if evaluations and $g(Z)$ do not agree. After l sequential confirming tests it can be considered that $g(Z) = f(Z)$. The probability that polynomials differ after l succeeded tests is bounded above by $m\rho^l$. If we want to achieve a failure tolerance of $1 - \epsilon$, then l should be chosen

$$l \geq \log_\rho(\epsilon/m) > \log_\rho(\epsilon/|X_1| + |X_2|).$$

The total number of transmitted bits is at most

$$(b+2)(m+l) + b$$

and if the node has recovered differences of sets of objects then it has to send back $d(1,2)(b+1)$ bits of information.

Computational cost is bounded by recalculation of $g(Z)$, which is done m times. As Gaussian elimination has complexity $O(m^3 \log\log|\mathbf{X}|)$, then the total computation cost is $\text{Computation}(\mathfrak{A}) = O(m^4 \log\log|\mathbf{X}|)$.

It is possible to reduce the computational cost by sending an increased amount of evaluation values after each round. The amount of evaluation values is increased c times after each successful round.

The total number of evaluation values sent is $(c^N - 1)/(c-1)$ for some $N > 0$ and the total number of rounds is $N = \lceil \log_c[(m+l)(c-1)+1] \rceil$. In the worst case, only one value is left to the last round. Then extra $(c-1)(m+l-1)$ evaluation values are sent. The total number of bits transmitted by one node is at most

$$2b + 1 + (b+1)c(m+l-1) + \Big\lceil \log_c[(m+l)(c-1)+1] \Big\rceil.$$

The receiving node has to return $d(1,2)(b+1)$ bits of information to finish the algorithm.

In this case, as the number of rounds is proportional to $\log_c(m+l)$, computational complexity is reduced to

$$O((m+l)^3 \log_c(m+l) \log\log|\mathbf{X}|).$$

2.2 Bloom filters

In [Blo70], a new data structure was introduced, which allows very efficient probabilistic testing of an element inclusion in a set. This data structure was modified in [GM11], to allow deletion, lookup and listing of the elements included in the data structure.

An algorithm for pairwise set reconciliation using invertible Bloom filters was described in [EGUV11]. We give a description of the corresponding data structures and describe the use of the data structure in a pairwise set reconciliation algorithm.

2.2.1 Description of Bloom filter

The *filter* $\mathcal{F}$ is an bit array of length N. Initially, all of the bits of the filter are set to 0. We call the each bit of the filter as *cell*.

There are H *hash functions* $h_i : \mathbf{X} \to [N]$, $i \in [H]$. An object x is added into the filter by hashing the object with H hash functions h_i. The resulting integers are used as indices which denote the cells which are set to 1.

To test if an object x is stored in the filter $\mathcal{F}$, it is hashed through H hash functions h_i and the resulting indices denote which filter cells are checked if they are 1. If all of the checked cells are 1, then the object x is included in the filter $\mathcal{F}$ with some probability. Otherwise, if some of the cell is 0, then the object is not included in the filter.

It can be seen that the Bloom filter is probabilistic data structure as it may allow false inclusions if the checked cells are set to 1 by other objects. However, the data structure is deterministic for checking exclusions as the cells are set 1 for the same object if the same hash functions are used.

Adding an object to Bloom filter is illustrated in Figure 2.1.

For the analysis on the probability of false inclusion, we assume that the hash functions are uniform. That is, for any object x and for arbitrary hash function h_i, $i \in [H]$, we have

$$\Pr\{\forall j \in [N] : h_i(x) = j\} = \frac{1}{N}.$$

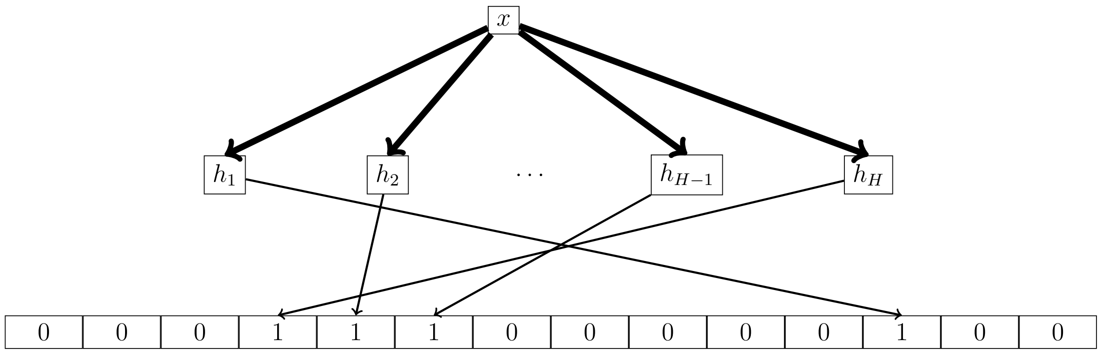


Figure 2.1: Adding an object x to Bloom filter

For a single object, the probability that single hash function sets a specific cell to 1 is $1/N$ and that the cell is not is not set is $1-1/N$. As there are H hash functions, then the probability that a cell is not set to 1 after all hash functions have set the bits is $(1-1/N)^H$. If there are n objects, then the probability that a cell is 0 is $(1-1/N)^{Hn}$ and that the cell is 1 is $1-(1-1/N)^{Hn}$.

Asymptotically,

$$\begin{aligned} 1-\left(1-\frac{1}{N}\right)^{Hn} &= 1-\left(\left(1+\frac{-1}{N}\right)^{N}\right)^{Hn/N} \\ &\approx 1-\left(e^{-1}\right)^{Hn/N} \\ &\approx 1-e^{-Hn/N}. \end{aligned}$$

Thus, for the test to be false inclusive, all H functions must output an index where the cell is set to 1. The probability of false inclusion is

$$\Pr\{x \notin \mathcal{F} : \text{inclusion test is positive}\} \approx \left(1-e^{-Hn/N}\right)^H . \tag{2.1}$$

If n and N are fixed, then we can find an optimal H if we take a logarithm of Equation (2.1) and take a derivate

$$\frac{d(H\ln(1-e^{-Hn/N}))}{dH} = \ln\left(1-e^{-Hn/N}\right) + H\frac{\frac{n}{N}e^{-Hn/N}}{1-e^{-Hn/N}}. \tag{2.2}$$

The right side of Equation (2.2) is 0 if $H = \frac{N}{n}\ln 2$.

2.2.2 Description of invertible Bloom filter

The invertible Bloom filter (IBF) allows for inserting, removing and listing of the key-value pairs (K, V), $K \in [|\mathbf{X}|]$, $V \in \mathbf{X}$, into the data structure. Similarly to standard Bloom filters, we require H hash functions $h_i : [|\mathbf{X}|] \to [N]$, $i = 1, \dots, H$.

In addition to the requirement that hash functions are uniform, we also require that they are distinct for same K, meaning that they all output distinct values. Here we assume that the distinction is satisfied by dividing the filter to H subfilters of size N/H. The hash function h_i then outputs its values to i-th subfilter. We see that this assumption does not contradict the requirement of hash function being uniform.

In the standard Bloom filter, the filter was a bit array of length N but in IBF, the filter consists of N cells with the fields count, keySum, valueSum. All of the fields are initially set to 0.

The definitions of the functions Insert, Remove, Get and ListEntries are given in Algorithms 1, 2, 3 and 4.

Algorithm 1 Insert(K, V)

```
1: for all $h_i, i = 1, \dots, H$ do
2:     $\mathcal{F}[h_i(K)].\mathsf{count} += 1$
3:     $\mathcal{F}[h_i(K)].\mathsf{keySum} += K$
4:     $\mathcal{F}[h_i(K)].\mathsf{valueSum} += V$
5: end for
```

Algorithm 2 Remove(K, V)

```
1: for all $h_i, i = 1, \dots, H$ do
2:     $\mathcal{F}[h_i(K)].\mathsf{count} -= 1$
3:     $\mathcal{F}[h_i(K)].\mathsf{keySum} -= K$
4:     $\mathcal{F}[h_i(K)].\mathsf{valueSum} -= V$
5: end for
```

To analyze the success probabilities of the functions Get and ListEntries, we have to assume that Insert and Remove operations are done correctly, i.e. an existing key-value pair is not inserted into the filter and a nonexistent key-value pair is not removed from the filter.

The Get function is always correct as it only returns non-null value if the count in the cell is 1. As other fields in the same cell are updated simultaneously while changing count field, then Get only returns the value if the valueSum field contains only one value and it corresponds to the keySum value.

Algorithm 3 Get(K)

```
 1: for all $h_i, i = 1, \ldots, H$ do
 2:     if $\mathcal{F}[h_i(K)].\mathsf{count} = 0$ then
 3:         return null
 4:     else if $\mathcal{F}[h_i(K)].\mathsf{count} = 1$ then
 5:         if $\mathcal{F}[h_i(K)].\mathsf{keySum} = K$ then
 6:             return $\mathcal{F}[h_i(K)].\mathsf{valueSum}$
 7:         else
 8:             return null
 9:         end if
10:     end if
11: end for
12: return "not found"
```

Algorithm 4 ListEntries()

```
1: output = []
2: while there is an $i \in [N]$ with $\mathcal{F}[i].\mathsf{count} = 1$ do
3:     append $(\mathcal{F}[i].\mathsf{keySum}, \mathcal{F}[i].\mathsf{valueSum})$ to output.
4:     Remove($\mathcal{F}[i].\mathsf{keySum}, \mathcal{F}[i].\mathsf{valueSum}$)
5: end while
6: return output
```

As the filter is divided into smaller subfilters of size N/H and the hash functions are uniform, then the probability that key-value pair hashes into a specific cell is H/N. The probability that a it hashes to other cells in the subfilter is $1 - H/N$. If n key-value pairs are hashed, then the probability that all but one key-value pair is hashed to other cells is $(1-H/N)^{n-1}$. Using similar argumentation as in Section 2.2.1, we obtain that

$$\left(1 - \frac{H}{N}\right)^{n-1} \approx e^{-H(n-1)/N} \approx e^{-H(n-1)/N}.$$

Thus, the probability that Get returns "not found" for a key-value pair included in the filter is $\left(1 - e^{-Hn/N}\right)^H$, as it has to hold for all H hash functions.

We say that function ListEntries succeeds if it returns all key-value pairs stored in the filter.

In [GM11], the success probability of ListEntries was analyzed. The constant c_H was given by its inverse

$$c_H^{-1} = \sup\left\{\alpha : 0 < \alpha < 1; \forall y \in (0,1), 1 - e^{-H\alpha y^{H-1}} < y\right\}.$$

The constant c_H is used to find a failure probability $O(t^{-c})$ for desired constant c where $t = N/c_H$. The relation between the filter size and the failure probability is given by Theorem 2.

Theorem 2 ([GM11], Theorem 1)**.** *As long as N is chosen so that $N > (c_H + \epsilon)t$ for some $\epsilon > 0$,* ListEntries *fails with probability $O\left(t^{-H+2}\right)$ whenever $n \leq t$.*

2.2.3 Using invertible Bloom filter for set reconciliation

Invertible Bloom filters can be used to perform pairwise set reconciliation. The nodes construct IBFs from their sets of objects. The IBFs are transferred to each other and they are subtracted. As the subtraction cancels out common objects, then the resulting IBFs contain only insertions of objects in sets differences.

We construct a new data structure which is also called filter and denoted by $\mathcal{F}$. The filter consists of N cells. For the set reconciliation the concept of key is irrelevant, and so the cells in the filter do not contain keySum field. If the indexing of the objects is not done using keys, then we need to change the domain space of the hash functions h_i to $\mathbf{X}$.

The subtraction function Subtract is equivalent to Remove function for ordinary IBFs. However, we required that only objects included in the filter are removed, but set differences consists only of objects which have not been included in the filter. Without modifications, the ListEntries function would return incorrect objects. For example, if objects x_1 and x_2 are hashed into a specific cell and x_3 is subtracted from this cell, then ListEntries would output an object $x_1 + x_2 - x_3$.

To diminish the rate of false entries returned by ListEntries, an additional field hashSum is added to the cells and a checksum hash function $\mathbf{h} : \mathbf{X} \to [|\mathbf{X}|]$ is required. The hashSum field contains the sum of hashed values of the value. During listing, it is checked if the fields valueSum and hashSum match, i.e. $\mathbf{h}(F[j].\mathsf{valueSum}) = F[j].\mathsf{hashSum}$. In the previous example, $\mathbf{h}(x_1+x_2-x_3) \neq \mathbf{h}(x_1)+\mathbf{h}(x_2)-\mathbf{h}(x_3)$ and thus the listing function does not output any value.

We rewrite the Insert function and the Remove function as functions Encode and Subtract to correspond with the modifications. The definitions of the corresponding functions are written in Algorithms 5 and 6.

The ListDifference function is a modification of ListEntries which also checks for the additional hash function. The definition of ListDifference is given in Algorithm 7.

The failure probability of listing differences was studied in [EGUV11].

Theorem 3 ([EGUV11], Theorem 1)**.** *Let X_1 and X_2 be two sets having at most m elements in their symmetric differences, and let $\mathcal{F}_1 =$ Encode(X_1) and $\mathcal{F}_2 =$*

Algorithm 5 Encode(X)

```
1: $\mathcal{F} = []$
2: for all $x_i \in X$ do
3:     for all $h_j, j = 1, \ldots, H$ do
4:         $\mathcal{F}[h_j(x_i)].\mathsf{count} += 1$
5:         $\mathcal{F}[h_j(x_i)].\mathsf{valueSum} += x_i$
6:         $\mathcal{F}[h_j(x_i)].\mathsf{hashSum} += \mathbf{h}(x_i)$
7:     end for
8: end for
9: return $\mathcal{F}$
```

Algorithm 6 Subtract($\mathcal{F}_1, \mathcal{F}_2$)

```
1: $\mathcal{F} = []$
2: for all $j \in [N]$ do
3:     $\mathcal{F}[j].\mathsf{count} = \mathcal{F}_1[j].\mathsf{count} - \mathcal{F}_2[j].\mathsf{count}$
4:     $\mathcal{F}[j].\mathsf{valueSum} = \mathcal{F}_1[j].\mathsf{valueSum} - \mathcal{F}_2[j].\mathsf{valueSum}$
5:     $\mathcal{F}[j].\mathsf{hashSum} = \mathcal{F}_1[j].\mathsf{hashSum} - \mathcal{F}_2[j].\mathsf{hashSum}$
6: end for
7: return $\mathcal{F}$
```

Algorithm 7 ListDifference($\mathcal{F}$)

```
1: output = []
2: while there is an $i \in [N]$ with $\mathcal{F}[i].\mathsf{count} = 1$ do
3:     if $\mathbf{h}(\mathcal{F}[i].\mathsf{valueSum}) = \mathcal{F}[i].\mathsf{hashSum}$ then
4:         value = $\mathcal{F}[i].\mathsf{valueSum}$
5:         append value to output
6:         $\mathcal{F} = \text{Subtract}(\mathcal{F}, \text{Encode}(\mathsf{value}))$
7:     end if
8: end while
9: return output
```

Encode(X_2) *be invertible Bloom filters with* $N = (H + 1)m$ *cells and with at least* $\Omega(H \log m)$ *bits in each* hashSum *field. Then with probability* $O(m^{-H})$ *we fail to recover* X_1 *and* X_2 *by applying the* Subtract *operation to* $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *and then applying the* ListDifference *operation to the resulting Bloom filter.*

From Theorem 3 we see that the filter size is linear to the size of the difference if the number of hash functions is fixed. To construct a filter with sufficient size such that the ListDifference function would not fail, we need to estimate the size of the difference before filter construction.

In [EGUV11], the authors proposed an algorithm named Strata Estimator to predict the size of the set difference. The universe $\mathbf{X}$ is partitioned into disjoint subsets $\mathbf{X}_i$ of size $1/2^{i+1}$, $i \in [\log|\mathbf{X}|]$. If an object from set X belongs to i-th subset $\mathbf{X}_i$, then it is inserted into i-th filter $\mathcal{F}_i$. All of the filters $\mathcal{F}_i$ have an fixed size of 80 cells.

The filters are transmitted to other node which subtracts corresponding filters from its filters and tries to list set differences for each of the filter. The filter which covers the smallest partition of the universe gives a good estimate on the size of the set difference.

2.2.4 Computational and communication cost

The description of full pairwise set reconciliation algorithm $\mathfrak{A}$ is given using the descriptions of running a Strata Eliminator and using IBFs to find the set differences.

To start the set reconciliation process, a node v_1 sends its Strata Estimator to node v_2. The node v_2 estimates the size m of the symmetric difference and constructs an IBF of size $O(d \log|\mathbf{X}|)$ and sends it to node v_1. Node v_1 extracts the differences $d(1,2)$ and $d(2,1)$ using ListDifference and sends $d(1,2)$ to node v_2.

The computation consists of creating the estimator, which has the computational complexity $O(|X_{\text{all}}| \log|\mathbf{X}|)$; creating the IBF for reconciliation, which has same computational complexity and extracting the differences, where there is an algorithm in [EGUV11], which has the computational complexity of $O(m \log|\mathbf{X}|)$. However, first two steps can be precomputed and only the last step is done during the set reconciliation process. Thus, the computational cost of the online phase of the algorithm is only

$$\text{Computation}(\mathfrak{A}) = O(m \log|\mathbf{X}|).$$

The communication cost is dominated by the size of the estimator. As the sizes of filters in the estimator are fixed and there are $\log|\mathbf{X}|$ filters which store elements which can be represented by $\log|\mathbf{X}|$ bits, then the size of the estimator is $O(\log|\mathbf{X}| \log|\mathbf{X}|)$. Thus the communicational cost of the algorithm is

$$\text{Communication}(\mathfrak{A}) = O(\log|\mathbf{X}| \log|\mathbf{X}|).$$

If the size of the symmetric difference is known, then the communicational cost is $\text{Communication}(\mathfrak{A}) = O(m \log|\mathbf{X}|)$, achieving optimal cost.

The round cost of the algorithm is $\text{Time}(\mathfrak{A}) = 3$ using the estimator and $\text{Time}(\mathfrak{A}) = 2$ if the estimator is not used.

2.3 Data exchange protocol

The broadcast set reconciliation problem was studied in [RSS10]. The graph in their model is a full graph. In their algorithm, nodes broadcast linear combinations of their objects which are used to extract missing objects for each node. Based on these ideas, we extended the algorithm to an arbitrary graph. This work is covered in Section 3.3.

2.3.1 Bounds on the number of transmissions

Let the graph be defined by nodes $\mathcal{V}$. Let $\mathcal{E}$ be a set of edges such that the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a complete graph. Let k be the number of nodes. We denote by X_i, $i \in [k]$ the sets of objects associated with each of the node. The complement of each set of objects is denoted as $\overline{X}_i = X_{\text{all}} \setminus X_i$. We denote by $n_i = |X_i|$, $i \in [k]$ the sizes of the sets of nodes. Let $n = |X_{\text{all}}|$ be the size of the reconciled set of objects. We denote $n_{\mathsf{min}} = \min_{i \in [k]} n_i$ and $n_{\mathsf{max}} = \max_{i \in [k]} n_i$. The objects are from a finite field $\mathbb{F}$, so the universe is also $\mathbf{X} = \mathbb{F}$.

We observe that all of the nodes need to receive at least the amount of objects they do not possess. The exact lower bound is given in the following lemma.

Lemma 4 ([RSS10], Lemma 1)**.** *The minimum number of transmissions* $\mathrm{Time}(\mathfrak{A})$ *is greater or equal to* $n - n_{\mathsf{min}}$*. If all nodes initially have the same number of objects* $n_{\mathsf{min}} < n$*, then the minimum number of transmissions is greater or equal to* $n - n_{\mathsf{min}} + 1$*.*

Proof. The first part follows from the fact that each node needs to receive at least $n - n_i$ objects. The second part follows from the fact that a transmitting node does not benefit from its own transmissions. □

The upper bound on the number of transmissions can be given through a naive set reconciliation algorithm which always succeeds.

Lemma 5 ([RSS10], Lemma 2)**.** *For* $|\mathbf{X}| \leq k$*, it holds that*

$$\mathrm{Time}(\mathfrak{A}) \leq \min_{i \in [k]} \left\{ |\overline{X}_i| + \max_{j \in [k]} |\overline{X}_j \cap X_i| \right\}.$$

Proof. The algorithm works in two phases:

1. pick a node v_i and reconcile its set by transmitting all objects in $\overline{X}_i$. This requires $|\overline{X}_i|$ transmissions;

2. this node transmits linear combinations of its packets to reconcile the sets of other nodes.

After the first phase, each node v_j knows all the packets in $X_j \cup \overline{X}_i$. For reconciliation, it needs objects from $\overline{X}_j \cap X_i$. Using network coding methods from [JSC+05], these objects can be transmitted using $\max_{j \in [k]} |\overline{X}_j \cap X_i|$ messages if $|\mathbf{X}| \leq k$.

Running this algorithm with i which minimizes the number of messages gives the upper bound. □

2.3.2 Data exchange protocol

The algorithm is used to create linear combinations of objects and these are broadcast to neighbouring nodes. Each of the nodes receives these messages and solves a system of linear equations to extract missing objects to recover X_{all}.

For a linear combination y, let $\Gamma_y \in \mathbb{F}^n$ be the vector of linear coefficients such that $y = \Gamma_y \cdot (x_1, \ldots, x_n)$. As the field $\mathbb{F}$ is a vector space, then the messages y also belong to the universe $\mathbf{X}$.

We consider the vector spaces Y_i which are spanned by linear coefficients Γ_y where $y \in X_i$. Formally, $Y_i = \langle \{\Gamma_y | y \in X_i\} \rangle$.

The algorithm for set reconciliation is given in Algorithm 8.

Algorithm 8 Information Exchange algorithm as given in [RSS10]

```
1: for all $i \in [k]$ do
2:     $Y_i = \langle \{\Gamma_y | y \in X_i\} \rangle$
3: end for
4: while there is a node $v_i$ with $\dim Y_i < n$ do
5:     while exists distinct $v_i, v_j \in \mathcal{V}$ such that $Y_i = Y_j$ do
6:         $\mathcal{V} = \mathcal{V} \setminus \{v_i\}$
7:     end while
8:     Find a node $v_i$ with a vector space $Y_i$ of maximum dimension (if there are
       multiple such nodes choose an arbitrary of them).
9:     Select a vector $b \in Y_i$ such that $b \notin Y_j$ for each $j \neq i$.
10:    Let node $v_i$ broadcast message $b \cdot (x_i, \ldots, x_n)$.
11:    for all $l \in [k]$ do
12:        $Y_l = \langle \{a | a \in Y_l \cup \{b\}\} \rangle$
13:    end for
14: end while
```

In this algorithm, the dimensions of vector spaces Y_i are increased each round. The vector spaces are increased with adding a vector b from the vector space of such node which has the highest dimension. The nodes which have the same vector spaces are considered as one.

We see that the maximum number of rounds is n as the dimension increases by one at each round and the dimension is unbounded from below.

As only a single field element is broadcast during a round, then the communicational cost is $\text{Communication}(n \log |\mathbf{X}|)$.

Chapter 3

New reconciliation protocols

The research goal of this Thesis is finding methods for set reconciliation in arbitrary networks. We have seen from Sections 2.1 and 2.2 that there exist algorithms which can achieve optimal communicational, computational cost and number of messages. However, little is known about algorithms working with arbitrary graphs. In Section 2.3, Lemma 4 provided a lower bound for a set reconciliation algorithm in broadcast network with complete underlying graph.

Extending known results to larger graphs is not straight-forward. Our first result in Section 3.1 shows, that if optimizing the transmission schedule, it is trivially possible to reduce the communicational cost by one third compared to performing pairwise set reconciliation between all nodes. Thus it is possible to gain efficiency if taking into consideration the structure of the graph.

In our second result in Section 3.2, we used characteristics which describe graphs to achieve minimal number of rounds. Our preliminary experiment gave good results in large proportion of the cases and failed only in a small subset of the cases. Furthermore, if the experiment failed, then difference with a counter-example was minute. Even though this approach requires further study, it can be used if no other good results are found as it is rather easy to implement.

The main contribution of this Thesis is in Section 3.3, where we develop an analytical framework for describing the set reconciliation problem in an arbitrary broadcast graph. If there exists a possible set reconciliation protocol for some specific graph, the framework allows to construct it using the solution to a rank-optimization problem. Even though we have used ideas from [RSS10], our idea and technique is novel and could be possibly extended. The idea is based on observation that the adjacency matrix of the graph could be incorporated into a rank-optimization problem.

3.1 Improvements to the method of characteristic polynomials

If there are more than two parties that need to reconcile their sets, using set reconciliation algorithm based on interpolation of characteristic polynomials between each pair is not optimal, as this leads to transmission of redundant information. We propose the following modification to the protocol.

Let there be three nodes v_1, v_2, v_3, and let X_1, X_2 and X_3 be the sets of objects owned by v_1, v_2 and v_3, respectively. We assume that an upper bound $\overline{m}$ on the size of any two pairwise symmetric difference is known. We denote nodes v_1 and v_2 pairwise union as

$$X_{1,2} = (X_1 \cap X_2) \cup \Delta_{1,2} \cup \Delta_{2,1}.$$

Pairwise unions can be simplified to

$$X_{1,2} = (X_1 \cap X_2) \cup \Delta_{1,2} \cup \Delta_{2,1} = X_2 \cup \Delta_{1,2}$$

and

$$X_{2,3} = X_2 \cup \Delta_{3,2}.$$

If $X_{1,2}$ and $X_{2,3}$ are known then $X_{\text{all}} = X_{1,2} \cup X_{2,3}$. Now the node v_1 and the node v_3 do not have $X_{\text{all}} \setminus X_1$ and $X_{\text{all}} \setminus X_3$ accordingly. But $X_{\text{all}} \setminus X_1 = X_{\text{all}} \setminus (X_{1,2} \setminus \Delta_{2,1}) = (X_{\text{all}} \setminus X_{1,2}) \cup \Delta_{2,1}$ and $X_{\text{all}} \setminus X_3 = (X_{\text{all}} \setminus X_{2,3}) \cup \Delta_{2,3}$ and thus can be recovered from sets known to node v_2.

This discussion can be summarised as a protocol for reconciling sets between three parties:

Algorithm 9 Set reconciliation with three parties using characteristic polynomials

```
1: Hosts $v_1$ and $v_3$ evaluate $\chi_{X_1}(Z)$ and $\chi_{X_3}(Z)$ at $\overline{m}$ evaluation points $\mathsf{Ev}$.
2: The evaluation values $\chi_{X_1}(\mathsf{Ev})$, $\chi_{X_3}(\mathsf{Ev})$ and sizes of the sets $|X_1|$, $|X_3|$ are sent
   to host $v_2$.
3: Host $v_2$ recovers $\Delta_{A,B}$, $\Delta_{B,A}$, $\Delta_{C,B}$ and $\Delta_{B,C}$.
4: Host $v_2$ recovers $X_{A,B}$ and $X_{B,C}$.
5: Host $v_2$ recovers $X_{\text{all}}$.
6: Host $v_2$ sends $S \setminus X_1$ to host $v_1$ and $S \setminus X_3$ to host $v_3$.
7: Hosts $v_1$ and $v_3$ recover $X_{\text{all}}$.
```

In this protocol, nodes v_1 and v_3 send values of $\overline{m}$ evaluation points and sizes of their sets. Because node v_2 has all the knowledge about differences, it can send to

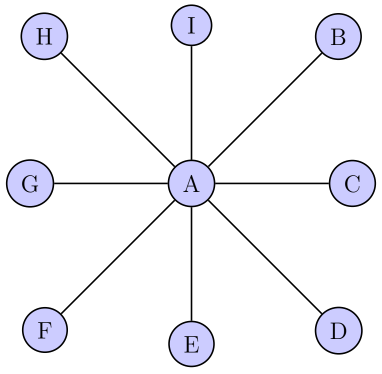


Figure 3.1: Star graph

the nodes v_1 and v_3 only elements missing from X_1 and X_3. It sends elements from $\Delta_{2,1} \cup \Delta_{3,1}$ and $\Delta_{2,3} \cup \Delta_{1,3}$ to the nodes v_1 and v_3 respectively. In total, we have to transmit $|\Delta_{1,3} \cup \Delta_{2,3}| + |\Delta_{3,1} \cup \Delta_{2,1}|$ elements. By looking at disjoint sets, we can simplify:

$$\begin{aligned}
|\Delta_{1,3} \cup \Delta_{2,3}|+|\Delta_{3,1} \cup \Delta_{2,1}| = \\
&=|\Delta_{2,1}| + |\Delta_{2,3}| + |\Delta_{1,3} \setminus \Delta_{2,3}| + |\Delta_{3,1} \setminus \Delta_{2,1}| \\
&=|\Delta_{2,1}| + |X_1 \setminus (X_2 \cup X_3)| + |\Delta_{2,3}| + |X_3 \setminus (X_1 \cup X_2)| \\
&=|\Delta_{2,1}| + |\Delta_{1,2}| - |\Delta_{1,2} \cap \Delta_{3,2}|+ \\
&+|\Delta_{2,3}| + |\Delta_{3,2}| - |\Delta_{1,2} \cap \Delta_{3,2}| \\
&=2(\overline{m} - |\Delta_{1,2} \cap \Delta_{3,2}|).
\end{aligned}$$

Because no more than $2\overline{m}$ elements are sent in this stage, then total number of transmitted bits is bounded above by

$$2(b+1)\overline{m} + 2b + 2b\overline{m} = 2((2b+1)\overline{m} + b)$$

During the protocol, sets are evaluated 3 times and interpolation is done 2 times.

In this algorithm, node v_2 acted as a proxy between nodes v_1 and v_3. Similar construction, where one node acts as a master, could also be used to perform set reconciliation between a large number of nodes, but this leads to uneven distribution of computational cost while having balanced use of communication. The algorithm can be extended to star graphs (see Figure 3.1) and to graphs that contain a star graph as a subgraph.

It has not been considered if requirement would be to have even computational cost across nodes. Intuitively, in this case the amount of communication would increase, as after reconciliation of two nodes sets upper bounds of differing elements increase and more evaluation values should be sent.

Very recently, a similar observation for reconciliation algorithm using invertible Bloom lookup tables was done independently in [MP13].

3.2 Network matching approach

We consider the unicast network model where the edges are undirected. In the unicast network model, pairs of nodes perform pairwise set reconciliation. Our goal is to find an optimal matching algorithm such that the number of rounds for a set reconciliation problem is minimal.

We have constructed a framework [Kub14] which can be used to compare different matching algorithms and to find counter-examples. This framework was used to test if maximum weight matching algorithm gives good results against randomly chosen matchings.

The maximum weight matching algorithm looks for matchings in a graph such that the sum of weights for edges in the matching is maximal. For each of the edges, we used the size of the symmetric difference of sets of nodes at the endpoints of the edge. Formally, $\forall e = \{v_i, v_j\} \in \mathcal{E} : w(e) = \mathrm{dist}(i, j)$, where $w(e)$ is the weight associated with the edge e.

We can intuitively see that maximum weight matching can not be optimal matching algorithm. As the maximum weight matching algorithm takes into account only the size of the symmetric differences and not the objects in nodes sets, then after a round, weights on other edges may increase. However, we believed that on average the maximum weight matching algorithm could diminish the effects of this anomaly.

3.2.1 Experimental results

We constructed arbitrary graphs with 5 to 19 nodes. Let the number of nodes be k. For the graph being connected we require that the number of edges was greater or equal to $k - 1$. For allowing only a single edge between any two pairs of nodes we require that the number of edges is smaller than $k(k - 1)/2$. For each k, we tested all possible edge set sizes satisfying the above restrictions.

For each number of nodes and edges, we constructed 10 arbitrary connected graphs. The requirement that the graph has to be connected is trivial, as otherwise there exists no reconciliation algorithm.

The random graph construction algorithm is a probabilistic algorithm which

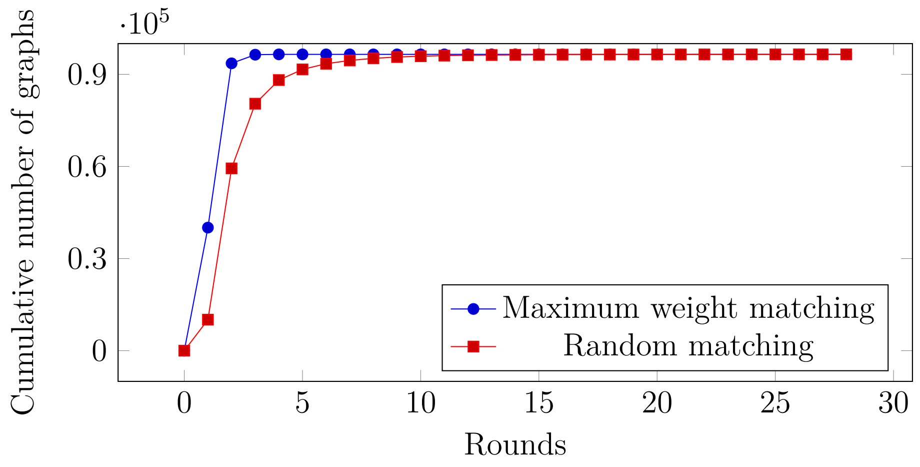


Figure 3.2: Number of graphs which are reconciled depending on the number of rounds

works in iterations. During each of the iterations, a node which is not an endpoint for any edge is selected. An edge connecting this node and a random node in a largest connected subgraph is added to the set of edges. If there exists no such node then an edge between random nodes is added if these two nodes do not have an edge. The exact implementation details can be seen from the source code of the framework at [Kub14].

For each of the graphs, we assigned a subset of 100-element set to each of the node. We compared the number of rounds required for reconciliation if the matchings were chosen by maximum weight matching algorithm to if the matchings were chosen randomly.

There were in total of 9650 graphs and 96500 test cases. With maximum weight matching, all graphs were reconciled with up to 4 rounds and mean number of required rounds was 2. In Figure 3.2, we have illustrated how many graphs could be reconciled with given number of rounds. We have compared the maximum weight matching algorithm with random matching algorithm.

There were 122 cases where maximum weight matching was inferior to random matching. In all of these cases, random matching required only one fewer round than maximum weight matching.

We have illustrated one counter-example in Figures 3.3 and 3.4. In these figures, red circles represent nodes and lines represent edges. The numbers on the edges denote the size of the symmetric difference of the sets of nodes at endpoints. With

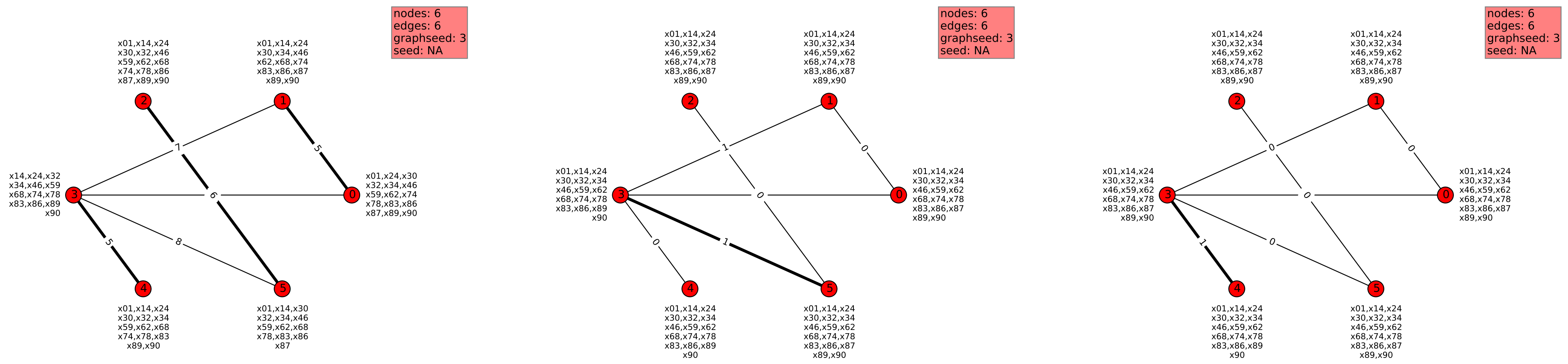

nodes: 6
edges: 6
graphseed: 3
seed: NA
x01,x14,x24 x30,x32,x46 x59,x62,x68 x74,x78,x86 x87,x89,x90
x01,x14,x24 x30,x34,x46 x62,x68,x74 x83,x86,x87 x89,x90
x14,x24,x32 x34,x46,x59 x68,x74,x78 x83,x86,x89 x90
x01,x24,x30 x32,x34,x46 x59,x62,x74 x78,x83,x86 x87,x89,x90
x01,x14,x24 x30,x32,x34 x59,x62,x68 x74,x78,x83 x89,x90
x01,x14,x30 x32,x34,x46 x59,x62,x68 x78,x83,x86 x87
7
5
6
5
8
nodes: 6
edges: 6
graphseed: 3
seed: NA
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x89 x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x89 x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
1
0
0
0
1
nodes: 6
edges: 6
graphseed: 3
seed: NA
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x89 x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
0
0
0
1
0

Figure 3.3: Rounds of reconciliation if matchings are chosen by maximum weight matching algorithm

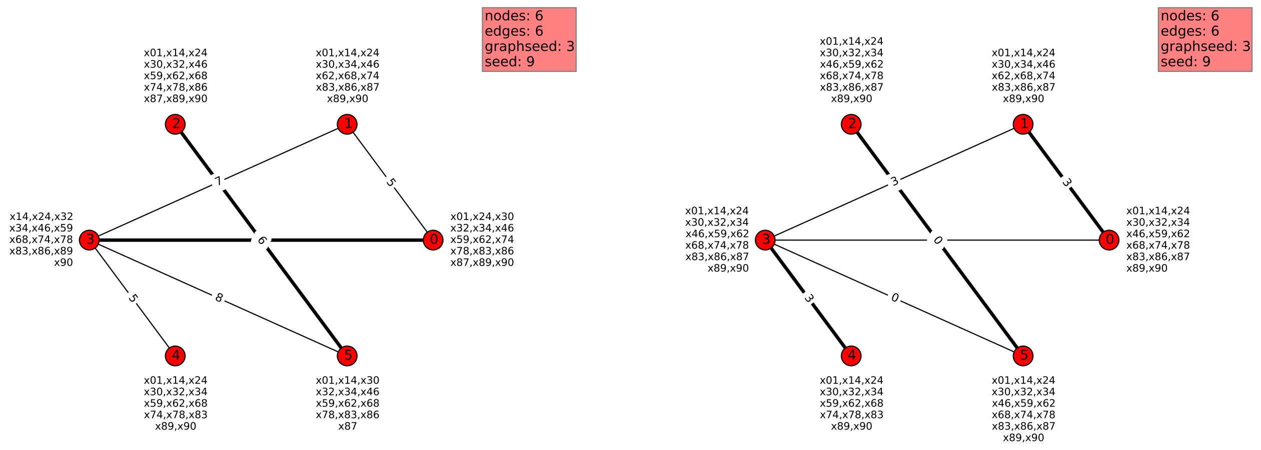

nodes: 6
edges: 6
graphseed: 3
seed: 9
x01,x14,x24 x30,x32,x46 x59,x62,x68 x74,x78,x86 x87,x89,x90
x01,x14,x24 x30,x34,x46 x62,x68,x74 x83,x86,x87 x89,x90
x14,x24,x32 x34,x46,x59 x68,x74,x78 x83,x86,x89 x90
x01,x24,x30 x32,x34,x46 x59,x62,x74 x78,x83,x86 x87,x89,x90
x01,x14,x24 x30,x32,x34 x59,x62,x68 x74,x78,x83 x89,x90
x01,x14,x30 x32,x34,x46 x59,x62,x68 x78,x83,x86 x87
7
5
6
5
8
nodes: 6
edges: 6
graphseed: 3
seed: 9
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x34,x46 x62,x68,x74 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
x01,x14,x24 x30,x32,x34 x59,x62,x68 x74,x78,x83 x89,x90
x01,x14,x24 x30,x32,x34 x46,x59,x62 x68,x74,x78 x83,x86,x87 x89,x90
3
3
0
3
0

Figure 3.4: Rounds of reconciliation if matchings are chosen randomly

each of the node, there are objects which are associated with that node. The thick edges are edges belonging to the matching. The legend shows the used parameters. We see that three rounds are required if the matchings are chosen using maximum weight matching algorithm and two rounds if matchings were chosen randomly.

3.3 Data exchange protocol extension to set reconciliation problem

We consider the generalization of the problem of data exchange. In [RSS10], a connection between the number of transmissions and the minimum rank of a matrix family was established.

Hereafter, we use the broadcast network model. In a broadcast network model, messages transmitted by a node v_i are received by all its neighbours in $S(v_i)$. This model is relevant in wireless networks where the messages are received by nodes in such proximity that they are able to successfully decode the messages. The messages passed to longer distances need to be retransmitted at intermediate nodes. As each of the nodes can possibly receive messages from several nodes at a time, then all of the received messages could be taken into account while transmitting new messages.

In the current setting, let the graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be undirected. This setting is artificial as in wireless networks there may exist nodes with different transmission capabilities. If this is the case, a node with higher transmission power may send messages to neighbouring nodes but some of these nodes may not be able to transmit messages back to this node.

Our goal in this model is to reduce the number of transmissions. This goal leads to finding a set reconciliation algorithm which minimizes $\mathrm{Time}(\mathfrak{A})$. To achieve this goal, we study how many rounds are required for any set reconciliation algorithm and reduce the number of messages during each round.

We define a property of a graph which gives the minimal number of rounds for any set reconciliation algorithm for this graph.

Definition 4. The set reconciliation problem in a graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is l-**solvable** if for any set reconciliation algorithm $\mathfrak{A}$ the rounds cost $\mathrm{Rounds}(\mathfrak{A}) \geq l$ and l is the largest such value.

We recall that the number of nodes is k. Let $n = |X_{\text{all}}|$ be the size of the union of the sets.

The solution to the set reconciliation problem for 1-solvable graphs was given in [RSS10]. We use the adjacency matrix of the graph $\mathcal{H}$ to filter out the messages that each of the nodes receives. This allows to generalize the set reconciliation algorithm to general graphs.

3.3.1 Possession matrix of the graph

Matrix families

Let $\mathbb{F}$ be a finite field. Let $\mathbb{A}_i$, $i \in [k]$, be a family of $(n \times n)$-dimensional matrices over $\mathbb{F}$. For the matrix family $\mathbb{A}_i$, $i \in [k]$, we use a special symbol $*$ to denote an arbitrary element in $\mathbb{F}$. The element in j-th column and t-th row of $\mathbb{A}_i$, for $t \in [n]$, is $*$ if $x_j \in X_i$ and 0 otherwise.

Let $\mathbb{A}$ be a family of matrices of the form (3.1), where each $\mathbb{A}_i$, $i \in [k]$, is a family of matrices that was defined earlier.

$$\mathbb{A} := \begin{bmatrix} \mathbb{A}_1 \\ \mathbb{A}_2 \\ \vdots \\ \mathbb{A}_k \end{bmatrix}, \tag{3.1}$$

Since the matrix families $\mathbb{A}_i$, $i \in [k]$, describe what objects each node v_i has, the matrix family $\mathbb{A}_i$ is called the *possession matrix* of the node v_i. Similarly, the matrix family $\mathbb{A}$ denotes the objects all of the nodes have and $\mathbb{A}$ is called the possession matrix of the graph $\mathcal{H}$.

As the symbols $*$ in these matrices can take any value in the field $\mathbb{F}$, then $\mathbb{A}_i$ and $\mathbb{A}$ consist of many distinct matrices with elements from the field $\mathbb{F}$. This approach allows to operate using known operators on a whole set of matrices.

We call a member A of a matrix family $\mathbb{A}$ as a transmission matrix. As $\mathbb{A}$ consists of matrix families $\mathbb{A}_i$, $i \in [k]$, as in Equation (3.1), then the transmission matrix A can also be written as

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{bmatrix}. \tag{3.2}$$

For each of the A_i from Equation (3.2), the row vectors are denoted as $\boldsymbol{t}_{i,j} = (t_{i,j,1}, \dots, t_{i,j,n})$, $j \in [n]$. The set of messages the node v_i transmits is

$$T_i = \left\{ \sum_{m \in [n]} t_{i,j,m} x_m | j \in [n] \right\}.$$

Without loss of generality, we can assume that only messages where the coefficients $\boldsymbol{t}_{i,j}$ are linearly independent, are transmitted. The row vectors can be transformed such that only linearly independent row vectors are non-zero. Then, if for some $j \in [k]$, if $\boldsymbol{t}_{i,j}$ is a zero vector, a message 0 is transmitted. These messages can be omitted.

The number of transmissions is the number of linearly independent row vectors of the transmission matrix A_i, $i \in [k]$. As the transmission matrix A_i is a member of the matrix family $\mathbb{A}_i$, then it is possible to find the maximum number of transmissions for a node v_i by finding the matrix with the highest rank. The following definition of maximum-rank captures this property between the number of transmissions and the possession matrix.

Definition 5. The max-rank of the matrix family $\mathbb{A}$ is defined as

$$\text{max-rank}(\mathbb{A}) = \max_{A \in \mathbb{A}} \text{rank}(A).$$

Given $\mathbb{A}_i$, we define an operator B, which replaces the symbols $*$ in the rows with canonical vectors and removes linearly dependent rows from the matrix.

Similarly, operator B_j takes as an input the possession matrix $\mathbb{A}$ and returns $B_j(\mathbb{A}) = B(\mathbb{A}_j)$ for $\mathbb{A}_j$ as in Equation (3.1).

Example 2. *For a fixed $i \in [k]$, let*

$$\mathbb{A}_i = \begin{bmatrix} * & 0 & * & * & 0 \\ * & 0 & * & * & 0 \\ * & 0 & * & * & 0 \\ * & 0 & * & * & 0 \\ * & 0 & * & * & 0 \end{bmatrix}.$$

After replacing the symbols $*$ in each row with canonical vectors and replacing redundant rows, we obtain

$$B(\mathbb{A}_i) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

The algebra of the matrix families

We consider the family of matrices over $\mathbb{F}$ (with symbol $*$ for an arbitrary element in $\mathbb{F}$) as a matrix over $\mathbb{F}^* = \mathbb{F} \cup \{*\}$ with a newly defined algebra.

For simpler manipulation of matrix families, we define basic operations for the matrices with the entries in $\mathbb{F}^*$.

As the symbol $*$ in the matrix family denotes any element from the field $\mathbb{F}$, then addition should comply with this definition. If 0 is added to any element, then the sum is also any element from the field. Similarly, if any two non-zero field elements are added, then we obtain some field element. Only if we add 0 to 0, then the resulting element is uniquely defined as 0. This motivates for the following definition of addition for the matrix family elements.

Definition 6. The addition of the elements of the matrix family $\mathbb{A}$ is defined by the following addition table:

$+$	0	$*$
0	0	$*$
$*$	$*$	$*$

. (3.3)

If we represent 0 as logical false and $*$ as logical true, then the addition of the elements of the matrix family is equivalent to logical OR operation.

During multiplication of the field elements, if one of the elements is zero, then the result is also zero. If a random field element is multiplied with non-zero field element, then the result is some field element, which corresponds to a star in the matrix family.

Definition 7. The multiplication of the elements of the field $\mathbb{F}$ with elements in $\mathbb{F}^*$ is defined by the following multiplication table:

$\cdot$	0	$\neq 0$
0	0	0
$*$	0	$*$

, (3.4)

where $\neq 0$ is any non-zero field element.

Similarly to addition as defined in Table (3.3), if we represent zero as logical false and all other field elements as logical true, and matrix family $\mathbb{A}$ element 0 as logical false and element $*$ as logical true, then the multiplication of a field element and a matrix family element can be considered as logical AND operation.

Given these operations, the matrix family multiplication with a matrix over the field $\mathbb{F}$ is defined as usual matrix multiplication but with the addition and multiplication operations as defined in Definitions 6 and 7. The following example illustrates the new definition of matrix family multiplication with a matrix.

Example 3. *Let a (3×3)-dimensional matrix C over the field $\mathbb{F}$ be*

$$C = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Let $\mathbb{A}$ be a (3×3)-dimensional matrix family defined as

$$\mathbb{A} = \begin{bmatrix} * & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & * \\ \mathbf{0} & * & \mathbf{0} \end{bmatrix}.$$

The element $\mathbf{0}$ in the matrix family $\mathbb{A}$ is written in bold to distinguish it from the element 0 in $\mathbb{F}$.

Multiplying the matrix family $\mathbb{A}$ from the left by the matrix C results in

$$\begin{aligned} C\mathbb{A} &= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} * & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & * \\ \mathbf{0} & * & \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} 1\cdot * + 1\cdot\mathbf{0} + 0\cdot\mathbf{0} & 1\cdot\mathbf{0} + 1\cdot\mathbf{0} + 0\cdot * & 1\cdot\mathbf{0} + 1\cdot * + 0\cdot\mathbf{0} \\ 1\cdot * + 1\cdot\mathbf{0} + 1\cdot\mathbf{0} & 1\cdot\mathbf{0} + 1\cdot\mathbf{0} + 1\cdot * & 1\cdot\mathbf{0} + 1\cdot * + 1\cdot\mathbf{0} \\ 0\cdot * + 1\cdot\mathbf{0} + 1\cdot\mathbf{0} & 0\cdot\mathbf{0} + 1\cdot\mathbf{0} + 1\cdot * & 0\cdot\mathbf{0} + 1\cdot * + 1\cdot\mathbf{0} \end{bmatrix} \\ &\overset{\textit{Table 3.4}}{=} \begin{bmatrix} * + \mathbf{0} + \mathbf{0} & \mathbf{0} + \mathbf{0} + \mathbf{0} & \mathbf{0} + * + \mathbf{0} \\ * + \mathbf{0} + \mathbf{0} & \mathbf{0} + \mathbf{0} + * & \mathbf{0} + * + \mathbf{0} \\ \mathbf{0} + \mathbf{0} + \mathbf{0} & \mathbf{0} + \mathbf{0} + * & \mathbf{0} + * + \mathbf{0} \end{bmatrix} \\ &\overset{\textit{Table 3.3}}{=} \begin{bmatrix} * & \mathbf{0} & * \\ * & * & * \\ \mathbf{0} & * & * \end{bmatrix} \end{aligned}$$

Note 1. In the following sections, by abusing the notation, we use the product of integer matrices with matrices over $\mathbb{F}^*$. In that case, every non-zero integer is interpreted as 1 in $\mathbb{F}$ and integer zero is interpreted as $\mathbb{F}$. The multiplication is then performed as defined in Definition 7.

The following example illustrates the multiplication of a integer matrix with a matrix family.

Example 4. *Let a* (3×3)*-dimensional integer matrix* C *be*

$$C = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 5 & 6 \\ 0 & 7 & 8 \end{bmatrix}.$$

Let $\mathbb{A}$ *be a* (3×3)*-dimensional matrix family be the same as defined in Example 3. Multiplying* C *with* $\mathbb{A}$ *results in*

$$\begin{aligned} C\mathbb{A} &= \begin{bmatrix} 1 & 2 & 0 \\ 4 & 5 & 6 \\ 0 & 7 & 8 \end{bmatrix} \begin{bmatrix} * & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & * \\ \mathbf{0} & * & \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} 1 \cdot * + 2 \cdot \mathbf{0} + 0 \cdot \mathbf{0} & 1 \cdot \mathbf{0} + 2 \cdot \mathbf{0} + 0 \cdot \mathbf{0} & 1 \cdot \mathbf{0} + 2 \cdot * + 0 \cdot \mathbf{0} \\ 4 \cdot * + 5 \cdot \mathbf{0} + 6 \cdot \mathbf{0} & 4 \cdot \mathbf{0} + 5 \cdot \mathbf{0} + 6 \cdot * & 4 \cdot \mathbf{0} + 5 \cdot * + 6 \cdot \mathbf{0} \\ 0 \cdot * + 7 \cdot \mathbf{0} + 8 \cdot \mathbf{0} & 0 \cdot \mathbf{0} + 7 \cdot \mathbf{0} + 8 \cdot * & 0 \cdot \mathbf{0} + 7 \cdot * + 8 \cdot \mathbf{0} \end{bmatrix} \\ &\overset{Note\ 1}{=} \begin{bmatrix} * + \mathbf{0} + \mathbf{0} & \mathbf{0} + \mathbf{0} + \mathbf{0} & \mathbf{0} + * + \mathbf{0} \\ * + \mathbf{0} + \mathbf{0} & \mathbf{0} + \mathbf{0} + * & \mathbf{0} + * + \mathbf{0} \\ \mathbf{0} + \mathbf{0} + \mathbf{0} & \mathbf{0} + \mathbf{0} + * & \mathbf{0} + * + \mathbf{0} \end{bmatrix} \\ &\overset{Table\ 3.3}{=} \begin{bmatrix} * & \mathbf{0} & * \\ * & * & * \\ \mathbf{0} & * & * \end{bmatrix} \end{aligned}$$

3.3.2 Adjacency matrix in set reconciliation

It is important to obtain the precise degree of solvability. The following trivial lemma gives the smallest such n.

Lemma 6. *Let* D *be the adjacency matrix of the graph. Let* l *be the smallest positive integer such that* $\sum_{i=1}^{l} D^i$ *is a positive matrix. Then the graph is* l*-solvable.*

Proof. The proof consists of two steps. We firstly show that there exists no smaller l', $l' < l$, such that the graph is l'-solvable. Then, a set reconciliation algorithm running in l iterations is constructed.

By contradiction we assume that there exists l' such that $l' < l$. As the element in r-th row and s-th column of D^i denotes the number of paths of length i between the nodes v_r and v_s, then there exists r^* and s^* such that the shortest path between the nodes v_{r^*} and v_{s^*} has length l. Now, if one of these nodes has an unique object x, then the minimum number of protocol iterations is l, as the distance between the other host and the object decreases only by one during each iteration. So the minimum number of iterations is l, contradicting our assumption.

We can construct a naive set reconciliation algorithm which reconciles the sets in l rounds. The set reconciliation algorithm transmits all objects of each of the node during each of the round. Upon receiving the objects, each of the node v_i adds new objects to its set X_i. As the longest path has distance l, then after l rounds the sets of objects are reconciled. $\square$

We can now formulate a lemma which shows the connection between the possession matrices of the graph after a single iteration of the set reconciliation protocol.

Lemma 7. *Let $\mathbb{A}$ be the possession matrix as defined in Equation (3.1). Let D be the adjacency matrix of the graph. Let E be a $(n \times n)$-dimensional ones matrix. After performing one round of the protocol, the new possession matrix $\mathbb{A}_+$ is related to $\mathbb{A}$ as*

$$\mathbb{A}_+ = (D \otimes E)\mathbb{A}.$$

Proof. We analyze, how elements in $\mathbb{A}_+$ are calculated.

From the definition of $\mathbb{A}$ in Equation (3.1), the matrix families $\mathbb{A}_i$, $i \in [k]$, have n identical rows. If we define a $(n \times 1)$-dimensional integer matrix R as

$$R = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix},$$

then, according to Note 1, we can write $\mathbb{A}_i = \mathbb{A}_i^* \otimes R$, where $\mathbb{A}_i^*$ is a $(1 \times n)$-dimensional matrix which consists of a single row of $\mathbb{A}_i$. From the definition of tensor product, this also means that

$$\mathbb{A} = \mathbb{A}^* \otimes R,$$

where

$$\mathbb{A}^* = \begin{bmatrix} \mathbb{A}_1^* \\ \vdots \\ \mathbb{A}_k^* \end{bmatrix}.$$

The right hand side of the claim can be written as

$$\begin{aligned}(D \otimes E)\mathbb{A} &= (D \otimes E)(\mathbb{A}^* \otimes R) \\ &\overset{(1)}{=} (D\mathbb{A}^*) \otimes (ER) \\ &= (D\mathbb{A}^*) \otimes (nR) \\ &\overset{(2)}{=} (D\mathbb{A}^*) \otimes R.\end{aligned}$$

The equality (1) holds because of the properties of the tensor product. The equality (2) holds because as mentioned in Note 1, any non-zero entry in nR is mapped to the field element 1 in $\mathbb{F}$ and thus we can omit a non-zero factor.

We now look at the product $D\mathbb{A}^*$ and later apply the tensor product of R.

Let $\mathbb{A}^* = (a^*_{i,j})^{j=1,\dots,n}_{i=1,\dots,k}$, $D = (d_{i,j})^{j=1,\dots,k}_{i=1,\dots,k}$ and $D\mathbb{A}^* = (\overline{a}_{i,j})^{j=1,\dots,n}_{i=1,\dots,k}$.

We perform matrix multiplication according to Definitions 6 and 7. The elements of $D\mathbb{A}^*$ are

$$\overline{a}_{i,j} = \sum_{\rho \in [k]} d_{i,\rho} a^*_{\rho,j}.$$

The element $\overline{a}_{i,j}$ is $*$, if there exists $\rho \in [k]$ such that $d_{i,\rho}$ is non-zero and $a^*_{\rho,j}$ is a star. This corresponds to the nodes v_i and v_ρ having an edge and the node v_ρ having an object x_j.

On the other hand, if $\overline{a}_{i,j} = *$, then this corresponds to the node v_i having an object x_j. Since during the set reconciliation protocol the objects between the nodes having an edge are reconciled, these two conditions coincide as $\overline{a}_{i,j}$ denotes if the node has an element after an iteration of the protocol.

If we take the tensor product $(D\mathbb{A}^*) \otimes R$, then we obtain the possession matrix of the graph $\mathcal{H}$ after an iteration of the protocol as defined in Equation (3.1). $\square$

This lemma can be extended for protocols with several rounds. We denote the possession matrix $\mathbb{A}$ before the first round as $\mathbb{A} = \mathbb{A}^{(0)}$. We denote by $\mathbb{A}^{(i)}$ the possession matrix after i-th round of the protocol.

Corollary 2. *Let $\mathbb{A}^{(0)}$ be the possession matrix of the graph before the first run of the protocol. Let the matrix E be a $(n \times n)$-dimensional ones matrix. The possession matrix after i-th round is*

$$\mathbb{A}^{(i)} = (D^i \otimes E)\mathbb{A}^{(0)}.$$

Proof. From the properties of the tensor product and matrix products:

$$\begin{aligned}
\mathbb{A}^{(i)} &= (D \otimes E)^i \mathbb{A}^{(0)} \\
&= (D^i \otimes E^i)\mathbb{A}^{(0)} \\
&= (D^i \otimes n^{i-1}E)\mathbb{A}^{(0)} \\
&= n^{i-1}(D^i \otimes E)\mathbb{A}^{(0)} \\
&\overset{(1)}{=} (D^i \otimes E)\mathbb{A}^{(0)}
\end{aligned}$$

The equality (1) holds since by Note 1, any non-zero integer entry in $(D^i \otimes E)$ is mapped to the element 1 in $\mathbb{F}$ and $n^{i-1} > 0$, the factor n^{i-1} can be omitted. □

3.3.3 Broadcast set reconciliation algorithm using rank optimization problem

The following theorem is the main result of this Thesis.

Theorem 4. *Let the graph $\mathcal{H}$ be l-solvable undirected graph defined by the adjacency matrix D. Let $\mathbb{A}$ be the corresponding possession matrix of the graph. Then there exists an iterated data exchange protocol with l rounds and τ transmissions, where*

$$\tau = \sum_{i=1}^{l} \min_{A^{(i)} \in (D^{i-1} \otimes E)\mathbb{A}} \sum_{j=1}^{k} \operatorname{rank} A_j^{(i)}$$

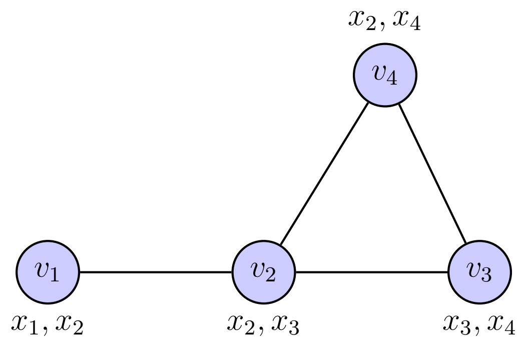
for matrices $A^{(i)}$ which are subject to

$$\operatorname{rank} \begin{bmatrix} (\operatorname{diag}(D_{j,\star}) \otimes I)\, A^{(i)} \\ B_j((D^{i-1} \otimes E)\mathbb{A}) \end{bmatrix} = \text{max-rank}\left[(\operatorname{diag}(e_j) \otimes I)(D^i \otimes E)\mathbb{A}\right], \quad \forall j \in [k], \tag{3.5}$$

where the matrix I is $(n \times n)$-dimensional identity matrix and the matrix E is $(n \times n)$-dimensional ones matrix.

Consider the following motivating example, which illustrates the usefulness of the condition (3.5).

Example 5. *Assume that the graph $\mathcal{H}$ consist of four nodes. The universe of files is $X = \{x_1, x_2, x_3, x_4\}$. Thus, $k = 4$ and $n = 4$.*



$$D = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 3.5: The graph $\mathcal{H}$ and the corresponding adjacency matrix D used in Example 5

For these parameters, the (4×4)*-dimensional matrices* I *and* E *are*

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

The $(n \times n)$*-dimensional possession matrices for the nodes* v_j*,* $j \in [k]$ *are*

$$\mathbb{A}_1 = \begin{bmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ * & * & 0 & 0 \\ * & * & 0 & 0 \end{bmatrix},$$

$$\mathbb{A}_2 = \begin{bmatrix} 0 & * & * & 0 \\ 0 & * & * & 0 \\ 0 & * & * & 0 \\ 0 & * & * & 0 \end{bmatrix},$$

$$\mathbb{A}_3 = \begin{bmatrix} 0 & 0 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix},$$

$$\mathbb{A}_4 = \begin{bmatrix} 0 & * & 0 & * \\ 0 & * & 0 & * \\ 0 & * & 0 & * \\ 0 & * & 0 & * \end{bmatrix}.$$

The $(nk \times n)$-dimensional possession matrix of the graph $\mathcal{H}$ is

$$\mathbb{A}^{(0)} = \begin{bmatrix} * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & * & * & * & * & 0 & 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & 0 & * & * & * & * & * & * & * & * & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & * & * & * & * & * & * & * & * \end{bmatrix}^T .$$

The smallest l such that $\sum_{i=1}^{l} D^i$ is positive, is $l = 2$. Using Lemma 6, we obtain that the number of rounds is $l = 2$.

We calculate the right-hand side of the Equation (3.5). Multiplying $\mathbb{A}$ by $(D^1 \otimes E)$, we obtain

$$\mathbb{A}^{(1)} = \left(D^1 \otimes E\right) \mathbb{A} =$$

$$= \begin{bmatrix} * & * & * & * & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\ 0 & 0 & 0 & 0 & * & * & * & * & * & * & * & * & * & * & * & * \end{bmatrix}^T .$$

After the first round, the corresponding sets of objects are

$$\begin{aligned} X_1 &= \{x_1, x_2, x_3\}, \\ X_2 &= \{x_1, x_2, x_3, x_4\}, \\ X_3 = X_4 &= \{x_2, x_3, x_4\}. \end{aligned}$$

We see that the ranks in the right-hand side of Equation (3.5) denote the sizes of the hosts sets.

$$j = 1: \quad (\operatorname{diag}(e_1) \otimes I)\mathbb{A}^{(1)} = \begin{bmatrix} * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T ,$$

$$j = 2: \quad (\operatorname{diag}(e_2) \otimes I)\mathbb{A}^{(1)} = \begin{bmatrix} 0 & 0 & 0 & 0 & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T ,$$

$$j = 3: \quad (\operatorname{diag}(e_3) \otimes I)\mathbb{A}^{(1)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & * & * & * & * & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & * & * & * & * & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & * & * & * & * & 0 & 0 & 0 & 0 \end{bmatrix}^T ,$$

$$j = 4: \quad (\mathrm{diag}(e_4) \otimes I)\mathbb{A}^{(1)} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&*&*&*&* \\ 0&0&0&0&0&0&0&0&0&0&0&0&*&*&*&* \\ 0&0&0&0&0&0&0&0&0&0&0&0&*&*&*&* \end{bmatrix}^T.$$

It is trivial to compute the rank on these matrices. In a sub-block containing symbols $*$, we count the number of non-zero columns. We can replace the symbols $*$ in each of these rows with a canonical vector, forming a max-rank matrix. The number of columns defines how many canonical vectors we can use.

$$\begin{aligned} j = 1: &\qquad \text{max-rank}\left((\mathrm{diag}(e_1) \otimes I)\mathbb{A}^{(1)}\right) = 3, \\ j = 2: &\qquad \text{max-rank}\left((\mathrm{diag}(e_2) \otimes I)\mathbb{A}^{(1)}\right) = 4, \\ j = 3: &\qquad \text{max-rank}\left((\mathrm{diag}(e_3) \otimes I)\mathbb{A}^{(1)}\right) = 3, \\ j = 4: &\qquad \text{max-rank}\left((\mathrm{diag}(e_4) \otimes I)\mathbb{A}^{(1)}\right) = 3. \end{aligned}$$

If the right-hand side of the Equation (3.5) shows the size of the sets the nodes should have after a round, then the left-hand side of the Equation (3.5) shows what is achieved after specific transmission.

For each of the nodes, the left-hand side of the Equation (3.5) shows the size of its set after it has received the combination of the elements from the other nodes.

More specifically, the upper part of the concatenated matrix denotes the received vectors. The lower part denotes the elements the node already has. The rank of this matrix thus estimates the number of elements in the host's set after the round.

As the required $A^{(i)}$ must satisfy Equation (3.5) for all $j \in [k]$, then this implies that after a round the required sizes of the sets are achieved.

We look at some $A^{(1)}$, which satisfies Equation (3.5). Even though the Theorem 4 iterates over all transmission matrices $A^{(1)} \in \mathbb{A}$, it is possible to reduce the search space. We see that x_1 is unique to the node v_1, thus it is mandatory for the node v_1 to send some linear combination including x_1. We also see, that pairs (v_2, v_3), (v_3, v_4) and (v_2, v_4) have the elements x_3, x_4 and x_2 in common, respectively. Therefore, it is not necessary to send redundant information.

$$A^{(1)} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \end{bmatrix}^T,$$

$$A^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T,$$

$$A^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T.$$

As the rank of all these examples is 3*, then we can pick any* $A^{(1)}$*. We take the first one to see what is happening on the left-hand side of the Equation (3.5).*

We can find $B_j\left((D^0 \otimes E)\mathbb{A}\right), j \in [k]$*. As per definition of* B_j*:*

$$B_1(\mathbb{A}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

$$B_2(\mathbb{A}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$B_3(\mathbb{A}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$B_4(\mathbb{A}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Now, we find the upper part of the matrix, concatenate it with matrix found previously and calculate the rank

$$\left(\operatorname{diag}(D_{1,\star}) \otimes I\right) A^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T,$$

$$\left(\operatorname{diag}(D_{2,\star}) \otimes I\right) A^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T,$$

$$\left(\operatorname{diag}(D_{3,\star}) \otimes I\right) A^{(1)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T,$$

$$(\mathrm{diag}(D_{4,\star}) \otimes I)\, A^{(1)} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \end{bmatrix}^T .$$

$$\begin{bmatrix} (\mathrm{diag}(D_{1,\star}) \otimes I)\, A^{(1)} \\ B_1(\mathbb{A}) \end{bmatrix} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}^T ,$$

$$\begin{bmatrix} (\mathrm{diag}(D_{2,\star}) \otimes I)\, A^{(1)} \\ B_2(\mathbb{A}) \end{bmatrix} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}^T ,$$

$$\begin{bmatrix} (\mathrm{diag}(D_{3,\star}) \otimes I)\, A^{(1)} \\ B_3(\mathbb{A}) \end{bmatrix} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1 \end{bmatrix}^T ,$$

$$\begin{bmatrix} (\mathrm{diag}(D_{4,\star}) \otimes I)\, A^{(1)} \\ B_4(\mathbb{A}) \end{bmatrix} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1 \end{bmatrix}^T .$$

$$\begin{aligned} \mathrm{rank} \begin{bmatrix} (\mathrm{diag}(D_{1,\star}) \otimes I)\, A^{(1)} \\ B_1(\mathbb{A}) \end{bmatrix} &= 3, \\ \mathrm{rank} \begin{bmatrix} (\mathrm{diag}(D_{2,\star}) \otimes I)\, A^{(1)} \\ B_2(\mathbb{A}) \end{bmatrix} &= 4, \\ \mathrm{rank} \begin{bmatrix} (\mathrm{diag}(D_{3,\star}) \otimes I)\, A^{(1)} \\ B_3(\mathbb{A}) \end{bmatrix} &= 3, \\ \mathrm{rank} \begin{bmatrix} (\mathrm{diag}(D_{4,\star}) \otimes I)\, A^{(1)} \\ B_4(\mathbb{A}) \end{bmatrix} &= 3. \end{aligned}$$

For this specific transmission matrix $A^{(1)}$, the node v_1 transmits the messages $T_1 = \{x_1\}$, the node v_2 transmits the messages $T_2 = \{x_2 + x_3\}$, the node v_3 transmits the messages $T_3 = \{x_4\}$ and the node v_4 does not transmit anything as $T_4 = \emptyset$.

The node v_1 can recover the object x_3 from the linear equation $x_3 = T_{2,1} - x_2 = x_2 + x_3 - x_2$. The node v_2 receives $x_1 = T_{1,1}$ from the node v_1. The node v_3 recovers the object x_2 from the linear equation $x_2 = T_{2,1} - x_3 = x_2 + x_3 - x_3$. The node v_4 recovers the object x_3 from the linear equation $x_3 = T_{2,1} - x_2 = x_2 + x_3 - x_2$.

We now move on to the second round $i = 2$.

The possession matrix $\mathbb{A}^{(2)}$ can be calculated from $\mathbb{A}^{(0)}$ or $\mathbb{A}^{(1)}$:

$$\mathbb{A}^{(2)} = (D \otimes E)\,\mathbb{A}^{(1)} = \left(D^2 \otimes E\right)\mathbb{A}^{(0)} =$$

$$= \begin{bmatrix} * & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \end{bmatrix}^T .$$

This possession matrix is expected as the Lemma 6 gave the number of rounds to be $l = 2$. This also means that $X_1 = X_2 = X_3 = X_4 = \{x_1, x_2, x_3, x_4\}$ after the second round.

We omit the calculation of the ranks in the right-hand side of the equation. It can be easily checked that the ranks are all equal to 4.

A transmission matrix $A^{(2)}$ which satisfies the Equation (3.5) is

$$A^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

This corresponds to only the node v_2 transmitting $x_1 + x_4$.

We are omitting further calculations as they are done similarly to the ones in round $i = 1$. As $\tau_1 = \sum_{j=1}^{k} \operatorname{rank} A_j^{(1)} = 3$ and $\tau_2 = \sum_{j=1}^{k} \operatorname{rank} A_j^{(2)} = 1$, then the data exchange protocol requires only $\tau = \tau_1 + \tau_2 = 4$ transmissions to reconcile the sets of the nodes.

For the proof, we formulate the following lemma.

Lemma 8. *Let $\mathcal{H}$ be an undirected graph defined by the adjacency matrix D. Let the possession matrix of the graph $\mathcal{H}$ be $\mathbb{A}$ as defined in Equation (3.1). There exists a transmission matrix $A \in \mathbb{A}$ as defined in Equation (3.2) such that*

$$\operatorname{rank}\begin{bmatrix} (\operatorname{diag}(D_{j,\star}) \otimes I)A \\ B_j(\mathbb{A}) \end{bmatrix} = \text{max-rank}\left[(\operatorname{diag}(e_j) \otimes I)(D \otimes E)\mathbb{A}\right]$$

for all $j \in [k]$ and where E and I are $(n \times n)$-dimensional ones and identity matrices, respectively.

Proof. We use the same $(n \times 1)$-dimensional matrix R as defined in the proof of Lemma 7.

The right-hand side of the equation can be written as

$$\begin{aligned}(\mathrm{diag}(e_j) \otimes I)(D \otimes E)\mathbb{A} &= (\mathrm{diag}(e_j) \otimes I)(D \otimes E)(\mathbb{A}^* \otimes R)\\ &= (\mathrm{diag}(e_j)D\mathbb{A}^*) \otimes (IER)\\ &= (\mathrm{diag}(e_j)D\mathbb{A}^*) \otimes (ER)\\ &= (\mathrm{diag}(e_j)D\mathbb{A}^*) \otimes (nR)\\ &\overset{(1)}{=} (\mathrm{diag}(e_j)D\mathbb{A}^*) \otimes R\end{aligned}$$

The equality (1) holds because $n > 0$ and all non-zero integers are mapped to field element 1, thus we can omit the factor.

Let $D = (d_{i,j})_{i=1,\dots,k}^{j=1,\dots,k}$, $\mathbb{A}^* = (a^*_{i,j})_{i=1,\dots,k}^{j=1,\dots,n}$ and $D\mathbb{A}^* = (\overline{a}_{i,j})_{i=1,\dots,k}^{j=1,\dots,n}$.

From the proof of Lemma 7, we know that

$$\overline{a}_{i,j} = \sum_{\rho\in[k]} d_{i,\rho}a^*_{\rho,j}.$$

If $D\mathbb{A}^*$ is multiplied from the left by $\mathrm{diag}(e_j)$, then in the resulting matrix only j-th row is non-zero. The elements in j-th row are $\overline{a}_{j,\eta}$, $\eta \in [n]$.

The resulting $(k \times n)$-dimensional matrix $\mathrm{diag}(e_j)D\mathbb{A}^*$ is

$$\mathrm{diag}(e_j)D\mathbb{A}^* = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \dots & \vdots \\ 0 & \dots & 0 \\ \sum_{\rho\in[k]} d_{j,\rho}a^*_{\rho,1} & \dots & \sum_{\rho\in[k]} d_{j,\rho}a^*_{\rho,n} \\ 0 & \dots & 0 \\ \vdots & \dots & \vdots \\ 0 & \dots & 0 \end{bmatrix}. \tag{3.6}$$

The max-rank of the matrix family $(\mathrm{diag}(e_j)D\mathbb{A}^*) \otimes R$ is the number of the symbols $*$ in the non-zero row of the matrix in Equation (3.6). The max-rank can not be larger because $*$ denotes the elements of the matrix family which can be chosen freely. This max-rank can be achieved because the dimension of R is $(n \times 1)$ and this means that up to n independent row vectors can be chosen.

We look at the upper part of the left-hand side. Let $A = (a^?_{i,j})_{i=1,\dots,kn}^{j=1,\dots,n}$. As we are showing the existence of the transmission matrix A, the values of the elements $a^?_{i,j}$ are not known. We show how the values are chosen.

The matrix $\operatorname{diag}(D_{j,\star}) \otimes I$ is

$$\operatorname{diag}(D_{j,\star}) \otimes I = \begin{bmatrix} d_{j,1} & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & \ddots & 0 & \dots & \dots & \dots & 0 \\ \vdots & 0 & d_{j,1} & 0 & \dots & \dots & 0 \\ \vdots & \dots & 0 & \ddots & 0 & \dots & 0 \\ \vdots & \dots & \dots & 0 & d_{j,k} & 0 & 0 \\ \vdots & \dots & \dots & \dots & 0 & \ddots & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 & d_{j,k} \end{bmatrix}$$

If this matrix is multiplied by A, we obtain

$$(\operatorname{diag}(D_{j,\star}) \otimes I)A = \begin{bmatrix} d_{j,1}a^?_{1,1} & \dots & d_{j,1}a^?_{1,n} \\ \vdots & \dots & \vdots \\ d_{j,1}a^?_{n,1} & \dots & d_{j,1}a^?_{n,n} \\ \vdots & \dots & \vdots \\ d_{j,k}a^?_{(k-1)n+1,1} & \dots & d_{j,1}a^?_{(k-1)n+1,n} \\ \vdots & \dots & \vdots \\ d_{j,k}a^?_{kn,1} & \dots & d_{j,1}a^?_{kn,n} \end{bmatrix}. \tag{3.7}$$

The element in the j-th row and i-th column in the matrix family in Equation (3.6) is $*$, if there exists γ such that $d_{j,\gamma} > 0$ and $a^*_{\gamma,i} = *$. In this case, we can choose some $s \in [n]$ and set $a^?_{(\gamma-1)n+s,i}$ to 1. Then also $d_{j,\gamma}a^?_{(\gamma-1)n+s,i} > 0$.

As we can choose s from the sequence $[n]$, then a different s can be chosen for every $i \in [n]$. After setting an element to 1 in every column $i \in [n]$, we set the values of all other elements in A to 0. Because the ones in the matrix in Equation (3.7) are all in distinct rows and in distinct columns, then the rank of the matrix $(\operatorname{diag}(D_{j,\star}) \otimes I)A$ equals to the max-rank of the matrix family $(\operatorname{diag}(e_j)D\mathbb{A}^*) \otimes R$.

The matrix $A = (a^?_{i,j})$ belongs to the matrix family $\mathbb{A}$, because if $a^*_{i,j} = *$, then $a_{(i-1)n+s,j} = *, s \in [n]$, in $\mathbb{A}$ and only elements in these places are set to 1 in A.

Because there exists $\mathcal{A} \in \mathbb{A}$ such that $\operatorname{rowspace}(B_j(\mathbb{A})) = \operatorname{rowspace}(\mathcal{A})$ and the upper part has achieved at least the max-rank of this family, then the rank of the left-hand side does not change if we concatenate $B_j(\mathbb{A})$ to $(\operatorname{diag}(D_{j,\star}) \otimes I)A$. $\square$

The proof of this lemma showed that the transmission matrix exists. However, it may not be optimal. For example, there can exist A which results in no transmission by some of the nodes, as we saw in Example 5.

We can now move on to the proof of Theorem 4.

Proof of Theorem 4. Using Lemmas 7 and 8 we observed that matrices $A^{(i)}$ exist.

Let $A^{(i)} = (a^{(i)}_{\rho,\eta})^{\eta=1,\ldots,n}_{\rho=1,\ldots,kn}$. Let $A^{(i)}_s$, $s \in [k]$, be the matrices as in Equation (3.2).

For every $s \in [k]$, let $\boldsymbol{t}^{(i)}_{s,r} = (t^{(i)}_{s,r,1}, \ldots, t^{(i)}_{s,r,n})$, be the r-th row vectors of $A^{(i)}_s$. These row vectors are the linear coefficients for the objects transmitted by the node v_s during the i-th iteration of the protocol. At the i-th iteration of the protocol, the message space for the node v_s is

$$T^{(i)}_s = \left\{ \sum_{m\in[n]} t^{(i)}_{s,r,m} x_m | r \in [n] \right\} \tag{3.8}$$

$$= \left\{ \boldsymbol{t}^{(i)}_{s,r} \cdot \boldsymbol{x} | r \in [n] \right\}, \tag{3.9}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)$ and the product is a dot product of the vectors.

The node v_s transmits at the i-th iteration the linearly independent subset of messages from Equation (3.9). Thus the number of transmissions by the node v_s during i-th iteration of the protocol is the rank of the matrix $A^{(i)}_s$. If summed over all $s \in [k]$, then this corresponds to the number of transmissions in the Theorem 4.

As each of the nodes receives messages from the nodes it has an edge to, then the node v_t, $t \in [k]$, receives the messages from the node v_s if $d_{t,s} = 1$. The received messages from the node v_s are $T^{(i)}_s$ as defined in Equation (3.9).

If we consider the messages not received by the node v_t as 0, then we can write that the node v_t receives the messages

$$d_{t,s} \sum_{m\in[n]} t^{(i)}_{s,r,m} x_m,$$

for all $s \in [k]$, $r \in [n]$.

Now, if we write $t^{(i)}_{s,r,m} = a^{(i)}_{(s-1)n+r,m}$, then the messages received by the node v_t form a matrix

$$\begin{bmatrix} d_{1,1}a^{(i)}_{1,1} & \ldots & d_{1,1}a^{(i)}_{1,n} \\ \vdots & \ldots & \vdots \\ d_{1,1}a^{(i)}_{n,1} & \ldots & d_{1,1}a^{(i)}_{n,n} \\ \vdots & \ldots & \vdots \\ d_{k,1}a^{(i)}_{(k-1)n+1,1} & \ldots & d_{k,1}a^{(i)}_{(k-1)n+1,n} \\ \vdots & \ldots & \vdots \\ d_{k,1}a^{(i)}_{kn,1} & \ldots & d_{k,1}a^{(i)}_{kn,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}. \tag{3.10}$$

From Lemma 7, the matrix family $(D^i \otimes E)\mathbb{A}$ denotes the possession matrix of the graph after the round i. Thus the matrix family $(\operatorname{diag}(e_j) \otimes I)(D^i \otimes E)\mathbb{A}$ denotes the possession matrix of the node v_j after the round i. The max-rank of this matrix family denotes the number of the elements the node v_j has after the i-th round.

From the proof of Lemma 8, we see that the left-hand side of the condition equals to the matrix in Equation (3.10). As the condition holds for the given $A^{(i)}$, then the rank of the matrix in Equation (3.10) concatenated with the matrix $B_j(\mathbb{A})$ is the max-rank of the right-hand side of the condition. As the rank equals to the max-rank, then the received messages are sufficient to recover all the missing objects. □

For comparison, we consider the naive set reconciliation protocol, where all nodes sequentially transmit all objects they possess. The number of transmissions by the node v_s at the i-th round for this protocol is max-rank$(\mathbb{A}_s^{(i-1)})$. It can be easily seen, that the number of the transmissions for the protocol defined in Theorem 4 is upper bounded by the number of transmission by this naive protocol.

Chapter 4

Conclusion and future work

We studied the problem of set reconciliation. The goal of set reconciliation algorithms is to find a union of distinct sets while optimizing the communicational cost. The expected communicational cost should be dependent only on the size of the symmetric difference of the sets being reconciled. Such algorithms are known for settings where there are only two nodes in the network.

These settings however do not model the actual networks where there are many nodes with arbitrary connections between them. Good algorithms for set reconciliation in general networks are still unknown. Moreover, even the optimal communication cost in this setting is unknown.

In this Thesis, we categorized different network models. Some models were given in the survey [MV12], but our definitions are more detailed and we included two new models. The broadcast network model captures the properties of wireless networks and is relevant in many applications. The threshold reconciliation model was proposed as a research subject.

We studied the set reconciliation problem in the star graph in an unicast network model. In this setting, compared to performing set reconciliation pairwise, the communicational cost was reduced by factor of $1/k$ where k is the number of the nodes in the network. The result suggested that if the topology of the network is known, then it is possible to reduce the communicational cost. Very recently, a similar observation for set reconciliation using invertible Bloom lookup tables was made by Mitzenmacher and Pagh in [MP13].

Similarly to gossip protocols, a node could choose a random neighbouring node and perform pairwise set reconciliation with this node. We defined the weights on the edges in the network graph according to the number of different objects in the communicating hosts. We experimented with an approach where communicating pairs were chosen according to maximum weight matching. The results show that this approach gave better results in 99.8% of the tested cases.

For set reconciliation problem in broadcast networks, we formulated an analytical framework, where the number of transmissions is given by the solution of algebraic minimization problem. Solving that problem yields a set reconciliation protocol that uses linear coding for transmitting the objects. Even though we do not know the optimal communicational cost in this model, the resulting protocol outperforms the naive set reconciliation algorithm where all items are transmitted by each node. To our knowledge, this is first algebraic analytical framework for a set reconciliation protocols in a broadcast network model.

There are still many open questions related to the set reconciliation problem. There may be better approaches to solve the problem for arbitrary unicast and broadcast network. For example, techniques from the area of gossip protocols could provide some insight into the reconciliation algorithms, but the relation between gossip protocols and set reconciliation is still not fully established.

The results of this research can potentially be used in the industry as cloud storage platforms have become widely used. Providing more efficient synchronization algorithms could lead to reducing the costs of data exchange. It would be interesting to implement the developed methods and to test their efficiency in practice.

Experiments on network matching

It could be further studied, how different matching algorithms work with unicast networks. The experiments which were covered in Section 3.2, provide only initial intuition and to make further conclusions, different models could be tested.

The experiments could be conducted in a virtual model using generated data or tested in a more realistic environment. In addition, distributed file sharing models should be considered for content distribution networks.

Also, it is not completely clear how the round cost and time cost are related. We assumed that trying to minimize the number of rounds, we could also minimize the number of messages transmitted. However, this could be false assumption. It would be interesting to construct matching algorithms, which would reduce the number of transmitted messages and to compare these algorithms to algorithms which minimize the number of rounds.

Security guarantees of the algorithms

It would be interesting to study the security of the proposed algorithms. In larger graphs, it is assumed that the nodes possess some side-information. If there is an adversary somewhere in the graph, then what does it need to possess to eavesdrop transmitted messages?

For example, could some of the algorithms be used in multi-party computation for secure information sharing?

Relation between unicast and broadcast networks

We presented an algebraic framework for the analysis of set reconciliation algorithms in broadcast networks using linearly coded messages. It would be interesting to obtain analogous results for unicast networks.

Metric in an arbitrary graph

The size of the symmetric difference was a good metric for pairwise set reconciliation. How can one give a similar metric which describes the distribution of the messages in the whole graph? Having a good characteristic of the graph, it is easier to analyze the performance of set reconciliation algorithm.

Some possible metrics are the sum of pairwise distances; the sum of $|X_{\text{all}} \setminus X_i|$ for all $i \in [k]$; and the number of messages needed for reconciliation.

Practical implementations

We have written an implementation of set reconciliation algorithm using characteristic polynomials which is available at [Kub13]. We also wrote a library to test different matching algorithms which is available at [Kub14].

Practical implementations could gain traction in the open-source community and this could lead to new ideas for optimizing the algorithms. Also, the current algorithms have been given in mathematical notation. We have not specified how the nodes should interact and how to represent information. For example, in Section 2.1, the objects were elements of a field but this does not describe the model if we want to reconcile files.

Bibliography

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[EGUV11] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the difference?: Efficient set reconciliation without prior context. *SIGCOMM Comput. Commun. Rev.*, 41(4):218–229, August 2011.

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables. In *Communication, Control, and Computing, 49th Annual Allerton Conference on*, pages 792–799. IEEE, 2011.

[JSC$^+$05] Sidharth Jaggi, Peter Sanders, Philip A. Chou, Michelle Effros, Sebastian Egner, Kamal Jain, and Ludo M. G. M. Tolhuizen. Polynomial time algorithms for multicast network code construction. *Information Theory, IEEE Transactions on*, 51(6):1973–1982, June 2005.

[KK08] Ralf Koetter and Frank R. Kschischang. Coding for errors and erasures in random network coding. *Information Theory, IEEE Transactions on*, 54(8):3579–3591, 2008.

[KS98] Erich Kaltofen and Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. *Mathematics of Computation of the American Mathematical Society*, 67(223):1179–1197, 1998.

[Kub13] Ivo Kubjas. Polynomial reconciliation solver. `https://github.com/ivokub/polyrecon`, 2013.

[Kub14] Ivo Kubjas. Network matching library. `https://github.com/ivokub/netmatching`, 2014.

[MM13] Jaban Meher and M. Ram Murty. Ramanujan's proof of Bertrand's postulate. *The American Mathematical Monthly*, 120(7):650–653, August-September 2013.

[MP13] Michael Mitzenmacher and Rasmus Pagh. Simple multi-party set reconciliation. *CoRR*, abs/1311.2037, 2013.

[MTZ03] Yaron Minsky, Ari Trachtenberg, and Richard Zippel. Set reconciliation with nearly optimal communication complexity. *Information Theory, IEEE Transactions on*, 49(9):2213–2218, 2003.

[MV12] Michael Mitzenmacher and George Varghese. The complexity of object reconciliation, and open problems related to set difference and coding. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 1126–1132, Oct 2012.

[RSS10] Salim Y. El Rouayheb, Alexander Sprintson, and Parastoo Sadeghi. On coding for cooperative data exchange. *CoRR*, abs/1002.1465, 2010.

[SB93] Josef Stoer and Roland Z. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, New York, 2nd edition, 1993.

[Tri99] Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, The Australian National University, 1999.

[Zip93] Richard E. Zippel. *Effective Polynomial Computation*. Kluwer Academic Press, Boston, 1993.

Appendix A

Pairwise set reconciliation using characteristic polynomials

```
import solve_recon

# Define sets and evaluation points, base and upper bound
set1 = [1, 2, 9, 12, 33]
set2 = [1, 2, 9, 10, 12, 28]
evpoints = [-1, -2, -3, -4, -5]
base = 97
m = 5

# Evaluates sets at given points
ev1 = solve_recon.evaluate(set1, evpoints, base)
ev2 = solve_recon.evaluate(set2, evpoints, base)
print ev1 # [mpz(58), mpz(19), mpz(89), mpz(77), mpz(4)]
print ev2 # [mpz(15), mpz(54), mpz(68), mpz(77), mpz(50)]

# Divides evaluation values
intvalues = solve_recon.divide(ev1, ev2, base)
print intvalues # [mpz(75), mpz(74), mpz(17), mpz(1), mpz(35)]

# Calculate polynomial bounds
d1, d2 = solve_recon.poly_bounds(set1, set2, m)
print d1, d2 # 2 3

```

```
# Create system we are going to solve
constraints = solve_recon.create_equations(evpoints,
              intvalues, d1, d2, base)
print constraints
# [[96 1 22 75 22 21]
#  [95 1 92 51 23 83]
#  [94 1 41 51 80 17]
#  [93 1 81  4 96 17]
#  [92 1 95 78 62 62]]

# Solve it
solved = solve_recon.solve(constraints, base)
print solved
# [[1 0 0 0 53 64]
#  [0 1 0 0 94  0]
#  [0 0 1 0 53 59]
#  [0 0 0 1 23 86]
#  [0 0 0 0  0  0]]

# See which values are independent
indep, dep = solve_recon.indep_solutions(solved, base)

# Output polynomial coefficients
coef_n, coef_d = solve_recon.rat_poly_sing(solved,
                 indep, d1, d2, base, coeff = True)
print coef_n # [(0, mpz(3)), (1, mpz(11))]
             # == x^2 + 11 x + 3
print coef_d # [(0, mpz(1)), (1, mpz(63)), (2, mpz(6))]
             # == x^3 + 6 x^2 + 63 x + 1

# or the polynomials themselves
fn_n, fn_d = solve_recon.rat_poly_sing(solved,
             indep, d1, d2, base, coeff = False)

# Test, if the polynomials are correct
sol1, sol2 = solve_recon.test(fn_n, fn_d, base)
print sol1  # [33]
print sol2  # [10, 28]
```