Entity Resolution for Distributed Probabilistic Data

Naser Ayat, Reza Akbarinia, Hamideh Afsarmanesh, Patrick Valduriez

Entity Resolution for Distributed Probabilistic Data

Naser Ayat[#1], Reza Akbarinia[*2], Hamideh Afsarmanesh[#3], Patrick Valduriez[*4]

[#]Informatics Institute, University of Amsterdam, Amsterdam, Netherlands

[1]s.n.ayat@uva.nl, [3]h.afsarmanesh@uva.nl

[*]INRIA and LIRMM, Montpellier, France

[2,4]{firstname.lastname@inria.fr}

ABSTRACT

The problem of entity resolution over probabilistic data (ERPD) arises in many applications that have to deal with probabilistic data. In many of these applications, probabilistic data is distributed among a number of nodes. The simple, centralized approach to the ERPD problem does not scale well as large amounts of data need to be sent to a central node. In this paper, we present FD, a fully distributed algorithm for dealing with the ERPD problem over distributed data, with the goal of minimizing bandwidth usage and reducing processing time. FD is completely distributed and does not depends on the existence of *certain* nodes. We validated FD through implementation over a 75-node cluster. We used both synthetic and real-world data in our experiments. Our performance evaluation shows that FD can achieve major performance gains in terms of bandwidth usage and response time.

1. INTRODUCTION

In recent years, we have been witnessing much interest in uncertain data management using probabilistic approaches in many application areas such as data integration [18], sensor networks [9, 13], information extraction [12], etc. Much research effort has been devoted to several aspects of uncertain data management, including data modeling [23, 8], skyline queries [6, 21], top-k queries [11, 24], nearest neighbor search [29], spatial queries [28], XML documents [4, 14], etc.

The main difference between a traditional certain database and an uncertain database is that an uncertain database represents a set of possible database instances, called *possible worlds*, rather than a single one.

An important problem that arises in many applications such as information integration is that of Entity Resolution (ER) [10]. ER is the process of identifying tuples that represent the same real-world entity. The problem of ER is challenging since the same entity can be encoded in different ways due to a variety of reasons such as different formatting conventions, abbreviations, and typographic errors.

The problem of *entity resolution over probabilistic data* (which we call ERPD) arises in many application domains that have to deal with probabilistic data, ranging from sensor databases to scientific data management. In this paper, we are interested in the following formulation of the ERPD problem[1]. Let e be an uncertain entity represented by multiple possible alternatives, i.e. tuples, each with a membership probability. Let D be an uncertain database composed of a set of uncertain entities. Then, given e, D, and a similarity function F, the problem is to find the entity-tuple pair (t, t_i) (where $t \in e, t_i \in D$) such that (t, t_i) has the highest cumulative probability to be the most similar in all possible worlds. This entity-tuple pair is called the *most probable match pair* of e and D, denoted as MPMP(e, D).

There have been recent proposals dealing with the ERPD problem [19, 20, 7]. They all deal with the uncertain data which is stored in a central database. However, many real-life applications, in which the ERPD problem arises, produce uncertain data distributed among a number of databases. Dealing with the ERPD problem for distributed data is quite important for such applications. Let us give two examples of such applications from image retrieval and scientific data management domains.

Example 1. Matching images in facial image database. For investigation purposes, the police keeps a database of facial image of all persons who leave the country. To gather such database, the police has installed supervenience video cameras in all ports including airports, seaports, and land border ports. In each port, video cameras capture the video of the persons who leave the country. A face recognition system then is used to extract each individual's facial image from captured videos, and store its feature vector as a tuple in a local database. Since the detected face might be moving in the video and the inherent uncertainty of the face recognition methods, each feature vector is associated with a probability value representing its degree of certainty (e.g. as in [30]). The local databases are connected through a distributed system which provides a query interface for querying the set of all databases from each port. A witness of a crime scene describes the facial features of a perpetrator. Since the witness is not sure about some of the features, the police represents the perpetrator's face using an uncertain entity, say e, consisting of a number of alternative feature

[1]Some texts (e.g. [25]) refer to this formulation of the ER problem as *Identity Resolution*. [7]

vectors each associated with a confidence value. An interesting query for the police is the following: *in the distributed database of all ports, find the person who is most probably the same person as e, where date > x (i.e. the date of the crime).*

Example 2. Finding astronomical objects in astrophysics data. In astrophysics, as well as in other scientific disciplines, the correlation and integration of observational data is the key for gaining new scientific insights. Astronomical observatories, distributed all over the world, produce data about sky surveys, some of which is uncertain [22]. In its simplified form, each observatory can maintain a single uncertain relation *Objects* that contains data about the observed *astronomical objects* in the sky surveys. Each *object* is represented using a number of alternative tuples each with a membership probability showing its degree of certainty. The alternatives are disjoint, meaning that at most one of them can be true. The astrophysics researchers who want to gather information about a particular astronomical object, which has been represented by an uncertain entity e, is very interested in the following query: *among astronomical objects observed in a sky region in all distributed observatories, find the object which is most probably the same object as a given object e.*

A straightforward approach to answer the above queries is to ask all distributed nodes to send their databases to a central node who deals with the problem of ER by using one of the existing centralized solutions, e.g. [7]. However, this approach is very expensive and does not scale well neither in the size of databases, nor in the number of nodes. Therefore, using a distributed algorithm for dealing with the ERPD problem over distributed data is inevitable.

In this paper, we propose FD, a fully distributed algorithm for dealing with the ERPD problem over distributed data, with the goal of minimizing bandwidth usage and reducing processing time. To the best of our knowledge, FD is the first proposal that deals with the ERPD problem over distributed data. It has the following salient features. First, it uses the novel concepts of *Potential* and *essential-set* to prune data at local nodes. This leads to a significant reduction of bandwidth usage compared to the baseline approaches. Second, its execution is completely distributed and does not depend on the existence of certain nodes. We validated FD through implementation over a 75-node cluster and simulation using both synthetic and real-world data. The results show very good performance, in terms of bandwidth usage and response time.

The rest of the paper is organized as follows. In Section 2, we make precise our assumptions and formally define the problem. In Section 3, we present FD, and analyze its communication cost. In Section 5, we report the performance evaluation of FD through implementation and simulation. Section 6 discusses related work, and Section 7 concludes.

2. PROBLEM DEFINITION

In this section, we first describe the probabilistic data model which we consider. Then, we define the problem of entity resolution over distributed probabilistic data.

Probabilistic Data Model

For representing an uncertain entity, we use the x-tuple data model [5], and represent each entity as a x-tuple. In each x-tuple there are several possible tuples, called alternatives. Each alternative is associated with a probability value showing its likelihood of truth. The alternatives are disjoint, meaning that at most one of them can be true, and the sum of the confidence values of the alternatives is less than or equal to one. As an example of x-tuple, consider the entity e in Figure 1(a). Notice that either $t_{e,1}$ or $t_{e,2}$ is true.

An uncertain database (called also x-relation) consists of a set of x-tuples that are subject of independent probabilistic events. Figure 1(b) shows an example of x-relation, named *Objects*. In the *Objects* relation, $t_{1,1}$ and $t_{1,2}$ together form one x-tuple, and $t_{2,1}$, by itself, form another x-tuple.

An uncertain database D can be interpreted as the set of all possible certain database instances, called possible worlds and denoted as $PW(D)$, each with a probability of occurrence. Figure 1(c) shows the possible worlds set of the uncertain database *Objects* and the probability of each member of this set.

We define the possible worlds set of an uncertain entity e and an uncertain database D, denoted by $PW(e, D)$, as follows:

$$PW(e, D) = \{w \mid w = \{t\} \times v, t \in e, v \in PW(D)\}.$$

As an example, Figure 2(b) shows all eight members of *PW(e, Objects)* together with their probability of occurrence.

tuple	Type	Distance	Brightness	Color	P
$\mathbf{t_{e,1}}$	Quasar	12.2	18.2	0.12	0.4
$\mathbf{t_{e,2}}$	Binary Star	11.5	14.8	0.4	0.5

(a)

	tuple	Type	Distance	Brightness	Color	P
$\mathbf{x_1}$	$\mathbf{t_{1,1}}$	Blazar	20.1	22.1	0.85	0.6
	$\mathbf{t_{1,2}}$	Quasar	20.4	18.4	0.64	0.4
$\mathbf{x_2}$	$\mathbf{t_{2,1}}$	Pulsar	80.12	19.64	0.67	0.4

(b)

PW(Objects)	P(w)
$\{t_{1,1}\}$	$0.6 \times (1 - 0.4) = 0.36$
$\{t_{1,2}\}$	$0.4 \times (1 - 0.4) = 0.24$
$\{t_{1,1}, t_{2,1}\}$	$0.6 \times 0.4 = 0.24$
$\{t_{1,2}, t_{2,1}\}$	$0.4 \times 0.4 = 0.16$

(c)

Figure 1: a) Example of uncertain entity e in the x-tuple model, b) example of x-relation *Objects*, c) the possible worlds of *Objects*

Entity Resolution in a Probabilistic Database

In this paper, we are interested in finding the *most probable match pair* (MPMP) defined as follows.

DEFINITION 1. MPMP*: Let D be an uncertain database, e an uncertain entity, and $PW(e, D)$ the set of possible worlds of e and D. Let $w \in PW(e, D)$ be a possible world and $P(w)$ be the probability that w occurs. Let the most similar entity-tuple pair of a world w, denoted as $msp(w)$, be the pair that has the highest similarity value among the pairs in w. Let ρ be an entity-tuple pair in $e \times D$. Let $P_{msp}(\rho, D)$*

Pair	Rank
$(t_{e,1}, t_{1,2})$	1
$(t_{e,1}, t_{1,1})$	2
$(t_{e,2}, t_{2,1})$	3
$(t_{e,2}, t_{1,2})$	4
$(t_{e,1}, t_{2,1})$	5
$(t_{e,2}, t_{1,1})$	6

(a)

w	PW(e, Objects)	MSP(w)	P(w)
w_1	$\{(t_{e,1}, t_{1,1})\}$	$(t_{e,1}, t_{1,1})$	$0.4 \times 0.36 = 0.144$
w_2	$\{(t_{e,1}, t_{1,2})\}$	$(t_{e,1}, t_{1,2})$	$0.4 \times 0.24 = 0.096$
w_3	$\{(t_{e,1}, t_{1,1}), (t_{e,1}, t_{2,1})\}$	$(t_{e,1}, t_{1,1})$	$0.4 \times 0.24 = 0.096$
w_4	$\{(t_{e,1}, t_{1,2}), (t_{e,1}, t_{2,1})\}$	$(t_{e,1}, t_{1,2})$	$0.4 \times 0.16 = 0.064$
w_5	$\{(t_{e,2}, t_{1,1})\}$	$(t_{e,2}, t_{1,1})$	$0.5 \times 0.36 = 0.18$
w_6	$\{(t_{e,2}, t_{1,2})\}$	$(t_{e,2}, t_{1,2})$	$0.5 \times 0.24 = 0.12$
w_7	$\{(t_{e,2}, t_{1,1}), (t_{e,2}, t_{2,1})\}$	$(t_{e,2}, t_{2,1})$	$0.5 \times 0.24 = 0.12$
w_8	$\{(t_{e,2}, t_{1,2}), (t_{e,2}, t_{2,1})\}$	$(t_{e,2}, t_{2,1})$	$0.5 \times 0.16 = 0.08$

(b)

Pair	P_{msp}
$\boldsymbol{(t_{e,1}, t_{1,1})}$	$0.144 + 0.096 = 0.24$
$(t_{e,2}, t_{2,1})$	$0.12 + 0.08 = 0.2$
$(t_{e,2}, t_{1,1})$	0.18
$(t_{e,1}, t_{1,2})$	$0.096 + 0.064 = 0.16$
$(t_{e,2}, t_{1,2})$	0.12
$(t_{e,1}, t_{2,1})$	0

(c)

Figure 2: a) Entity-tuple pairs ranked based on their similarity; b) possible worlds space of e and *Objects*, MSP in each world; c) all pairs and their P_{msp}.

be the aggregated probability that pair ρ is the most similar pair in $PW(e, D)$, i.e.

$$P_{msp}(\rho, D) = \sum_{w \in PW(e,D) \wedge \rho = MSP(w)} P(w)$$

Then, the most probable match pair of e and D, MPMP(e, D)*, defined as follows:*

$$MPMP(e, D) = \arg\max_{\rho \in e \times D} P_{msp}(\rho, D)$$

In other words, MPMP(e, D) *is the pair that has the highest probability to be the most similar.*

Given an uncertain database D and an uncertain entity e, the problem of ERPD consists of finding MPMP(e, D). At some points in this paper, we refer to the entity e, in the ERPD problem, as the entity resolution query or query in short.

Without loss of generality, we assume that all similarity scores between e's alternatives and the tuples of D are distinct.

The following example illustrates the MPMP concept.

Example 3. Consider the entity e and the *Objects* database in the Figures 1(a) and 1(b) respectively. Let F be a similarity function which ranks all possible entity-tuple pairs of e and *Objects* (i.e. set of pairs $e \times Objects$) as they appear in Figure 2(a). Figure 2(b) shows the eight possible worlds of $PW(e, Objects)$, the probability of each world, and the most similar pair (MSP) in each world. Figure 2(c) shows the P_{msp}) of all entity-tuple pairs. For instance, entity-tuple pair $(t_{e,1}, t_{1,2})$ is MSP in w_2 and w_4, thus, $P_{msp})(t_{e,1}, t_{1,2})$ is the sum of the probabilities of w_2 and w_4. We observe that entity-tuple pair $(t_{e,1}, t_{1,1})$ is MPMP since it has the maximum P_{msp}) among all entity-tuple pairs.

Entity Resolution for Distributed Probabilistic Data

In this paper, we consider that the uncertain database is fragmented over a number of nodes in a distributed system. We make no specific assumption about the topology of the distributed system architecture which can be very general, e.g. an unstructured P2P system or a cluster. In the distributed system, each node knows some other nodes, its neighbors, to communicate with.

Now we define the problem of entity resolution for distributed probabilistic data as follows. Let e be an uncertain entity issued at a query originator p. Let TTL (Time To Live) determine the maximum hop distance which the user wants the entity resolution message be sent. Let D be the union of the uncertain databases that are in the schema of e and maintained by nodes that can be accessed through TTL hops from the query originator. Our goal is to find the most-probable match-pair MPMP(e, D) while minimizing the communication cost.

3. DISTRIBUTED COMPUTATION OF MOST-PROBABLE MATCH-PAIR

One possible approach for computing MPMP is to move all relevant data of nodes to a central node, e.g. the query originator, where MPMP is computed using a centralized algorithm. However, the problem with this approach is that the query originator becomes a communication bottleneck since it must receive a large amount of data from other nodes. In addition, it becomes a processing bottleneck, as it must process a large amount of data. In this section, we propose a fully distributed algorithm called FD, for computing MPMP. Our algorithm avoids the problems of the above approach by : 1) pruning the data that have no chance to be MPMP, thus reducing the communication cost significantly; 2) distributing the processing of MPMP over a large number of nodes.

3.1 Algorithm Overview

The FD algorithm starts at the query originator, the node at which a user issues a query involving an uncertain entity e to be resolved. The query originator performs some initialization. First, it sets TTL with a value which is either specified by the user or default. Second, it gives e a unique identifier, denoted by *eid*, which is made of a unique node-ID and a query counter managed by the query originator. Nodes use *eid* to distinguish between new queries and those received before. After initialization, the entity resolution proceeds in the following phases done at each node that receives the query:

- **Query forward.** e is included in a message that is broadcast by the query originator to its reachable neighbors. Each node p that receives the message including e from node q performs the following steps. If it is the first time of receiving the query, then the node p saves the id of q as its parent, else discards the message and makes a new message including *eid* and sends it to q to indicate that the query has been received from another node. Then p decrements TTL by one, if $TTL > 0$, it makes a new message including e, *eid*, new TTL and the query originator's address; sends the message to all neighbors except q; and saves the number of sent messages to the neighbors.

- **Extract the essential-set.** The core idea of this phase is that for computing the most probable match pair, the query originator does not need all entity-tuple pairs maintained at p, but only a subset of them that we call *essential-set*. In this phase, p extracts the essential-set of its local data and saves it locally until receiving the essential-sets of its neighbors to which it has sent the query.

- **Merge-and-backward essential-sets.** In this phase, p unifies its essential-set with those received from its neighbors into a set of entity-tuple pairs $essential_{pq}$, and sends $essential_{pq}$ to its parent, the node from which it received the query.

- **MPMP computation and data retrieval.** During the first three phases of the algorithm, the query originator receives a number of merged essential-sets from its neighbors. It unifies these sets with its local essential-set into the set $essential_{unified}$, and computes MPMP(e, D) and asks the node which contains the data to return the data content.

In the next subsections, we describe the details of FD algorithm phases.

3.2 Extract the Essential-set

At each node p, our FD algorithm prunes the data that have no chance to be the (global) most probable match pair, i.e. MPMP(e, D). For this, FD needs to extract the essential-set of each node which we define as follows. Let e be the given entity. Suppose D_p is the database maintained by p and n_p is the number of tuples in D_p. Let S_p be the set of all entity-tuple pairs at p, i.e. $S_p = e \times D_p$. We define the *essential-set* of S_p, denoted as $essential(S_p)$ by using its complement: $essential^c(S_p)$ is a subset of S_p whose members can never be MPMP(e, D).

The alternatives of e are mutually exclusive, thus to find $essential(S_p)$, it is sufficient to compute the essential-set for each alternative $t \in e$, and then unify the essential-sets of all alternatives of e. More precisely, we have:

$$essential(S_p) = \bigcup_{t \in e} essential(S_{p,t}), \text{where } S_{p,t} = \{t\} \times D_p$$

Now, we consider an alternative $t \in e$, and explain the process of finding $essential(S_{p,t})$.

Let $L_p = \{(t, t_{p,1}), \ldots, (t, t_{p,n_p})\}$ be the list of $S_{p,t}$ pairs sorted in decreasing order of the similarities between t and D_p tuples. In other words, we have:

$$F(t, t_{p,1}) > \ldots > F(t, t_{p,n_p}),$$

where F is the given similarity function.

In the last step of the FD algorithm, we combine entity-tuple pairs from other nodes with the pairs in list L_p. Let $\rho = (t, t_q)$ be an entity-tuple pair from a node other than p. Pair ρ may be inserted in any index of L_p, say index $i \in [1, n_p + 1]$, based on the similarity between t and t_q. The question in pruning is whether this pair has any chance to be $MPMP(e, D)$ or not. The answer to this question depends on the value of $P_{msp}(\rho, D)$, i.e. the probability that the pair ρ is the most similar pair[2] (see Definition 1 in Section 2).

[2]Notice that $P_{msp}(\rho, D)$ is the global P_{msp} value of pair ρ, while $P_{msp}(\rho, D_p)$ is its local P_{msp} value at node p. Generally, $P_{msp}(\rho, D_p) \geq P_{msp}(\rho, D)$.

However, $P_{msp}(\rho, D)$ depends not only on the pairs that are at node p, but also on the pairs of other nodes. Thus, we cannot compute the exact value of $P_{msp}(\rho, D)$ locally, but we can compute an upper bound on this value. We denote such upper bound as the *Potential* of the index i of list L_p. More precisely,

$$Potential(i) = max P_{msp}(\rho, D) \quad (1)$$

where $i \in [1..n_p + 1]$.

For instance $Potential(1)$ is an upper bound on the (both global and local) P_{msp} value of a pair which is inserted in the first location of list L_p, i.e. before pair $(t, t_{p,1})$. The following lemma computes the Potential of index i of list L_p.

LEMMA 1. *Let i be an index in range $[1..n_p + 1]$. Let Y be the set of x-tuples formed by considering correlations between the tuples $\{t_{p,1}, \ldots, t_{p,i-1}\}$, then*

$$Potential\,(i) = P(t) \times \prod_{x \in Y} (1 - P(x)).$$

We include the proof of Lemma 1 in Appendix B.

COROLLARY 1. *Potential is a monotonically decreasing function.*

Intuitively, Corollary 1 says that the higher is the index, the lower is its potential.

Let $local\_max$ be the maximum local P_{msp} value of pairs in list L_p, i.e. $local\_max = max P_{msp}(\rho, D_p), \rho \in L_p$. We use $local\_max$ to define the *stop* index of list L_p as the smallest index in $[1..n_p + 1]$ where

$$Potential(stop) < local\_max. \quad (2)$$

The following lemma provides the basis for pruning the pairs in list L_p.

LEMMA 2. *Let stop be the stop index of list L_p. Then,*

$$\forall i \in [stop, n_p], L_p[i] \neq \arg\max_{\rho \in L_p} P_{msp}(\rho, D).$$

We include the proof of Lemma 2 in Appendix B.

COROLLARY 2. *Let stop be the stop index of list L_p. Then,*

$$\forall i \in [stop, n_p], L_p[i] \neq MPMP(e, D).$$

Intuitively, corollary 2 says that the pair at the stop index and any pair after it have no chance to be the most probable match pair. Thus, $essential(S_{p,t})$ is the set of L_p pairs whose index is smaller than the stop index.

Algorithm

Algorithm 1 describes the details of the steps which are performed for finding $essential(S_p)$. Steps 3-19 are repeated for every alternative of e, say t, and at each iteration compute $essential(S_{p,t})$. Step 3 computes set $S_{p,t}$ and step 4 sorts its pairs based on the similarity between the pair elements in descending order according to similarity function F, and stores the result in list L. Steps 5-7 compute list T as the second elements of list L and do some initialization. Steps 9-16 are repeated until finding the *stop* index of L, and in each iteration, they process the pair at index i of list L. Steps 10-12 compute $P_{msp}(L[i], D_p)$ as the intersection of

two independent probabilistic events: t occurs; and among tuples $T[1]$ to $T[i]$, only $T[i]$ occurs. To calculate the probability of the latter event, step 10 considers correlation among tuples to group tuples $T[1]$ to $T[i]$ into the set of x-tuples Y and step 11 removes the x-tuple containing $T[i]$ from Y and stores the result in x-tuple set X. Steps 13-15 update the current maximum P_{msp} of the pairs which we have processed so far. Steps 16 computes $Potential(i+1)$ using the x-tuple set Y which has already been computed in step 10. Step 17 checks if all pairs in the list L have been processed or $i+1$ is the *stop* index of L. If the condition holds, then the algorithm stops processing list L, else it continues by processing the next pair in L. Step 19 adds pairs $L[1]$ to $L[stop-1]$ to the *essential-set*. Finally, step 21 returns the *essential-set*.

Example

Let us illustrate the process of extracting essential-set using an example. Consider uncertain entity e and uncertain database D_p (maintained at node p) shown in Figures 3(a) and 3(b) respectively. In this example, e consists of a single alternative t, where $P(t) = 0.8$. The set of existing entity-tuple pairs in node p, i.e. set S_p, can be computed as $S_p = e \times D_p$. To prune S_p, we sort its pairs based on their similarity in descending order. The resulted list, denoted as L, is shown in Figure 3(c).

The *Potential* of the first location in list L, i.e. $Potential(1)$, is equal to the probability of the event that t occurs, which is equal to $P(t) = 0.8$. The P_{msp} of the first entity-tuple pair in L, i.e. $(t, t_{p,3})$, is equal to the probability of the event that t and $t_{p,3}$ occur, thus, $P_{msp}((t, t_{p,3}), D_p)$ is equal to $P(t) \times P(t_{p,3}) = 0.08$. The *Potential* of the second location in list L, i.e. $Potential(2)$, is equal to the probability of the event that t occurs but $t_{p,3}$ does not occur, which is equal to $P(t) \times (1 - P(t_{p,3})) = 0.72$. This means that the maximum possible value for P_{msp} of an entity-tuple pair which comes in $L[2]$ is 0.72. Since the *Potential* is greater than the current maximum value of P_{msp}, i.e. 0.08, we continue processing the list.

The P_{msp} of the second pair in L, i.e. $(t, t_{p,7})$, is equal to the probability of the event that t and $t_{p,7}$ occur but $t_{p,3}$ does not occur, which is equal to $P(t) \times P(t_{p,7}) \times (1 - P(t_{p,3})) = 0.504$. The *Potential* of the third location in list L, i.e. $Potential(3)$, is equal to the probability of the event that t occurs but neither $t_{p,3}$ nor $t_{p,7}$ occurs, which is equal to $P(t) \times (1 - P(t_{p,3})) \times (1 - P(t_{p,7})) = 0.216$. At this point, we stop processing the list since the *Potential* is less than the current maximum value of P_{msp}, i.e. 0.504. Therefore, the *stop* index of L_p is 3 and $essential(S_p)$ is qual to $\{(t, t_{p,3}), (t, t_{p,7})\}$. To provide better intuition, the P_{msp} and *Potential* values for other pairs are also shown in Figure 3(c).

t	P(t)
t	0.8

(a) e

t	P(t)
$t_{p,1}$	0.4
$t_{p,2}$	0.7
$t_{p,3}$	0.1
$t_{p,4}$	0.2
$t_{p,5}$	0.9
$t_{p,6}$	0.8
$t_{p,7}$	0.7
$t_{p,8}$	0.9

(b) D_p

i	L[i]	P_{msp}(L[i], D_p)	Potential(i)
1	$(t, t_{p,3})$	0.08	0.8
2	$(t, t_{p,7})$	0.504	0.72
3	$(t, t_{p,8})$	0.1944	0.216
4	$(t, t_{p,1})$	0.00864	0.0216
5	$(t, t_{p,2})$	0.00907	0.01296
6	$(t, t_{p,4})$	0.00078	0.00389
7	$(t, t_{p,5})$	0.00280	0.00311
8	$(t, t_{p,6})$	0.00025	0.00031
9	-	-	0.000062

(c) List L

Figure 3: An example of uncertain entity e, database D_p, and the pruning process

Algorithm 1 finding the *essential-set*

Input:
Entity e
Database D_p
Similarity function F
Output: $essential(S_p)$, where $S_p = e \times D_p$

```
1: essential ← ∅
2: for all t ∈ e do
3:     S_{p,t} ← {t} × D_p
4:     L ← Sort(S_{p,t}, F)
5:     T ← {t' | (t, t') ∈ L}
6:     local_max ← −1
7:     i ← 0
8:     repeat
9:         i ← i + 1
10:        Y ← set of x-tuples involved in {T[1], ..., T[i]}
           // removing the x-tuple containing T[i]
11:        X ← Y − {x | x ∈ Y ∧ T[i] ∈ x}
12:        P_msp ← P(t) × P(T[i]) × ∏_{x∈X}(1 − P(x))
13:        if P_msp > local_max then
14:            local_max ← P_msp
15:        end if
16:        Potential ← P(t) × ∏_{x∈Y}(1 − P(x))
17:    until (Potential < local_max) ∨ (i = |L|)
18:    stop ← i + 1
19:    essential ← essential ∪ {L[1], ..., L[stop − 1]}
20: end for
21: return essential
```

3.3 Merge-and-backward essential-sets

After extracting its essential-set, each node p waits for receiving the essential-sets of its children (the nodes to which p has sent the query). After receiving the essential-set of its children (or after a default wait time), p merges its essential-set with those received from its children, and extracts a new essential-set denoted by $essential_{pq}$, and sends it to its parent.

In order to minimize network traffic, nodes do not bubble up the data items of entity-tuple pairs (which could be large), but only some needed information about them. The information that is put in the sent essential-set for each entity-tuple pair $(t_i, t_j), t_i \in e, t_j \in D_q$, is a vector (i, a, j, x, s, p) where i is the index of t_i in e, a is the address of node q which owns t_j, j is the index of tuple t_j in the database D_q maintained by q, x is the x-tuple to which t_j belongs, s is the similarity score between t_i and t_j, and p is the probability of tuple t_j.

3.4 MPMP computation and data retrieval

Algorithm 2 computing MPMP(e, D)

Input: Set of entity-tuple pairs $essential_{unified}$
Output: MPMP(e, D)

1: $current\_max \leftarrow -1$
2: **for all** $t \in e$ **do**
3: $Potential \leftarrow P(t)$
4: $S_t \leftarrow \{(t, t') \mid (t, t') \in essential_{unified}\}$
5: $length \leftarrow |S_t|$
6: $i \leftarrow 1$
7: **while** $(Potential > current\_max) \wedge (i \leq length)$ **do**
8: $L \leftarrow$ Sort S_t pairs based on their similarity
9: $T \leftarrow \{t' \mid (t, t') \in L\}$
10: $Y \leftarrow$ set of x-tuples involved in $\{T[1], \ldots, T[i]\}$
11: $X \leftarrow Y - \{x \mid x \in Y \wedge T[i] \in x\}$
12: $P_{msp} \leftarrow P(t) \times P(T[i]) \times \prod_{x \in X}(1 - P(x))$
13: **if** $P_{msp} > current\_max$ **then**
14: $current\_max \leftarrow P_{msp}$
15: $MPMP \leftarrow L[i]$
16: **end if**
17: $Potential \leftarrow P(t) \times \prod_{x \in Y}(1 - P(x))$
18: $i \leftarrow i + 1$
19: **end while**
20: **end for**
21: return MPMP

When the query originator receives its children's essential-sets, it merges them with its local essential-set into the set $essential_{unified}$. Theorem 1 shows that $essential_{unified}$ contains all entity-tuple pairs which are needed for computing MPMP(e, D).

THEOREM 1. *The entity-tuple pairs in set $essential_{unified}$ are sufficient for computing MPMP(e, D).*

We include the proof of Theorem 1 in Appendix B. Algorithm 2 shows the detailed steps which the query originator performs to compute MPMP(e, D). Notice that:

- $current\_max$ does not represent the maximum value of P_{msp} of the pairs in one list,i.e. related to alternative $t \in e$, but the current maximum P_{msp} value of the pairs which we have visited so far. Thus, we reset it only once in the beginning of the algorithm.
- We use $Potential$ to stop early in visiting the pairs of list L. Notice that we may discard a list of pairs altogether because the maximum possible P_{msp} of the pairs in that list (i.e. $P(t)$, $t \in e$) is less than the current maximum P_{msp} value that we got so far.
- Since the set S_t consists of a number of sorted lists, the sort function in step 8 uses the sort-merge algorithm to merge these sorted lists.

Using Algorithm 2, the query originator computes MPMP(e, D) and asks the node which contains it to return the data content which is then returned to the user.

We provide an example of all phases of the FD algorithm in Appendix C.

4. ANALYSIS OF COMMUNICATION COST

In this section, we analyze the communication cost of FD, and as we will see, it is not very high. We measure the communication cost in terms of number of messages and number of bytes which should be transferred over the network in order to execute a query by our algorithm. The messages transferred can be classified as: (1) forward messages, for forwarding the query to nodes. (2) backward messages, for returning the essential-sets from nodes to the query originator. (3) retrieve message, to request and retrieve the MPMP. Let us first formalize the distributed system model that we use in our analysis.

4.1 Distributed System Model

Let P be the set of the nodes in the distributed system. Let Q be an entity resolution query at the query originator p, i.e. the node at which the query is issued. Let $P_Q \subseteq P$ be a set containing the query originator and all nodes that receive Q. We model the nodes in P_Q and the links between them by a graph $G(P_Q, E)$ where P_Q is the set of vertices in G and E is the set of the edges. There is an edge $p - q$ in E if and only if there is a link between the nodes p and q in the distributed system. Two nodes are called neighbor, if and only if there is an edge between them in G. The number of neighbors of each node $p \in P_Q$ is called the degree of p and is denoted by $d(p)$.

A peer $p \in P_Q$ may receive Q from some of its neighbors. The first node, say q, from which p receives Q, is the parent of p in G, so p is a child of q. A node may have some neighbors that are neither its parent nor its children.

4.2 Forward messages

Forward messages are the messages that we use to forward Q to the nodes. According to the basic design of our algorithm, each node in P_Q sends Q to all its neighbors except its parent. Let p_o denote the query originator. Let $G(P_Q, E)$ be a graph representing the distributed network, such that P_Q is the set of nodes and E is the links between the nodes. By our FD algorithm, each node $p \in \{P_Q - \{p_o\}\}$, sends Q to $d(p) - 1$ nodes, where $d(p)$ is the degree of p in G. The query originator sends Q to all of its neighbors, in other words to $d(p_o)$ nodes. Then, the sum of all forward messages m_{fw} can be computed as

$$m_{fw} = d(p_o) + \sum_{p \in \{P_Q - \{p_o\}\}} (d(p) - 1)$$

We can write m_{fw} as follows:

$$m_{fw} = \left(\sum_{p \in \{P_Q\}} (d(p) - 1) \right) + 1 = \left(\sum_{p \in \{P_Q\}} d(p) \right) - |P_Q| + 1 \quad (3)$$

We use the average degree of the graph G, denoted as $d(G)$, to simplify (3). $d(G)$ is defined as the average degree of nodes in G and can be computed as

$$d(G) = \frac{\sum_{p \in P_Q} d(p)}{|P_Q|}$$

Substituting $d(G)$ in (3), we have

$$m_{fw} = (d(G) - 1) \times |P_Q| + 1$$

From the above discussion, we can derive the following Lemma.

LEMMA 3. *The number of forward messages in the FD algorithm is* $(d(G)-1)\times|P_Q|+1$.

PROOF. Implied by the above discussion. □

In our underlying applications, e.g. astronomy application, the average degree of nodes is low, that is each node is usually connected to a small number of other nodes. Thus, the total number of forward messages is not very high compared to the number of nodes. For example, if the average degree of the system is 4, i.e. $d(G)=4$, then we have $m_{fw}=3\times|P_Q|+1$.

Let b_t be the average size of a tuple in Q in bytes, and $|Q|$ be the number of alternative tuples of Q. Then, the total size of data transferred by forward messages, denoted by b_{fw}, can be computed as $((d(G)-1)\times|P_Q|+1)\times|Q|\times b_t$.

4.3 Backward messages

In the Merge-and-Backward phase, each node in P_Q, except the query originator, sends its merged essential-set to its parent. Therefore, the number of backward messages, denoted by m_{bw}, is $m_{bw}=|P_Q|-1$.

In the query forward phase of the algorithm, nodes in P_Q are arranged in a tree, called *query-tree*, with the query originator as its root. For our modeling, we assume that the query-tree is a k-ary tree (i.e. $k=d(G)$) in which the root has k children and all intermediate nodes has exactly $k-1$ children. Moreover, we assume that all leaves are at the same level. These assumptions, however, are mostly for illustration purposes. In practice, nodes are organized in arbitrary tree topologies.

Let h be the height of the tree, with the root at level $l=0$. The total number of nodes, i.e. $|P_Q|$, can be computed as $|P_Q|=(\sum_{l=2}^{h}(k-1)^l)+k+1$.

Let $S(l)$ be the total size of data transferred in the Merge-and-Backward phase by each node which resides in the level l of the query-tree. Let b_{es} be the average size of the essential-set of a node. In the Merge-and-Backward phase, each node at level h of the query-tree, i.e. leaf nodes, sends its essential-set to its parent. Thus, $S(h)=b_{es}$. Also, each intermediate node at level l, $l\neq 0$, of the query-tree receives exactly $k-1$ essential-sets from its children which reside at level $l+1$; merges them with its essential-set; and send the merged essential-set to its parent. Thus, for each intermediate node we have $S(l)=(k-1)\times S(l+1)+b_{es}$; thereby yielding the following recurrence relation for $S(l)$:

$$S(l)=\begin{cases}(k-1)\times S(l+1)+b_{es} & \text{for } 0<l<h\\ b_{es} & \text{for } l=h\end{cases}$$

By solving this recurrence relation, we have

$$S(l)=\frac{1-(k-1)^{h-l+1}}{1-(k-1)} \tag{4}$$

Since there are exactly $k\times(k-1)^{l-1}$ nodes at level l, $0<l\leq h$, thus the total data transfer of the Merge-and-Backward phase, denoted as b_{bw}, can be computed as:

$$b_{bw}=\sum_{l=1}^{h}(k\times(k-1)^{l-1}\times S(l))$$

By substituting $k=d(G)$ and $S(l)$ from (4) into the above equation, b_{bw} can be written as

$$b_{bw}=\frac{b_{es}\times d(G)\times\left(1+(h\times(d(G)-2)-1)\times(d(G)-1)^h\right)}{(2-d(G))^2}$$

Let b_{pa} be the size of an entity-tuple pair in bytes, and η be the average number of entity-tuple pairs of the essential-set which have the same alternative of Q as their first element. Then, b_{es}, i.e. the average size of the essential-set in each node, can be computed as $b_{es}=|Q|\times\eta\times b_{pa}$.

In Section 5, we show that η is very small and almost independent from the number of tuples which are maintained at a node. However, η is dependent to the correlation between the probability of the tuples and their similarity to Q's alternatives.

Let us show with an example that b_{bw} is not significant. Consider that 10,000 nodes receive Q (including the query originator), thus $|P_Q|=10,000$. Assume that $d(G)=4$. Thus, the height of the query-tree, i.e. h, is equal to 8. Our experiments show that η is about 2.2 when similarity and probability are not correlated. Consider Q has two alternative tuples. Since the actual data contents of the entity-tuple pair (t_i,t_j) is not transferred during the Merge-and-Backward phase, we set b_{pa} to 23, i.e. 1 bytes for i, 4 bytes for j, 6 bytes for the address of the node in which t_j is maintained, 4 bytes for the x-tuple to which t_j belongs, 4 bytes for the similarity score of t_i to t_j, and 4 bytes for the probability of t_j. As a result, b_{bw} is less than 10 megabytes for a distributed system that contains 10,000 nodes.

4.4 Retrieve message

By retrieve messages, we mean the message sent by the query originator to request the MPMP and the message sent by the node owning the MPMP to return it. Therefore, the number of retrieve messages, denoted by m_{rt}, is $m_{rt}=2$. The total size of data transferred by these messages, denoted by b_{rt}, can be computed as $b_{rt}=m_{rt}\times b_t$, where b_t is the average size of a tuple.

5. PERFORMANCE EVALUATION

We evaluated the performance of FD (Distributed Algorithm) through implementation and simulation. The implementation over a 75-node cluster was useful to validate our algorithm in a realistic experimental environment. The simulation allowed us to study the performance of our algorithm under various conditions.

The rest of this section is organized as follows. In section 5.1, we describe our experimental and simulation setup, and the algorithms used for comparison. In section 5.2, we evaluate the response time of FD. Section 5.3 presents the evaluation of communication cost based on the bandwidth usage and the number of exchanged messages among nodes.

5.1 Experimental and simulation setup

In our implementation and simulation, we compare FD with two baseline algorithms. The first algorithm is a centralized algorithm which we call FC (Fully Centralized). With FC, all nodes that receive the query send thier data to the query originator where the most probable most similar pair is computed using a centralized algorithm. The details of the centralized processing by FC can be found in Appendix A. The second comparing algorithm is denoted

Table 1: Parameters

Parameter	Values
data: tuple's data items size	Normally distributed random, Mean = 1 KB, Variance = 16 KB
sim: similarity score	Normally distributed random, Mean = 0.5, Variance = 0.04
p: probability	Normally distributed random, Mean = 0.4, Variance = 0.04
N: number of tuples maintained at each node	Uniformly distributed random integer in range [4500..5500]
d_x: x-tuple's maximum number of alternatives	3 for R table and 2 for the query
Cor: correlation between *sim* and p	0
Upstream bandwidth	Normally distributed random, Mean = 56 Kbps, Variance = 32 Kbps
Downstream bandwidth	8 × Upstream bandwidth
Latency	Normally distributed random, Mean = 200 ms, Variance = 100 ms
Number of nodes	10,000
TTL: Time To Live	100

as SCC (Score Confidence Centralized), in which every node receiving Q extracts a list containing the score-probability pair for each tuple in its database and for each alternative tuple in the Q; then sends the extracted list directly to the query originator for centralized processing.

We implemented FD, FC, and SCC in Java, and tested them using a cluster of 75 nodes connected by a 1-Gbps network. Each node of cluster has a dual-quad-core 2.4 GHz processor and 24 GB memory. We make each node act as a node in the distributed system described in Section 4.1. We determined the node neighbors using the topologies generated by the BRITE universal topology generator [1]. Thus, each node only is allowed to communicate with the nodes that are its neighbors in the topology generated by BRITE.

To study the scalability of FD far beyond 75 nodes and to play with various performance parameters, we implemented a simulator using the PeerSim simulation kernel [2] and the Java programming language. We use the event driven engine of PeerSim to be able to simulate the delay in sending messages and also the bandwidth of nodes. We assign a random delay, denoted as latency, to communication ports to simulate the delay for sending a message between two nodes in a real distributed system. Also, we assign an upstream and a downstream bandwidth to each node. To simulate a node, we use a PeerSim's node that performs all tasks that must be done by a node for executing FD, FC, and SCC algorithms. We implemented each of the three algorithms as a protocol in PeerSim. We used PeerSim's WireKOut topology generator that randomly selects k neighbors for each node in the network. We used undirected links between nodes and set k to 10.

The experimental and simulation parameters are listed in Table 1. Notice that *bandwidth* and *latency* parameters are used only in our simulation. Unless otherwise specified, we use the values in this table for our tests. Each node has a table $R(data, sim, p)$ in which attribute *data* is a random real number with normal distribution with a mean of 1 KB (Kilobytes) and a variance of 16 KB, *sim* is a random real number in the interval [0..1] with normal distribution with a mean of 0.5 and a variance of 0.04, p is a random real number in the interval (0..1] with normal distribution with a mean of 0.4 and a variance of 0.04. Attribute *data* represents the data item that is returned back to the user as the result of the query and its value simulates the size of the data item. Attribute *sim* is used for computing the similarity between the tuple and the tuples in the query, and attribute p is the confidence value of the tuple. We introduce a number of parameters to control the characteristics of the R table. The number of tuples in R is denoted as N. The maximum number of alternatives that an x-tuple can have is denoted as d_x. The correlation between *sim* and p is denoted as *Cor*. To generate each tuple in R, we use a normal distribution for generating attribute *data*, and a bivariate normal distribution with a given correlation for generating *sim* and p attributes. We repeat this process to generate N different tuples. Then, we generate d as a uniform random number in $[1, d_x]$, and repeatedly pick d tuples at random and group them into an x-tuple; if their confidence values add up to more than 1, we relinquish them and take another set of tuples until we form a valid x-tuple. We repeat this process until we group all tuples in valid x-tuples. We generate the query needed for experiments, in the same way that we generated a valid x-tuple. As Table 1 shows, we use the following default values for N, d_x, and *Cor* unless otherwise specified. N is a random number, uniformly distributed over all nodes, which is greater than 4500 and less than 5500. We use the default value $d_x = 3$ for the database and $d_x = 2$ for the query, and we use the default value $Cor = 0$.

For our implementation, we generate the R table and the query in the same way as our simulation, except for the data item size which is no longer simulated by a real number but with an array containg data.

Unless otherwise specified, we use the following values for the other simulation parameters. The *upstream bandwidth* of nodes is a random number with normal distribution with a mean of 56 Kbps (Kilobits per second) and a variance of 32 Kbps. The *downstream bandwidth* of each node is set to a value equal to 8 times of its *upstream bandwidth*. The *latency* for sending messages between any two nodes is also a random number with normal distribution with a mean of 200 milliseconds and a variance of 100 milliseconds.

Running the simulator on a machine with 16 GB of memory, allows us to perform tests up to 10,000 nodes, after which the simulation data no longer fit in RAM and makes our tests difficult. This is quite sufficient for our tests. Therefore, the number of nodes of the system is set to be 10,000, unless otherwise specified.

In all of our tests, we set TTL to a high value, i.e. 100, to be sure that all nodes receive the query although the maximum hop-distance to other nodes from the query originator is much less than 100 with the topology that we use for our distributed system.

We repeat each simulation 10 times with the same query but with a different random number seed and average the

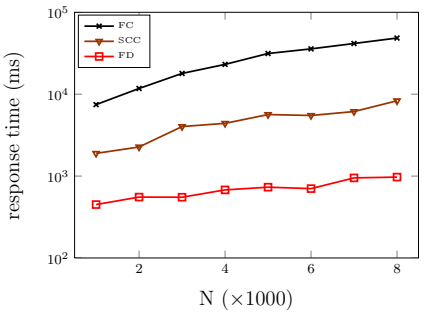


Figure 4: Response time vs. number of tuples (on cluster)

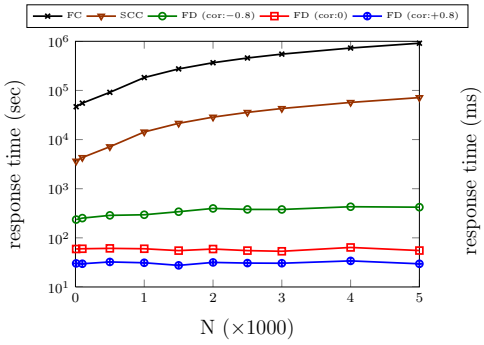


Figure 5: Response time vs. number of tuples

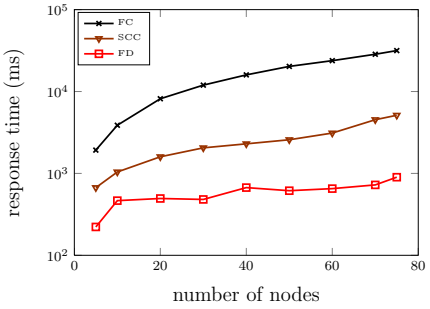


Figure 6: Response time vs. number of nodes (on cluster)

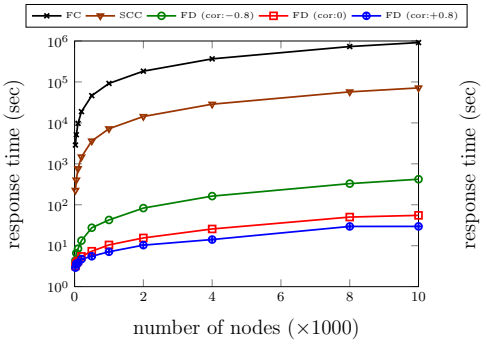


Figure 7: Response time vs. number of nodes

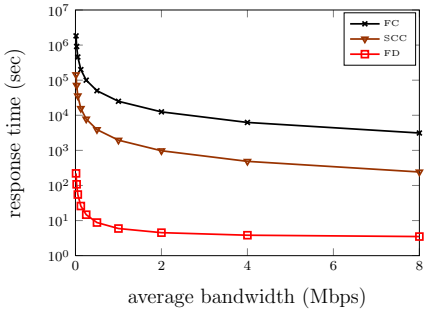


Figure 8: Effect of bandwidth on response time

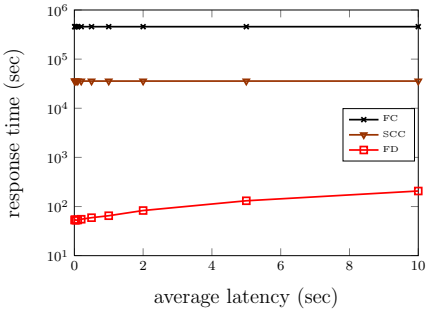


Figure 9: Effect of latency on response time

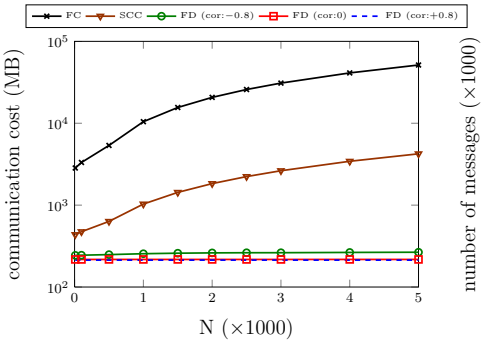


Figure 10: Effect of number of tuples on communication cost

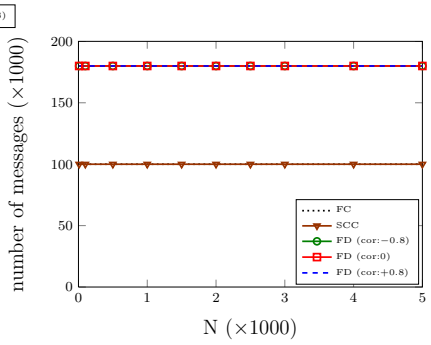


Figure 11: Number of exchanged messages vs. number of tuples

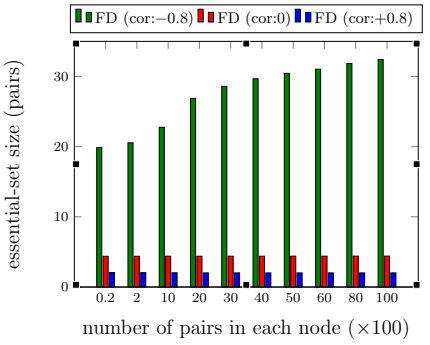


Figure 12: essential-set size vs. number of pairs in each node

outcomes.

5.2 Response time

5.2.1 Scale up

In this section we study the response time of distributed entity resolution by varying the number of tuples, i.e. N, and the number of nodes. The response time is the time elapsed from submitting the query to a node to sending the result of the query to the user. The response time includes local processing time and data transfer time. To study the effect of different correlations between similarity and confidence values, we ran experiments using three different correlations, i.e. negative, zero, and positive correlations.

We used our implementation over the cluster to study how the response time increases with increasing the number of tuples in each node. Figure 4 shows the response times of FD, FC, and SCC with N increasing up to 5,000. Using simulation, Figure 5 shows the response times of the three algorithms with N increasing up to 5,000 and the other simulation parameters set as in Table 1.

While FD significantly outperforms the other two algorithms, its response time is affected only very little with increasing N. As we expected, the negative correlation between similarity and confidence increases the response time

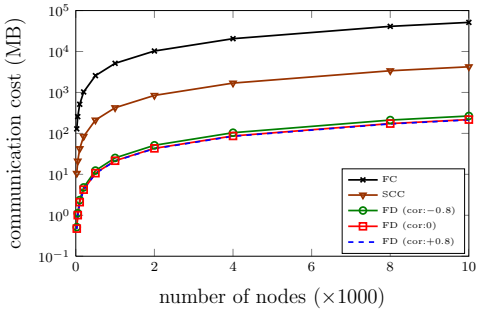


Figure 13: Effect of the number of nodes on the communication cost

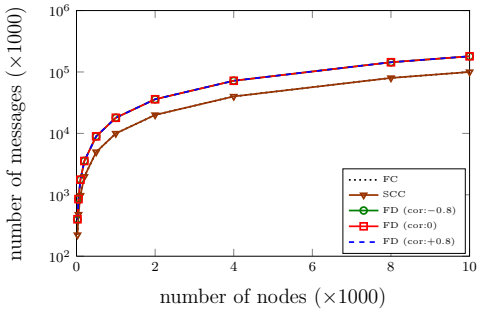


Figure 14: Effect of number of nodes on the number of exchanged messages

since it increases the number of score-confidence pairs at each node, and this increases the response time. Using independent random variables for similarity and confidence, i.e. zero correlation, decreases the response time and the positive correlation even decreases it more. Different correlations between similarity and confidence do not have any impact on FC and SCC algorithms, since they always send the whole database or the extracted similarity-confidence pairs of the whole database respectively.

We also used our implementation over the cluster to study the effect of the number of nodes on response time. Figure 4 shows the response times of FD, FC, and SCC with the number of nodes increasing up to 75 and the other experimental parameters set as in Table 1. Using simulation, Figure 7 shows the response times of the three algorithms with the number of nodes increasing up to 10,000 and the other simulation parameters set as in Table 1. FD always significantly outperforms the other two algorithms and the performance difference increases significantly in the favor of FD as the number of nodes increases. These figures show excellent scale up of FD since response time logarithmically increases with increasing the number of nodes. We also observe that negative correlation between similarity and confidence increases the response time, but zero and positive correlations decrease the response time. Also the performance difference between different correlations increases as the number of nodes increases.

The experimental results correspond with the simulation results. However, the response time of implementation over the cluster is better than that of simulation because the cluster has a high-speed network.

To sum up, the reason of excellent scalability of FD versus both the database size and the number of nodes is its distributed execution. In FC and SCC algorithms, a central node, i.e. the query originator, is responsible for query execution, and this makes them inefficient.

5.2.2 *Effect of latency and bandwidth*

In this section, we study the effect of latency and bandwidth on response time. In the previous simulation tests the latency and upstream bandwidth were normally distributed random numbers with mean values of 200 ms and 56 Kbps respectively. In this test, we vary the mean values of the latency and bandwidth and study their effects on response time. For both experiments on bandwidth and latency, we set both N and the number of nodes to 5,000 and other simulation parameters set as in Table 1.

Figure 8 shows how response time decreases with increasing bandwidth. Increasing the bandwidth has strong, similar effect on all three algorithms. FD outperforms the other two algorithms for all tested bandwidths.

Figure 9 shows how response time evolves with increasing latency. Latency has little effect on the FC and SCC algorithms, because in these algorithms the nodes return their results directly to the query originator, and do not bubble up the results. Although FD outperforms the other algorithms for all the tested values, high latency, e.g. more than 500 ms, has strong impact on it and increases its response time much. However, below 500 ms, latency does not have much effect on FD's response time.

5.3 Communication cost

In this section, we study the communication cost of FD. We measure the communication cost in terms of the number of bytes, which should be transferred on the network for processing a query Q. We also measure the number of exchanged messages during the execution of an algorithms. To study the effect of different similarity-confidence correlations, we ran experiments using three different correlations, i.e. negative, zero, and positive correlations.

Figure 10 shows how communication cost evolves with the number of tuples in each node increasing up to 5,000 and the other simulation parameters set as in Table 1. This figure shows that FD significantly outperforms the other two algorithms. Moreover, while increasing N has strong effect on FC and SCC algorithms, it has a very little effect on FD. Also as in scalability experiments, negative correlation between similarity and confidence values increases the communication cost, and zero or positive correlation decreases it.

Figure 11 shows the number of messages exchanged during the execution of the three algorithms with the number of tuples in each node increasing up to 5,000 and the other simulation parameters set as in Table 1. This figure shows that the database size has no effect on the number of exchanged messages. Although FD exchanges more messages than the other two algorithms, since the sizes of these messages are much smaller than the sizes of the messages produced by the other two algorithms, FD's communication cost is significantly smaller than theirs.

Figure 11 also shows that different similarity-confidence correlations has no effect on the number of exchanged messages in FD.

We ran experiments to compare the average size of the

essential-set with the number of entity-tuple pairs which exist at a node. To measure the average essential-set size, we calculated the sum of the essential-set of all nodes and divided it by the number of nodes. In these experiments, we used uncertain entities with exactly 2 alternatives for the query. Figure 12 shows how the average size of the essential-set (in number of entity-tuple pairs) changes with the number of entity-tuple pairs in each node (i.e. $2 \times N$) increasing up to 10,000 and the other simulation parameters set as in Table 1. This Figure shows that the correlation between similarity and confidence has a strong effect on the size of the essential-set. The average size of the essential-set is almost constant for positive and zero correlations, i.e. 2 and 4.4 pairs respectively, but the essential-set size increases from 19.8 to 32.4 pairs for the negative correlation. These observations indicate that the size of the essential-set is very small and almost independent from the number of entity-tuple pairs which exist at the nodes. This means that our pruning algorithm performs quite effectively.

We also ran experiments to study the effect of the number of nodes on communication cost. Figure 13 shows the communication costs of the three algorithms with the number of nodes increasing up to 10,000 and the other simulation parameters set as in Table 1. As this figure shows, FD significantly outperforms the other two algorithms and the performance difference increases significantly in the favor of FD as the number of nodes increases. Again as we expect, negative correlation between similarity and confidence increases the communication cost, but zero and positive correlations decrease the communication cost.

Figure 14 shows the number of messages exchanged during the execution of the three algorithms with the number of nodes increasing up to 10,000 and the other simulation parameters set as in Table 1. This figure shows that increasing the number of nodes increases the number of messages in the three algorithms. The number of exchanged messages in FD is higher than the other two algorithms but, as we discussed earlier because of the small size of these messages, FD significantly outperforms the other algorithms based on communication cost. Again as we expect, different similarity-confidence correlations has no effect on the number of exchanged messages in FD.

5.4 Case study on real data

In this section, we report the result of applying the three algorithms on real data. As real-world database, we used a facial image database which we extracted from video. We downloaded 900 videos tagged with the keyword *"wedding ceremony"* from YouTube[3], and used 2 fps sampling method and the pittpatt software [3] to extract 5010 distinct facial images each associated with a confidence value, from the videos. Then, we used the *bag of words* model [17] with a codebook of 250 visual words to represent each facial image with a vector containing 500 real numbers in range [0..1] each associated with a confidence value. We randomly selected one of the vectors as the query, and randomly selected 250 vectors among other vectors for each of the 20 nodes in the network. We also used the cosine similarity metric for measuring the similarity between vectors. The other parameters set as in Table 1. The result of applying the three algorithms on these real data is summarized in Table 2. As we expected, the result of applying the algorithms on real data confirms the result we observe on synthetic data.

[3]http://www.youtube.com

Table 2: Result on real data

	FC	SCC	FD
Response time (sec)	457.2	7.2	3.4
Communication cost (MB)	22.03	0.75	0.47
Number of messages	174	174	313
Average essential-set size (pairs)	-	-	2.42

6. RELATED WORK

In the literature, considerable attention has been devoted to the problem of entity resolution for *certain* data (refer to [10] for a survey). The problem has been presented under various terms such as entity resolution, duplicate detection, etc. Recently, there have been some proposals dealing with entity resolution over probabilistic data (ERPD) [19, 20, 7]. The proposals in [19, 20] consider the duplicate detection of a relation and hence generate a partitioning of the relation's tuples. The proposal in [7] adopts the possible worlds semantics for uncertain data for defining the semantics for the ERPD problem and proposes an algorithm for computing it. In this paper, we use the semantics presented in [7] for the ERPD problem. However, our work differs from that of [19, 20, 7], in that they deal with the ERPD problem in centralized systems but ours deals with the problem in distributed systems. To the best of our knowledge, our work is the first proposal that deals with the problem of ERPD for distributed data.

Among the prior work in database literature, top-k, and nearest neighbor query processing over distributed uncertain data are relevant to ours. While there are a number of proposals, e.g. [15, 26, 29], that deal with nearest neighbor queries over uncertain data, to the best of our knowledge, we are not aware of any proposal that deals with this problem over distributed uncertain data. Recently, there have been some proposals dealing with the problem of top-k query processing for distributed uncertain data [27, 16]. In [27], the authors present a top-k query processing system for a wireless sensor network in which sensor nodes are grouped into clusters, where cluster heads are selected to perform localized data processing and to report aggregated results to the base station. Cluster heads use a user-specified probability threshold to find a rank boundary for pruning data gathered from sensors before reporting to the base station. In [16], the authors present a proposal for ranking queries for distributed uncertain data. They use the concept of expected score and approximate it to reduce the communication cost and also processing time. Our work differs from these proposals because our problem definition is completely different. We look for an entity-tuple pair with the maximum probability of being the most similar pair, while [27] looks for tuples which have a probability higher than a user-specified threshold to be in the query result, and [16] is a proposal for approximating the expected score of the query results and ranking them.

7. CONCLUSION

In this paper, we proposed FD, a fully distributed algorithm for dealing with the entity resolution problem over distributed probabilistic data, with the objective of minimizing network traffic. FD uses the novel concepts of *potential* and *essential-set* to prune data at local nodes. This leads to a significant reduction in bandwidth usage and response time compared to the baseline approaches. FD requires no global information, and does not depend on the existence of certain nodes.

We validated the performance of FD through simulation using a simulator which we implemented using the PeerSim simulation kernel and the Java programming language. The simulation results show that response time of FD increases logarithmically with increasing the number of nodes. The simulations also show that FD's response time is almost independent from the size of the database in nodes. The results also show the excellent performance of FD, in terms of communication cost, compared with two baseline algorithms.

8. REFERENCES

[1] BRITE, http://www.cs.bu.edu/brite/.

[2] PeerSim, http://peersim.sourceforge.net/.

[3] Pittsburgh Pattern Recognition, http://www.pittpatt.com/.

[4] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB J.*, 18(5), 2009.

[5] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *Proc. of VLDB*, 2006.

[6] M. J. Atallah and Y. Qi. Computing all skyline probabilities for uncertain data. In *Proc. of PODS*, 2009.

[7] N. Ayat, R. Akbarinia, H. Afsarmanesh, and P. Valduriez. Entity resolution for uncertain data. In *BDA*, 2012.

[8] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *Proc. of VLDB*, 2006.

[9] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of VLDB*, 2004.

[10] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1), 2007.

[11] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proc. of SIGMOD Conference*, 2008.

[12] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1), 2006.

[13] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. *PVLDB*, 1(1), 2008.

[14] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query evaluation over probabilistic XML. *VLDB J.*, 18(5), 2009.

[15] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *Proc. of DASFAA*, 2007.

[16] F. Li, K. Yi, and J. Jestes. Ranking distributed probabilistic data. In *Proc. of SIGMOD*, 2009.

[17] F.-F. Li and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Proc. of CVPR*, 2005.

[18] M. Magnani and D. Montesi. Uncertainty in data integration: current approaches and open problems. In *Proc. of MUD*, 2007.

[19] D. Menestrina, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with data confidences. In *Proc. of CleanDB*, 2006.

[20] F. Panse, M. van Keulen, A. de Keijzer, and N. Ritter. Duplicate detection in probabilistic data. In *Proc. of ICDE Workshops*, 2010.

[21] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proc. of VLDB*, 2007.

[22] L. Peng, Y. Diao, and A. Liu. Optimizing probabilistic query processing on continuous uncertain data. *PVLDB*, 4(11), 2011.

[23] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *Proc. of ICDE*, 2006.

[24] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE*, 2007.

[25] J. Talburt. *Entity resolution and information quality.* Morgan Kaufmann Pub, 2010.

[26] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *Proc. of EDBT*, 2009.

[27] M. Ye, X. Liu, W.-C. Lee, and D. L. Lee. Probabilistic top-k query processing in distributed sensor networks. In *Proc. of ICDE*, 2010.

[28] M. L. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis. Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. *TKDE*, 21(1), 2009.

[29] S. M. Yuen, Y. Tao, X. Xiao, J. Pei, and D. Zhang. Superseding nearest neighbor search on uncertain spatial databases. *TKDE*, 22(7), 2010.

[30] S. K. Zhou, V. Krüger, and R. Chellappa. Probabilistic recognition of human faces from video. *CVIU*, 91(1-2), 2003.

APPENDIX

A. FC ALGORITHM

In this section, we present a basic algorithm, which we call FC (Fully-Centralized), for computing MPMP.

The idea behind FC is to move all relevant data of nodes to a central node, e.g. the query originator, where MPMP is computed using a centralized algorithm. Let D be the database containing the database of the query originator and all databases which it has received from other nodes. Then, the query originator can compute MPMP(e, D) as follows. Let S be the set of all entity-tuple pairs at the query originator, i.e. $S = e \times D$. Let $\rho = (t, t_i)$ be an entity-tuple pair in S. Since alternative tuples of e are mutually

exclusive, there is no possible world in $PW(e, D)$ containing pair ρ together with pair $\rho' = (t', t_j)$, where $\rho' \in S$, and $t \neq t'$. Thus, to compute $P_{msp}(\rho, D)$, we just have to consider the subset of entity-tuple pairs in S which have t as their first elements, say set $S_t \subseteq S$, where $S_t = \{t\} \times D$. We use this fact to compute $P_{msp}(\rho)$.

Let $L = \{(t, t_1), \ldots, (t, t_n)\}$ be the list of S_t pairs sorted based on the similarity between pair elements in descending order. Let $\rho = (t, t_i)$ be the entity-tuple pair which lies in the ith index of list L. We can calculate $P_{msp}(\rho, D)$ as the intersection of two independent events: t occurs; and among tuples t_1 to t_i, only t_i occurs. Considering tuple correlations in calculating the probability of the latter event, we can calculate $P_{msp}(\rho, D)$ as

$$P_{msp}(\rho, D) = P(t) \times P(t_i) \times \prod_{x \in X} (1 - P(x)) \quad (5)$$

where P is the occurrence probability of a tuple or x-tuple, and X is the set of x-tuples formed by considering correlations between the tuples t_1 to t_i while the x-tuple containing t_i is omitted from it. Using (5), the centralized algorithm computes P_{msp} of all entity-tuple pairs in set S and returns the pair with maximum such probability.

B. PROOFS

In this section, we provide the proofs of the lemmas and the theorem presented in Section 3.

B.1 Proof of Lemma 1

Let $S_t = t \times D$ and $L = \{(t, t_1), \ldots, (t, t_n)\}$ be the list of S_t pairs sorted based on their similarity in descending order. Let $\rho = (t, t_q)$ resides in the index j of list L, i.e. $L[j] = \rho$. Using equation (5), we have

$$P_{msp}(\rho, D) = P(t_q) \times P(t) \times \prod_{x \in X} (1 - P(x)) \quad (6)$$

where X is the set of x-tuples formed by considering correlations between the tuples t_1 to t_j while the x-tuple containing t_q is omitted from it. It is clear that the value of $P(t_q)$ which maximizes RHS(6) is equal to one. The set of tuples t_1 to t_{j-1} can be partitioned into two sets T_1 and T_2, where $T_1 = \{t_{p,1}, \ldots, t_{p,i-1}\}$ is a subset of D_p and T_2 is a subset of $D - D_p$. Let X_1 and X_2 be the set of x-tuples formed by considering correlations between the tuples in T_1 and T_2, respectively. Since all members of an x-tuple reside within the same node, x-tuple set X in RHS(6) can be partitioned into two disjoint sets $X = X_1$ and X_2, and, setting $P(t_q)$ to one, equation (6) can be rewritten as

$$P_{msp}(\rho, D) = P(t) \times \prod_{x \in X_1} (1 - P(x)) \times \prod_{x \in X_2} (1 - P(x)) \quad (7)$$

Notice that since we set $P(t_q)$ to one, no x-tuple can contain it. Set X_1 is fixed, but we can make any assumtion about set X_2 to maximize RHS(7). Each x-tuple x in set X_2 reduces RHS(7) by the factor of $1-P(x)$, thus, RHS(7) is maximized when $X_2 = \emptyset$. In such case, RHS(7) is equal to the asserted value in the lemma.

B.2 Proof of Lemma 2

Let j be the index of a pair in list L_p with maximum local P_{msp} value, i.e. $P_{msp}(L_p[j], D_p) = local\_max$. Let i be an index in list L_p, where $i \in [stop, n_p]$. Let $S_t = \{t\} \times D$ and $L = \{(t, t_1), \ldots, (t, t_n)\}$ be the list of S_t pairs sorted based on their similarity in descending order. Let j' and i' respectively be the index of pairs $L_p[j]$ and $L_p[i]$ in list L, i.e. $L[j'] = L_p[j]$ and $L[i'] = L_p[i]$. To prove the lemma, we show that

$$P_{msp}(L[i'], D) < P_{msp}(L[j'], D) \quad (8)$$

We have

$$P_{msp}(L[i'], D) = P(t) \times P(t_{i'}) \times \prod_{x \in X_{i'}} (1 - P(x)) \quad (9)$$

where $X_{i'}$ is the set of x-tuples formed by considering correlations between the tuples t_1 to $t_{i'}$ while the x-tuple containing $t_{i'}$ is omitted from it. The set of tuples t_1 to $t_{i'}$ can be partitioned into two sets $T_{i',1}$ and $T_{i',2}$, where $T_{i',1} = \{t_{p,1}, \ldots, t_{p,i-1}\}$ is a subset of D_p and $T_{i',2}$ is a subset of $D - D_p$. Let $X_{i',1}$ and $X_{i',2}$ respectively be the set of x-tuples formed by considering correlations between the tuples in T_1 and T_2, while the x-tuple containing $t_{i'}$ is omitted from $X_{i',1}$. Since all members of an x-tuple reside within the same node, x-tuple set $X_{i'}$ in RHS(9) can be partitioned into two disjoint sets $X_{i',1}$ and $X_{i',2}$, and equation (9) can be rewritten as

$$P_{msp}(L[i'], D) = P(t) \times P(t_{i'}) \times \prod_{x \in X_{i',1}} (1 - P(x)) \times \prod_{x \in X_{i',2}} (1 - P(x)) \quad (10)$$

Since $L[i']$, $L_p[i]$, $(t, t_{i'})$, and $(t, t_{p,i})$ refer to the same pair, equation (10) can be written as

$$P_{msp}(L[i'], D) = P_{msp}(L_p[i], D_p) \times \prod_{x \in X_{i',2}} (1 - P(x)) \quad (11)$$

Using the same notation, we can write $P_{msp}(L[j'], D)$ as

$$P_{msp}(L[j'], D) = P_{msp}(L_p[j], D_p) \times \prod_{x \in X_{j',2}} (1 - P(x)) \quad (12)$$

Based on the definition of stop index, it is clear that $stop > j$, thus yielding $i > j$. Thus, $i' > j'$ and we have

$$(\forall x \in X_{j',2}, \exists y \in X_{i',2} \mid x \subseteq y) \Rightarrow \prod_{y \in X_{i',2}} (1 - P(x)) \leq \prod_{x \in X_{j',2}} (1 - P(y)) \quad (13)$$

Moreover, we know that

$$P_{msp}(L_p[j], D_p) = local\_max > P_{msp}(L_p[i], D_p) \quad (14)$$

Using (11), (12), (13) and (14), we have

$$P_{msp}(L[i'], D) < P_{msp}(L[j'], D) \quad (15)$$

Since $L[i'] = L_p[i]$ and $L[j'] = L_p[j]$, (15) implies that $L_p[i] \neq \arg\max_{\rho \in L_p} P_{msp}(\rho, D)$.

B.3 Proof of Theorem 1

Let S be the set of all entity-tuple pairs at nodes which receive the query, i.e. $S = e \times D$. We show that we do not need any entity-tuple pair $\rho = (t, t'), \rho \in S - essential_{unified}$ for computing MPMP(e, D).

Let L be the list of pairs in set $essential_{unified}$ which have alternative $t \in e$ as their first element, and sorted based on their similarity in descending order. Let $stop_p$ be the stop index of a node, say node p, which comes before the stop indices of other nodes in list L. Using Lemma 2, pairs which

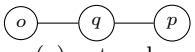
come at or after $stop_p$ in L, cannot be the pair of L with maximum P_{msp}. Thus, the pair of L with maximum P_{msp} lies in the range $[1..stop_p - 1]$. Now, we show that there is no entity-tuple pair $\rho = (t, t'), \rho \in S - essential_{unified}$, which may come before *stop* in list L, and thus, is needed for computing the pair of L with maximum P_{msp}. Pair ρ is either maintained at node p or at a node other than p, say q. In the former case, ρ comes after $stop_p$ in list L since $stop_p$ is the stop index p. Also in the latter case, ρ comes after $stop_p$ in list L since ρ comes after the stop index of q which itself comes after $stop_p$ in list L. Thus, using the pairs $L[1]$ to $L[stop-1]$, we can compute the pair with maximum P_{msp} and thereby MPMP(e, D).

C. FD EXAMPLE

In this section, we illustrate the FD algorithm with an example.

Consider a network consisting of three nodes o, q, and p as shown in Figure 15(a) and let these nodes respectively contain D_O, D_q, and D_p databases which are shown in Figures 15(c), 15(d), and 15(e), respectively. Suppose that the user submits entity e, shown in Figure 15(b), to the node o, then FD performs the following phases for computing MPMP(e, D), where $D = D_O \cup D_q \cup D_p$:

1. In the *"query forward"* phase, node o forwards e to node q, and node q forwards e to node p.
2. In the *"extract the essential-set"* phase, each node extracts its essential-set as shown in Figures 15(f), 15(g), and 15(h). In the figures, we abbreviate *Potential* as *Po*. Also, the pairs that are transferred to the essential-set, are shown in bold, and since FD stops at the stop index, the values of P_{msp} and *Potential* for the pairs after the stop index are not shown. Notice that the stop index of list L_q if equal to $|L_q| + 1$, thus all pairs in this list are transferred into the essential-set.
3. In the *"merge-and-backward essential-sets"* phase, node p sends its essential set, i.e. $essential(S_p)$, to node q. Then, q merges the received set with its own essential set, i.e. $essential(S_q)$, into set $essential_{pq}$ (shown in Figure 15(i)), and sends it to node o.
4. In the *"MPMP computation and data retrieval"* phase, node o merges its essential set, i.e. $essential_O$, with the received set from q, i.e. $essential_{pq}$, into set $essential_{unified}$ whose members are shown in Figure 15(j). Then, node o computes the MPMP as shown in Figure 15(j). The computed MPMP is equal to $(t, t_{p,4})$, thus node o asks node p for the data of the tuple $t_{p,4}$.



(a) network

t	P(t)
t	0.3

(b) e

t	P(t)
$t_{O,1}$	0.6
$t_{O,2}$	0.4
$t_{O,3}$	0.3
$t_{O,4}$	0.7
$t_{O,5}$	0.3
$t_{O,6}$	0.5

(c) D_O

t	P(t)
$t_{q,1}$	0.2
$t_{q,2}$	0.1
$t_{q,3}$	0.2
$t_{q,4}$	0.2
$t_{q,5}$	0.1
$t_{q,6}$	0.1

(d) D_q

t	P(t)
$t_{p,1}$	0.2
$t_{p,2}$	0.9
$t_{p,3}$	0.8
$t_{p,4}$	0.8
$t_{p,5}$	0.7
$t_{p,6}$	0.9

(e) D_p

Pair	P_{msp}	Po
$\boldsymbol{(t, t_{p,4})}$	0.24	0.3
$(t, t_{p,6})$	-	0.06
$(t, t_{p,1})$	-	-
$(t, t_{p,5})$	-	-
$(t, t_{p,3})$	-	-
$(t, t_{p,2})$	-	-

(f) L_p

Pair	P_{msp}	Po
$\boldsymbol{(t, t_{q,3})}$	0.06	0.3
$\boldsymbol{(t, t_{q,6})}$	0.02	0.24
$\boldsymbol{(t, t_{q,4})}$	0.04	0.22
$\boldsymbol{(t, t_{q,1})}$	0.03	0.17
$\boldsymbol{(t, t_{q,5})}$	0.01	0.14
$\boldsymbol{(t, t_{q,2})}$	0.01	0.12
-	-	0.11

(g) L_q

Pair	P_{msp}	Po
$\boldsymbol{(t, t_{O,5})}$	0.09	0.3
$\boldsymbol{(t, t_{O,2})}$	0.084	0.21
$\boldsymbol{(t, t_{O,1})}$	0.08	0.13
$(t, t_{O,4})$	-	0.05
$(t, t_{O,3})$	-	-
$(t, t_{O,6})$	-	-

(h) L_O

Pair
$(t, t_{q,3})$
$(t, t_{p,4})$
$(t, t_{q,6})$
$(t, t_{q,4})$
$(t, t_{q,1})$
$(t, t_{q,5})$
$(t, t_{q,2})$

(i) $essential_{pq}$

Pair	P_{msp}	Po
$(t, t_{q,3})$	0.06	0.3
$(t, t_{p,4})$	0.19	0.24
$(t, t_{O,5})$	-	0.05
$(t, t_{q,6})$	-	-
$(t, t_{q,4})$	-	-
$(t, t_{O,2})$	-	-
$(t, t_{q,1})$	-	-
$(t, t_{q,5})$	-	-
$(t, t_{q,2})$	-	-
$(t, t_{O,1})$	-	-

(j) L

Figure 15: a) Illustration of different phases of the FD algorithm using a simple example