Fuzzy Data Deduplication at Edge Nodes in Connected Environments

Sylvana Yakhni[1], Joe Tekli[1*], Elio Mansour[2], and Richard Chbeir[3]

[1] E.C.E. Department, Lebanese American University, 36 Byblos, Lebanon
silvana.yakhni@lau.edu, joe.tekli@lau.edu.lb

[2] Scient Analytics, 75007 Paris, France
elio.mansour@scient.io

[3] University of Pau and Pays Adour, 64600 Anglet, France
richard.chbeir@univ-pau.fr

Abstract. The Internet of Things (IoT) is ushering-in the era of connected environments, i.e., networks of physical objects that are embedded with sensors and softwar, connecting and exchanging data with other devices and systems. The huge amount of data produced by such systems calls for solutions to reduce the amount of data being handled and transmitted over the network. In this study, we investigate data deduplication as a prominent pre-processing method that can address such a challenge. Data deduplication techniques have been traditionally developed for data storage and data warehousing applications, and aim at identifying and eliminating redundant data items. Few recent approaches have been designed for sensor networks and connected environments, yet existing solutions mostly rely on crisp thresholds and provide minimum-to-no expert control over the deduplication process, disregarding the domain expert's needs in defining redundancy. In this study, we propose a new approach for Fuzzy Redundancy Elimination for Data Deduplication in a connected environment. We use simple natural language rules to represent domain knowledge and expert preferences regarding data duplication boundaries. We then apply pattern codes and fuzzy reasoning to detect duplicate data items at the outer-most edge (sensor node) level of the network. This reduces the time required to hard-code the deduplication process, while adapting to the domain expert's needs for different data sources and applications. Experiments on a real-world dataset highlight our solutions' potential and improvement compared with existing solutions.

Keywords. Connected Environments, Fuzzy Reasoning, Data Redundancy, Data Deduplication, Internet of Things (IoT), Wireless Sensor Networks.

1 Introduction

The Internet of Things (IoT) is ushering-in the era of connected environments, i.e., networks of physical objects that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems (e.g., smart hospitals, smart buildings, and smart cities) [1, 2]. According to a recent survey conducted by VoucherCloud in [3], 2.5+ quintillions of sensory data are currently generated every day, and over 123×10^9 IoT devices are expected to be connected within the next 10 years [4]. This huge amount of data highlights several challenges including network bandwidth, consumption of network energy, cloud storage, and I/O throughput. These call for data pre-processing and filtering techniques to reduce the amount of data being handled and transmitted over the network. In this study, we investigate data deduplication as a prominent pre-processing method that can address such challenges. Data Deduplication techniques have been traditionally developed for data storage and data warehousing applications, and

* Corresponding author

aim at identifying and eliminating redundant data items, where only one unique copy of the data is stored [5]. Similarly, data deduplication in connected environments aims at eliminating redundant measurements produced by sensing devices. For instance, there is no need to store and transmit similar temperature measurements produced by a sensor if they are almost identical within a given timespan. Such measurements would be considered redundant and need to be eliminated, where only relevant changes are processed by the system. In this context, few recent approaches have been designed for handling data in connected environments, e.g., [5-9], yet most existing solutions rely on crisp evaluation thresholds and provide the domain expert with minimum-to-no control over the deduplication process (i.e., which data need to be duplicated and which data should be kept intact) hence overlooking the expert's requirements and application needs in defining redundancy, e.g., [6, 9].

In this study, we propose a new approach for Fuzzy Redundancy Elimination for Data Deduplication (FREDD) in a connected environment. It uses simple natural language rules to represent domain knowledge and expert preferences regarding data duplication boundaries. It then applies pattern codes and fuzzy reasoning to detect duplicates on the edge of the network. This reduces the time required to hard-code the deduplication process, while adapting to the domain expert's needs and application requirements. Experiments on a real-world dataset highlight our solution's potential and improvement compared with existing approaches.

The remainder of this paper is organized as follows. Section 2 briefly reviews the related works. Section 3 describes our framework. Section 4 describes our experimental evaluation and results, before concluding in Section 5 with ongoing directions.

2 Related Works

Data deduplication techniques have been initially developed for data storage and data warehousing systems, e.g., [10-12], and have been recently investigated in the context of IoT and connected environments.

In [9], the authors address data redundancies at the core of the network using a supervised machine learning solution based on Support Vector Machines (SVM). They build an aggregation tree for the given size of the network and then apply SVM to recognize data redundancies. The authors target temporal and spatial redundancies once the data is consolidated in a central node, which provides a redundancy-free data repository that can be mined using dedicated data processing techniques (cf. *challenge 1*). However, redundancies are not handled at the edge level, and data exchange between devices at the edge remains costly due to unnecessary communications. In [13], the authors provide a data deduplication technique in healthcare-based IoT, and introduce a Controlled Window-size based Chunking Algorithm (CWCA) to identify cut-points in sensor data distributions. Yet similarly to [9], the solution in [13] only performs data deduplication at sink nodes and does consider redundancies at edge devices (cf. *challenge 1*). In [14], the authors focus on the spatial distribution of sensors in the environment, and how it can be managed to prevent redundancies. The authors build a graph of nodes and events in order to detect "redundant" nodes: i.e., nodes producing identical events. Redundant nodes are either relocated or put into sleep mode using a circle packing technique to enhance coverage, while minimizing energy usage during relocation. This work only handles redundancy from a sensor deployment perspective (i.e., avoiding deploying sensors that provide the same type of data in the same area).

In a continuous sensing setup, triggering mechanisms are available to restrict the number of transmissions between the sensor node and the monitoring node without degrading the tracking of the sensed measurements, e.g., [15, 16]. These mechanisms can also be used for filtering redundant spatial-temporal data, in order to trigger transmissions

from sensor node (edge) to monitoring node (sink) only when there are changes in the sensed measurements. These approaches fall into the category of edge-based data deduplication solutions, and rely on a simple crisp deviation threshold δ_v. More recently, the authors in [6, 17] proposed a Data Redundancy Management Framework (DRMF) that handles data redundancies at the edge device level considering both static and mobile devices. It clusters the data based on expert/system defined crisp deviation threshold, while also keeping track of the temporal and spatial-temporal spread (or coverage) of each cluster (i.e., sets of redundant data). The algorithm sorts all data tokens and checks if the current token belongs to the current cluster by comparing it with the cluster's centroid, considering a expert-defined deviation threshold δv. Otherwise, a new cluster is created with the current data item added as its centroid. In another relevant study in [18], the authors introduce the Redundancy Elimination Data Aggregation (REDA) algorithm to perform deduplication at individual edge nodes and also among nodes in the same cluster. Assuming that data is presented as a set of scalar values (e.g., temperature, humidity), the range of numbers is divided into crisp intervals that depend on the domain expert requirements, then a lookup table is generated for each cluster containing the ranges of each interval and their associated pattern codes (e.g., 10-15°C are associated with pattern code 1, 16-20° are associated with pattern code 2). Note that both [6, 18] rely on crisp deviation thresholds, and share the same limitations of crisp processing.

Discussion: Few recent solutions have been designed to handle data deduplication in connected environments. They utilize crisp thresholds where even the slightest variations in the sensed measurements are processed similarly to extremely large variations (e.g., given a temperature variation threshold δ_v = 1°C, variations of 1.5 °C and 20.5 °C are processed exactly the same). Similarly, variations which are slightly below the threshold are entirely ignored (e.g., given a temperature variation threshold δ_v=1°C, a variation of 0.99 °C goes unprocessed). Relying on crisp thresholds leads to i) missing certain relevant redundancies or ii) removing certain data values that might not be redundant. Also, existing solutions provide minimum-to-no expert intervention and adaptability in the deduplication process.

3 FREDD Framework

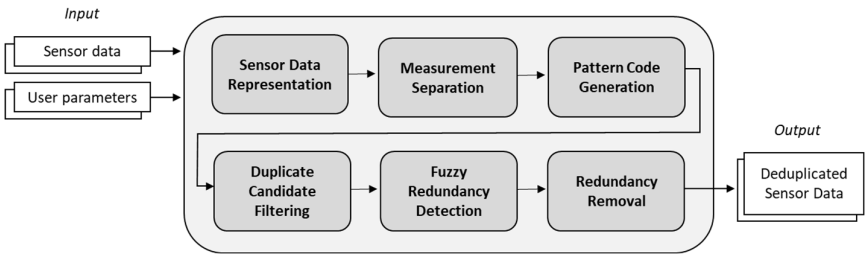


Fig. 1. Simplified diagram describing FREDD's architecture

To address the aforementioned challenges, we introduce FREDD: a new framework for Fuzzy Redundancy Elimination for Data Deduplication in a connected environment. FREDD detects data duplicates at edge (source) nodes, in order to minimize network traffic and bandwidth consumption. It combines simple natural language rules with a fuzzy inference mechanism designed to adapt the deduplication process following the expert's needs. FREDD's core architecture is depicted in Fig. 1. It consists of six main modules: i) *sensor data representation* which defines the spatial and temporal representations of data

measurements/items, ii) *measurement separation* which separates the input data into measurement-based data collections, iii) *pattern code generation* which associates data items with pattern codes based on expert-defined lookup tables, iv) *duplicate candidate filtering* which determines whether data items are candidates for fuzzy duplication, v) *fuzzy redundancy detection* which identifies duplicate data items using fuzzy reasoning based on expert-defined condition-action rules, and vi) *redundancy removal* which eliminates redundant data items and produces deduplicated data.

3.1. Sensor Data Representation

Connected environments contain diverse devices each embedding one or more sensors that provide data from the real world. Static devices are immobile; therefore, the data generated by such devices can be redundant temporally. Mobile devices produce data while moving around the environment, which potentially generates spatial-temporal redundancies. Here, we restrict our presentation to static devices dealing with temporal redundancies, and report spatial-temporal redundancies to a later dedicated study. We adopt a set of formal definitions from [6] that allows us to describe data items considering the temporal dimension:

Definition 1 - Data Items: A data item d is defined as a 4-tuple:

$$d = \langle m\ ;\ v\ ;\ t\ ;\ s \rangle \quad \textbf{(1)}$$

where m is the data measurement, v is the data value, t is the creation temporal stamp of d (cf. Definition 2), and s is the data source that sensed/created d ●

Definition 2 - Temporal Stamp: A temporal stamp t is defined as a single discrete temporal value represented as a 2-tuple:

$$t = \langle format\ ;\ value \rangle \quad \textbf{(2)}$$

where *format* is a string indicating the format of the date-time value of t (e.g., "dd-MM-yyyy hh:mm:ss"), and *value* is the timestamp value (e.g., 10-11-2020 15:34:23 following the sample time format mentioned above) ●

Table 1. Sample sensory data items

Measurement *m*	**Value *v***	**Time stamp *t***		**Source *s***
		format	*value*	
Humidity	92 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	S1
Temperature	16 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	S1
Humidity	94 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Temperature	19.5 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	S1
Humidity	103 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1
Humidity	104 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	S1

Consider the motivating example of a smart building, hosting a set of static sensing devices that provide *humidity* and *temperature* measurements (among others) from the environment. Devices have built-in memories to buffer chunks of sensory data before transmission to the network's sink nodes. Table 1 shows the representation of sample sensory data after being sensed/produced by an edge device (source) S_1 embedding two sensors producing *humidity* and *temperature* measurements respectively.

3.2. Measurement Separation

Since the device can embed various sensors, its internal memory might store different measurements (i.e., features such as *humidity* and *temperature* in Table 1). Therefore, in order to detect redundancies in the data stored locally on the edge device, we start by filtering the data into collections having the same measurements. To illustrate the measurement filtering process, the data shown in Table 1 produces two distinct data collections: the first for *humidity* data (first four tuples - cf. Table 2), and the second for *temperature* data (containing the last tuples). Consequently, the measurement data collections are processed separately for data deduplication. Note that the domain expert decides about the selection of measurements to be processed for deduplication.

Table 2. Measurement separation of the data from Table 1

a. *Humidity* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Source *s***
		format	*value*	
Humidity	92 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	S1
Humidity	94 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Humidity	103 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1
Humidity	104 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	S1

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Source *s***
		format	*value*	
Temperature	16 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	S1
Temperature	19.5 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1

3.3. Pattern Generation Code

The pattern code generation module transforms ranges of data item values for a given measurement (e.g., *humidity*, *temperature*) into interval values that are defined based on reference lookup tables. Edge and sink devices handling the same measurements refer to the corresponding measurement lookup tables (e.g., *humidity* lookup table, or *temperature* lookup table), where lookup tables are created based on expert preferences or application requirements. Here, we distinguish between two kinds of lookup tables allowing: i) *disjoint* data ranges, and ii) *intersecting* data ranges.

Table 3. Sample *disjoint* value lookup tables for *humidity* and *temperature* measurements, considering ranges 90-110 µg/m [1] and 15-28 °C respectively

a. Disjoint *humidity* data ranges

Interval values	[90, 96] µg/m^3	]96, 104] µg/m^3	]104, 110] µg/m^3
Pattern code	H1	H2	H3

b. Disjoint *temperature* data ranges

Interval values	[15-19] °C	]19-24] °C	]24-28] °C
Pattern code	T1	T2	T3

[1] Microgram Per Cubic Meter

Table 4. Sample *intersecting* value lookup tables for *humidity* and *temperature* measurements, considering ranges 90-110 µg/m³ and 15-28 °C respectively

a. Intersecting *humidity* data ranges

Interval values	[90, 98] µg/m³	[94, 106] µg/m³	[102, 110] µg/m³
Pattern code	H1	H2	H3

b. Intersecting *temperature* data ranges

Interval values	[15-20] °C	[18-25] °C	[23-28] °C
Pattern code	T1	T2	T3

Disjoint data ranges (cf. Table 3) allow simple pattern code generation, yet they produce disconnected pattern codes where values on the range boundaries might be misrepresented (e.g., it is not clear which pattern code can be assigned with values 96.2 µg/m³ or 104.7 µg/m³ following Table 3). *Intersecting* data ranges (cf. Table 4) allow the generation of combined pattern codes when the target value belongs to more than one range (e.g., *humidity* values 103, 104, and 105 µg/m³ belong to both H2 and H3 pattern codes following Table 4).

In this study, we consider *intersecting* ranges to allow more efficient processing (duplicate candidate filtering, cf. Section 3.4) and more accurate data deduplication (fuzzy redundancy detection, cf. Section 3.5)

Table 5 shows the pattern codes generated for the data items from our running example in Table 2, considering the above look-up tables.

Table 5. Value, zone, and combined pattern codes for sample data from Table 1

a. *Humidity* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Source *s***
			format	*value*	
Humidity	92 µg/m³	{H1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	S1
Humidity	94 µg/m³	{H1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Humidity	103 µg/m³	{H2,H3}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1
Humidity	104 µg/m³	{H2,H3}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	S1

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Source *s***
			format	*value*	
Temperature	16 °C	{T1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	S1
Temperature	19.5 °C	{T1,T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1

3.4. Duplicate Candidate Filtering

Since sensor data items are produced and ordered per sensing time stamp, each data item to be deduplicated is evaluated with its previous one to check if the data is duplicate or not. Our duplicate candidate filtering algorithm is depicted in Fig. 2. It accepts as input two consecutive data items and produces as output a decision of whether the data items are duplicates, non-duplicates, or candidates for deduplication, based on the following rules: i) if two data items share one value-zone pattern code, then they are considered duplicates (cf. Fig. 2, lines 4-5), ii) if the data items share one or more value-zone pattern codes, they are considered as candidates for deduplication (lines 6-7), and iii) if the data items do not share any value-zone pattern code, they are considered as non-duplicates (lines 8-9),

Table 6 shows the output of the filtering algorithm applied on the input data from Table 5, where 6 data items are identified as either duplicates/non-duplicates, such that 2 of the

original 8 items need to be further considered for fuzzy deduplication. Depending on the data patterns generated in the target connected environment, duplicate filtering can significantly reduce the number of data items to be processed for fuzzy redundancy detection, thus significantly improving overall processing performance especially at the device level (cf. experimental results in Section 4).

Algorithm 1 – Duplicate Candidate Filtering

Input: DataItem1, DataItem2
Output: DeduplicationStatus

```
Begin
 1    pattern1 ← pattern code for DataItem1
 2    pattern1 ← pattern code for DataItem2
 3    interLen ← length of intersection between DataItem1 and DataItem2
 4    if (pattern1 = pattern2) and (interLen =1) then
 5         DeduplicationStatus ← Duplicates
 6    else if interLen > 1 then
 7         DeduplicationStatus ← Candidates
 8    else
 9         DeduplicationStatus ←  NotDuplicates
End
```

Fig. 2. Pseudocode of our *duplicate candidate filtering* algorithm

Table 6. Output of the filtering algorithm applied on input data from Table 5

Duplicate Candidate for Deduplication Non-Duplicate

a. *Humidity* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Source *s***
			format	*value*	
Humidity	92 µg/m^3	H1	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	S1
Humidity	94 µg/m^3	H1	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Humidity	103 µg/m^3	H2 H3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1
Humidity	104 µg/m^3	H2 H3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	S1

Duplicates
Non-Duplicates
Candidates

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Source *s***
			format	*value*	
Temperature	16 °C	{T1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	S1
Temperature	19.5 °C	{T1,T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1

Non-Duplicates
Candidates
Duplicates

3.5. Fuzzy Redundancy Detection

The fuzzy redundancy detection module's overall process is shown in Fig. 3. It accepts as input data items that are candidates for redundancy detection, and then produces as output their deduplication status (i.e., *duplicates* or *non-duplicates*).

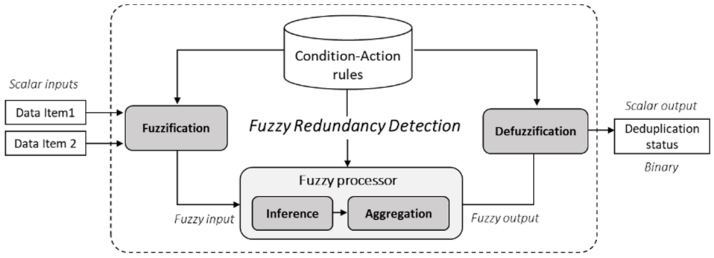
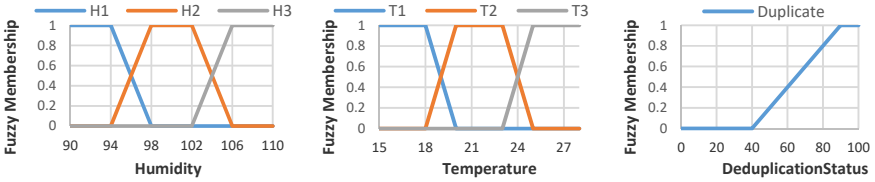


Fig. 3. Simplified diagram describing the *fuzzy redundancy detection* module's process

Fuzzification: First, the scalar data item values are fuzzified, producing linguistic values associated with fuzzy membership degrees (e.g., *humidity* value 103 µg/m^3 becomes 75% H2 and 25% H3 following Fig. 4). The fuzzy partitions for every measurement are defined based on the corresponding lookup table ranges, where the fuzzy membership functions can be defined following the expert and application needs (cf. Fig. 4.a and b). The output *deduplication status* variable represents a percentage value using one membership function varying from 0-to-100% duplication (cf. Fig. 4.c).



a. Input *humidity* fuzzy partitions, cf. Table 4.a

b. Input *temperature* fuzzy partitions, cf. Table 4.b

c. Output *deduplication status* fuzzy partitions

Fig. 4. Input *humidity* and *temperature* fuzzy partitions, and output *deduplication status* fuzzy partitions defined using the trapezoidal function following the lookup tables in Table 4

Condition-action rules: As for the fuzzy agent's condition-action rules, they reflect the common sense logic applied by an domain expert to determine whether two data items are duplicates or not, based on their measurement's look-up tables:

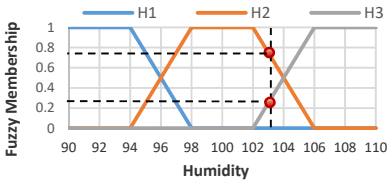
Rule 1. **IF** (Humidity_Item1 is H1) **AND** (Humidity _Item2 is H1) **THEN** DedupStatus is *Duplicate*
Rule 2. **IF** (Humidity_Item1 is H2) **AND** (Humidity _Item2 is H2) **THEN** DedupStatus is *Duplicate*
Rule 3. **IF** (Humidity_Item1 is H3) **AND** (Humidity _Item2 is H3) **THEN** DedupStatus is *Duplicate*
Rule 4. **IF** (Temp_Item1 is T1) **AND** (Temp_Item2 is T1) **THEN** DedupStatus is *Duplicate*
Rule 5. **IF** (Temp_Item1 is T2) **AND** (Temp_Item2 is T2) **THEN** DedupStatus is *Duplicate*
Rule 6. **IF** (Temp_Item1 is T3) **AND** (Temp_Item2 is T3) **THEN** DedupStatus is *Duplicate*

Inference: Fuzzy inference consists in applying the concerned condition-action rules on the fuzzified data in order to produce fuzzy outputs. The logical connectors in the condition-action rules are translated into mathematical formulas that operate on the fuzzy data. In our agent, we adopt *Mamdani's implication* operator as the default inference function given its common usage in the literature [19, 20].

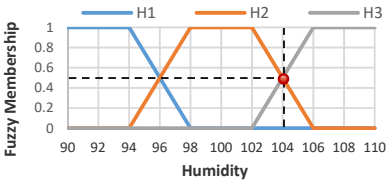
Aggregation: It allows grouping the outputs of multiple inference operations executed on multiple condition-action rules, in order to produce on single fuzzy output result. In our agent, we adopt the *maximization* aggregation function (Formula 5) given its usage in the literature [19, 21]. Others formulas like *bounded sum* and *weighted sum* can be utilized.

1. Fuzzification: Given *case 1*'s input data: Humidity_DataItem1 = 103 µg/m^3 and Humidity_DataItem2 = 104 µg/m^3, we compute the corresponding fuzzy membership values following the *humidity* fuzzy functions in Fig. 4.a (reported below):

- For Humidity_DataItem1:
  $f_{H1}(103) = 0$, $f_{H2}(103) = 0.75$, and $f_{H3}(103) = 0.25$



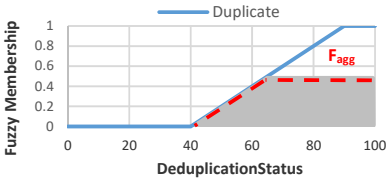
- For Humidity_DataItem2:
  $f_{H1}(104) = 0$, $f_{H2}(104) = 0.5$, and $f_{H3}(104) = 0.5$



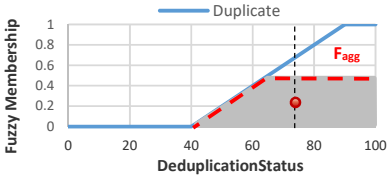
2. **Condition-Action rules:** Based on the input membership values, the following condition-action rules are invoked:

- ***Rule 2***: H2(Humidity_DataItem1) ∧ H2(Humidity_DataItem2) ⇒ *Duplicate*(DedupStatus)
- ***Rule 3***: H3(Humidity_DataItem1) ∧ H3(Humidity_DataItem2) ⇒ *Duplicate*(DedupStatus)

4. **Inference and Aggregation:** By applying *Mamdani*'s inference mechanism and the *maximization* aggregation function, $F_{agg} = F_{max} = max(f_{Rule2}, f_{Rule3})$, the agent produces the fuzzy coverage areas subsumed by the inference membership functions (represented in grey color).



4. **Defuzzification:** The *center of gravity* defuzzification function is applied on the fuzzy coverage area to compute the center of gravity point (represented as a red dot), and then identify the corresponding deduplication status (on the *x* axis) as the agent's output = 76%.



6. **Result:** Given $dedup_{threshold}$ = 75% in our running example, and since the output of the defuzzification step is 76% ≥ $dedup_{threshold}$, the agent's final output becomes: dedupStatus = *duplicates*

Fig. 5. Fuzzy redundancy detection process for the *humidity* sample case (cf. Table 6)

Deduplication: It allows transforming the fuzzy output produced by the aggregation function into a crisp output that represents the final result of the agent. In our agent, we adopt *center of gravity* (Formula 6) given its common usage in the literature [19, 21]. Other formulas like *maximum to the left* and *maximum to the right* can be utilized.

Mamdani's implication:

Given fuzzy sets f_1, f_2:

$$f_1 \Rightarrow_{\text{Mamdani}} f_2 \equiv f_1 \wedge f_2 \equiv min(f_1, f_2) \quad \textbf{(4)}$$

where ∧ is the AND fuzzy logic operator[1]

Maximization aggregation:

Given fuzzy sets $f_1, f_2, \ldots, f_n$: **(5)**

$$F_{agg} = F_{Max} = max(f_1, f_2, \ldots, f_n)$$

Center of gravity defuzzification:

Given aggregate fuzzy set F_{Agg}: **(6)**

$$x = \frac{\int x \times F_{agg}(x) \times dx}{\int F_{agg}(x) \times dx}$$

Computation example: We consider in Table 6 two cases for *humidity* and *temperature* measurements studied in our motivation scenario. The detailed computation process for *humidity* described in Fig. 5 (a similar computation process is executed for *temperature*).

[1] The AND fuzzy logic operator can be any t-norm function, including *min* which is commonly adopted in the literature.

The agent recommends that input 103 µg/m³ and 104 µg/m³ data values are duplicates with a 76% fuzzy membership degree, which seems reasonable given the *humidity* lookup tables and value ranges defined in Table 4 (H2 and H3 fuzzy partitions intersect between [102, 106] µg/m³, where 103 is much closer to the 102 µg/m³ boundary of H2 than to the 106 µg/m³ boundary of H3, but also 103 µg/m³ and 104 µg/m³ are close to each other). Given our running example data from Table 6, the identified *humidity* and *temporal* redundancies following the fuzzy redundancy detection process are shown in Table 7.

Table 7. Output of the fuzzy redundancy detection process applied on the data from Table 6

a. *Humidity* data collection

			Time stamp *t*			
Measurement *m*	**Value *v***	**Value Pattern Code**	*format*	*value*	**Source *s***	
Humidity	92 µg/m³	H1	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	S1	Duplicates
Humidity	94 µg/m³	H1	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1	
Humidity	103 µg/m³	H2 H3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1	Duplicates
Humidity	104 µg/m³	H2 H3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	S1	

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Source *s***	
			format	*value*		
Temperature	16 °C	{T1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	S1	
Temperature	19.5 °C	{T1,T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	S1	Duplicates
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	S1	
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1	

3.6. Redundancy Removal

Once redundancies are identified, the redundancy removal process occurs. Here, domain experts might have different needs for redundancy removal. This component summarizes a sequence of redundancies into one representative data item following an expert-chosen redundancy removal function (e.g., *media*, *mean*, maximum, minumum,) representative. Experts provide their requirements in the form of simple consumer requests that the module processes to execute the required redundancy removal functions. For instance, Table 8 shows the *humidity* and *temporal* redundancies that are removed using the *median* function.

Table 8. Output of redundancy removal using the *median* function applied on Table 7

a. *Humidity* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Source *s***
		format	*value*	
Humidity	92 µg/m³	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	S1
Humidity	103 µg/m³	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Source *s***
		format	*value*	
Temperature	16 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	S1

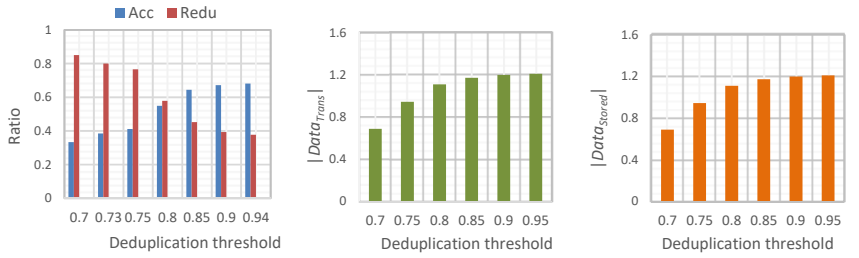
4 Experimental Evaluation

We have implemented our *FREDD* framework as a web-based application, using methods from the *jFuzzyLogic* open source library [22, 23] in implementing our fuzzy logic agent, to allow easy manipulation for domain experts in operating and evaluating the system[1]. We considered Intel Lab Berkeley dataset [24] obtained from 54 Micra2Dot sensors providing weather data including temperature, humidity, light, as well as the list of Cartesian coordinates for each of the 54 sensors, and the time when each data measurement is collected. In our empirical evaluation, we consider 20k *humidity* and *temperature* data measurements collected from sensor S1 on 28/2/2004.

We utilize four evaluation metrics: i) *deduplication accuracy* (acc): time series similarity between the original data and the deduplicated data [5], ii) *data reduction ratio* (*redu*) is defined as the ratio of the difference between the original data and the duplicated data, iii) *size of transmitted data* ($|data_{trans}|$) represents the size of the data transmitted from the edge devices to the sink device (a good deduplication solution would reduce the size of data transmitted over the network in order to gain in network bandwidth), and iv) *size of stored data* ($|data_{stored}|$) represents the size of the data stored at the sink device (a good deduplication solution would reduce the size of the data stored at the sink to gain in processing speed and throughput at the sink level). The system implementation, experimental datasets, and test results are available online[2].

4.1. Fuzzy Deduplication Threshold Evaluation

We vary the fuzzy deduplication threshold, allowing the fuzzy redundancy detection process to decide on the deduplication status of candidate data items, and evaluate FREDD's behavior accordingly. Results in Fig. 6 show that when the threshold increases: i) *acc* increases while ii) *redu* decreases. This is due to the fact that a higher deduplication threshold means less candidate pairs are considered for duplication. Also, the size of data transmitted to the sink ($|data_{trans}|$) and the size of data stored at the sink ($|data_{stored}|$) are both increased with the increase in deduplication threshold. This is mainly due to the decrease in *redu*, resulting in more data being sent and processed at the sink node. Fine-tuning the evaluation metric values can be handled automatically as a multi-objective optimization problem, e.g., [25-27]. We report this to a dedicated study.



a. *Acc* and *redu* results

b. Size of data transmitted to the sink ($|data_{Trans}|$)

d. Size of data stored at the sink ($|data_{Stored}|$)

Fig. 6. Deduplication quality metrics obtained with varying fuzzy deduplication thresholds

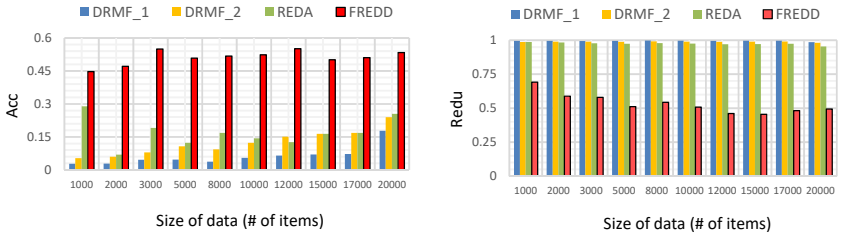
[1] We adopt a three-layer architecture: i) a *Web API* layer that allows client-side applications to communicate with the server to request data, etc.; ii) a *Business Logic* layer where *FREDD*'s main decision making processes are implemented; and iii) a *Data Access* layer where data storage and retrieval take place.

[2] http://sigappfr.acm.org/Projects/FREDD/

4.2. Baseline Comparisin with Existing Approaches

We conducted a comparative study to assess FREDD's effectiveness with respect to recent alternatives in the literature: i.e., REDA [18] and DRMF [6]. To test REDA, we consider the crisp humidity ranges shown in Table 4. To test FREDD, we consider the fuzzy humidity ranges in Fig. 4 and we set the deduplication threshold to 0.8. We also consider two variations of DRMF: i) the first one with a deviation threshold equal to one quarter of the width of the crisp range $\delta = 3/4$ (which we refer to as DRMF_1), and ii) the second one with a deviation threshold equal to one eighth of the width of the crisp range $\delta = 3/8$ (which we refer to as DRMF_2). Results in Fig. 7 show that FREDD consistently achieves the best *acc* results across all data variations compared with both REDA and DRMF1/2. This is due to FREDD's fuzzy processing capability, allowing to detect approximate redundancies and process them for deduplication, compared with the crisp decision-making processes performed by REDA and DRMF.

To further explain the results in Fig. 7, we conduct a second experiment where we compare the decision-making behavior of each algorithm applied on different pairs of humidity data measurement; the first data item is fixed at a certain value, while the second item is varied within a controlled range. Fig. 8 shows the percentage of deduplication produced by each algorithm for a first humidity value of 39.5 µg/m^3, and the second value with a variation range of ± 2.5 µg/m^3. Results for exising solutions show that all values that lie between [38, 41] µg/m^3 are considered automatic duplicates (i.e., 100% duplicates).



a. Deduplication accuracy (*acc*) **b.** Data reduction ratio (*redu*)

Fig. 7. Comparison of the deduplication quality metrics between RED, DRMF1/2 and FREDD, when varying the number of data measurements of *dataset1*

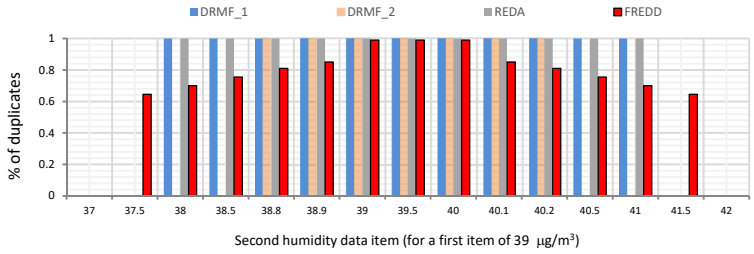


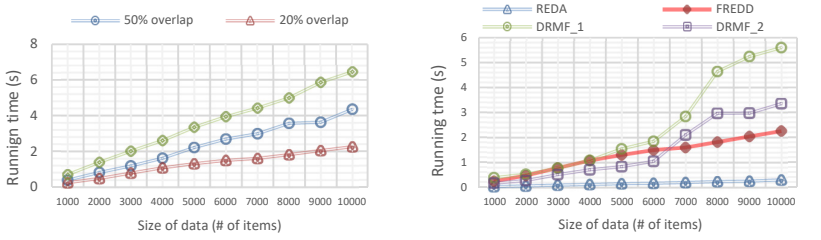
Fig. 8. Percentage of deduplicates with first humidity data fixed at 39.5 µg/m^3 and varying the second between [37, 42] µg/m^3

In contrast, each pattern code range in FREDD is divided into: i) a crisp range where pairs are automatically considered duplicates (i.e., from [39, 40] µg/m^3), and ii) a fuzzy range (i.e., between [37, 39] µg/m^3 and [40, 42] µg/m^3) where boundaries from different

other ranges overlap. In the fuzzy range, the deduplication decision is made based on a fuzzy inference system and a set of fuzzy rules, allowing the percentage of duplicates to vary accordingly (e.g., for a second value of 38 µg/m^3, the percentage of duplicates is 70%). Less duplicate pairs are considered automatic duplicates and the accuracy of the deduplication process increases accordingly (as shown in Fig. 8).

4.3. Performance Evaluation

We have also compared FREDD's time complexity with its recent alternatives, REDA, DRMF_1 and DRMF_2. FREDD's complexity simplifies to: $O(N \times E^2)$ where N designates the number of data items considered per edge device, and E the number of edge devices considered per sink node. Tests were carried out on a PC with an *Intel I7* system with 2.9 GHz CPU/16GB RAM. Fig. 9.a highlights the linear complexity of FREDD's deduplication process when varying the number of data items per edge node, reflecting $O(N)$ time complexity. Fig. 9.b shows running time results considering a fix data size per edge device =1000 items and a fixed number of edges per sink node = 10. Results show that REDA is the most efficient approach due to its fast and crisp pattern code assignment approach. FREDD requires more processing time than REDA due to its fuzzy computation process. DRMF is seemingly the most time consuming approach due to its data clustering process.



a. Edge-level processing time when varying the number of data items

b. Time performance compared with its alternatives, considering a fixed data size of 1000 items per edge

Fig. 9. Time performance results

5 Conclusion

This paper introduces a new approach for Fuzzy Redundancy Elimination for Data Deduplication (FREDD) in a connected environment. It uses natural language rules to represent domain knowledge and expert preferences regarding data duplication boundaries. It then applies pattern codes and fuzzy reasoning to detect duplicates on the general network infrastructure including both the edge level and the sink level of the network. Experiments highlight our solution's potential and improvement compared with existing solutions.

We are currently investigating the use of parametric learners [28, 29] and meta-heuristic algorithms [30, 31] to (semi) automatically configure the pattern codes' interval ranges and their fuzzy rules based on expert or data related features. We are currently investigating data deduplication at the sink level of the network [32], where data is aggregated from multiple edge nodes, including edge node mobility, edge node coverage area overlapping, and inter-edge collaboration. In the future, we plan to investigate data recovery [33, 34] in connected environments, including damage assessment and recovery from deduplicated data.

References

[1] Nižetić S. et al., *Internet of Things (IoT): Opportunities, Issues and Challenges towards a Smart and Sustainable Future.* Journal of Cleaner Production, 2020. 274: 122877.

[2] Lytras M., et al., *Enabling Technologies and Business Infrastructures for Next Generation Social Media: Big Data, Cloud Computing, IoT and VR.* Journal of Universal Computer Science, 2015, 21(11): 1379-1384.
[3] VoucherCloud, *The Uses of Big Data.* www.vouchercloud.com/resources/everyday-big-data, 2018.
[4] IoT Analytics, *State of IoT 2021.* https://iot-analytics.com/number-connected-iot-devices/ (Feb. 2023), 2021.
[5] Ismael W., et al., *An In-Networking Double-Layered Data Reduction for Internet of Things (IoT).* Sensors, 2019. 19(4): 795.
[6] Mansour E., et al., *Data Redundancy Management in Connected Environments.* Inter. Confe. on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (Q2SWinet), 2020, 75-80.
[7] Qutub B., et al., *Data Reduction in Low Powered Wireless Sensor Networks.* Wireless Sensor Networks-Technology and Applications, 2012. 10.5772/50178.
[8] Li S. et al., *EF-Dedup: Enabling Collaborative Data Deduplication at the Network Edge.* IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019. pp. 986–996.
[9] Patil P. and Kulkarni U., *SVM-based Data Redundancy Elimination for Data Aggregation in Wireless Sensor Networks.* Advances in Comput., Comm. & Info. (ICACCI), 2013, 1309–1316.
[10] Christen P., *A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication.* IEEE Transactions on Knowledge and Data Engineering, 2012. 24(9): 1537-1555.
[11] Malhotra J. and Bakal J., *A Survey and Comparative Study of Data Deduplication Techniques.* Inter. Conf. on Pervasive Computing (ICPC), 2015. pp. 1-5.
[12] Bhalerao A. and Pawar A., *A Survey on Data Deduplication for Efficiently Utilizing Cloud Storage for Big Data Backups.* Trends in Electronics and Informatics (ICEI), 2017. pp. 933-938.
[13] Ullah A. et al., *Secure Healthcare Data Aggregation and Deduplication Scheme for FoG-Orineted IoT.* IEEE Inter. Conf. on Smart Internet of Things (SmartIoT) 2019. pp. 314–319.
[14] Chowdhury S. and Benslimane A., *Relocating Redundant Sensors in Randomly Deployed Wireless Sensor Networks.* IEEE Global Communications Conf. (GLOBECOM), 2018. pp. 1-6.
[15] Santini S. and Romer K., *An Adaptive Strategy for Quality-based Data Reduction in Wireless Sensor Networks.* Inter. Conf. on Networked Sensing Systems (INSS'06), 2006. 14407470.
[16] Liansheng T. and Wu M., *Data Reduction in Wireless Sensor Networks: A Hierarchical LMS Prediction Approach.* IEEE Sensors journal, 2015. 16.6 (2015): 1708-1715.
[17] Shahzad F., et al., *Data Redundancy Management Framework for Connected Environments.* Computing journal, 2022. 104(7): 1565-1588.
[18] Khriji S., et al., *Redundancy Elimination for Data Aggregation in Wireless Sensor Networks.* Inter. Multi-Conference on Systems, Signals & Devices (SSD'18), 2018. 2018: 28-33.
[19] Salloum G. and Tekli J., *Automated and Personalized Nutrition Health Assessment, Recommendation, and Progress Evaluation using Fuzzy Reasoning.* Inter. J. of Human-Computer Studies, 2021. 151:102610.
[20] Bouchon-Meunier B., et al., *Compositional Rule of Inference as an Analogical Scheme.* Fuzzy Sets and Systems, 2003. 138(1): 53-65.
[21] Ross T. J., *Fuzzy Logic with Engineering Applications.* Wiley; 4th edition 2016. 580 p.
[22] Cingolani P. and Alcala-Fdez J., *jFuzzyLogic: a Robust and Flexible Fuzzy-Logic Inference System Language Implementation.* In IEEE Inter. Conf. on Fuzzy Systems, 2012. pp. 1-8.
[23] Cingolani P. and Alcalá-Fdez J., *jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming.* Int. Journal Comput. Intell. Syst., 2013. 6(1): 61–75.
[24] Bodik P., et al., *Intel Lab Data.* http://db.csail.mit.edu/labdata/labdata.html (assessed in Feb. 2023), 2019.
[25] Hopfield J., *The Effectiveness of Neural Computing.* IFIP World Computer Congress (WCC), 1989. 402-409.
[26] Zou F., et al., *A Reinforcement Learning Approach for Dynamic Multi-objective Optimization.* Information Sciences, 2021. 546: 815-834.
[27] Salloum G. and Tekli T., *Automated and Personalized Meal Plan Generation and Relevance Scoring using a Multi-Factor Adaptation of the Transportation Problem.* Soft Computing, 2022. 26(5):2561-2585.
[28] Abboud R. and Tekli J., *Integration of Non-Parametric Fuzzy Classification with an Evolutionary-Developmental Framework to perform Music Sentiment-based Analysis and Composition.* Soft Computing, 2019. 24(13): 9875-9925
[29] Wen X., *Using Deep Learning Approach and IoT Architecture to Build the intelligent Music Recommendation System.* Soft Computing, 2021. 25(4): 3087-3096.
[30] Azar D., et al., *A Combined Ant Colony Optimization and Simulated Annealing Algorithm to Assess Stability and Fault-Proneness of Classes Based on Internal Software Quality Attributes.* Inter. J. of AI, 2016. 14:2.
[31] Nguyen T., *A Novel Metaheuristic Method based on Artificial Ecosystem-based Optimization for Optimization of Network Reconfiguration to Reduce Power Loss.* Soft Computing, 2021. 25(23): 14729-14740.
[32] Yakhni S., et al., *Using Fuzzy Reasoning to Improve Redundancy Elimination for Data Deduplication in Connected Environments.* Soft Computing, 2023. https://doi.org/10.1007/s00500-023-07880-z.
[33] Haraty R. and El Sai M., *Information Warfare: a Lightweight Matrix-based Approach for Database Recovery.* Knowlegde and Information Systems 2017. 50(1): 287-313 (2017).
[34] Haraty R., et al., *Data Damage Assessment and Recovery Algorithm from Malicious Attacks in Healthcare Data Sharing Systems.* Peer Peer Network Applications, 2016. 9(5): 812-823 (2016).