

Soft Querying JSON Datasets with Personalized Preferences and Aggregations

Paolo Fosci^a and Giuseppe Psaila^b

University of Bergamo, DIGIP, Viale Marconi 5, 24044 Dalmine (BG), Italy
paolo.fosci@unibg.it, giuseppe.psaila@unibg.it

Keywords: Soft Web Intelligence, User-Defined Fuzzy Evaluators, Novel Constructs in J-CO-QL+, Soft Querying on JSON Datasets, Preferences Among Linguistic Predicates.

Abstract: Soft conditions are a powerful and established formal tool to select data on the basis of linguistic predicates. In previous work, the *J-CO* Framework (and its query language) was used to perform *Soft Web Intelligence*, i.e., a practical interpretation of the concept of *Web Intelligence* that exploits soft conditions to search for desired items in *JSON* datasets acquired from Web sources. However, the effectiveness of soft conditions depends on how elementary conditions are combined: in this sense, a plethora of proposals are available, such as the vector p -norm.

This paper shows how a generic concept, named “user-defined fuzzy evaluator”, that has been recently introduced in the query language, actually allows users to define their own operators, so as to express advanced operators such as “and possibly”. The paper also shows how the AND operator defined as a vector p -norm actually behaves, depending on different configurations of parameters, so as to let the reader understand how to use it in practice.

1 INTRODUCTION

The concept of *Soft Web Intelligence* was introduced in (Fosci and Psaila, 2022b; Fosci and Psaila, 2023c) and later refined in (Fosci and Psaila, 2023a): the concept gives a practical interpretation to the general concept of *Web Intelligence*, which was introduced more than two decades ago in (Yao et al., 2001). Specifically, the vision behind *Soft Web Intelligence* is to view the World-Wide Web as a giant source of information and datasets, that must be gathered from web sources, possibly stored within NoSQL (Not Only SQL) data stores, integrated and queried; soft querying, based on Fuzzy Sets and Fuzzy Logic (introduced in (Zadeh, 1965)), a formal approach that is widely recognized as a key concept in Artificial Intelligence, could be the right tool to deal with imprecise and linguistic conditions to be evaluated on the mass of gathered data.

Tasks of *Soft Web Intelligence* can actually be performed by exploiting the *J-CO* Framework (Fosci and Psaila, 2022b; Fosci and Psaila, 2023a); it is a pool of software tools under development at the Univer-

sity of Bergamo (Italy); the framework is designed to provide analysts and data engineers with sophisticated capabilities to gather, integrate and query *JSON* datasets (Fosci and Psaila, 2021a). In particular, its query language, named *J-CO-QL+*, is undergoing a continued evolution with the addition of novel constructs. Specifically, the authors are now unifying two distinct concepts, to simplify the language and provide a more powerful construct for evaluating membership degrees to fuzzy sets, so as to widen the potential application fields of the framework. Indeed, in the current version of the language, the novel concept of “Fuzzy Evaluator” unifies the former concepts of “Fuzzy Operator” (see (Fosci and Psaila, 2022b; Fosci and Psaila, 2023c)) and of “Fuzzy Aggregator” (see (Fosci and Psaila, 2023b; Fosci and Psaila, 2023a)).

The concept of “Fuzzy Evaluator” not only unifies two concepts that were previously managed through two distinct linguistic constructs; it also opens the way to define sophisticated fuzzy concepts that it was not possible to specify previously, such as preferences among predicates based on sophisticated approaches. An interesting example is constituted by logical operators defined through the “Vector p -norm” (see (Bordogna and Psaila, 2008; Fosci and Psaila, 2023b)),

^a  <https://orcid.org/0000-0001-9050-7873>

^b  <https://orcid.org/0000-0002-9228-560X>

in which preferences among predicates can be expressed, so as to give unequal importance to predicates. The availability of a linguistic tool that jointly supports user-defined preferences and aggregations further expands the way tasks of *Soft Web Intelligence* can be carried on.

This paper, after a discussion about previous work (Section 2), introduces the novel statement `CREATE FUZZY EVALUATOR`, showing how to define various types of aggregators, both for aggregating raw data and for expressing preferences among linguistic predicates (Section 3). Then, the practical exploitation of fuzzy evaluators is discussed (Section 4), by relying on the same case study formerly presented in (Fosci and Psaila, 2023a), so as to see the improvements that are enabled by the novel concept. Section 5 presents a sensitivity analysis with different importance weights for linguistic predicates, so as to understand how the behaviors of the Vector p -norm varies. Finally, Section 6 draws conclusions and future work.

2 PREVIOUS WORK

This paper relies on several previous works. Hereafter, the main ones are discussed. A detailed review of related work on the topic is not reported, because the current proposal is unique and not counterparts are available. Furthermore, readers that are interested in the relation with *Web Intelligence* can refer to (Fosci and Psaila, 2022b; Fosci and Psaila, 2023a).

2.1 The J-CO Framework

The *J-CO* Framework is a pool of software tools whose goal is to provide analysts with powerful support for gathering, integrating and querying possibly-large collections of *JSON* datasets. The core of the framework is its query language, named *J-CO-QL*⁺.

The current organization of the framework is not different from the one that was presented in (Fosci and Psaila, 2023a). Consequently, it is not necessary to report it in this paper (the interested reader can refer to (Fosci and Psaila, 2023a)). The following provides a short overview of the main features that are offered by the *J-CO* Framework.

- *J-CO-QL*⁺ is the query language of the framework. By means of it, users (such as analysts and data scientists) can write scripts that acquire *JSON* datasets from Web sources and *JSON* stores, transform and integrate them, perform soft queries and save results again into *JSON* stores.
- *J-CO-QL*⁺ instructions are made through declarative statements. Scripts are sequences of state-

ments that receive and generate a “temporary collection”, i.e., a pool of *JSON* documents.

- Fuzzy sets are natively supported: for each single *JSON* document, it is possible to evaluate its membership degrees to several fuzzy sets at the same time. This “fuzzification” process is made possible by adding, to each document, a special root-level field, named `~fuzzysets`, holding the membership-degree values to different fuzzy sets. Through membership degrees to fuzzy sets, it is possible to exploit the concept of “linguistic predicate”, so as to exploit vague or imprecise conditions.

2.2 Fuzzy Logical Aggregators

When dealing with multiple fuzzy sets, to which an entity can belong, and with a multitude of entities that belong to the same fuzzy set, several interpretations of the concept of “aggregation” can be provided. Indeed, a plethora of proposals can be found in the literature.

Many fuzzy aggregators, such as “t-norm” and “t-conorm” operators, see (Farahbod and Eftekhari, 2012), consider the aggregation of “pairs of fuzzy sets” for the same entity, for example the classical `AND` and `OR` operators in the fuzzy version, where a fuzzy set denotes a linguistic property. For example, if *CheapFlat* is the linguistic predicate that denotes if a flat belongs to the fuzzy set of “cheap flats”, while *LargeFlat* is the linguistic predicate that denotes if a flat belongs to the fuzzy set of “large flats”,

CheapFlat AND LargeFlat

denotes a complex linguistic condition that looks for those flats that are “cheap and large”. How to obtain the resulting membership degree? Many proposals for t-norm and t-conorm operators are available in the literature; the simplest one is to use the minimum membership degree¹, where for the `OR` operator (the corresponding t-conorm operator of the t-norm operator `AND`), the maximum membership degree is considered².

In an orthogonal way (studied in (Fosci and Psaila, 2023a)), it is possible to aggregate entities in groups (or in categories) $G_j = \{x_{j,1}, x_{j,2}, \dots\}$ of items $x_{j,i}$ that belong to the same group G_j because they share some common properties or are samples of the same category of items. Each $x_{j,i}$ singularly may belong to a fuzzy set A , thus it is provided with a membership $\mu_A(x_{j,i})$. Consequently, the set $\bar{A}$ of groups G_j can be seen as a partition of A : with this vision, the membership of a group G_j to $\bar{A}$ should be derived by somehow

¹ $\mu_{\text{CheapFlat AND LargeFlat}} = \min(\mu_{\text{CheapFlat}}, \mu_{\text{LargeFlat}})$

² $\mu_{\text{CheapFlat OR LargeFlat}} = \max(\mu_{\text{CheapFlat}}, \mu_{\text{LargeFlat}})$

aggregating memberships $\mu_A(x_{j,i})$, group by group. Alternatively, if items $x_{j,i}$ do not have a membership, their properties might have to be aggregated to obtain the final membership of the G_j group.

Popular fuzzy aggregators of this type are “Weighted aggregation” (see (Dombi and Jónás, 2022)) and “Ordered Weighted Aggregation” (OWA) (Yager, 1988; Li and Yen, 1995).

2.3 The Vector p -norm

When soft conditions are written by means of linguistic predicates, the basic interpretation of AND and OR as the minimum (respectively, maximum) membership degree to the fuzzy sets that correspond to the linguistic predicates is overly simplified. Furthermore, “preferences” for linguistic predicates can be conceived as their “importance”.

In the literature, the “Vector p -norm” was proposed as an elegant mathematical formalization to cope with this situation. The reader can refer to (Bordogna and Psaila, 2009). Hereafter, the general formulas are reported.

Given a list $P = (p_1, \dots, p_n)$ of linguistic predicates p_i , $\mu_i(x)$ denotes the membership degree of the item x to the fuzzy set associated with p_i . Consider the list $I = (i_1, \dots, i_n)$, where i_i is the “importance degree” for the predicate p_i .

The membership degree computed by the AND operator in the p -norm version is formulated as in Equation 1:

$$\mu_{\text{AND}}(x) = 1 - \sqrt[p]{\frac{\sum_{k=1}^n (i_k)^p \times (1 - \mu_k(x))^p}{\sum_{k=1}^n (i_k)^p}} \quad (1)$$

To understand the rationale, in the case of $n = 2$ (bi-dimensional case), all $i_k = 1$ (no preferences) and $p = 2$, the square root computes the Minkowski distance between the point (μ_1, μ_2) and the point $(1, 1)$. Consequently, the final membership degree is maximized when the Minkowski distance between the two above-mentioned points is minimized. In practice, when all $i_k = 1$, the points (μ_1, μ_2) that are equidistant from the point $(1, 1)$ have the same final membership degree (“iso-membership” line), because they are on the same circle that is centered in the point $(1, 1)$ (see, for example, the black solid line in Figure 1).

The OR operator is defined as in Equation 2:

$$\mu_{\text{OR}}(x) = \sqrt[p]{\frac{\sum_{k=1}^n (i_k)^p \times \mu_k(x)^p}{\sum_{k=1}^n (i_k)^p}} \quad (2)$$

Reasoning for the case $n = 2$, all $i_k = 1$ and $p = 2$, the final membership degree is the Minkowski distance between the point (μ_1, μ_2) and the point $(0, 0)$

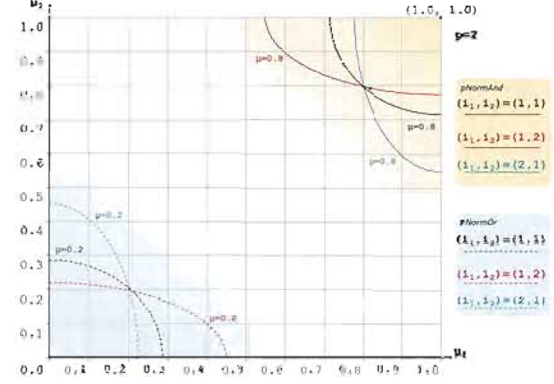


Figure 1: Iso-membership lines for the AND (solid lines) and the OR (dashed lines) defined as Vector p -norm, with different importance degrees.

(the final membership degree is maximized when the Minkowski distance from the point $(0,0)$ is maximized). Consequently, points on the same circle centered on the point $(0,0)$ have the same final membership degree (see, for example, the black dashed line in Figure 1).

Keeping $n = 2$ and varying the importance degrees i_k , circles become ellipsis. The reader can see this effect in Figure 1.

Looking at solid lines, the blue line is obtained with $(i_1, i_2) = (2, 1)$, i.e., μ_1 doubles the importance of μ_2 ; all the points on this line denotes pairs (μ_1, μ_2) giving the same final membership degree $\mu = 0.8$; clearly, a small variation of μ_1 must be compensated by a much greater variation of μ_2 , to obtain the same final membership degree. Similarly, the red solid line corresponds to the case $(i_1, i_2) = (1, 2)$, i.e., μ_1 has half the importance of μ_2 . Notice that the three solid lines determine the same final membership degree of $\mu = 0.8$; consequently, a point on the main diagonal has the same final membership, independently of the importance degrees: as an effect, the three solid lines depicted in Figure 1 cross on one single point, because they correspond to the same final membership degree.

Similarly, the same happens for the dashed lines in Figure 1, which correspond to the OR operator: the three dashed lines denotes the same final membership degree of $\mu = 0.2$.

Notice that, in the case $i_k = 1$, the limit for p going to infinite of the AND and OR operators are the classical definitions based on *min* and *max*, respectively. Conversely, when $p = 1$, AND and OR behave in the same way, so their semantics cannot be distinguished.

The reader can see that the logical operators defined on the basis of the Vector p -norm are a form of fuzzy aggregation: given a vector of membership

Listing 1. *J-CO-QL⁺*: fuzzy evaluator highCumulativeRain.

```

1. CREATE FUZZY EVALUATOR highCumulativeRain
PARAMETERS rainData TYPE ARRAY
FOR ALL rd IN rainData
AGGREGATE rd AS av
EVALUATE av
POLYLINE [ ( 0, 0.0), ( 50, 0.0), (100, 0.1),
            (200, 0.7), (300, 0.9), (400, 1.0) ];

```

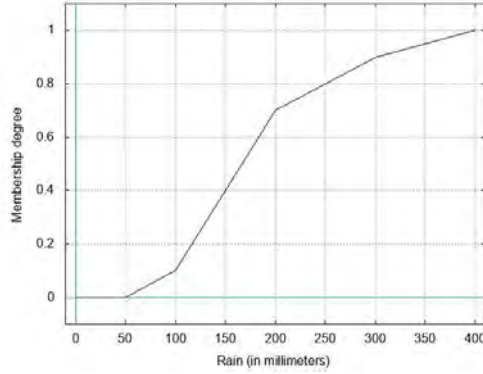


Figure 2: Membership function for the fuzzy evaluator highCumulativeRain.

degrees and a vector of importance degrees, they are aggregated to compute the final membership degree.

The above definitions rely on the “Minkowski distance”, one of the concepts of the “Minkowski geometry” (Thompson, 1996). These concepts are widely used in Mathematics (see, for example, (Craata and Malusa, 2007)) and in various applied sciences (Merigó and Gil-Lafuente, 2008),

3 FUZZY EVALUATORS IN THE J-CO FRAMEWORK

As far as linguistic capabilities are concerned, the novel contribution that is presented in this paper is the concept of “Fuzzy Evaluator”, recently introduced in *J-CO-QL⁺* (the query language of the *J-CO* Framework).

The novel statement `CREATE FUZZY EVALUATOR` unifies the two previously-existing statements `CREATE FUZZY OPERATOR` (Fosci and Psaila, 2021b; Fosci and Psaila, 2022a) and `CREATE FUZZY AGGREGATOR` (Fosci and Psaila, 2023a): indeed, a fuzzy evaluator can be used for computing membership degrees of a *JSON* document by moving from elementary properties, arrays of simple values, arrays of documents, pools of membership degrees, all together at the same time. This way, users can define libraries of highly complex fuzzy evaluators, enabling them to express sophisticated soft queries on collected *JSON* datasets.

Listing 2. *J-CO-QL⁺*: fuzzy evaluator owaRain.

```

2. CREATE FUZZY EVALUATOR owaRain
PARAMETERS rainData TYPE ARRAY
SORT rd IN rainData
BY rd TYPE NUMERIC ASC AS sRainData
FOR ALL srd IN sRainData
LOCALLY ( POS^2 - (POS-1)^2 )
/ (COUNT(sRainData)^2) AS w
AGGREGATE srd * w AS av
EVALUATE av
POLYLINE [(0.00, 0.0), (0.10, 0.0), (0.15, 0.7),
            (0.20, 0.8), (0.50, 0.9), (0.80, 1.0)];

```

A fuzzy evaluator is characterized by the following features.

- The fuzzy evaluator receives a list of “formal parameters”, which can be either simple values or documents or arrays; the actual values are used to evaluate the final membership degree.
- A “precondition” is a Boolean condition that is evaluated on the actual parameters, so as to ensure that the computation can be performed properly.
- A pool of “internal variables” can be derived, so as to perform preliminary computations.
- Internal array variables can be derived by sorting input arrays.
- A pool of “aggregated values” can be computed, by scanning arrays that are received as actual parameters; the resulting aggregated values become internal variables.
- A “resulting value” is computed, by evaluating an expression that can exploit both parameters and internal variables.
- If the resulting value is in the range $[0, 1]$, it can be returned as the “output membership degree”. If this is not the case, it becomes the x -axis value of a “membership function” that is specified as a polyline; the corresponding y -axis value (in the range $[0, 1]$) becomes the output membership degree.

The syntax of the statement for defining fuzzy evaluators will be discussed hereafter.

Simple Fuzzy Aggregation. Listing 1 reports the definition of a very simple fuzzy evaluator (named highCumulativeRain), whose goal is computing a cumulative sum of numerical values (millimeters of rain) within an array, so as to obtain a membership degree that denotes high cumulative rain.

The reader can see the clauses `PARAMETERS`, which defines the formal parameters, and `PRECONDITION`, which specifies the Boolean precondition. Specifically, the evaluator receives the array `rainData`, whose values are numbers denoting millimeters of rain.

The “resulting value” is computed by the `EVALUATE` clause; the expression in the clause contains only the internal variable `av`, which is computed

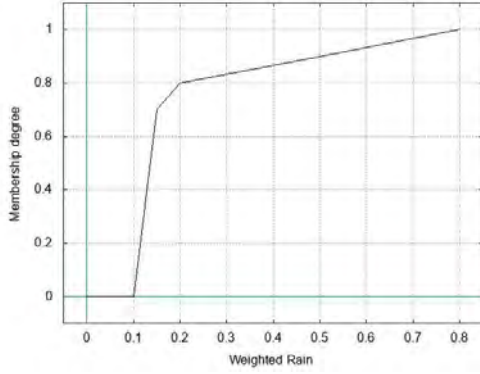


Figure 3: Membership function for the fuzzy evaluator owaRain.

by the previous clause `FOR ALL`. The clause `FOR ALL` is substantially straightforward: it scans the input array `rd`, summing its values into the internal variable `av` (see the sub-clause `AGGREGATE`). Thus, the resulting value is the sum of millimeters of rain.

The resulting value becomes the x -axis value of the membership function, which is defined as a polyline function by the clause `POLYLINE` (the function is depicted in Figure 2): the corresponding y -axis value is returned as output membership degree.

More Complex Fuzzy Aggregation. Listing 2 shows a more complex fuzzy evaluator, named `owaRain`. It performs the “Ordered Weighted Aggregation” (OWA, see (Yager, 1988)): a monotone function is used to determine the weights of each item in the array to aggregate, in such a way that the array is sorted beforehand.

A novel clause, named `SORT`, derives a novel internal array, named `sRainData`, which is obtained by sorting the input array `rainData` in ascending order.

The clause `FOR ALL` now contains an extra sub-clause `LOCALLY`, which defines a variable whose value is locally associated to each single scanned value. This way, a specific weight w is computed for each single value; thus, the aggregated value `av` is obtained by summing the products of each single value (denoted as `srd`) by its weight w .

Again, this aggregated value is the x -axis value of the membership function, which is depicted in Figure 3.

A Fuzzy Evaluator for Generic AND Based on the Vector p -norm. A fuzzy evaluator can be used to aggregate membership degrees, even in a non-trivial way, such as the operators `AND` and `OR` defined through the vector p -norm (see Equation 1 and Equation 2). Listing 3 reports the fuzzy evaluator `pNormAND`, which implements Equation 1.

The evaluator receives three formal parameters,

Listing 3. $J\text{-CO-QL}^+$: fuzzy evaluator `pNormAND`.

```
3. CREATE FUZZY EVALUATOR pNormAND
PARAMETERS
  p          TYPE NUMERIC,
  importances TYPE ARRAY,
  memberships TYPE ARRAY
PRECONDITION
  p >= 1
  AND
  COUNT(memberships) = COUNT(importances)
FOR ALL
  m IN memberships
  LOCALLY
    (1-m)^p AS x,
    importances[POS]^p AS i
  AGGREGATE
    i*x AS numerator,
    i AS denominator
EVALUATE
  1 - (numerator/denominator)^(1/p);
```

respectively the exponent (or root) p , the array of importance degrees (named `importances`) and the array of membership degrees (named `memberships`). The precondition must check that p is no less than 1 and that the two arrays have the same size.

The complexity of Equation 1 is managed by the `FOR ALL` clause, which is able to deal with several local variables and with several aggregated variables at the same time. Specifically, the x is the p -th power of the complement of a membership value m , while i is the p -th power of the corresponding importance degree. The `AGGREGATE` clause computes the sum of the product of i by x (named `numerator`, as well as the sum of i (named `denominator`).

This way, the clause `EVALUATE` can subtract to 1 the result of the division of `numerator` by `denominator`. Since the resulting number is in the range $[0, 1]$ and it is already the desired output membership degree, it is returned as it is (no polyline is specified).

The reader can notice that the fuzzy evaluator is still declarative and elegant. Furthermore, it is an example of how a fuzzy evaluator can be used to extend linguistic capabilities of $J\text{-CO-QL}^+$: novel aggregations of membership degrees can be easily defined by the user, to be used in soft conditions. Users can certainly create their own library of fuzzy evaluators, to be exploited when necessary.

The statement `CREATE FUZZY EVALUATOR` incorporates the former statements `CREATE FUZZY OPERATOR` and `CREATE FUZZY AGGREGATOR`. Indeed, a former fuzzy operator (Fosci and Psaila, 2021b) encompassed only the clauses `PARAMETERS`, `PRECONDITION`, `EVALUATE`, and `POLYLINE`; furthermore, arrays were not allowed as parameters. In contrast, a former fuzzy aggregator (Fosci and Psaila, 2023a) provided the same clauses that are presented in this paper, but only one clause `SORT` was allowed after the clause `PRECONDITION`, as well as only one clause `FOR ALL` was possible. In contrast, a fuzzy evaluator can be defined as a free sequence of clauses `SORT`, `FOR ALL`, and `DERIVE` (which is not shown in this paper, whose goal is to derive novel internal variables).

```

{
  "city"      : "Osio Sopra",
  "province"  : "BG",
  "sensorId"  : 5856,
  "latitude"  : 45.6338645090807,
  "longitude" : 9.55608425579086,
  "rainData" : [
    {
      "timestamp" : "12/05/2023 21:00:00",
      "value"      : 2.2
    },
    ..., // other rain data
    {
      "timestamp" : "24/05/2023 17:00:00",
      "value"      : 47.8
    }
  ]
}

```

Figure 4: Example of document in the starting collection MeasuredRain.

4 CASE STUDY AND QUERY

This section provides the second contribution of the paper, i.e., showing how to exploit preferences in soft conditions by using a generic fuzzy concept, such as user-defined fuzzy evaluators.

To do that, the section relies on the same case study that was introduced in (Fosci and Psaila, 2023a).

4.1 Case Study

In (Fosci and Psaila, 2023a), a dataset of rain data was collected on May 2023. The dataset was made available on an institutional Open-Data portal³: it describes measurements of rain performed by rain sensors, located in the Lombardy region of Italy, on May 2023.

Figure 4 reports one of the 207 documents in the collection MeasuredRain, that was obtained by preprocessing source datasets (see (Fosci and Psaila, 2023a)). Each single document describes a sensor and the measurements that it has taken: the field `sensorId` reports the identifier of the sensor; the fields `city`, `province`, `longitude` and `latitude` briefly represent the position of the sensor. Notice the array field `rainData` (highlighted by a blue box), which contains simple documents describing measurements of rain (the field `value` reports millimeters of rain, and the field `timestamp` reports when the measurement was taken). The average number of documents in the array field `rainData` is above 4400.

The application goal was to discover those sensors that, in the considered time-window, measured high peaks of rain possibly with significant cumulative rain. Specifically, the addressed problem was formalized as reported in Problem 1 (taken from (Fosci

and Psaila, 2023a)).

Problem 1. *Given the measurements of rain collected in the MeasuredRain collection, find out those sensors that measured high peaks of rain, possibly with significant cumulative rain.*

The analysis was made in fuzzy terms: the membership of each sensor to two fuzzy sets named, respectively, `PeaksOfRain`, denoting those sensors that measured peaks of rain, and `SignificantRain`, denoting those sensors that measured a significant amount of rain in the monitored period, were evaluated. Then, the memberships to the fuzzy sets `PeaksOfRain` and `SignificantRain` were combined together by means of a weighted fuzzy aggregator, whose weights were predetermined, to discover those sensors that reported interesting data.

In this work, the same dataset is exploited, as well as the same problem is addressed; however, instead of using a weighted fuzzy aggregator, the adoption of the generalized AND operator based on the vector p -norm is explored; consequently, the fuzzy evaluator `pNormAND` (reported in Listing 3) will be used, so as to express preferences between memberships to fuzzy sets (i.e., preferences between linguistic predicates).

The problem of using the operator AND based on the vector p -norm with preferences is to comprehend how to choose p and i_k (the importance weights, or preferences); indeed, the fuzzy evaluator `pNormAND` is parametric as far as they are concerned. Consequently, it is worth considering several different configurations, so as to study the behavior of the operator with different configurations of its parameters.

Important Disclaimer. The authors are not experts in meteorology or weather sensors. This case study was defined solely to showcase the capabilities of the *J-CO* Framework and should not be interpreted as scientifically accurate or professionally verified, as far as the meteorological side is concerned.

4.2 Query

Problem 1 can be practically interpreted as described by Query 1, enabling the writing of a *J-CO-QL*⁺ script.

Query 1. *Consider the universe of sensors.*

The fuzzy set that is named PeaksOfRain denotes those sensors that measured peaks of rain; this corresponds to the linguistic predicate P_1 : “is x a sensor that measured peaks of rain?”.

The second fuzzy set is named SignificantRain and denotes sensors that measured a significant amount of rain in the

³Open-Data portal of Regione Lombardia:
<https://www.dati.lombardia.it/>

Listing 4. *J-CO-QL⁺*: retrieval and soft querying.

```

4. USE DB webist2023
   ON SERVER MongoDB 'http://127.0.0.1:27017';

5. GET COLLECTION MeasuredRain@webist2023;

6. FILTER
   CASE WHERE WITH .rainData
   GENERATE
   CHECK FOR
     FUZZY SET SignificantRain
       USING highCumulativeRain(EXTRACT_ARRAY(
         .value FROM ARRAY .rainData)),
     FUZZY SET PeaksOfRain
       USING owaRain(EXTRACT_ARRAY(
         .value FROM ARRAY .rainData));

7. JOIN OF COLLECTIONS temporary AS t,
   Configurations@webist2023 AS c
   SET FUZZY SETS KEEP LEFT
   CASE WHERE
     KNOWN FUZZY SETS SignificantRain, PeaksOfRain
   GENERATE
   CHECK FOR
     FUZZY SET Wanted
       USING pNormAND (.c.p, .c.importances,
         MEMBERSHIP_ARRAY( [ PeaksOfRain,
           SignificantRain ] ) )
   ALPHACUT 0.80 ON Wanted
   BUILD {
     .city      : .t.city,
     .province  : .t.province,
     .sensorId  : .t.sensorId,
     .dateStart : MIN_ARRAY(.t.rainData, STRING,
       .timestamp),
     .dateEnd   : MAX_ARRAY(.t.rainData, STRING,
       .timestamp),
     .label     : .c.label,
     .ranking   : MEMBERSHIP_TO (Wanted) }
   DEFUZZIFY;

8. SAVE AS Results@webist2023;

```

monitored period; it corresponds to the linguistic predicate P_2 : “is x a sensor that measured significant rain?”.

A third fuzzy set that is named *Wanted* denotes those sensors that are in the fuzzy set *PeaksOfRain* “and possibly” in the fuzzy set *SignificantRain*, selecting only sensors whose membership to the fuzzy set *Wanted* is no less than 0.80. Exploit many different configurations of parameters for the evaluator *pNormAND*, so as to find the configuration that best meets the request “ P_1 and possibly P_2 ”.

To achieve the requirement to use several configurations for parameters of the fuzzy evaluator *pNormAND*, a collection of documents, named *Configurations*, is created. The structure of *JSON* documents in this collection will be presented later. Hereafter, the *J-CO-QL⁺* script is presented.

Listing 4 actually performs Query 1; notice that the first line number is 4, because the three definitions of fuzzy evaluators reported in previous listings are part of the query itself. Hereafter, the query is presented in details.

Acquiring Data. Line 4 of Listing 4 specifies the database to connect with. Specifically, the database *webist2023* is managed by *MongoDB*.

On Line 5 of Listing 4, the instruction *GET COLLECTION* retrieves the content of the collection *MeasuredRain* from the database *webist2023*; this collection becomes the new temporary collection of the process (see Section 2.1). Figure 4 shows an example of document in the collection.

Document Fuzzification with Fuzzy Evaluators. The instruction *FILTER*, on Line 6 of Listing 4, evaluates the belonging of each document in the current temporary collection to the fuzzy sets *PeaksOfRain* and *SignificantRain*. Hereafter, it is explained in details.

- The condition *CASE WHERE* selects (in a Boolean way) those documents that have the field *rainData*. The remainder of the instruction will work only on these documents; any other documents are discarded.
- The block *GENERATE* actually generates the output documents, by possibly performing several actions, including evaluating memberships to fuzzy sets through the clause *CHECK FOR*.
- The clause *CHECK FOR* may contain different branches *FUZZY SET*, one for each fuzzy set whose membership has to be evaluated. There are two branches *FUZZY SET* on line 6.
- The first branch *FUZZY SET* evaluates the membership to the fuzzy set *SignificantRain*. To perform this task, the soft condition *USING* (which actually provides the membership to the fuzzy set under consideration) exploits the fuzzy evaluator *highCumulativeRain* defined in Listing 1. To call the fuzzy evaluator, an array of numbers must be provided as actual parameter: since the array field *rainData* contains nested documents, the special built-in function *EXTRACT_ARRAY* creates a novel array of numbers by projecting the array field *rainData* on the inner (numerical) field value. The membership provided by the fuzzy evaluator *highCumulativeRain* becomes the membership degree of the current *JSON* document to the fuzzy set *SignificantRain*. The current document is then fuzzified by adding the special root-level field *~fuzzysets*, so as to report the computed membership (see Section 2.1).
- The second branch *FUZZY SET* evaluates the membership of the current document to the fuzzy set *PeaksOfRain*, through the fuzzy evaluator *OwaRain* (reported in Listing 2). Apart from the fact that the evaluator *OwaRain*

```

{
  "city"      : "Osio Sopra",
  "province"  : "BG",
  "sensorId"  : 5856,
  "latitude"  : 45.6338645090807,
  "longitude" : 9.55608425579086,
  "rainData"  : [...],
  "~fuzzysets" : {
    "PeaksOfRain" : 0.930466293345489,
    "SignificantRain" : 0.75399998486042
  }
}

```

Figure 5: Example of document generated by the instruction FILTER on Line 6.

performs an OWA aggregation (instead of a cumulative aggregation) the branch behaves similarly to the previous one: the evaluator is called by passing the array of values obtained by projecting the array `rainData` on the inner field value by means of the special built-in function `EXTRACT_ARRAY`; then, the computed membership becomes the degree to the fuzzy set `PeaksOfRain`.

Definitely, the goal of the fuzzy set `PeaksOfRain` is to denote (through the membership) those sensors that measured a peak of rain; the OWA approach allows for doing that, because the items with the highest values (typically, two or three) gain the greatest weights; consequently, many days of rain with few millimeters of rain do not contribute significantly to the aggregated membership: in contrast, two days with heavy rain on a mass of dry days strongly contribute to achieve high membership.

A second field is added into the field `~fuzzysets`, denoting the membership degree to the novel fuzzy set.

The fuzzified documents of the collection `MeasuredRain`, i.e., enriched with membership degrees to fuzzy sets, constitutes now the current temporary collection. Figure 5 shows an example of document generated by the instruction `FILTER`: notice the special root-level field `~fuzzysets` (highlighted by a yellow box), with the memberships to the fuzzysets `PeaksOfRain` and `SignificantRain`.

Soft Querying with Fuzzy Evaluators. The instruction `JOIN OF COLLECTIONS`, on Line 7 of Listing 4, actually performs Query 1. The goal of the instruction is to apply the fuzzy evaluator `pNormAND` to the fuzzy sets `SignificantRain` and `PeaksOfRain` with different configurations of parameters. To achieve this purpose, a second collection of documents, named `Configurations`, is considered. Figure 6 shows an example of document in the collection `Configurations`: considering Equation 1, in each document the field `p` reports the value of the parameter p and the array field `importances` holds the values of the parameters i_k ; the field `label` iden-

```

{
  "label"      : "p:7-i1:3-i2:1",
  "p"          : 7,
  "importances" : [3, 1]
}

```

Figure 6: Example of document in the collection Configurations.

```

{
  "c" : {
    "label"      : "p:7-i1:3-i2:1",
    "p"          : 7,
    "importances" : [3, 1]
  },
  "t" : {
    "city"      : "Osio Sopra",
    "province"  : "BG",
    "sensorId"  : 5856,
    "latitude"  : 45.6338645090807,
    "longitude" : 9.55608425579086,
    "rainData"  : [...],
    "~fuzzysets" : {
      "PeaksOfRain" : 0.930466293345489,
      "SignificantRain" : 0.75399998486042
    }
  },
  "~fuzzysets" : {
    "PeaksOfRain" : 0.930466293345489,
    "SignificantRain" : 0.75399998486042
  }
}

```

Figure 7: Example of temporary document generated by the instruction `JOIN OF COLLECTIONS` on Line 7 after the clause `SET FUZZY SETS`.

tifies each configuration of parameters with a string that briefly resumes it.

Hereafter, the instruction `JOIN OF COLLECTIONS` is explained in details.

- For each couple (t, c) , where t is a document in the temporary collection, and c is a document in the collection `Configurations` in the `webist2023` database, the instruction `JOIN OF COLLECTIONS` generates a novel document with two root-level fields named `t` and `c` holding, respectively, the t document and the c document.
- The clause `SET FUZZY SETS` fuzzifies the novel document and the option `KEEP LEFT` fills the field `~fuzzysets` with the fuzzy sets in the t document. Figure 7 shows an example of temporary document generated by the instruction `JOIN OF COLLECTIONS` after the clause `SET FUZZY SETS`: notice the field `t` holding the document t (highlighted by a blue-box), the field `c` holding the document c (highlighted by a red-box), and the field `~fuzzysets` (highlighted by a yellow-box) with the memberships from the document t .
- The following condition `CASE WHERE` selects those documents belonging to the fuzzy sets `PeaksOfRain` and `SignificantRain`.
- The block `GENERATE` actually generates, filters and restructures the output documents.
 - The clause `CHECK FOR` contains one branch

```

{
  "city"      : "Osio Sopra",
  "province"  : "BG",
  "sensorId"  : 5856,
  "label"     : "p:7-i1:3-i2:1",
  "dateStart" : "01/05/2023 00:00:00",
  "dateEnd"   : "31/05/2023 23:00:00",
  "ranking"   : 0.914731773101206
}

```

Figure 8: Example of document in the collection Results.

FUZZY SET to evaluate the degree of the current *JSON* document to the fuzzy set Wanted. The goal is to exploit the fuzzy evaluator `pNormAND` in order to aggregate the fuzzy sets `PeaksOfRain` and `SignificantRain` with the configuration of parameters provided by the document in `c` the `Configurations` collection. Thus, in the soft condition USING, the fuzzy evaluator `pNormAND` is called passing as actual parameters the field `c.p` for the parameter p , the array field `c.importances` for the parameters i_k , and, by means of the $J\text{-CO-QL}^+$ built-in function `MEMBERSHIP_ARRAY`, the array of memberships for the parameters μ_k .

The returned value of the fuzzy evaluator `pNormAND` is the membership degree to the fuzzy set Wanted. Thus, the fuzzy set Wanted is added to the special field `~fuzzysets`.

- The clause `ALPHACUT` discards those *JSON* documents whose membership degree to the fuzzy set Wanted is less than 0.80. In this way, only documents that describe sensors that actually measured peaks of rain and possibly significant rain during the monitored period (or very close to this situation) are selected.
- The block `BUILD` flattens and restructures the output documents. In particular, the $J\text{-CO-QL}^+$ built-in functions `MIN_ARRAY` and `MAX_ARRAY` are used to extract, from the array `rainData`, respectively, the starting date (field `dateStart`) and the final date (field `dateEnd`) of the time-window of the considered rain data. Notice also the use of the $J\text{-CO-QL}^+$ built-in function `MEMBERSHIP_TO` to assign to the field `ranking` the value of the membership degree to the fuzzy set Wanted.
- The final option `DEFUZZIFY` actually “defuzzifies” documents by removing the special field `~fuzzysets`, so as documents become again classical crisp *JSON* documents.

Figure 8 reports an example of document generated by the instruction `JOIN OF COLLECTIONS`.

Saving the Results. The resulting collection, which contains documents describing sensors of interest on the basis of Problem 1, is finally saved, by Line 8

Table 1: List of values of the parameters in the importances field of documents in the collection Configurations.

i1	1	2	1	3	1	3	2
i2	1	1	2	1	3	2	3

Table 2: Table of the values provided by the fuzzy evaluator `pNormAND` with different configurations of parameters.

		[i1, i2]						
		[1, 1]	[2, 1]	[1, 2]	[3, 1]	[1, 3]	[3, 2]	[2, 3]
p	1	0.84	0.87	0.81	0.89	0.80	0.86	0.82
	2	0.82	0.87	0.78	0.90	0.77	0.85	0.79
	3	0.80	0.88	0.76	0.91	0.76	0.85	0.77
	4	0.79	0.88	0.76	0.91	0.75	0.84	0.76
	5	0.79	0.88	0.76	0.91	0.75	0.84	0.76
	6	0.78	0.88	0.75	0.91	0.75	0.84	0.76
	7	0.78	0.88	0.75	0.91	0.75	0.84	0.76

of Listing 4, with the name `Results` in the database `webist2023`.

5 DISCUSSION

The $J\text{-CO-QL}^+$ script reported in Listing 4 exploits the *JSON* documents reported in the collection `Configurations`, in order to apply the fuzzy evaluator `pNormAND` with different configurations for parameters p , i_1 , and i_2 ; the goal is to study the behavior of the Vector p -norm on the field. 49 different configurations were considered, which are hereafter presented.

- As far as p is concerned, seven integer values, from 1 to 7, were used.
- As far as the pair of importance degrees $[i_1, i_2]$ is concerned, 7 different combinations were considered, for each distinct value of p . These combinations are reported in Table 1; for the sake of the completeness, both cases in which $i_1 > i_2$ (which gives more importance to μ_1 than to μ_2) and in which $i_1 < i_2$ (which gives more importance to μ_2 than to μ_1) are considered.

The sample document, presented in Figure 4 and Figure 8, was exploited to understand how the final membership degree (in the field `ranking`) varies, provided that $\mu_1 = 0.930466293345489$ and $\mu_2 = 0.75399998486042$ (see Figure 7). Table 2 reports the values of the field `ranking` when the parameters p (in rows) and $[i_1, i_2]$ (in columns) vary. The same data are depicted in Figure 9.

Each line of Figure 9 corresponds to a configuration for $[i_1, i_2]$ (“iso-configuration” line), connecting the results that were obtained for varying values of p

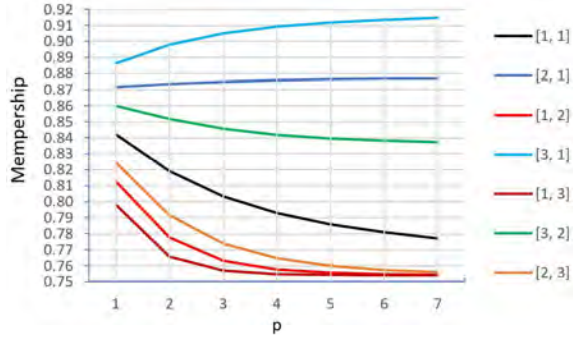


Figure 9: Iso-configuration lines. Graph of the values provided by the fuzzy evaluator $p\text{NormAND}$ with different configurations of parameters.

(on the x-axis). Figure 9 is crucial, as far as the following discussion is concerned: indeed, it allows the reader to visualize the different behaviors of the Vector p -norm, when the parameters p , i_1 , and i_2 vary. The goal is to improve the consciousness of users for properly exploit such an aggregation method.

- The black line corresponds to the case of equal importance degrees $i_1 = i_2 = 1$. Notice how, with increasing values of p , the resulting membership degree tends to the minimum value between μ_1 and μ_2 .
- The orange, light red, and red lines correspond to the configurations in which $i_1 < i_2$. These lines are below the black line, meaning that the lower membership degree μ_2 gains increasing importance in the final membership degree. In other words, the final membership degree is dominated by $\mu_2 < \mu_1$; thus, the corresponding lines must be below the black line.
- The light blue, blue and green lines correspond to those configurations in which $i_1 > i_2$. Notice that the configuration $(i_1, i_2) = (3, 1)$ (light blue line), behaves as expected.

In contrast, the configuration $(i_1, i_2) = (3, 2)$ (green line) lies below the blue line (configuration $(i_1, i_2) = (2, 1)$): indeed, the difference $\Delta = i_1 - i_2 = 1$ in both cases; however, Δ is only 33% of $i_1 = 3$ (green line), while Δ is 50% of $i_1 = 2$ (blue line). This shows that the important feature that determines the behavior of the Vector p -norm is the ratio of the distance between i_1 and i_2 on $\max(i_1, i_2)$, to manage the importance of a predicate with respect to another. The blue line has a ratio of 50%, which is greater than the ratio of 33% for the green line; consequently, the blue line is above the green line, because the strength of $\mu - 1$ is higher than in the green line (remember that $\mu_1 > \mu_2$).

Table 3: Execution times of the script in Listings 1, 2, 3, and 4 applied on the input collections.

#	Instruction	Execution Time (ms)	%
1	CREATE FUZZY EVALUATOR	1	0.00%
2	CREATE FUZZY EVALUATOR	0	0.00%
3	CREATE FUZZY EVALUATOR	0	0.00%
4	USE DB	2	0.00%
5	GET COLLECTION	2,723	1.86%
6	FILTER	140,677	95.96%
7	JOIN OF COLLECTIONS	3,126	2.13%
8	SAVE AS	65	0.04%
Total Time (ms)		146,594	

- Finally, notice that the light blue and blue lines increase, while the other decreases. The explanation is the following: with $p \rightarrow \infty$, the final membership degree becomes as in Equation 3.

$$\mu_{\text{AND}}(x) = \min_{k=1}^n \left(1 - \frac{i_k}{\max(i_1, \dots, i_n)} (1 - \mu_k(x)) \right) \quad (3)$$

Thus, an iso-configuration line is ascending (respectively, descending) when its membership degree for $p = 1$ is less than (respectively, greater than) its limit as $p \rightarrow \infty$.

To conclude, Table 3 presents data related to the performance (in terms of execution time) of the script in Listings 1, 2, 3, and 4. The experiment was conducted on a PC powered by a Processor Intel quad-Core i7-8550-U, running at 1.80 GHz, equipped with 16 GB RAM and 250 GB Solid State Drive. For each instruction, the execution time (in milliseconds) and the percentage of the total time are reported.

The input collections, *MeasuredRain* and *Configurations*, consisted of 207 and 49 documents, respectively, while the resulting collection, named as *Results*, comprised 57 documents.

The total execution time was less than 147 seconds, with nearly 96% of the time (140 seconds) consumed by the instruction *FILTER*, primarily due to the sorting operation on the array parameter *rainData*, specified by the clause *SORT* in the fuzzy evaluator *owaRain*.

Although performance analysis is an important consideration, it is beyond the scope of this paper. The primary objective here is to demonstrate the *J-CO* Framework as a practical and efficient tool for data analysis under soft conditions. For readers that are interested in an in-depth performance evaluation, the studies in (Psaila and Fosci, 2021) and (Fosci and Psaila, 2023c) provide a thorough investigation of this topic.

6 CONCLUSIONS

This paper showed how the novel concept of “fuzzy evaluator”, that was recently introduced in *J-CO-QL⁺*, the query language of the *J-CO* Framework, effectively allows users to exploit complex definitions for the fuzzy interpretation of the AND (and OR) operator. The concept of fuzzy evaluator unifies and extends the former concepts of “fuzzy operator” and “fuzzy aggregator”, formerly used to perform tasks of *Soft Web Intelligence* on JSON datasets acquired from Web sources (see (Fosci and Psaila, 2022b; Fosci and Psaila, 2023b; Fosci and Psaila, 2023a)).

The main contribution of the paper is to show how an interpretation of the AND (and OR) operator on the basis of the Vector p -norm, so as to express the concept of “ P_1 and possibly P_2 ”, which relies on the idea of “linguistic predicates with unequal importance”. Indeed, with the concept of fuzzy evaluator, *J-CO-QL⁺* has gained a further flexibility, allowing users to define their own (non-trivial) evaluators that capture complex and personalized semantics. At the end, since using the AND operator defined as a Vector p -norm is not trivial — specifically, choosing the right configuration for its parameters — its behavior is studied in Section 5.

Due to the lack of space, the behavioral analysis of the AND operator defined as a Vector p -norm is limited. As future work, we plan to continue this study, including both the AND and the OR operators. Indeed, the *J-CO* Framework (with its query language) is a very powerful tool for performing such kind of studies. Furthermore, it is possible to envision the creation of a library of fuzzy evaluators to provide complex and varied interpretations of the AND and the OR operators.

The *J-CO* Framework is available on a Github page⁴.

ACKNOWLEDGEMENTS

This study was funded by the European Union - *NextGenerationEU*, in the framework of the *GRINS - Growing Resilient, INclusive and Sustainable project (GRINS PE00000018 – CUP F83C22001720001)*. The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the European Union, nor can the European Union be held responsible for them.

⁴Github repository of the *J-CO* Framework:
<https://github.com/JcoProjectTeam/JcoProjectPage>

REFERENCES

- Bordogna, G. and Psaila, G. (2008). Modeling soft conditions with unequal importance in fuzzy databases based on the vector p -norm. *Proceedings of the IPMU, Malaga*.
- Bordogna, G. and Psaila, G. (2009). Soft aggregation in flexible databases querying based on the vector p -norm. *I. J. of Uncertainty, Fuzziness and Knowledge-Based Systems*, 17(supp01):25–40.
- Crasta, G. and Malusa, A. (2007). The distance function from the boundary in a minkowski space. *Transactions of the American Mathematical Society*, 359(12):5725–5759.
- Dombi, J. and Jónás, T. (2022). Weighted aggregation systems and an expectation level-based weighting and scoring procedure. *European Journal of Operational Research*, 299(2):580–588.
- Farahbod, F. and Eftekhari, M. (2012). Comparison of different t -norm operators in classification problems. *arXiv preprint arXiv:1208.1955*.
- Fosci, P. and Psaila, G. (2021a). J-co, a framework for fuzzy querying collections of json documents. In *Flexible Query Answering Systems: 14th International Conference, FQAS 2021, Bratislava, Slovakia, September 19–24, 2021, Proceedings 14*, pages 142–153. Springer International Publishing.
- Fosci, P. and Psaila, G. (2021b). Towards flexible retrieval, integration and analysis of json data sets through fuzzy sets: a case study. *Information*, 12(7):258.
- Fosci, P. and Psaila, G. (2022a). Soft integration of geo-tagged data sets in j-co-ql+. *ISPRS International Journal of Geo-Information*, 11(9):484.
- Fosci, P. and Psaila, G. (2022b). Towards soft web intelligence by collecting and processing json data sets from web sources. In *Proceedings of the 18th I. C. on Web Inf. Systems and Technologies*.
- Fosci, P. and Psaila, G. (2023a). Enhancing soft web intelligence with user-defined fuzzy aggregators. In *WEBIST 2023*, pages 258–267.
- Fosci, P. and Psaila, G. (2023b). Fuzzy aggregators in practice: Meta-model and implementation. In *International Conference on Soft Computing Models in Industrial and Environmental Applications*, pages 56–68. Springer Nature Switzerland Cham.
- Fosci, P. and Psaila, G. (2023c). Soft querying powered by user-defined functions in j-co-ql+. *Neurocomputing*, 546:126311.
- Li, H. and Yen, V. C. (1995). *Fuzzy sets and fuzzy decision-making*. CRC press.
- Merigó, J. M. and Gil-Lafuente, A. M. (2008). Using the owa operator in the minkowski distance. *International Journal of Economics and Management Engineering*, 2(9):1032–1040.
- Psaila, G. and Fosci, P. (2021). J-co: A platform-independent framework for managing geo-referenced json data sets. *Electronics*, 10(5):621.
- Thompson, A. C. (1996). *Minkowski geometry*. Cambridge University Press.

- Yager, R. R. (1988). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on systems, Man, and Cybernetics*, 18(1):183–190.
- Yao, Y., Zhong, N., Liu, J., and Ohsuga, S. (2001). Web intelligence (wi) research challenges and trends in the new information age. In *Asia-Pac. C. on Web Intelligence*, pages 1–17. Springer.
- Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, 8(3):338–353.