Enhancing Soft Web Intelligence with User-Defined Fuzzy Aggregators

Paolo Fosci[a] and Giuseppe Psaila[b]
University of Bergamo - DIGIP, Viale Marconi 5, 24044 Dalmine (BG), Italy

Keywords: Soft Web Intelligence, User-Defined Fuzzy Aggregators, Novel Constructs in J-CO-QL+, Soft Querying on JSON Data Sets.

Abstract: In our previous work, we proposed Soft Web Intelligence as the interpretation of the general notion of Web Intelligence in the current technological panorama, in such a way *JSON* data sets are acquired from the Internet, stored within *JSON* document stores and then processed and queried by means of soft computing and soft querying methods. Specific extensions to the *J-CO* Framework and to its query language (named *J-CO-QL*$^+$) made possible to practically implement the concept.
However, any "data intelligence" activity does not exclude aggregating data, but *J-CO-QL*$^+$ did not provide statements for defining "user-defined fuzzy aggregators". In this paper, we present the novel constructs introduced into *J-CO-QL*$^+$ to allow users to define and use their own fuzzy aggregators, so as to evaluate membership degrees to fuzzy sets moving from array fields within processed *JSON* documents. This way, complex soft queries are enabled, so as to enhance Soft Web Intelligence.

1 INTRODUCTION

Two decades ago, Web Intelligence was proposed in (Yao et al., 2001) as an approach to exploit the large amount of data and information that is possible to obtain from the World-Wide Web; the foreseen key technology was Artificial Intelligence (AI), because it was recognized that the semi-structured or totally unstructured form of information that is published on the Web made classical approaches to Business Intelligence substantially unsuitable.

Two decades later, the format that has become very popular is *JSON* (JavaScript Object Notation), due to its syntactic simplicity and its ease to be processed in programming languages. This success has been accompanied by the advent of *JSON* document stores, i.e., DBMSs (Database Management Systems) that natively store and query *JSON* data sets. Consequently, data scientists and data engineers very often deal with Web Intelligence scopes in which they have to gather, integrate, query and publish *JSON* data sets.

In (Fosci and Psaila, 2022b), we envisioned the notion of "Soft Web Intelligence": soft computing and soft querying (based on Fuzzy-Set Theory) can actually provide the tools to perform Web Intelligence tasks that must process *JSON* data sets coming from web sources. Fuzzy-Set Theory and Fuzzy Logic are indeed tools that belong to the AI world, consequently we conceived the idea of "Soft Web Intelligence" as a natural evolution of Web Intelligence.

The *J-CO* Framework is a pool of software tools that is under development at the University of Bergamo (Italy); its goal is to provide analysts and data engineers with sophisticated capabilities to gather, integrate and query *JSON* data sets; its query language, named *J-CO-QL*$^+$, is undergoing a continued evolution with the addition of novel constructs: specifically, we are currently introducing constructs for further extending its capabilities for performing soft querying on *JSON* data sets. In order to achieve a practical support to our vision, in (Fosci and Psaila, 2022b) we introduced extensions to the *J-CO* Framework specifically designed to practically realize *Soft Web Intelligence*; furthermore, through a practical case study, we showed that they are effective.

A typical computational activity that is performed in Business Intelligence is "aggregating measures of facts", so as to provide an aggregated view of events described by the analyzed data. It is reasonable to guess that aggregating data should be a typical task to do in *Soft Web Intelligence*, but in our previous works on *J-CO-QL*$^+$ aggregation was not considered, because times were not mature. Now, it is the time to introduce aggregators in *J-CO-QL*$^+$, to further extend

[a] https://orcid.org/0000-0001-9050-7873
[b] https://orcid.org/0000-0002-9228-560X

Fosci, P. and Psaila, G.
Enhancing Soft Web Intelligence with User-Defined Fuzzy Aggregators.
DOI: 10.5220/0012179100003584
In *Proceedings of the 19th International Conference on Web Information Systems and Technologies (WEBIST 2023)*, pages 258-267
ISBN: 978-989-758-672-9; ISSN: 2184-3252

the support to *Soft Web Intelligence*; specifically, we consider "fuzzy aggregators", which can be used to rank documents by aggregating either values in array fields or membership degrees to multiple fuzzy sets.

In this paper, we propose a novel construct, named `CREATE FUZZY AGGREGATOR`, that we added to *J-CO-QL*$^+$ so as to define "user-defined fuzzy aggregators"; its clauses drive users through the definition of a fuzzy aggregator, giving them the intuition of the semantic model. Through a practical case study, we will show how to exploit fuzzy aggregators in a scope of *Soft Web Intelligence*.

The paper is organized as follows. Section 2 presents the background of our work. Section 3 introduces the vision of *Soft Web Intelligence* and presents the main features of the *J-CO* Framework. Section 4 introduces the novel construct that we added to *J-CO-QL*$^+$ to declare user-defined fuzzy aggregators, together with the semantic model. Section 5 addresses a practical case study by exploiting user-defined fuzzy aggregators. Finally, Section 6 draws conclusions and future work.

2 RELATED WORK

This paper connects two areas that usually are not considered together, i.e., Web Intelligence and fuzzy logic. This is not the first attempt we make, but now we consider fuzzy aggregators (that were not considered in our previous work (Fosci and Psaila, 2022b)).

Specifically, Web Intelligence was introduced in (Yao et al., 2001) to obtain useful information from Web content. The complexity of Web content suggested to rely on Artificial Intelligence (AI) to get rid of this complexity. But what is AI? Techniques for "Data Mining" are certainly considered as AI techniques and are exploited in Web Intelligence (Han and Chang, 2002), as well as neural networks are nowadays perceived as "the AI".

Nonetheless, in the literature, many papers tries to give a more specific yet wider interpretation of Web Intelligence, such as "Computational Web Intelligence" (Zhang and Lin, 2002), i.e., the adoption of "Computational Intelligence" in Web Intelligence, as well as "Brain Informatics" (Zhong et al., 2006), i.e., fostering Web Intelligence through techniques that come out from the study of the human brain.

Fuzzy Logic and Soft Computing belong to AI too: indeed, its capability of approximate reasoning based on "linguistic predicates" provides a significant contribution towards AI. Consequently, it is straightforward that fuzzy logic can be exploited for Web Intelligence. Zadeh, the creator of Fuzzy-Set Theory (Zadeh, 1965), had this vision clearly in his mind: indeed, in (Zadeh, 2004a; Zadeh, 2004b), he showed that soft computing could play an important role.
Remember that a Fuzzy Set A in a universe U is a mapping for each $x \in U$, $A : x \rightarrow [0,1]$, also denoted as $\mu_A : x \rightarrow [0,1]$. The co-domain is the set of "membership degrees" (or simply "memberships", for brevity in the following): each item x belongs to A with a degree; when the degree is $0 < \mu_A(x) < 1$, x belongs to A only partially; obviously, $\mu_A(x) = 1$ denotes full membership of x to A, while $\mu_A(x) = 0$ means hat x does not belong at all to A.

However, looking for papers about fuzzy logic and soft computing in Web Intelligence, very few works can be found. The paper (Kacprzyk and Zadrożny, 2010) exploits soft computing in a group decision-making system to express preferences, but Web Intelligence activities were not supported by soft computing. The paper (Poli, 2015) uses `FUZZYALGOL`, a fuzzy procedural programming language (Reddy, 2010), for soft querying Web sources.

Consequently, this is why in (Fosci and Psaila, 2022b) we proposed the concept of *Soft Web Intelligence*, trying to give a modern interpretation of the concept of Web Intelligence on the basis of the current technological panorama; in Section 3.1, we define the concept in a precise way.

Nevertheless, Data Intelligence activities (Alahakoon and Yu, 2015) ask for aggregation; consequently, *Soft Web Intelligence* asks for "fuzzy aggregations" (since it relies on fuzzy sets). A plethora of proposals for fuzzy aggregators can be found in the literature. Many of them, such as "t-norm" and "t-conorm" operators, see (Farahbod and Eftekhari, 2012), consider the aggregation of "pairs of items", for example the classical `AND` and `OR` operators in the fuzzy version. In this paper, we are focused on groups (or 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 $\overline{A}$ of groups G_j can be seen as a partition of A: with this vision, the membership of a group G_j to $\overline{A}$ should be derived by somehow aggregating memberships $\mu_A(x_{j,i})$, group by group. Alternatively, if $x_{j,i}$ has not a membership, its values 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).

3 SOFT WEB INTELLIGENCE AND THE J-CO FRAMEWORK

In this section, we introduce our vision about *Soft Web Intelligence*, moving from the original paper (Fosci and Psaila, 2022b). Then, we briefly introduce the *J-CO* Framework, which is the technical tool that has inspired the idea of *Soft Web Intelligence*.

3.1 Soft Web Intelligence

We conceived the notion of *Soft Web Intelligence* as an evolution of the generic idea of Web Intelligence, on the basis of the current technological panorama. The following definition provides a synthetic characterization of the concept, that has come out while working on it after (Fosci and Psaila, 2022b).

Definition 1. *"Soft Web Intelligence" is the continued acquisition, integration and querying of JSON data sets coming from or representing Web sources, by exploiting Soft Computing and Soft Querying, so as to use them for decision making and knowledge discovery.*

In some sense, we could say that Definition 1 instantiates the very generic definition of *Web Intelligence* provided more than 20 years ago (Yao et al., 2001). It is the result of the considerations made by Zadeh in (Zadeh, 2004a; Zadeh, 2004b), concerning the fact that *JSON* has imposed, in place of XML, as the most popular format to represent and share data over the Internet, as well as it is the result of the availability of *NoSQL* DBMSs known as "*JSON* document stores", which natively store *JSON* data sets.

Through Figure 1, we illustrate how we figure out *Soft Web Intelligence*.

- **Web Sources**. Different kinds of Web sources can be considered. Users usually think about Web pages, but many services provide *JSON* data sets. For example, Web Services can be contacted to provide either complete data sets or single pieces of data; this latter ones could be singularly collected into a global data set; typically, social media expose Web services that can be exploited to interact with the system; nowadays, *JSON* is the data format on which most of Web services rely.
  Open-Data portals are a common channel that is exploited by Public Administrations to publish data sets (possibly "Authoritative Data Sets") concerned with the administered territory or country. Among all formats, *JSON* and *GeoJSON* are becoming more and more popular in this context too.
  As a final example, the content of Web pages (i.e., HTML pages) could provide useful data sets and information. In this case, techniques of "Web Scraping" could be exploited for acquiring the content of HTML pages and represent their information content as *JSON* data sets.

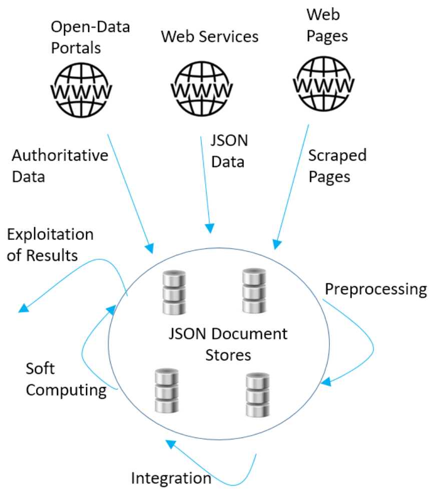

Figure 1: Vision of Web Intelligence.

- ***JSON* Document Stores**. A pool of databases managed by (potentially different) *JSON* document stores is the right solution to store *JSON* data sets acquired from Web sources. A very famous *JSON* store is *MongoDB*, but other products are available (such as *CouchDB*, chosen as the DBMS for the *HyperLedger Fabric* permissioned BlockChain platform (Bringas et al., 2019)).
- **Processing Activities**. Processing is the key factor for the success of *Soft Web Intelligence*. Certainly, "Preprocessing" is the first activity to perform on data, so as to remove noise and make formats homogeneous. The second activity is "Integration", i.e., integrating the different data sets possibly stored in different databases, by uniting those pieces of information that came from different sources. The final activity is "Soft Computing", in which techniques based on soft computing and soft querying are used to extract knowledge from integrated data. Clearly, the more powerful the support for soft computing and soft querying, t he greater the possibility to extract useful data and knowledge from the acquired data sets.

3.2 The J-CO Framework

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

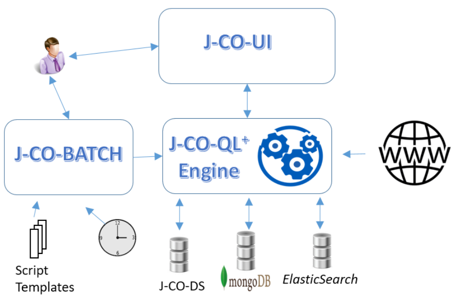

Figure 2: The *J-CO* Framework.

The current organization of the framework, which is the result of (Fosci and Psaila, 2022b), is depicted in Figure 2. We explain it hereafter.

- *J-CO-QL$^+$ Engine*. This component actually executes *J-CO-QL$^+$* scripts (i.e., queries). It is able to retrieve data from document databases (for example, managed by *MongoDB*) and save results into them; it also can send HTTP requests to Web sources to get *JSON* data sets directly from them.
- *J-CO-DS*. This component is a simple document store specifically introduced in (Psaila and Fosci, 2018) to store large or very large single *JSON* documents (such as many *GeoJSON* documents that cannot be stored within other *JSON* stores, such as *MongoDB*). In (Fosci and Psaila, 2022b), we extended its data model, in such a way it now supports three different types of collections:
  - *Static Collections* are the classical collections that contains *JSON* documents, similarly to other *JSON* stores (the content of a collection can be updated by the user or by the *J-CO-QL$^+$ Engine*);
  - *Dynamic Collections* automatically acquire the content of web sources at scheduled times, so as to provide images of web sources without need to access the Internet when they are processed;
  - *Virtual Collections* are associated to Web sources, but they do not manage any local copy of them, in that Web sources are accessed when the virtual collection is accessed by users or the *J-CO-QL$^+$ Engine* (thus, virtual collections provide a database view of Web sources).
- *J-CO-BATCH* (Introduced in (Fosci and Psaila, 2022b)) is an off-line executor of *J-CO-QL$^+$* scripts; in particular, it is possible to schedule the repeated execution of scripts. Another feature is the concept of "template": *J-CO-QL$^+$* scripts can be parameterized, so as to reuse them with different settings/configurations.
- *J-CO-UI* is the user interface, by means of which analysts can write *J-CO-QL$^+$* scripts, submit them to the *J-CO-QL$^+$ Engine* and inspect results.

```
CREATE FUZZY AGGREGATOR integrateRain
  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) ];
```

Listing 1: *J-CO-QL$^+$*: fuzzy aggregator `integrateRain`.

Data and Execution Models. For the sake of clarity, we briefly introduce the data and execution models of *J-CO-QL$^+$*.

- A *JSON* document is the basic computational unit. *JSON* documents are grouped within "collections": an instruction takes one or two collections as input and generates a new collection.
- A query or script in *J-CO-QL$^+$* is a sequence of instructions. They constitute a "piped flow", in such a way an instruction receives a "temporary collection" (generated by the previous instruction) as input and possibly generates a novel temporary collection as output. The "temporary" adjective denotes that the collection is not saved in any database, but is a temporary result of the process. Instructions can also acquire data either from *JSON* databases or from Web sources.
- For each single document, it is possible to independently evaluate its memberships to many fuzzy sets. These degrees are represented within the same document, by adding a special root-level field named `~fuzzysets`. It is a nested document that behaves as a key/value map: a field within it denotes the membership to a fuzzy set, in such a way the field name is the name of the fuzzy set, while the value (in the range $[0,1]$) is the degree. Specific clauses of *J-CO-QL$^+$* instructions evaluate soft conditions, by evaluating memberships.

4 USER-DEFINED FUZZY AGGREGATORS

User-defined fuzzy aggregators were missing in *J-CO-QL$^+$*: indeed, when documents in collections contain arrays, it is highly probable that their content should be somehow summarized, so as to determine the membership to some fuzzy set and contribute to soft querying. Hereafter, we present the novel statement named `CREATE FUZZY AGGREGATOR`.

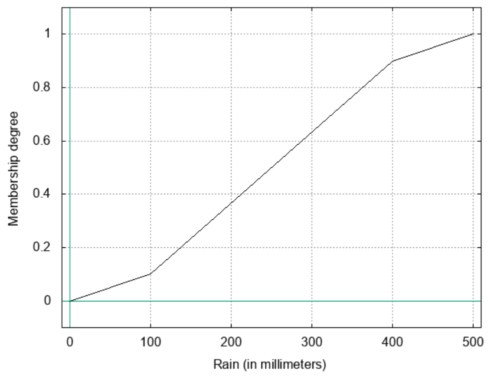

Figure 3: Membership function for the fuzzy aggregator `integrateRain`.

4.1 Basic Structure

Listing 1 reports the definition of a very simple fuzzy aggregator: its goal is to perform a cumulative aggregation (sum) of numerical values within an array. Hereafter, we explain it in details.

- The fuzzy aggregator is named `integrateRain`, as specified after the keywords `CREATE FUZZY AGGREGATOR`. The name is motivated by the fact that the aggregator has to aggregate the amount of rain (in millimeters).
- The clause `PARAMETERS` defines the formal parameters for the aggregator: in this case, only one parameter, named `rainData` is specified, which must be an array. Specifically, the aggregator is expected to receive the amount of rain, so each item in the array `rainData` denotes millimeters of rain.
- The clause `FOR ALL` scans all values in an array parameter, so as to perform aggregations. Specifically, all items `rd` in the array `rainData` are explored.
- The sub-clause `AGGREGATE` evaluates, for each item `rd` in the array `rainData`, an expression, whose value is aggregated into the "variable" specified after the keyword `AS`.
  Specifically, the expression to evaluate refers only to the item `rd`; this means that their values are aggregated into the value `av`.
- The clause `EVALUATE` is evaluated after the clause `FOR ALL` has scanned all items in the array and generated an aggregated value.
  Specifically, it refers to the `av` aggregated value only, meaning that the aggregated value is taken as it is. Clearly, since we aggregated generic numerical values, the result is not in the range $[0,1]$,

```
2. CREATE FUZZY AGGREGATOR 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)];
```

Listing 2: *J-CO-QL*$^+$: fuzzy aggregator `owaRain`.

so it cannot be considered as a membership. The next clause converts it into a membership.
- The clause `POLYLINE` defines a membership function as a polyline of points: while the x coordinate can be any real number, the y coordinate must be necessarily in the range $[0,1]$, because the membership function converts the value returned by the clause `EVALUATE` into a membership value.
  The polyline is depicted in Figure 3: notice that we considered as range of interest from 0 *mm* to 500 *mm* of rain (which is really a lot of rain). With 500 *mm*, the value 1 for the membership is reached: greater values of rain still obtain 1 as membership.

Resuming, the `integrateRain` fuzzy aggregator moves from an array of values, aggregates them and generates a membership to a fuzzy set.

4.2 Adding Local Derived Values

Listing 2 shows a more complex fuzzy aggregator, 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 the array is previously sorted. Hereafter, we explain the aggregator reported in Listing 2 in details.

- As for the aggregator reported in Listing 1, the aggregator `owaRain` receives one single array parameter, named `rainData`.
- The clause `SORT` generates a novel array, named `sRainData`, sorting each numeric item `rd` in the array `rainData` in ascending order.
- The clause `LOCALLY` evaluates, for each item `srd` in the array `sRainData`, a derived value that is used in the sub-clause `AGGREGATE`.
  Specifically, the $f(x) = x^2$ is used to compute the weight `w` of the item `srd`, whose position in the array `sRainData` is denoted by `POS` (the first item has `POS` $= 1$). Formally, the weight of the i-th item in `sRainData` is
  `w`$= f(i/|$`sRainData`$|) - f((i-1)/|$`sRainData`$|)$.
  Since f is a parable, items with highest position get the highest weights; furthermore, the array is

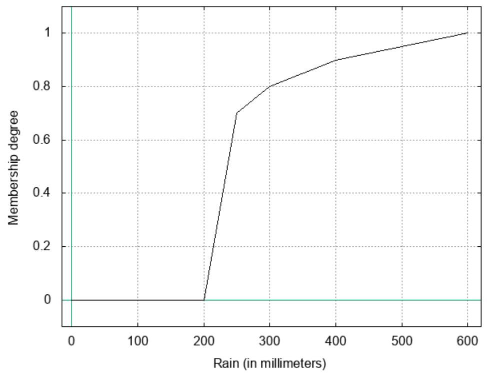

Figure 4: Membership function for the fuzzy aggregator `owaRain`.

sorted in ascending order, thus the highest values get the highest weights.

- The sub-clause `AGGREGATE` performs the aggregation, by summing all products of an item `srd` by its weight `w`.
  The resulting aggregated value, aliased as `av`, is then checked against the membership function defined by the clause `POLYLINE`, to obtain a membership. The polyline is depicted in Figure 4. Notice the shape, which is aimed at checking peaks of rain: below 200 *mm*, the membership is 0; between 200 and 250 *mm*, the membership raises quickly to 0.7, meaning that, after 250 *mm*, it is likely a peak; then, the membership progressively increases, thus denoting hard peaks of rain at 600 *mm*.

4.3 Multiple Aggregated Values

Listing 3 reports a further complex fuzzy aggregator, whose name is `weightedMemberships`. The goal is to aggregate memberships to fuzzy sets, provided that an array of weights is given. We now describe it.

- The aggregator receives two parameters: `memberships` is the array of memberships to aggregate; `w` is the array of weights.
- The size of the two vectors must be the same, because `w` contains the weight for each item in `memberships`. This constraint is expressed by the novel clause `PRECONDITION`: if this clause is not satisfied, the aggregation is not performed and an error is raised.
- In the clause `FOR ALL`, the sub-clause `LOCALLY` multiplies the `m` item by its weight (in the position `POS`). This value is aliased as `wm`.
- Two aggregated values must be computed: the first one is named `am` and is obtained by summing all the values `wn` computed by the clause `LOCALLY`; the second one is named `aw` and is obtained by summing all the weights in the array `w`.
- The clause `EVALUATE` assembles the two aggregated values, by dividing `am` by `aw`.
  By construction (the array `memberships` contains memberships) the result is in the range $[0,1]$, so it is already a membership. Since we do not want to modify it, no polyline is defined.

```
3. CREATE FUZZY AGGREGATOR weightedMemberships
     PARAMETERS
       memberships  TYPE ARRAY,
       w            TYPE ARRAY
     PRECONDITION   COUNT (memberships) = COUNT (w)
     FOR ALL        m IN memberships
       LOCALLY      m * w[POS] AS wm
       AGGREGATE    wm AS am
       AGGREGATE    w [POS] AS aw
     EVALUATE       am / aw;
```

Listing 3: *J-CO-QL*$^{+}$: fuzzy aggregator `weightedMemberships`.

4.4 Semantic Model

We conclude this section by formalizing the rich semantic model of fuzzy aggregators in *J-CO-QL*$^{+}$.

- A fuzzy aggregator is defined by means of a tuple
  $$\langle LV, AV, evalExpr, Points, Params, Precond\rangle$$
  where *LV* is the (possibly empty) set of "local values", *AV* is the (non empty) set of "aggregated values", *evalExpr* is the expression to evaluate with aggregated values, *Points* is the array of points that constitute the polyline, *params* is the list of parameters of the aggregator. *Precond* is the precondition: if it does not hold on *Params*, the aggregator is not evaluated.
- Each local value $lv_j \in LV$ is obtained as
  $$lv_j = le_j(e, POS, Params)$$
  where le_j is the expression used to evaluate the local value; le_j can be seen as a function of an item e and of its position *POS* in the array, as well as of the list of parameters.
- An aggregated value $av_k \in AV$ is evaluated as
  $$av_k = \Sigma_{e\in V} ae_k(e, pos(e), LV, Params)$$
  where ae_k is the expression used to evaluate the local value to aggregate; notice that ae_k depends on the item e, its position in the array, the set of local values and the list of parameters. The sum is performed for each item e in the V vector, where $V \in Params$.
- The final membership μ is obtained as
  $$\mu = Polyline(Points, evalExpr(AV, Params))$$
  where the expression *evalExpr* depends on the set *AV* of aggregated values and on the list of

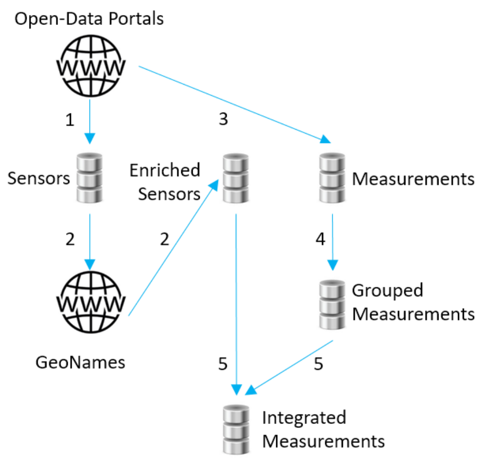

Figure 5: Preliminary Web-Intelligence process.

parameters.
The resulting value passes through the function*Polyline*, which receives the list *Points* of points specified in the clause `POLYLINE` (if missing, we assume that $Points = [(0,0),(1,1)]$).

The reader can see that, based on this semantic model, in *J-CO-QL*$^+$ it is possible to define a broad range of fuzzy aggregators. As far as we know, this is a novelty in the panorama of (soft) query languages.

In Section 5, we will show how to exploit fuzzy aggregators for soft querying data in *J-CO-QL*$^+$.

5 CASE STUDY AND QUERY

This section shows how to use the previously defined fuzzy aggregators to actually query a *JSON* data set. We introduce the case study (Section 5.1); then, we present the *J-CO-QL*$^+$ query in details (Section 5.2).

5.1 Case Study

The case study derives from the one we considered in (Fosci and Psaila, 2022b). In that work, the case study had to collect data from meteorological sensors and integrate them with registry data. The raw data were downloaded on-the-fly from an institutional Open-Data portal[1] and then pre-processed by a dedicated *J-CO-QL*$^+$ script to be integrated with geographical information.
In this work, we exploit the same Open-Data portal, but we downloaded and pre-processed rain measurements; in other words, we performed an activity of Web Intelligence that is depicted in Figure 5, by executing a pool of *J-CO-QL*$^+$ scripts that generate the source data set for the remainder of this paper. For the sake of space and since this pre-processing task is out of the scope of this paper, we quickly summarize it.

1. A collection of 1261 documents describing meteorological sensors is downloaded from the Open-Data portal[2]. Each document reports the identifier, the typology of measurement (rain, temperature, and so on), the coordinates and other less interesting data related to one sensor. The name of the city where the sensor is located is not reported.
2. Each sensor document is enriched with the city in which the sensor is located, by calling the online and freely-accessible *GeoName* service[3].
3. A second collection of $5,179,417$ documents is downloaded from the Open Data portal[4], describing the measurements made by sensors in the period from $01/05/2023$ to $31/05/2023$. Each document reports the identifier of the sensor that performed the measurement, the pure numerical value and the timestamp. No field regarding the typology (rain, temperature, etc) of the measurement is present.
4. Measurements made by the same sensor are grouped together: a novel collection of documents is obtained, such that a document corresponds to a sensor; it contains the sensor identifier and an array of measurements (made by that sensor) with their timestamp.
5. Finally, the collection of sensors is joined with the collection of grouped measurements: basically, a document fully describes a sensor and reports the array of measurements; only rain sensors are selected. A collection of 207 documents is obtained, related to $903,027$ measurements, which constitutes the initial collection for our case-study. The collection is saved within a *JSON* store, with name `MeasuredRain`.

Figure 6 reports a document in this collection: notice the array field named `rainData` (highlighted by a blue box), which contains simple documents describing measurements of rain (the field `value` reports millimeters of rain). Clearly, each single sensor is described by one single document.

[1]Open-Data portal of Regione Lombardia https://www.dati.lombardia.it/

[2]https://www.dati.lombardia.it/Ambiente/Stazioni-Meteorologiche/nf78-nj6b

[3]https://www.geonames.org/export/ws-overview.html

[4]https://www.dati.lombardia.it/Ambiente/Dati-sensori-meteo/647i-nhxk

```
4. USE DB Webist2023
     ON SERVER jcods 'http://127.0.0.1:17017';

5. GET COLLECTION MesauredRain@Webist2023;

6. FILTER
     CASE WHERE WITH .rainData
       GENERATE
         CHECK FOR
           FUZZY SET SignificantRain
             USING integrateRain(EXTRACT_ARRAY(
                         .value FROM ARRAY .raidData)),
          FUZZY SET PeaksOfRain
            USING owaRain (EXTRACT_ARRAY(
                         .value FROM ARRAY .raidData)),
           FUZZY SET Wanted
             USING weightedMemberships (MEMBERSHIP_ARRAY(
                 [PeaksOfRain, SignificantRain]), [2, 1])
         ALPHACUT 0.8 ON Wanted
         BUILD {
           .city       : .city,
           .province   : .province,
           .sensorId   : .sensorId,
           .dateStart  : .dateStart,
           .dateEnd    : .dateEnd,
           .ranking    : MEMBERSHIP_TO (Wanted)
         }
         DEFUZZIFY;

7. SAVE AS PeaksAndLongRain@Webist2023;
```

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

Once the input collection is ready, the problem to address in the case study might be the following.

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.*

Problem 1 can be thought as a soft query.

Query 1. *Consider the universe of sensors and two fuzzy sets in it: the first one is named `PeaksOfRain` and denotes those sensors that measured peaks of rain; the second one is named `SignificantRain` and denotes sensors that measured a significant amount of rain in the monitored period. A third fuzzy set that is named `Wanted` denotes those sensors that are in the fuzzy set `PeaksOfRain` (with weight* 2*) and possibly in the fuzzy set `SignificantRain` (with weight* 1*). Specifically, we are interested in sensors whose membership to the fuzzy set `Wanted` is no less than* 0.8.

Clearly, in order to evaluate the memberships of sensors to the desired fuzzy sets, it is necessary to aggregate measurements of rain.

5.2 Query

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

Acquiring Data. Line 4 specifies the database to connect with. Specifically, notice that the database

```
{
  "city"        : "Osio Sopra",
  "province"    : "BG",
  "sensorId"    : 5856,
  "latitude"    : 45.6338645090807,
  "longitude"   : 9.55608425579086,
  "rainData"    : [
    {
      "date"        : "12/05/2023 21:00:00",
      "value"       : 2.2
    },
    …,
    {
      "date"        : "24/05/2023 17:00:00",
      "value"       : 47.8
    }
  ]
}
```

Figure 6: Example of document in the starting `MeasuredRain` collection.

`webist2023` is managed by *J-CO-DS*, the *JSON* document store provided by the *J-CO* Framework (see Section 3).

On Line 5, the instruction `GET COLLECTION` actually retrieves he content of the collection `MeasuredRain` from the database `Webist2023`; the collection becomes the new temporary collection of the process.

Soft Querying with Fuzzy Aggregators. The instruction `FILTER` on Line 6 of Listing 4 actually performs the soft query. Hereafter, we explain it.

- The clause `CASE WHERE` selects (in a Boolean way) those documents that have the field `rainData`. The remainder of the instruction will work on these documents.
- 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` contains many different branches `FUZZY SET`, one for each fuzzy set under consideration. We have three branches `FUZZY SET` on line 6.
- The first branch `FUZZY SET` evaluates the membership to the fuzzy set `SignificantRain`. To do this, the soft condition `USING` (which actually provides the membership to the fuzzy set under consideration) exploits the fuzzy aggregator `integrateRain` defined in Listing 1.
  To call the fuzzy aggregator, an array of numbers must be provided as actual parameter: since the array field `rainData` contains nested documents; the special 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 aggregator `integrateRain` becomes the membership

```
{
  "city"        : "Osio Sopra",
  "province"    : "BG",
  "sensorId"    : 5856,
  "latitude"    : 45.6338645090807,
  "longitude"   : 9.55608425579086,
  "rainData"    : [
    {
      "date"        : "12/05/2023 21:00:00",
      "value"       : 2.2
    },
    ...,
    {
      "date"        : "24/05/2023 17:00:00",
      "value"       : 47.8
    }
  ],
  "~fuzzysets"  : {
    "PeaksOfRain"       : 0.930466311,
    "SignificantRain"   : 0.754000000,
    "Wanted"            : 0.871644207
   }
}
```

Figure 7: Example of temporary document generated by the instruction `FILTER` on Line 6 before the block `BUILD`.

of the current *JSON* document to the fuzzy set `SignificantRain`.
The field `~fuzzysets` is added to the *JSON* document, so as to report the computed membership.

- The second branch `FUZZY SET` evaluates the membership of the current document to the fuzzy set `PeaksOfRain`, through the fuzzy aggregator `OwaRain` (reported in Listing 2).
Apart from the fact that the aggregator `OwaRain` performs an OWA aggregation (instead of a cumulative aggregation) the branch behaves similarly to the previous one: the aggregator is called by passing the array of values obtained by projecting the array `rainData` on the inner field `value` by means of the special function `EXTRACT_ARRAY`; then, the computed membership becomes the degree to the fuzzy set `PeaksOfRain`.
Definitely, the goal of he 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 obtain high membership.
A second field is added into the field `~fuzzysets`, denoting the membership to the novel fuzzy set.

- The third branch `FUZZY SET` evaluates the degree of the current *JSON* document to the fuzzy set `Wanted`. The goal is to combine the memberships to the fuzzy sets `PeaksOfRain` and `SignificantRain` in such a way the former contributes with weight 2, while the latter contributes with weight 1.

```
{
  "city"        : "Osio Sopra",
  "province"    : "BG",
  "sensorId"    : 5856,
  "latitude"    : 45.6338645090807,
  "longitude"   : 9.55608425579086,
  "Wanted"      : 0.871644207
}
```

Figure 8: Example of document saved in the `PeaksAndLongRain` collection.

Clearly, the fuzzy aggregator (reported in Listing 3) `weightedMemberships` is exploited, but this time it is necessary to create an "array of memberships": this is done by the special function `MEMBERSHIP_ARRAY`. Specifically, this function creates a novel array by taking the previously calculated memberships to the listed fuzzy sets (i.e., `PeaksOfRain` and `SignificantRain`), whose values are taken from the special field `~fuzzysets`. The second actual parameter is a constant array that reports the weights to apply (i.e., 2 and 1, respectively); this way, requirements in Query 1 are met.
The resulting membership becomes the membership degree to the fuzzy set `Wanted`.
The third and final field is added to the field `~fuzzysets`, with the membership to the fuzzy set `Wanted`.

- The clause `ALPHACUT` discards *JSON* documents whose membership to the fuzzy set `Wanted` is less than 0.8. 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 final block `BUILD` restructures the output documents and the option `DEFUZZIFY` removes the special filed `~fuzzysets`, so as documents become again classical crisp *JSON* documents.

Saving the Results. The resulting collection, which contains documents describing sensors of interest on the basis of Problem 1, is finally saved by Line 7. The collection is named `PeaksAndLongRain`.

6 CONCLUSIONS

In this paper, we enhanced the potential application of *Soft Web Intelligence* by introducing the concept of "user-defined fuzzy aggregator". The concept allows users of the *J-CO-QL*$^+$ language that are involved in tasks of *Soft Web Intelligence* (enabled by the *J-CO* Framework) to directly perform complex aggregations of array fields in *JSON* documents, so as to directly obtain memberships to fuzzy sets by aggregating raw data; definitely, sophisticated soft queries are made possible.

The paper resumes the vision of *Soft Web Intelligence*, then introduces the novel statement `CREATE FUZZY AGGREGATOR`, by presenting three different examples of aggregators, together with its semantic model. Then, through a case study, a short yet sophisticated query is presented, which exploits all the three previously defined fuzzy aggregators for performing a complex soft query on rain data.

As a future work, we will finish to investigate the definition of user-define fuzzy aggregators, so as to cope with very complex situations; in this sense, we also plan to build a library of fuzzy aggregators to distribute with the *J-CO* Framework. Furthermore, we plan to investigate how web scraping tools could be effectively integrated within *Soft Web Intelligence*: indeed, we expect that these tools represent somehow uncertainty about the data they extract from Web pages, because this uncertainty could be easily managed with soft computing and soft querying. Definitely, although we already demonstrated the effectiveness of the *J-CO* Framework for integrating geographical data sets (see (Fosci and Psaila, 2022a)), we want to further push its capabilities towards soft querying, specifically by allowing users to define their complex constructs (see (Fosci and Psaila, 2023)).

The framework is available on a Github page[5].

REFERENCES

Alahakoon, D. and Yu, X. (2015). Smart electricity meter data intelligence for future energy systems: A survey. *IEEE Transactions on Industrial Informatics*, 12(1):425–436.

Bringas, P. G., Pastor, I., and Psaila, G. (2019). Can blockchain technology provide information systems with trusted database? the case of hyperledger fabric. In *I. C. on Flexible Query Answering Systems*, pages 265–277. Springer, Cham.

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. (2022a). Soft integration of geotagged 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. (2023). Soft querying powered by user-defined functions in j-co-ql+. *Neurocomputing*, 546:126311.

Han, J. and Chang, K.-C. (2002). Data mining for web intelligence. *Computer*, 35(11):64–70.

Kacprzyk, J. and Zadrożny, S. (2010). Soft computing and web intelligence for supporting consensus reaching. *Soft Computing*, 14(8):833–846.

Li, H. and Yen, V. C. (1995). *Fuzzy sets and fuzzy decision-making*. CRC press.

Poli, V. S. R. (2015). Fuzzy data mining and web intelligence. In *I. Conf. on Fuzzy Theory and Its Applications (iFUZZY)*, pages 74–79. IEEE.

Psaila, G. and Fosci, P. (2018). Toward an anayist-oriented polystore framework for processing json geo-data. In *Int. Conf. on Applied Computing 2018, Budapest; Hungary, 21-23 October 2018*, pages 213–222. IADIS.

Reddy, P. V. S. (2010). Fuzzyalgol: Fuzzy algorithmic language for designing fuzzy algorithms. *J. of Computer Science and Engineering*, 2(2):21–24.

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.

Zadeh, L. A. (2004a). A note on web intelligence, world knowledge and fuzzy logic. *Data & Knowledge Engineering*, 50(3):291–304.

Zadeh, L. A. (2004b). Web intelligence, world knowledge and fuzzy logic–the concept of web iq (wiq). In *I. C. on Knowledge-Based and Intelligent Inf. and Eng. Systems*, pages 1–5. Springer.

Zhang, Y.-Q. and Lin, T. Y. (2002). Computational web intelligence (cwi): synergy of computational int. and web technology. In *W. C. on Comp. Int.*., volume 2, pages 1104–1107. IEEE.

Zhong, N., Liu, J., Yao, Y., Wu, J., Lu, S., Qin, Y., Li, K., and Wah, B. (2006). Web intelligence meets brain informatics. In *I. Ws. on Web Intelligence Meets Brain Informatics*, pages 1–31. Springer.

[5]Github repository of the *J-CO* Framework: https://github.com/JcoProjectTeam/JcoProjectPage