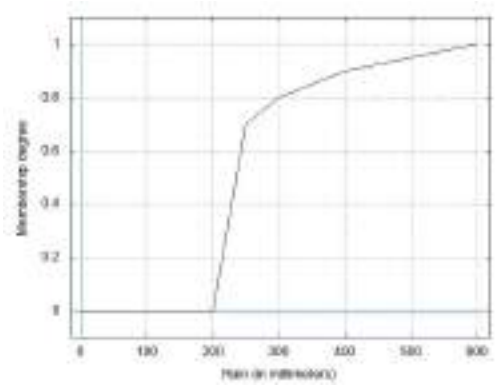

Figure 4: Membership function for the fuzzy aggregator `owaRain`.

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

```
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`.

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.

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

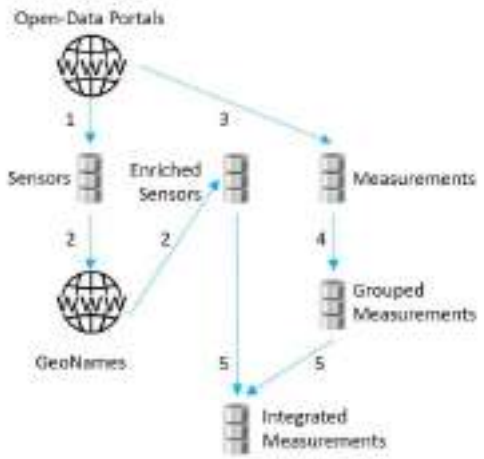

Figure 5: Preliminary Web-Intelligence process.

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 aggre-

gating 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 geodata. 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