Juzzy Constrained: Software for Constrained Interval Type-2 Fuzzy Sets and Systems in Java

Pasquale D'Alterio, Jonathan M. Garibaldi, Robert I. John and Christian Wagner
IMA and LUCID Research Groups
School of Computer Science, University of Nottingham
Nottingham, UK
{pasquale.d'alterio, jon.garibaldi, robert.john, christian.wagner}@nottingham.ac.uk

***Abstract*—Constrained interval type-2 (CIT2) fuzzy sets are a class of type-2 fuzzy sets that has been recently proposed as a way to extend type-1 membership functions to interval type-2 (IT2) while keeping a semantic connection between the IT2 fuzzy set and the concept it models. Recent work has shown how their mathematical properties can be used to design CIT2 fuzzy logic systems that are able to provide explanations for their outputs. Although the CIT2 representation can be a valuable alternative to the IT2 one, no software library for their implementation is available for the research community. The aim of this paper is to introduce a new Java library, *Juzzy Constrained*, that has been developed as an extension of the popular type-1 and type-2 Java toolkit *Juzzy*, adding support for CIT2 sets and systems. Throughout the paper, the main classes and the structure of the new library are described, together with a working example that illustrates how to build a CIT2 fuzzy system from scratch and how it can be used to produce explanations for the output.**

***Index Terms*—Juzzy, Constrained Interval Type-2, XAI, type-2 fuzzy logic**

I. Introduction

The use of interval type-2 (IT2) [1] fuzzy sets and systems has rapidly grown over time. The development of practical and fast defuzzification algorithms (e.g. [2]) together with the increase in performance [3]–[5], led to the development of IT2 fuzzy logic systems (FLS) in controller design [6], classification and regression [7], modeling of data and words [8].

More recently, fuzzy logic has been adopted to build interpretable systems: the use of the rule-base structure together with the partitioning of the input and output variables with fuzzy sets (FSs) with a clear semantic meaning (i.e. a linguistic label) allow for the design of FLSs that show an understandable decision process. In the context of explainable artificial intelligence (XAI) [9], many T1 FLSs have been produced with the ability of providing clear explanations for each of the classifications produced by the system [10]–[12]. Implementing IT2 FLS with the same characteristics, however, has proven to be more challenging. In fact, although T1 and IT2 FLS share the same rule-based structure, there is a difference in how the output of the system is produced. As discussed in other research works [13], [14], the use of centroid defuzzification procedures like the Karnik-Mendel algorithm (KM) [2], makes it hard to find a direct link between the endpoints of the interval centroid and the rule-base of the FLS as well as producing a detailed explanation like for T1 FLSs. To ensure a higher level of interpretability when going from T1 to IT2 and type-2 (T2) membership functions, constrained type-2 (CT2) [15] and constrained interval type-2 (CIT2) [13] have been introduced. By the imposition of additional mathematical constraints on the footprint of uncertainty (FOU) and the shape of the embedded sets (ES), they establish a standard process to "extend" T1 fuzzy sets to IT2 ones while keeping a strong semantic relation between the constrained set and the concept it models. It has also been shown [13], [14] how the additional constraints make it easier to explain the interval centroid produced by CIT2 FLS thanks to the use of embedded sets with a "meaningful" shape. Therefore, these properties make CIT2 FLSs a valuable alternative to IT2 FLSs in contexts in which producing explainable outputs is important (e.g. XAI applications).

This new class of fuzzy sets has recently started to be explored, with a particular focus on CIT2 fuzzy sets. Although many practical applications of CIT2 FLS have already been shown [13], [14], there is no library that can be used by the research community to easily deploy CIT2 FLS. The aim of this paper is to present the first software library, named *Juzzy Constrained*, that implements CIT2 fuzzy sets and systems and also to collect constructive feedback on the CIT2 representation from the research community. Written in Java, *Juzzy Constrained* is an extension of the already well-known Java library *Juzzy* [16] and follows its conventions to facilitate its use for developers. The new toolkit makes possible the design of CIT2 FLS using the defuzzification algorithms proposed in [13], [14] and is capable of using the constrained representation to provide human-readable explanations for the constrained interval centroids produced by the systems.

The rest of the paper is organized as follows: after a short description of other software libraries focused on T1 and T2 fuzzy logic (Sec. II), CIT2 fuzzy sets will be briefly described, highlighting their main characteristics and the motivations

behind their introduction (Sec. III). Then, the new library *Juzzy Constrained* will be analyzed, describing its structure, its main classes and its relation with *Juzzy* (Sec. IV). Finally, a working example will be presented: a CIT2 FLS will be built from scratch, with the help of code snippets to facilitate the understanding of the usage of the toolkit (Sec. V).

II. Related works

Many tools for the development of T1 and T2 FLS have been released over the years. One of the most famous ones is the Fuzzy Logic Toolbox for MATLAB[1]. It allows developers to design T1 fuzzy sets and systems through a set of functions or the use of a graphical interface. The sets built with the toolbox, the control surfaces and the rules can then be easily visualized. Similar toolboxes that include IT2 fuzzy sets have been proposed in [17]–[20].

Software libraries for different programming languages have also been released. In [21] a Python toolkit for the automatic generation of T1, IT2 and T2 fuzzy sets from data has been presented; [22], instead, describes a software library in R for the modelling of T1, IT2 and T2 FLS that also includes functions for the graphical visualization of fuzzy sets and control surfaces.

Software for the creation of fuzzy systems has also been included in famous suites for machine learning such as KEEL [23] and Weka [24]. Both offer various methods to learn fuzzy rules and sets from data (e.g. with the use of genetic algorithms) and to perform fuzzy clustering.

After the introduction of the IEEE Standard for Fuzzy Markup Language for the definition of fuzzy sets and systems in a "human-readable and hardware independent way" [25], new software libraries adhering to the novel standard have been developed such as JFML [26] and VisualJFML [27].

A. Juzzy

Juzzy Online [28], is a toolkit for the design, execution and sharing of T1 and T2 fuzzy sets and systems through the use of an online dashboard that is usable with no knowledge of programming.

A Java version of the same tool has been released. Juzzy [16] is a library for the implementation of T1, IT2 and T2 fuzzy sets and systems. It is written in Java, it is open-source and available online at http://juzzy.wagnerweb.net/, http://www.lucidresearch.org/software.html and on the Maven Central Repository[2]. The toolkit implements T2 fuzzy sets with the zSlices representation [29] and also supports multi-core execution of the code.

III. Constrained Interval Type-2 Fuzzy Sets

Constrained type-2 (CT2) and constrained interval type-2 (CIT2) fuzzy sets were proposed [15] as a new way to model vague concepts using T2 and IT2 fuzzy sets starting from a T1 set modeling the same word or concept. The rest of the paper and the library presented here will focus only on CIT2 since most of the research work in this area [13], [14], [30] is focused on CIT2 only.

CIT2 fuzzy sets start from a T1 fuzzy set, called *generator set* (GS), that models a given concept. The uncertainty in this case, is represented by the fact that the exact location on the x-axis of the GS is not known. This situation may arise, for example, when different people are asked to place the fuzzy set modeling the label *medium height* on the x-axis, as shown by the Gaussian MFs in black in Fig. 1

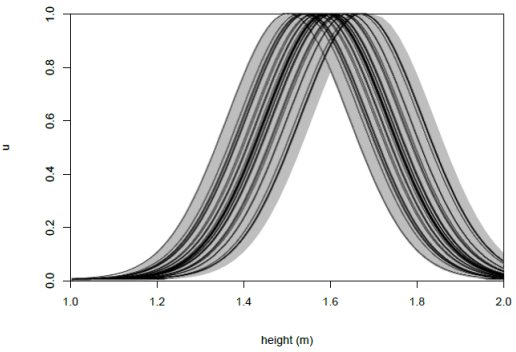

Figure 1. Different Gaussian fuzzy sets modeling the concept of *medium height* (in black). A possible FOU of a CIT2 that contains them is depicted in grey

This uncertainty causes a *blurring* around the GS that determines the footprint of uncertainty (FOU). The constrained approach proposed a standard way of carrying out this operation:

- The "width" of the blurring is modeled as an interval (in the continuous case) and called *displacement interval*.
- The FOU of the generated CIT2 fuzzy set is defined as the points covered by the translation along the x-axis of the GS within the displacement interval (e.g. the shaded gray area in Fig. 1).
- Within this FOU, the only embedded sets that are considered acceptable (*acceptable embedded sets*, AES) are the ones that have the same shape as the GS, i.e. the ones that represent a valid translation of the GS within the displacement interval. Only the AES are processed in operations that work with the embedded sets.

For more details, see [13]. Processing only AESs rather than all the embedded sets makes a significant difference in the explainability of some fuzzy operators, as analyzed in [13]. Specifically, for the centroid defuzzification, the endpoints of the constrained interval centroid are determined by acceptable embedded sets with a *meaningful* shape. As shown in Sec. V, the properties of these AES can be used to produce human-readable explanations for the system output while providing both a formal and intuitive understanding of how they have been obtained. Therefore, the CIT2 modeling, represents a valuable alternative to IT2 systems in all the cases in which it is important to understand the decision process of the classification model (e.g. in the XAI field).

[1]https://www.mathworks.com/products/fuzzy-logic.html

[2]https://search.maven.org/artifact/com.github.chwagnlucid/Juzzy/2.0/jar

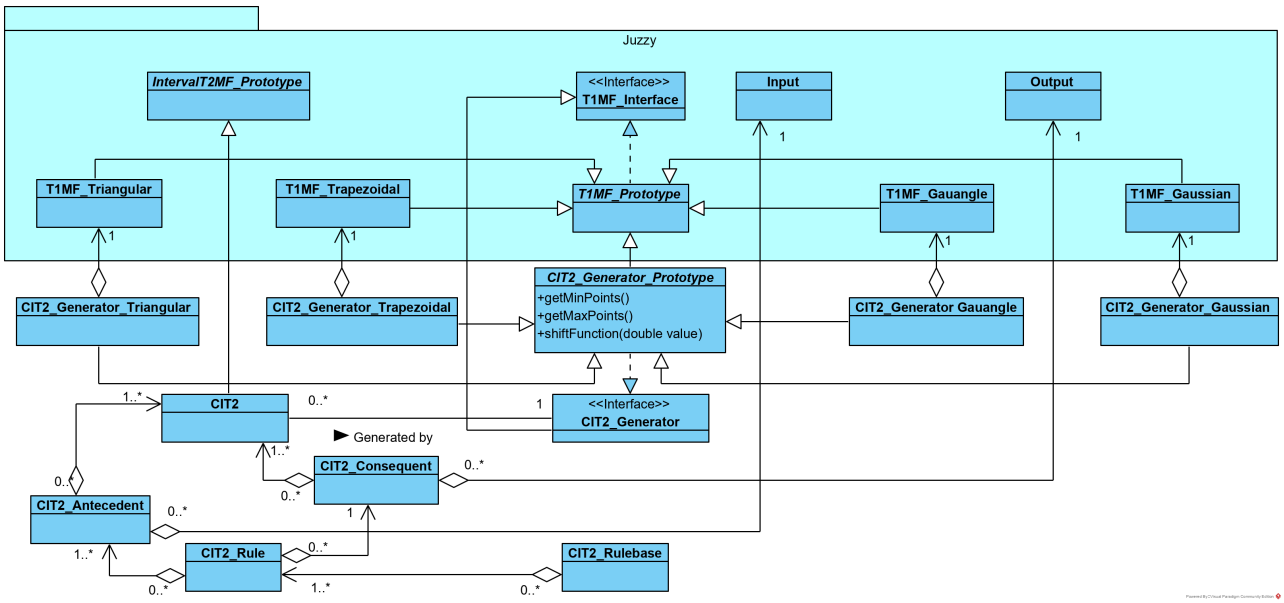

Figure 2. The class diagram of Juzzy Constrained

IV. Juzzy Constrained

The Java library for CIT2 fuzzy sets and systems has been conceived as an extension of Juzzy: it makes use of its T1 membership functions to define the generator sets of CIT2 fuzzy sets and also adopts some of its conventions (e.g. for the creation of rules) and utility classes (such as `Input` and `Output` to model the input and output variables of a CIT2 FLS). Therefore, for Juzzy Constrained to work, also Juzzy must be included in the given Java project.

The source-code released under the BSD 3-Clause license, the documentation and the JAR archive of Juzzy Constrained are freely available on GitHub at https://github.com/PasqualeDAlterio/JuzzyConstrained and http://www.lucidresearch.org/software.html. The library is also available on the Maven Central Repository and can be quickly included in any Java Maven project (for more information, see the GitHub page).

A. Library structure

The packages included in the library are shown in Fig. 3. The package `CIT2_Generator` includes the T1 membership functions that are usable as generator sets. Triangular, Gaussian, Gauangle and trapezoidal membership functions are currently supported. Each one of them is a wrapper of the correspondent `T1MF` defined in Juzzy; the main difference between them is that each of the `CIT2_Generator` must implement a method that returns all the points of local (and global) maximum and a method that returns all points of local minimum of the membership function, as requested by the `CIT2_Generator` interface. These two methods are needed to determine the upper and lower membership functions of the CIT2 fuzzy set, as discussed in Theorem 1, in the Appendix.

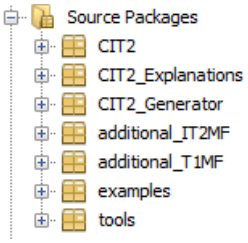

Figure 3. The package structure of the library

The `CIT2` package is the core of the library: it implements CIT2 fuzzy sets and systems using the same style used by Juzzy. Once the sets have been defined they can be used to build antecedents and consequents that are then organized in `CIT2_Rule`. Once the rules are created, they can be organized in a `CIT2_Rulebase` to implement a FLS.

`CIT2_Explanations` contains all the objects that are used in the generation of the explanation of the output of a `CIT2_Rulebase`. They mostly focus on organizing and formatting information in a humanly readable piece of text in order to show how the endpoints of the constrained centroid have been obtained. The remaining packages offer additional utilities and tools that were not originally implemented in Juzzy but are useful in the Juzzy Constrained context (e.g. an IT2 fuzzy set where the upper and lower bounds can have arbitrary shape or a T1 fuzzy set modeling the result of the inference operation).

Fig. 2 shows the class diagrams with all the main classes used in Juzzy Constrained and their relation with the original classes in Juzzy. Each `CIT2` fuzzy set, to be instantiated, needs a `CIT2_Generator`. This interface extends the `T1MF_Interface` defined in Juzzy, since the generator

set is a T1 set, and requires the implementation of three additional methods: `getMaxPoints`, `getMinPoints` and `shiftFunction`. The first two, as described earlier in this section, are needed to determine the boundary functions of the generated CIT2 fuzzy; the shifting method, instead, is needed to generate the acceptable embedded sets: since they are translations along the x-axis of the generator set, this method takes a real number `value` as an argument and returns a new T1 membership function representing the generator set shifted by `value`. Additionally, since CIT2 fuzzy sets are a special case of IT2 fuzzy sets, i.e. they have been obtained by adding a set of additional mathematical constrained to the original IT2 definition, the class `CIT2` extends the Juzzy abstract class `IntervalT2MF_Prototype`.

The generator sets implemented in the library extend `CIT2_Generator_Prototype`, i.e. an abstract class that already implements some functionalities that are used by all the generators provided. To add a new generator membership function, it is only required to implement the `CIT2_Generator` interface. This operation, as it will be shown in Sec. V, is straightforward for all the widely used T1 membership functions.

All the classes related to the the construction of a rule and a rule-base follow the same conventions used in Juzzy, making them easy to work with for the developers that are already used to the T1, IT2 and T2 rule-bases of the original library.

B. Defuzzification algorithms, other features and limitations

The toolkit provides two algorithms for the defuzzification of the output of a CIT2 FLS. The first one, implemented by the method `sampleCentroid` in the class `CIT2_Rulebase`, is based on the *sampling approach* proposed in [13], itself an adaptation for CIT2 sets of the *sampling method* for T2 fuzzy sets [31]. Since the extensive computation of the centroid by processing all the acceptable embedded sets has a prohibitive cost (similarly to what happens with "standard" T2 fuzzy sets) and each of these embedded sets only gives a small contribution to the final result, the idea is to calculate an approximation by sampling a subset of the acceptable embedded sets and use only them to compute the constrained centroid.

The other defuzzification algorithm included in Juzzy Constrained is the one presented in [14], based on the concept of *switch indices* instead of the *switch points* used by the KM procedure for IT2 fuzzy sets [2]. This approximation method is faster than the sampling one as it uses the properties of CIT2 fuzzy sets to quickly identify the small subset of acceptable embedded sets that will be used to determine the constrained centroid. For more details about this algorithm, please refer to [14]. This approach can also be used to produce human-readable explanations for CIT2 FLSs as shown in [32] and in Sec. V.

In addition to the methods implemented in Juzzy for the visualization of T1 and IT2 fuzzy sets, Juzzy Constrained integrates the popular Java graphical library JFreeChart[3]. This represents a more flexible way of building plots, since they are easily and widely customizable, while also giving the opportunity of better highlight the FOUs of the CIT2 and IT2 fuzzy sets, as shown in Fig. 4.

Being currently still under development, Juzzy Constrained has some limitations. Specifically, `CIT2_Rule` only implements the *and* operator in the antecedent composition and does so with the *min T-Norm*. In addition to that, each rule can currently has only one consequent. At the moment, this limitation can be overcome by replacing a rule with n consequents with n replicas of the rule, one per consequent. In future works, we plan on expanding the library by adding the support to multiple-consequent rules and more antecedent connectors.

V. Applications and Examples

This section will show how Juzzy Constrained can be used in practice to develop CIT2 FLS, starting from the creation of CIT2 fuzzy sets and then illustrating how they can be put together to make rules and rulebases.

The example analyzed in this paper is the *tipping problem*. This system has been chosen for its simplicity and *not* to show the full potential of CIT2 FLSs. A more thorough analysis of the advantages of the use of CIT2 FLSs and case studies on real world datasets can be found in [32]. The tipping problem has the following structure: it has 2 input variables, food and service, and the goal is to use them to determine the adequate percentage to give as tip.

The first thing to do, is to instantiate the generator sets. Their creation is identical to the creation of T1 fuzzy sets in Juzzy. Here there is an example of how the generator sets for the *service* membership functions can be created.

```
T1MF_Generator_Gauangle unfriendlyServiceMF=
    new T1MF_Generator_Gauangle("Unfriendly",0.0, 0.0, 6);
unfriendlyServiceMF.setLeftShoulder(true);
T1MF_Generator_Gauangle okServiceMF =
    new T1MF_Generator_Gauangle("OK",2.5, 5.0, 7.5);
T1MF_Generator_Gauangle friendlyServiceMF =
    new T1MF_Generator_Gauangle("Friendly",4, 10, 10);
friendlyServiceMF.setRightShoulder(true);
```

With the generator sets, it is possible to create CIT2 fuzzy sets. In addition to the generator sets, also the displacement intervals need to be specified. They determine how "wide" the shifting and therefore the FOU will be. In the example below, the positive shifting values `shifting_size_2` is used to generate the displacement interval `[-shifting_size_2, shifting_size_2]`.

```
CIT2 cit2_unfriendlyServiceMF =  new CIT2(
     unfriendlyServiceMF.getName(),
    unfriendlyServiceMF, shifting_size_2);
CIT2 cit2_okServiceMF = new CIT2(
    okServiceMF.getName(), okServiceMF,
    shifting_size_2);
CIT2 cit2_friendlyServiceMF = new CIT2(friendlyServiceMF.
     getName(), friendlyServiceMF, shifting_size_2);
```

The definition of the input and output variables, is taken from Juzzy since it uses the same `Input` and `Output` objects.

[3] http://www.jfree.org/jfreechart/

```
Input food = new Input("Food Quality", new Tuple(0,10));
Input service =new Input("Service Level", new Tuple(0,10));
Output tip = new Output("Tip", new Tuple(0,30));
```

The partitioning of the variables can then be plotted using JFreeChart as shown below. The results of this operation for the food, service and tip are shown respectively in Fig. 4, Fig. 5 and Fig. 6.

```
JFreeChartPlotter.plotMFs("Food partitioning", new CIT2[]{
     cit2_badFoodMF, cit2_greatFoodMF},
    food.getDomain(), 1000);
JFreeChartPlotter.plotMFs("Service partitioning", new CIT2
     []{cit2_friendlyServiceMF, cit2_okServiceMF,
     cit2_unfriendlyServiceMF}, service.getDomain(), 1000);
JFreeChartPlotter.plotMFs("Tip partitioning", new CIT2[]{
     cit2_lowTipMF, cit2_mediumTipMF, cit2_highTipMF}, tip.
     getDomain(), 1000);
```

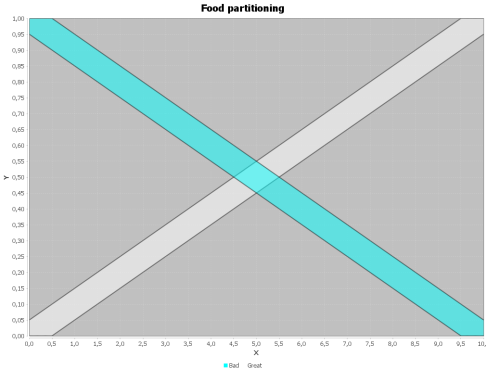

Figure 4. Partitioning of the *food* variable

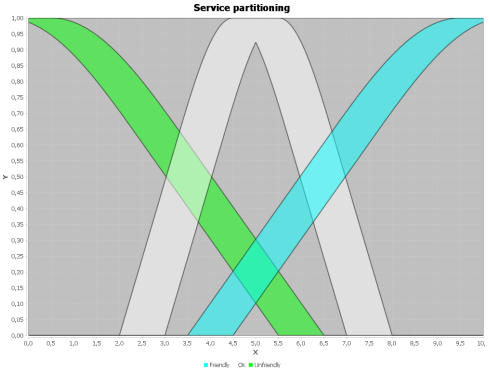

Figure 5. Partitioning of the *service* variable

Once the CIT2 fuzzy sets have been defined, they can be paired with the input and output variables to define the antecedents and the consesequents that will be used in the rulebase. In this case, Juzzy Constrained follows the same conventions used by Juzzy, making the creation of `CIT2_Antecedent` and `CIT2_Consequent` very similar to the creation of IT2 antecedents and consequents in the original library.

```
CIT2_Antecedent unfriendlyService =
    new CIT2_Antecedent(cit2_unfriendlyServiceMF, service);
CIT2_Antecedent okService =
    new CIT2_Antecedent(cit2_okServiceMF, service);
CIT2_Antecedent friendlyService =
    new CIT2_Antecedent(cit2_friendlyServiceMF, service);
```

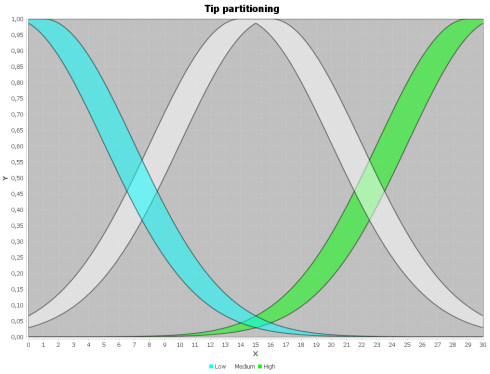

Figure 6. Partitioning of the *tip* variable

Once the antecedents and consequents have been defined, they can be put together to create the rulebase. Again, the initialization of a CIT2 rulebase is very similar to the creation of T1 and IT2 rulebases in Juzzy.

```
CIT2_Rulebase rulebase = new CIT2_Rulebase();
rulebase.addRule(new CIT2_Rule(new CIT2_Antecedent[]{
     badFood, unfriendlyService}, lowTip));
rulebase.addRule(new CIT2_Rule(new CIT2_Antecedent[]{
     badFood, okService}, lowTip));
rulebase.addRule(new CIT2_Rule(new CIT2_Antecedent[]{
     badFood, friendlyService}, mediumTip));
rulebase.addRule(new CIT2_Rule(new CIT2_Antecedent[]{
     greatFood, unfriendlyService}, lowTip));
rulebase.addRule(new CIT2_Rule(new CIT2_Antecedent[]{
     greatFood, okService}, mediumTip));
rulebase.addRule(new CIT2_Rule(new CIT2_Antecedent[]{
     greatFood, friendlyService}, highTip));
```

After the input values are set, there are two algorithms that can be used to do the inference and defuzzify the result: the sampling strategy [13] and the switch index method [14]. In the first case, the algorithm can be executed invoking the method `rulebase.samplingDefuzzification(50)` where 50 is the number of samples used to compute the constrained centroid. The function returns a `Tuple` representing the centroid.

```
food.setInput(7);
service.setInput(8);
Tuple constrained_centroid_sampling=
    rulebase.samplingDefuzzification(50);
Tuple constrained_centroid_si=
    rulebase.switchIndexDefuzzification(100);
ExplainableCentroid result=
    rulebase.explainableDefuzzification(100);
```

The switch index approach, instead, can be used in two different ways: using the method `rulebase.switchIndexDefuzzification(100)` where 100 is the level of discretization used to defuzzify the acceptable embedded sets, the library returns a `Tuple` containing the value of the constrained centroid, just like in the sampling method case; the method `rulebase.explainableDefuzzification(100)`, instead, returns the constrained centroid and the explanation for its generation in an `ExplainableCentroid` object.

As already discussed in other research works [13], [14], the properties of CIT2 fuzzy sets can be used to link the endpoints of the centroid to the specific acceptable embedded sets that

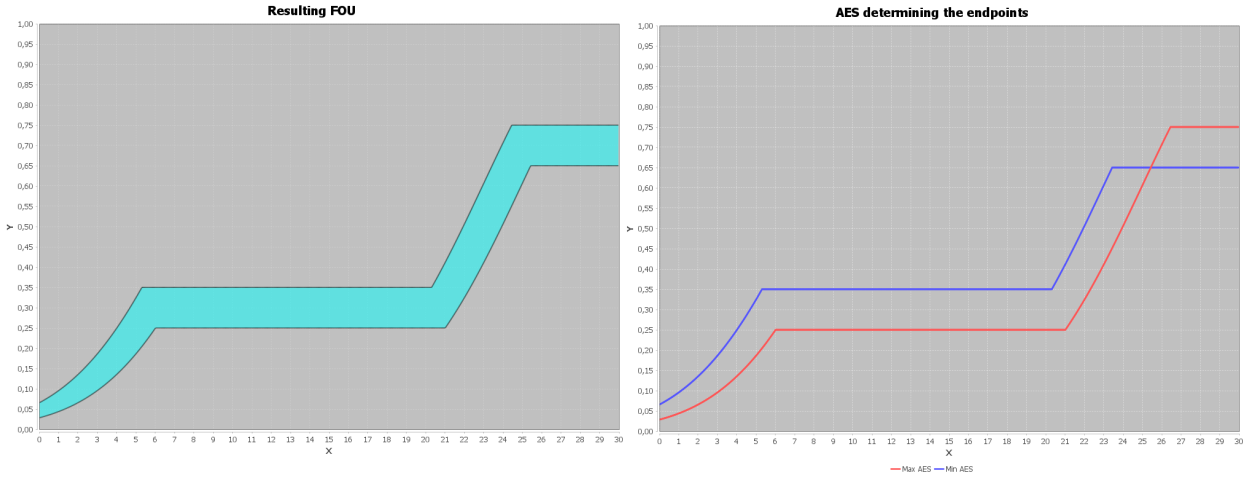

Figure 7. FOU obtained from the inference (on the left) and the acceptable embedded sets determining the endpoints of the constrained centroid (on the right)

generated them. They can then be used to determine which rules and input values led to the creation of the constrained interval centroid, in order to create a human-readable explanation. The selected acceptable embedded sets also have an interpretable structure: the consequent membership functions that contributed to their generation are clearly visible and so are the firing strengths of the rules they belong to (i.e. the heights at which they have been "truncated"). For other IT2 defuzzification procedures like the KM one, on the other hand, there is no guarantee that the chosen embedded sets will have any *meaningful* shape nor that it is possible to link them directly to the rules to produce an explanation. The ability to provide interpretable results when computing the constrained interval centroid is one of the reasons why CIT2 fuzzy sets can represent a valuable alternative to IT2 fuzzy sets in the context of XAI.

Once the `ExplainableCentroid` object is obtained, the acceptable embedded sets determining the constrained centroid can be plotted as shown below, together with the fired FOU. The plots for this example are show in Fig. 7.

```
JFreeChartPlotter.plotMFs("Resulting FOU", new
    IntervalT2MF_Interface[]{rulebase.getFiredFOU()}, tip.
    getDomain(), 1000);
JFreeChartPlotter.plotMFs("AES determining the endpoints",
    new T1MF_Interface[]{left_aes, right_aes}, tip.
    getDomain(), 1000);
System.out.println("The recommended tip percentage is in
    the range:"+result.getIntervalCentroid());
//Print the explanations
System.out.println(result.printableExplanation());
```

The `ExplainableCentroid` structure also stores the information necessary for the creation of the human-readable explanation using the method `result.printableExplanation()`. The piece of text below, links each of the endpoints of the constrained centroid to the rules in the rulebase that generated them, also showing the firing values of the rules, the input values and their membership degrees with respect to the antecedent membership functions.

```
The recommended tip percentage is in the range: left =
    18.06 and right = 19.67
The leftmost centroid (18.06) is obtained from firing the
    following rules:
Medium: 0.35 obtained because Food Quality IS Bad [0.25,
    0.35] AND Service Level IS Friendly [0.71, 0.88] using
    the UPPER membership degree of each input terms
High: 0.65 obtained because Food Quality IS Great [0.65,
    0.75] AND Service Level IS Friendly [0.71, 0.88] using
    the LOWER membership degree of each input terms

The rightmost centroid (19.67) is obtained from firing the
    following rules:
Medium: 0.25 obtained because Food Quality IS Bad [0.25,
    0.35] AND Service Level IS Friendly [0.71, 0.88] using
    the LOWER membership degree of each input terms
High: 0.75 obtained because Food Quality IS Great [0.65,
    0.75] AND Service Level IS Friendly [0.71, 0.88] using
    the UPPER membership degree of each input terms
```

A. *Adding a new CIT2 generator membership function*

Juzzy Constrained currently supports 4 types of generator sets: Gaussian, Gauangle, triangular and trapezoidal. To add additional shapes, it is necessary to define a new class that implements the `CIT2_Generator` interface. The new class needs to provide methods that return the points of minimum and maximum of the membership function (so that the FOU of the CIT2 fuzzy set can be determined, see Theorem 1, in the Appendix) and a method for the shifting of the generator set (to generate the acceptable embedded sets).

Although implementing the methods that determine the points of minimum and maximum may seem challenging, it is relatively easy for many shapes. In the code snippet below, the implementation of these method is shown for the trapezoidal membership function.

```
@Override
protected ArrayList<Interval> computeMinPoints()
{
   ArrayList<Interval> min_points=new ArrayList<>();
    min_points.add(new Interval(trapezoid.getA()));
    min_points.add(new Interval(trapezoid.getD()));
    return min_points;}
```

```
@Override
protected ArrayList<Interval> computeMaxPoints()
{
    ArrayList<Interval> max_points=new ArrayList<>();
    max_points.add(new Interval(trapezoid.getB(), trapezoid
        .getC()));
    return max_points;
}
```

The minimum and maximum points are stored in `Interval` objects which store generic intervals of the form $[a, b]$. The reason why intervals are used rather than points is that in some membership functions the points of minimum or maximum are infinite and all within a given interval. For example, in the case of a trapezoidal membership function, the points of maximum are all the points that make the shorter base, i.e. all the points in the segment $\overline{BC}$. The minimum points, instead, are only A and B; in this case the `Interval` object is initialized using a single value a, representing the interval $[a, a]$.

In other functions, the points of local minimum or maximum may not exist. For example, the Gaussian shape does not have any points of local minimum. In that situation, the `getMinPoints()` method can return a null value.

VI. Conclusion

In this paper, the new open-source library *Juzzy Constrained* has been presented. This toolkit, written in Java, has been developed as an extension of the fuzzy library *Juzzy* (for type-1 and type-2 fuzzy logic) and adds the support to constrained interval type-2 (CIT2) fuzzy sets and systems. This new class of fuzzy sets represents a useful alternative to the standard interval type-2 representation in the contexts in which a high level of interpretability is needed. Through the addition of some mathematical constraints, it ensure that a meaningful connection is kept between the shape of the footprint of the uncertainty, the embedded sets and the concept the CIT2 set is modeling. In the literature, it has also been shown how these properties can be used to produce explainable systems by processing only embedded sets with a *meaningful* shape for the determination of the interval centroid.

The paper demonstrates the library and showcases the properties and utility of CIT2 models using a worked, practical example, clearly highlighting the advantages of CIT2 FLSs from an XAI point of view.

The toolkit presented here, is the first one to support CIT2 fuzzy sets and systems. The aim of this paper is therefore to present the new library to the research community and also to receive feedback on the project and CIT2 fuzzy sets in general.

The structure of the library, its main classes and the defuzzification algorithms provided have been discussed, while in Sec. V a CIT2 fuzzy logic system is built from scratch, with the help of code snippets to facilitate the understanding of how the toolkit can be used.

Being still under development, the library has some limitation such as the fact that rules only support a single consequent or that the antecedents can only be connected using the *and* operator. In future works, we plan on improving these aspects, adding rules with multiple consequents and different connectors for the antecedents as well as making the library compliant with the fuzzy markup language.

References

[1] J. M. Mendel, R. I. John, and F. Liu, "Interval type-2 fuzzy logic systems made simple," *IEEE Transactions on Fuzzy Systems*, vol. 14, no. 6, pp. 808–821, Dec 2006.

[2] N. N. Karnik and J. M. Mendel, "Centroid of a type-2 fuzzy set," *Information Sciences*, vol. 132, no. 1-4, pp. 195–220, 2001.

[3] M. A. Sanchez, O. Castillo, and J. R. Castro, "Generalized type-2 fuzzy systems for controlling a mobile robot and a performance comparison with interval type-2 and type-1 fuzzy systems," *Expert Systems with Applications*, vol. 42, no. 14, pp. 5904 – 5914, 2015.

[4] A. H. M. Pimenta and H. A. Camargo, "Interval type-2 fuzzy classifier design using genetic algorithms," in *International Conference on Fuzzy Systems*, July 2010, pp. 1–7.

[5] L. Amador-Angulo, O. Castillo, and M. Pulido, "Comparison of fuzzy controllers for the water tank with type-1 and type-2 fuzzy logic," in *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*, June 2013, pp. 1062–1067.

[6] O. Castillo, L. Amador-Angulo, J. R. Castro, and M. Garcia-Valdez, "A comparative study of type-1 fuzzy logic systems, interval type-2 fuzzy logic systems and generalized type-2 fuzzy logic systems in control problems," *Information Sciences*, vol. 354, pp. 257 – 274, 2016.

[7] A. H. M. Pimenta and H. d. A. Camargo, "Genetic interval type-2 fuzzy classifier generation: A comparative approach," in *2010 Eleventh Brazilian Symposium on Neural Networks*, Oct 2010, pp. 194–199.

[8] F. Liu and J. M. Mendel, "Encoding words into interval type-2 fuzzy sets using an interval approach," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 6, pp. 1503–1521, Dec 2008.

[9] D. Gunning, "Explainable artificial intelligence (xai)," *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2017.

[10] J. M. Alonso, A. Ramos-Soto, E. Reiter, and K. van Deemter, "An exploratory study on the benefits of using natural language for explaining fuzzy rule-based systems," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2017, pp. 1–6.

[11] I. Baaj and J.-P. Poli, "Natural language generation of explanations of fuzzy inference decisions," in *International Conference on Fuzzy Systems (FUZZ-IEEE 2019)*, 2019.

[12] N. Potie, S. Giannoukakos, M. Hackenberg, and A. Fernandez, "On the need of interpretability for biomedical applications: Using fuzzy models for lung cancer prediction with liquid biopsy," in *International Conference on Fuzzy Systems (FUZZ-IEEE 2019)*, 2019.

[13] P. D'Alterio, J. M. Garibaldi, R. John, and A. Pourabdollah, "Constrained interval type-2 fuzzy sets," *IEEE Transactions on Fuzzy Systems*, pp. 1–1, 2020.

[14] P. D'Alterio, J. M. Garibaldi, R. I. John, and C. Wagner, "A fast inference and type-reduction process for constrained interval type-2 fuzzy systems," *IEEE Transactions on Fuzzy Systems*, 2020.

[15] J. M. Garibaldi and S. Guadarrama, "Constrained type-2 fuzzy sets," in *Advances in Type-2 Fuzzy Logic Systems (T2FUZZ), 2011 IEEE Symposium on*. IEEE, 2011, pp. 66–73.

[16] C. Wagner, "Juzzy – a java based toolkit for type-2 fuzzy logic," in *2013 IEEE Symposium on Advances in Type-2 Fuzzy Logic Systems (T2FUZZ)*, 04 2013.

[17] A. Taskin and T. Kumbasar, "An open source matlab/simulink toolbox for interval type-2 fuzzy logic systems," in *2015 IEEE Symposium Series on Computational Intelligence*, Dec 2015, pp. 1561–1568.

[18] O. Castillo and P. Melin, "Computational intelligence software: Type-2 fuzzy logic and modular neural networks," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 1820–1827.

[19] J. R. Castro, O. Castillo, and P. Melin, "An interval type-2 fuzzy logic toolbox for control applications," in *2007 IEEE International Fuzzy Systems Conference*, July 2007, pp. 1–6.

[20] J. R. Castro, O. Castillo, P. Melin, and A. Rodríguez-Díaz, *Building Fuzzy Inference Systems with a New Interval Type-2 Fuzzy Logic Toolbox*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 104–114.

[21] J. McCulloch, "Fuzzycreator: A python-based toolkit for automatically generating and analysing data-driven fuzzy sets," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2017, pp. 1–6.
[22] C. Wagner, S. Miller, and J. M. Garibaldi, "A fuzzy toolbox for the r programming language," in *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, June 2011, pp. 1185–1192.
[23] J. Alcala-Fdez, S. Garcia, F. J. Berlanga, A. Fernandez, L. Sanchez, M. J. del Jesus, and F. Herrera, "Keel: A data mining software tool integrating genetic fuzzy systems," in *2008 3rd International Workshop on Genetic and Evolving Systems*, March 2008, pp. 83–88.
[24] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, p. 10–18, Nov. 2009.
[25] "IEEE standard for fuzzy markup language," *IEEE Std 1855-2016*, pp. 1–89, May 2016.
[26] J. M. Soto-Hidalgo, J. M. Alonso, G. Acampora, and J. Alcala-Fdez, "JFML: A java library to design fuzzy logic systems according to the ieee std 1855-2016," *IEEE Access*, vol. 6, pp. 54 952–54 964, 2018.
[27] G. Acampora, J. Alcala-Fdez, R. Siciliano, J. M. Soto-Hidalgo, and A. Vitiello, "VisualJFML: A visual environment for designing fuzzy systems according to ieee std 1855-2016," in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, June 2019, pp. 1–6.
[28] C. Wagner, M. Pierfitt, and J. McCulloch, "Juzzy online: An online toolkit for the design, implementation, execution and sharing of type-1 and type-2 fuzzy logic systems," *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 2321–2328, 2014.
[29] C. Wagner and H. Hagras, "Toward general type-2 fuzzy logic systems based on zslices," *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 4, pp. 637–660, Aug 2010.
[30] P. D'Alterio, J. M. Garibaldi, and A. Pourabdollah, "Exploring constrained type-2 fuzzy sets," in *2018 IEEE World Congress on Computational Intelligence (WCCI 2018)*, July 2018.
[31] S. Greenfield, R. John, and S. Coupland, "A novel sampling method for type-2 defuzzification," in *Proc. UKCI 2005*, 09 2005, pp. 120–127.
[32] P. D'Alterio, J. M. Garibaldi, and R. I. John, "Constrained interval type-2 fuzzy classification systems for explainable AI (XAI)," in *2020 IEEE World Congress on Computational Intelligence (WCCI 2020)*, July 2020.

APPENDIX

A. Determining the boundary functions of a CIT2 fuzzy set

The `CIT2_Generator` interface used in this library requires a method that returns all the points of local maximum and one that returns all the points of local minimum of a membership function for it to be used as a generator set. The reason why these points are needed is to easily determine the boundary functions of the generated CIT2 fuzzy set. As shown in [13], these two membership functions for a generic CIT2 fuzzy set $\breve{A}$ can be expressed as:

$$\overline{\mu}_{\breve{A}}(x) = \sup_{S \in \text{CAES}_{\breve{A}}} \mu_S(x) \tag{1}$$

$$\underline{\mu}_{\breve{A}}(x) = \inf_{S \in \text{CAES}_{\breve{A}}} \mu_S(x) \tag{2}$$

where $\breve{A}$ is a CIT2 set and the CAES is the collection of its acceptable embedded sets. The following theorem proves that to determine the upper and lower bounds of the FOU of a CIT2 fuzzy set, it is sufficient to know the generator set, its points of local minimum and maximum and the displacement interval used.

Theorem 1. *Given a CIT2 fuzzy set $\breve{A}$, to determine its upper membership function $\overline{\mu}_{\breve{A}}$ it is sufficient to know the T1 generator set G (with a continuous membership function) its displacement interval $[a, b]$ with $a \leq 0, b \geq 0, a, b \in \mathbb{R}$ and the set M of all the local points of maximum of μ_G.*

Proof. To prove the theorem, we will show that the upperbound function of $\breve{A}$ $\overline{\mu}_{\breve{A}}$ can be expressed as:

$$\overline{\mu}_{\breve{A}}(x) = \begin{cases} \max\left(\star, \max_{k \in M}(\mu_G(k))\right) & M \neq \emptyset \\ \star & \text{otherwise} \end{cases} \tag{3}$$

where M is the set of all the local points of maximum of μ_G in $[x-b, x-a]$ and $\star$ is:

$$\star = \max\Big(\mu_G(x-a), \mu_G(x-b)\Big) \tag{4}$$

Since each S in (1) is obtained as a shifting of G using the values in the displacement interval (see [13] for more details), it can be rewritten as:

$$\overline{\mu}_{\breve{A}}(x) = \max_{z \in [a,b]} \mu_G(x-z) \tag{5}$$

Using (5), we can rewrite the upperbound membership function (3) as:

$$\max_{z \in [a,b]} \mu_G(x-z) = \begin{cases} \max\left(\star, \max_{k \in M}(\mu_G(k))\right) & M \neq \emptyset \\ \star & \text{otherwise} \end{cases} \tag{6}$$

At this point, we need to prove that the upperbound membership function (5) is determined either by $\mu_G(x-a)$ and $\mu_G(x-b)$ or by one of the points of maximum of μ_G in the interval $[x-b,\, x-a]$, i.e. by one of the points in M. To do so, we consider the two possible scenarios:

1) $\max_{z \in [a,b]} \mu_G(x-z) = \star$ (7)
2) $\max_{z \in [a,b]} \mu_G(x-z) \neq \star$ (8)

In (7), we assume that the upperbound membership function is determined by maximum between $\mu_G(x-a)$ and $\mu_G(x-b)$. In this case, (3) trivially holds, both when M is empty and when it contains at least one element. In (8), instead, we need to prove that when the upperbound membership function is not determined by $\mu_G(x-a)$ and $\mu_G(x-b)$, then it is be determined by one of the points of maximum in M. In fact, since the upperbound membership degree of x is different from both $\mu_G(x-a)$ and $\mu_G(x-b)$, it must be determined by another value w that is different from $x-a$ and $x-b$. Formally:

$$\exists\, w \in (x-b,\, x-a) : \forall z \in [a, b],\, \mu_G(w) \geq \mu_G(x-z) \tag{9}$$

By definition, w is a point of local maximum in $[x-b, x-a]$ and must therefore be equal to the maximum $k \in M$ in (6) when $M \neq \emptyset$. Therefore the thesis holds in 2) as well. Since (6) holds in all the possible cases, it is true. □

Similarly, it can be proven that to determine the lowerbound membership function of a CIT2 fuzzy set it is sufficient to know the generator set, its points of local minimum and the displacement interval used.