A fuzzy expert system architecture for data and event stream processing

Jean-Philippe Poli, Laurence Boudet

A Fuzzy Expert System Architecture For Data And Event Stream Processing

Jean-Philippe Poli, Laurence Boudet

CEA, LIST, Data Analysis and System Intelligence Laboratory, 91191 Gif-sur-Yvette cedex, France.

Abstract

The Internet of Things was born from the proliferation of connected objects and is known as the third era of information technology. It results in the availability of a huge amount of continuously acquired data which need to be processed to be more valuable. This leads to a real paradigm shift: instead of processing fixed data like classical databases or files, the new algorithms have to deal with data streams which bring their own set of requirements. Researchers address new challenges in the way of storing, querying and processing those data which are always in motion.

In many decision making scenarios, fuzzy expert systems have been useful to deduce a more conceptual knowledge from data. With the emergence of the Internet of Things and the growing presence of cloud-based architectures, it is necessary to improve fuzzy expert systems to support higher level operators, large rule bases and an abundant flow of inputs.

In this paper, we introduce a modular fuzzy expert system which takes data or event streams in input and which outputs decisions on the fly. Its architecture relies on both a graph-based representation of the rule base and the cooperation of four customizable modules. Stress tests regarding the number of rules have been carried out to characterize its efficiency.

Keywords: Fuzzy expert system, complex event processing, data stream

*Corresponding author

Email address: jean-philippe.poli@cea.fr (Jean-Philippe Poli, Laurence Boudet)

processing, rule base representation, policies

1. Introduction

The emergence of connected objects and mobile devices gave birth to the Internet of Things and is leading towards a continuous data acquisition from different devices and sensors. Before this third era of information technology, the data were stored in data warehouse, queried at once and manipulated by algorithms as a whole. With such data in motion, the use cases have changed: for instance, new database management paradigms are introduced, special efforts are made on data compression to avoid networks overload, and supervised or unsupervised learning algorithms are rethought.

Cugola and Margara [1] define the *Information Flow Processing* (IFP) domain as the domain of tools capable of processing information as it flows. Usually, the flow is coming from multiple sources and processed to extract relevant knowledge. They distinguish two subdomains:*Complex Event Processing* (CEP) and *Data Stream Processing* (DSP). Algorithms for processing such flows have to be fast and incremental [2] and are evaluated regarding two criteria: the number of passes over the stream (which must be as close as possible to 1) and the size of the workspace in memory [3].

On the one hand, DSP consists in processing data flows and in producing a new data flow as output. The Federal Standard defines a data stream as a "sequence of digitally encoded signals used to represent information in transmission". More formally, data stream can be defined [2, 3, 4] as a sequence of data items $x_1, \ldots, x_i, \ldots, x_n$ such that the items are read once in increasing order of the indexes i. A lot of tools have been introduced to process data streams. For instance, traditional database management systems, which work on persistent data, are replaced by data stream management systems whose queries run continuously: anytime new data arrive, the result of a query is updated [5, 6]. Other researches mainly include data streams mining with either clustering methods [7] or neural networks [8]. In [9, 10], the authors are revealing the open chal-

lenges which must be addressed in the domain of data stream mining, including privacy issues, developing a methodology for stream preprocessing, developing online monitoring systems and balancing resources.

On the other hand, CEP differs by the type of data items it considers: an item is a notification of event [11]. CEP are also associated with velocity: it aims at managing thousands of events per second [12], for instance, up to 125000 events for a financial software [13]. In this domain, processing mainly consists in filtering, gathering and combining those events to build a higher level information [14], to raise alerts or trigger processes. It makes use of the relationships which exist between events: indeed, events may be related in various ways (by cause, by timing, or by membership) [11].

Nowaday, DSP and CEP became usual considerations in many real world applications. We can cite for example: system monitoring and fault detection, home automation, security and finance [1]. However, in both data and event streams, the information may be incomplete and imprecise by nature [15]. For instance, sensors may be out of order or inaccurate, and data may be noisy. Fuzzy logic [16] has been specifically designed to mathematically represent uncertainty and vagueness and is a popular tool for dealing with imprecision in many real world problems. Taking advantage of fuzzy logic, fuzzy expert systems allow to easily represent human knowledge about data and phenomena and have been successfully applied to many domains [17, 18].

Fuzzy expert systems often come with a higher computational cost compared with boolean logic expert systems. For instance, fuzzy inference needs to assess the whole rule base to compute the outputs, whereas in boolean expert systems, a subset of the rules are applied one by one to produce the inference. Whereas fuzzy inference involves simple operations and simple functions to lower the computational cost, it has been showed that fuzzy rule bases need to be more complicated if only piecewise-linear functions (e.g. trapezoids...) are used instead of non-linear membership functions (e.g. sigmoids...) [19]. Consequently, expensive functions in terms of computation are needed to assess the aggregation and the defuzzification [20]. In addition, in real-world applications, it is

possible to have very large rule bases which require a great amount of processor time [21]. Moreover, to describe the relations between the data or the events, more sophisticated operators are needed for temporal [22, 23], spatial [24, 25] or even spatio-temporal [26] reasoning. These operators imply a higher computational cost. In addition, traditional fuzzy expert systems compute output values only when input values have changed. This is not compliant with event streams whose events are potentially arriving in an irregular manner: in such a case, expressions may change before the next event arrival (see section 3.3).

Our work aims at developing a fuzzy expert system to process information flows, handling the imprecision brought by noisy data, sensors or network problems with fuzzy logic. The motivation of our work is to provide an efficient fuzzy expert system in operational contexts. To enable human experts to author more complex rules, our system is able to efficiently assess complex fuzzy relations [23]. To ensure it can interface easily with the various information systems of our partners, we chose to avoid specific architectures (like GPU) and to develop a software for data and event stream processing on regular CPU platforms. Finally, in industrial applications, the efficiency is important not only because there must be a lot of rules, but also because the rules can be applied to a huge number of objects or events per second.

The paper is structured as follows: section 2 presents the related work about large rule base handling. Section 3 describes the architecture of our fuzzy expert system. Section 4 presents the implementation, and then the protocol and the results of experiments on both data and event streams. Finally, section 5 points out the conclusions.

2. Related work

To address real-world applications, fuzzy expert systems have to be able to process large rule bases very fast, and thus face the well-known combinatorial explosion problem. Indeed, in case of a conjunctive combination of the terms of all the inputs (also known as grid fuzzy rule structure), the number of rules

grows exponentially with respect to the number of inputs. For instance, there are p^n possible rules for n inputs with p terms each.

Combs and Andrews [27] tried to solve this problem by proposing a rule construction schema based on the union of single-antecedent rules, called Union Rule Configuration. In this case, the number of rules evolves only linearly in function of the number of inputs. Mendel and Liang [28] contested this approach stating that the two rule bases may not be equivalent and suggest to inquire whether the replacement makes sense.

Large rule bases are rarely given by human experts and are rather automatically inducted from datasets. Thus, automatic rule induction also faces the curse of dimensionality. To fix this issue, one can generate a partial rule base which does not cover the whole input space. Approaches based on input space partitioning by k-d trees [29] or quadtrees [30] result in nonuniform overlapping membership functions which are difficult to label with an understandable linguistic term. Jin [31] takes advantage of the conclusion drawn in [32] : optimal rules cover the extrema. The generated rule base is then checked for redundancy and potential inconsistency and optimized by a genetic algorithm. Hence, Jin's algorithm generates 27 rules from a training set of 20000 examples described by 11 inputs. In [33], authors present S-FRULER a genetic fuzzy system for regression problem capable of learning rules from big datasets. First, a multi-granularity fuzzy discretization is applied on the whole dataset, which is then split into several partitions. Each partition is then treated as an independent problem, which allows distribution. The processing consists in selecting variables randomly and then using genetic algorithms to induce rules. The rules are then combined into a unique rule base. Authors claim they obtain simple rule bases in terms of number of rules while maintaining a good precision. However, the different steps do not guarantee to have an optimal rule base since random variable selection, genetic algorithms and rule combination can add a lot of biases.

The interpretability of the rule base is not the only reason to decrease the number of rules: applying on information streams needs a fast processing of

the rules to make decisions on the fly. Fuzzy controllers have been introduced to overcome these drawbacks and are able to process inputs in real-time [34]. More recently, papers address the acceleration of fuzzy computation either with dedicated hardware [35] or with the help of Graphics Processing Units (GPU) [36]. However, to our experience, fuzzy expert softwares which run on classic CPU platforms are more convenient for many reasons. Firstly, they are easier to interface with an existing system than electronic chipsets. Then, DSP and CEP both rely on software intensive architectures. Moreover, in terms of scalability, it is possible to use from a single core of a machine to several machines and it can all be done transparently for the user ; for instance, it can take advantage of the virtualization as in cloud-based services.

The next section describes a suitable architecture for general-purpose fuzzy expert systems in which the problem of the rule base size in terms of computation speed is addressed by eliminating redundancy in the assessment. We also distribute the different roles like gathering inputs, computing and announcing results in different modules to be able to process information streams.

3. Architecture description

Fuzzy expert systems can infer regarding two main paradigms:

- Mamdani type, in which rule conclusions are fuzzy set;
- Takagi-Sugeno type, in which rule conclusions are a function of the inputs, i.e. a crisp value.

Whatever the type of inference, when a group of inputs change at a time t, all the rules containing at least one of those inputs have to be reevaluated. In information streams, inputs may change several times per second, or rules must be applied on thousands of incoming events per second ; the evaluation of the whole rule base may thus need a huge computation time. The distribution of the rule base is not always a good solution: for instance, monitoring of vehicles

or of patients need to process the same rule base over different input streams. However, the evaluation for each vehicle or each patient can be concurrent.

In this article, we introduce an architecture which tends to avoid the system saturation. In the remainder, without loss of generality, the examples will be given for a Mamdani type inference system.

3.1. Architecture overview

Figure 1 presents the overview of the proposed architecture. The modularity is ensured by a separation of the tasks and a customization provided by the use of policies. A policy is a set of parameters which customize the behavior of each module. The combination of the behaviors of all the modules enable to address a lot of applications and issues : regular or irregular data rate, delay before inference, etc. The architecture is composed of several modules :

- the **active input queue** gathers and groups the inputs by timestamps,
- the **scheduler** monitors the system (via the operating system) and to decide which inputs group has to be processed,
- the **evaluator** is in charge of the evaluation of the rules,
- the **output change broadcaster** informs the user about outputs changes.

The different modules help avoiding a system overload (for instance, the active input queue selects the inputs which should be treated) or user overfeeding (for instance, the output change broadcaster displays only the relevant information). We first introduce how we optimize the rule base representation by common subexpression elimination and the concept of expiration of expressions. We then describe each module of the architecture and give some examples of policies.

3.2. Rule base representation

The rule base in-memory model plays a major role in the efficiency of the fuzzy expert system. Expressions are usually modeled with a tree [37], as in

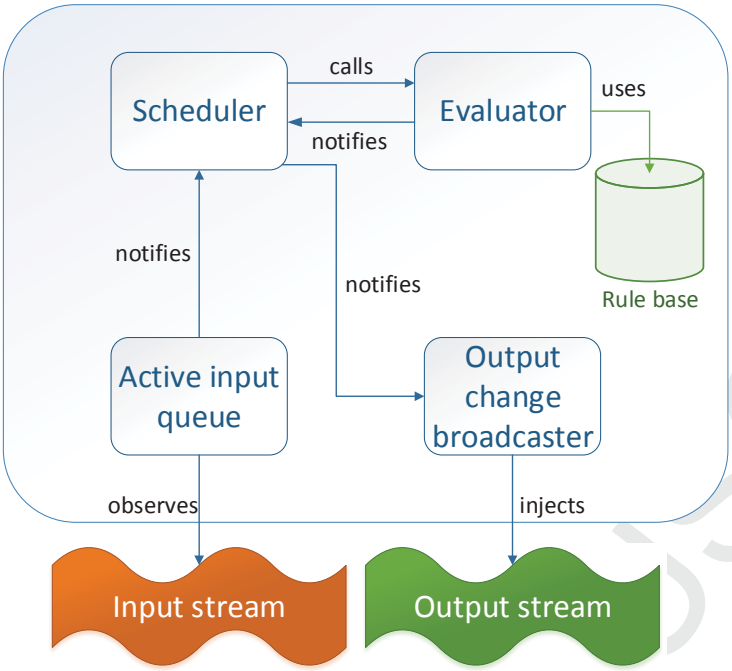
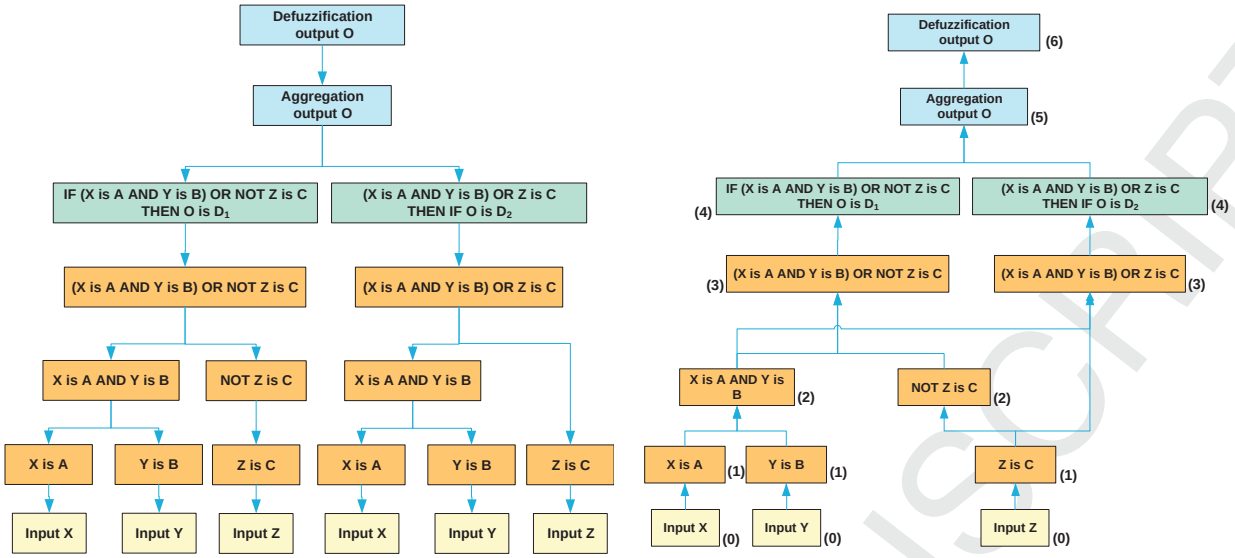


Figure 1: Architecture overview.

Figure 2(a). However, some expressions can be included in several rules or other expressions: thus, in a tree representation, it is difficult to check the redundancy of such expressions, and it is necessary to compute them several times when a group of inputs changed. This problem is known as common subexpression elimination (CSE).

To address the CSE problem in our architecture, we chose to represent each expression by a unique node: thus, the rule base is not represented by a tree anymore but by a graph (figure 2(b)). More precisely, we use an acyclic directed graph to avoid loops during the evaluation. In the graph, an edge $A \longrightarrow B$ means that if the value of the node A changes, it affects the node B and B has to be evaluated again. In this case, B is called a direct successor of A. In the graph we are considering, a node may have several direct successors. A node can represent fuzzy expressions (including fuzzy propositions) or rules, and we consider particular nodes for defuzzification and aggregation. Thus, the changes propagate from input nodes to output nodes. The propagation stops if there are no changes during the evaluation of the current node.

The propagation is achieved as particular breadth-first traversal of the graph. However, for a fuzzy relation of cardinality n, it is necessary to assess its n



(a) Tree-based representation (b) Graph-based representation

Figure 2: Representations of a base of two Mamdani-type rules.

predecessors before its own evaluation, otherwise it would be evaluated n times, and worst, at a certain time, its value would be inconsistent. To avoid this effect, we added a priority information to the nodes. Before starting the fuzzy inference engine, the graph is traversed and a recursive function $priority : Node \rightarrow integer$ is applied. Let N be the current node to be treated, the function $priority$ is defined as follow:

- if N is an input node, then $priority(N) = 0$,
- otherwise, let s_i be the direct successors of N:

$$priority(N) = \max_i(priority(s_i)) + 1.$$

Let X, Y and Z be three input linguistic variables, and A, B, C a term from respectively X, Y, Z. Let D_1 and D_2 be two terms of an output linguistic variable O. Then, the rule base is composed of two rules:

- `IF (`X `is` A `AND` Y `is` B`) OR NOT` Z `is` C `THEN` O `is` D_1,
- `IF (`X `is` A `AND` Y `is` B`) OR` Z `is` C `THEN` O `is` D_2.

In Figure 2(b), numbers in brackets represent the evaluation priority of each node; the three inputs are at the bottom of the figure and have a null priority,

which means they need to be evaluated first. We will develop in section 3.6 the use of the priority during evaluation.

To the best of our knowledge, current fuzzy expert system does not implement CSE. This is due to the fact that they only use classical fuzzy logic operators which are really fast to compute. For instance, t-norms and t-conorms use arithmetic operators and binary comparisons (Zadeh's operator min and max), whose complexity is $O(1)$, whereas most of temporal operators complexity is in $O(l)$ where l is a number of samples used [23] and most of spatial operators are at least in $O(n \times m)$ where $n \times m$ is the dimension of the image [24].

We can easily assess the number of nodes in the two representations of the rule base. Let N_t be the number of nodes in the tree-based one and N_g in the graph-based one :

$$\begin{aligned} N_t(n,p) &= \underbrace{n}_{inputs} + \underbrace{1}_{aggregation} + \underbrace{1}_{defuzzification} \\ &\quad + \underbrace{p^n}_{rules}\Big(\underbrace{p}_{propositions} + \underbrace{(n-1)}_{conjonctions} + \underbrace{1}_{implication}\Big) \end{aligned} \tag{1}$$

$$\begin{aligned} N_g(n,p) &= \underbrace{n}_{inputs} + \underbrace{n \times p}_{propositions} + \underbrace{\sum_{i=2}^{n} p^i}_{conjonctions} \\ &\quad + \underbrace{p^n}_{implications} + \underbrace{1}_{aggregation} + \underbrace{1}_{defuzzification} \end{aligned} \tag{2}$$

Using Landau notations, these equations show the number of nodes is asymptotically $O((p+n) \cdot p^n)$ for the tree-based rule base, and $O(p^n)$ for the graph-based one.

3.3. Expiration

Among the sophisticated relations we have implemented, temporal operators [23] and those which derived from them need a special attention when applied on event streams. The particularity of event streams is that the system is noticed of events irregularly. For instance, let us consider the fact "the temperature was

too hot on Monday from 2 am to 3 am". The system has received two events: at 2am, a temperature high enough to activate the fuzzy proposition "the temperature is too hot", and at 3am, a lower temperature such as "the temperature is too hot" is false. Now, we consider the temporal operator "occurrence" from [22] which indicates that a phenomenon has occurred on a certain scope in the past: for instance, it can express that "the temperature was too hot during the last 24 hours". Until the next Tuesday 3 am, the degree of truth of this occurrence is strictly greater than 0. After 24 hours, its degree of truth equals 0, whereas the system inputs have not changed since Monday 3 am.

Classical fuzzy expert systems cannot perform this trick since they need that inputs change to compute the outputs. We thus introduce in our system the notion of expiration. Some expressions in the rule base are represented by special nodes in the rule base graph, which are marked as "expirable". After being evaluated, the expirable nodes signal to the scheduler they must be evaluated again after a certain delay (see section 3.5). If an input which is connected to an expirable node changed before this delay, the expiration is simply postponed by the scheduler.

Thus, expirable components must provide an expiration frequency and a set of criteria to stop the expiration. The expiration frequency is a parameter which depends on the application and the set of criteria depend only on the definition of the operator. For instance, in home health patient monitoring applications, expressions usually expire every day for symptoms persistence, because doctors cannot have alerts that change too frequently, whereas in control applications, the expiration rate is usually less than 1 second to have a smooth behavior. More details can be found in [23].

3.4. Active input queue

Sensor networks are a particular case of information stream. Some sensors measure (data stream) and some others detect (event stream), but they usually work in an asynchronous way. Moreover, some delays can appear in such networks. The active input queue is thus in charge of several steps before the

engine can process them.

Firstly, it listens to the information stream to fetch the interesting values it contains. Then, it groups the input values by timestamp and enqueue. Finally, it signals the scheduler that a new group has been enqueued.

Different policies can be conceived for this component. For instance, in some applications, it is necessary to wait for delayed sensors or delayed network packets before signaling the scheduler. Conversely, it can ignore delays and late arrivals, and thus filter these data. It may also be seen as a firewall which protects the scheduler from irrelevant inputs.

3.5. Scheduler

The scheduler has an important role to play to limit the delay between the arrival of the data and the decision making. When a new input set is announced, it decides, regarding its own policy, whether it is important to compute it immediately, later or not at all.

In the simplest configuration, the scheduler just fetches the first element in the active input queue, asks the evaluator to assess this group of inputs and gives the results to the broadcaster. With the use of policies, his behavior can be more sophisticated. For instance, one particular configuration can monitor the system to determine how busy the CPU cores are and to decide whether a group of inputs can be skipped. Moreover, the scheduler implements the expiration. All the expirable components of the rule base whose evaluation has changed are placed in another queue, waiting to expire.

Another configuration may consist in evaluating on different processor cores of the machine. Each core receives a sub-part of the input set. A simple algorithm based on the graph representation of the rule base is used to separate independent inputs on different sub-parts: this may simply be achieved by finding connected components of graph with well-known algorithms of graph theory [38].

3.6. Evaluator

The evaluator is the component which evaluates the different expressions and rules in the rule base. For a set of inputs, it gives a particular set of outputs. It also takes advantage of the rule base representation to perform the computation only when necessary.

To compute the different nodes of the graph representing the rule base, the evaluator traverses the graph in a certain order. To ensure the right order, we use a priority queue Q. The priority queue Q places the nodes with the lowest priority at the front of the queue and can contain each node at most once. Algorithm 1 presents the general evaluation algorithm. The algorithm takes four parameters: I, a dictionary which maps each input which has changed to its value, E, a set of nodes which has expired and must be evaluated again, M, a dictionary which maps each node of the graph to its value, and finally G, the rule base graph. Eventually, I or E can be exclusively empty: when I is empty, the procedure is called just to evaluate the expired nodes, whereas when E is empty, it is called to evaluate the consequences of the input change. The *assess* function takes the node to evaluate and M : it fetches the operands values in M and applies the operator, then stores the value of the current node in M and finally returns $false$ if the value of the current node has not changed, *true* otherwise.

In figure 2(b), the priority queue ensures the node "(X is A AND Y is B) OR Z is C" is evaluated at the right time. It ensures that if several paths lead to the same node N, all nodes on the paths are assessed before N.

In fuzzy logic, different functions can be used for operators (conjunction, disjunction, negation, implication, aggregation, defuzzification) evaluation. The policies of the evaluator indicate which version of the operators must be used.

3.7. Output change broadcast

The broadcaster is also an important module because it is in charge of building the output stream. The last step is indeed to inform on the fly the calling

Algorithm 1 Evaluation of the rule base graph

Inputs:

▷ *I: dictionary mapping nodes and values representing inputs which have just changed*

▷ *E: set of expired nodes*

▷ *M: dictionary mapping nodes and values resulting of the previous evaluation*

▷ *G: rule base graph*

/* Initializes the priority queue Q */

Q $\leftarrow \emptyset$

/* Adds changed inputs into Q and update memory */

for all pair $\langle node, value \rangle$ in I **do**

  Q $\leftarrow$ priority_enqueue(Q, $node$, priority($node$))

  M $\leftarrow M \bigcup \langle node, value \rangle$

end for

/* Adds expired nodes into Q */

for all node n in E **do**

  $Q \leftarrow$ priority_enqueue(Q, n, priority(n))

end for

/* Rule base graph browsing */

while $Q \neq \emptyset$ **do**

  current $\leftarrow$ priority_first(Q)

  $Q \leftarrow$ priority_dequeue(Q)

  if assess($current$, M) **then**

    for all n such as $n \in$ successors($current$, G) **do**

      $Q \leftarrow$ priority_enqueue(Q, n, priority(n))

    end for

  end if

end while

return M

system or the user that some outputs have changed. The policies are used to determine when and how the outputs have to be broadcast. For instance, the changes can be gathered and sent at regular time intervals or only outputs which have changed are broadcast with their new values. In a more verbose style, the changes can be sent with a trace of the activated rules.

It may gather information from the graph and the evaluation of its node to build justifications (to explain why the decision has been made).

The next section introduces implementation considerations and shows some experiments we achieved to characterize the performances of the software on both data and event stream processing.

4. Implementation and experiments

The software has been developed in C# as an API (Application Programming Interface). This language is not compiled as C++, but it offers a good compromise between efficiency and the ease of interfacing with other systems (for instance by webservices) and runs on different platforms without recompiling.

The succession of the nodes evaluation has been implemented without any optimization regarding the current fuzzy inference methods : the software performs all the steps needed in Takagi-Sugeno or Mamdani type inferences. We wanted to be as general as possible to be able to take into account future or experimental inference methods. Moreover, all calculations involved in the different inferences (integrals, curves intersection, ...) are processed analytically, in opposition with, for instance, Matlab which represents a membership function by a given number of points.

In terms of memory usage, for now, all the rules are kept in memory at all time. We also have to store the values of the different nodes of the rule base. To implement efficiently expiration, which affects only some temporal operators [23], the values of expirable expression operands are stored in memory regarding a temporal scope to allow partial recalculation. Indeed, it is necessary to find

a good compromise between efficiency, scalability and flexibility regarding our industrial partners and our needs to experiment new operators and inference methods.

Implementation is still a work in progress to improve performances. For instance, from the results in [39], we improved the efficiency of the hash function and some data structures and divided by up to 170 the computation time of large rule bases. From the results in [40], we also improved memory usage in order to load larger rule bases.

Without loss of generality, we have then experimented the system in two different ways: either with a data stream or an event stream. On one hand, to test the ability of the system to process data streams, we measured the time to process rule bases, varying the number of inputs and terms to increase the number of rules at each test. On the other hand, we used an event simulator to test the event stream processing capabilities, which is used to benchmark CEP softwares. It is of course possible to address both event streams and data streams with the same rule base.

In the first series of experiments, all the inputs are changing at each time whereas in the second series of experiments, few inputs change simultaneously to reveal the potential of our system in real world applications of stream processing.

These tests have been processed on only one core of an Intel Xeon X5650 at 2.67GHz on a Windows server and 42GB of RAM to process several tests in parallel without interfering.

4.1. Data stream experiment

This experiment aims at comparing the performances of the evaluation of different rule bases regarding two different policies of the evaluator module :

- full recalculation mode : all the expressions and nodes are reassessed each time an input changes,
- partial recalculation mode : last values of the nodes are kept in memory and are reassessed only when needed,

and different rule base representations:

- tree-based representation (without CSE),
- graph-based representation (with CSE).

The graph based representation with partial recalculation is the default behavior of our system. Its modularity, through the use of policies, allows to easily switch between the different modes. The full recalculation mode is obtained by clearing M before each call of the evaluation procedure (algorithm 1) whereas the partial recalculation mode stores in memory the dictionary M.

4.1.1. Protocol

These experiments have been carried out on artificial rule bases and data sets whose generation is described hereafter. Let $\{v_i\}_{1 \leq i \leq n}$ be n input linguistic variables, each defined by p terms $T_i^1, \ldots, T_i^p$. Let w be an unique output linguistic variable whose terms are $W_1, ..., W_K$. Those input variables combine into rules by the full conjunctive combination principle :

$$\texttt{IF } v_1 \texttt{ is } T_1^{l_1} \texttt{ and } \ldots \texttt{ and } v_n \texttt{ is } T_n^{l_n} \texttt{ THEN } w \texttt{ is } W_k$$

where $T_i^{l_i}$ refers to a term of v_i with $1 \leq l_i \leq p$ and $k = \sum_{i=1}^{n} l_i - n + 1$. Thus, for a given couple (n, p), there are p^n possible combinations of those inputs (i.e. rules) and w has $K = n(p-1) + 1$ terms.

For the sake of simplicity, the terms $T_i^{l_i}$ of each variable v_i are defined by triangular membership functions on the domain $[0, p+1]$. By construction, the support of each term $T_i^{l_i}$ is $[l_i - 1; l_i + 1]$ and its kernel is $\{l_i\}$. The same construction is used for the terms W_k of w. Figure 3 shows an example of a linguistic variable characterized by 3 terms.

Each input variable v_i receives a data stream of 20 values, which have been generated following an uniform distribution $\mathcal{U}([0, p+1])$.

The architecture has been configured as follows: the active input queue is set in DSP mode, i.e. it waits to receive a value for each input. The scheduler evaluates this group as soon as possible, then the new value of the output is

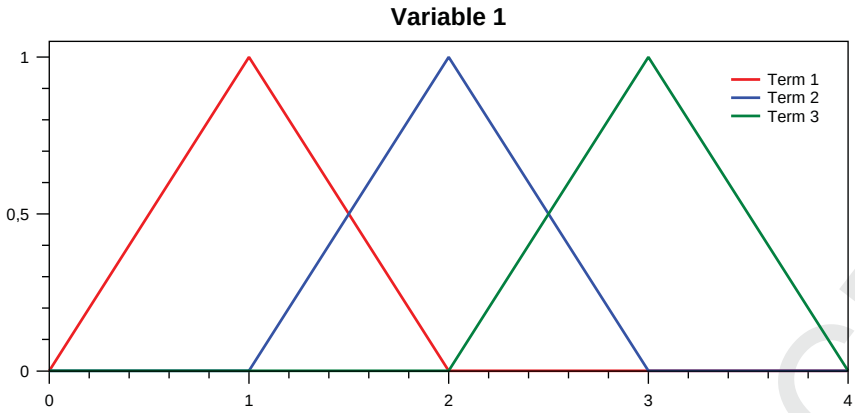


Figure 3: Linguistic variable with 3 terms defined on the domain [0, 4].

broadcasted. This is the most simple configuration of these modules. The different modes of evaluation of the architecture have been obtained by configuring the policy of the evaluator : in one case, it uses its memory functionality; in the other case, it has to compute all the values of the nodes again. The same input data streams have been used for both cases.

By making both the number of inputs n and the number of terms p vary from 2 to 10, we are able to assess the performance of the architecture on large rule bases and to draw some conclusions. Due to the computational cost, the largest configuration was obtained with 7 input variables and 9 linguistic terms with the graph-based representation (4782969 rules), whereas the tree-based representation allows to reach 9 input variables and 4 terms (262144 rules). Even if these are not realistic cases, it is useful to benchmark the proposed system.

4.1.2. Results

In this section, we first compare the average number of nodes being reevaluated in each mode and then compare the average evaluation time of the different rule bases. The averages are computed over the 20 values of the data stream to decrease the possible biases.

Figure 4 represents the number of nodes to be evaluated in each configuration and figure 5 represents the computation times regarding the number of rules. These figures show the results for the full and partial recalculation modes and

for the tree-based and graph-based representations of the rule base in log-scale. We can see that for the graph-based rule bases, both the computation time and the number of nodes are linear regarding the number of rules whereas for tree-based rule bases, it is weakly exponential. Point clouds in figure 5 confirm the intuition: storing the value of each node allows to stop propagating the changes, and strongly decreases the number of nodes to evaluate. For a rule base with 6 input variables and 8 terms (262144 rules), 1572920 nodes may be evaluated in the full recalculation mode with tree-based rule base and 561784 with a graph, whereas in the partial one, only 35248 nodes in average are evaluated for trees and 275 for graphs.

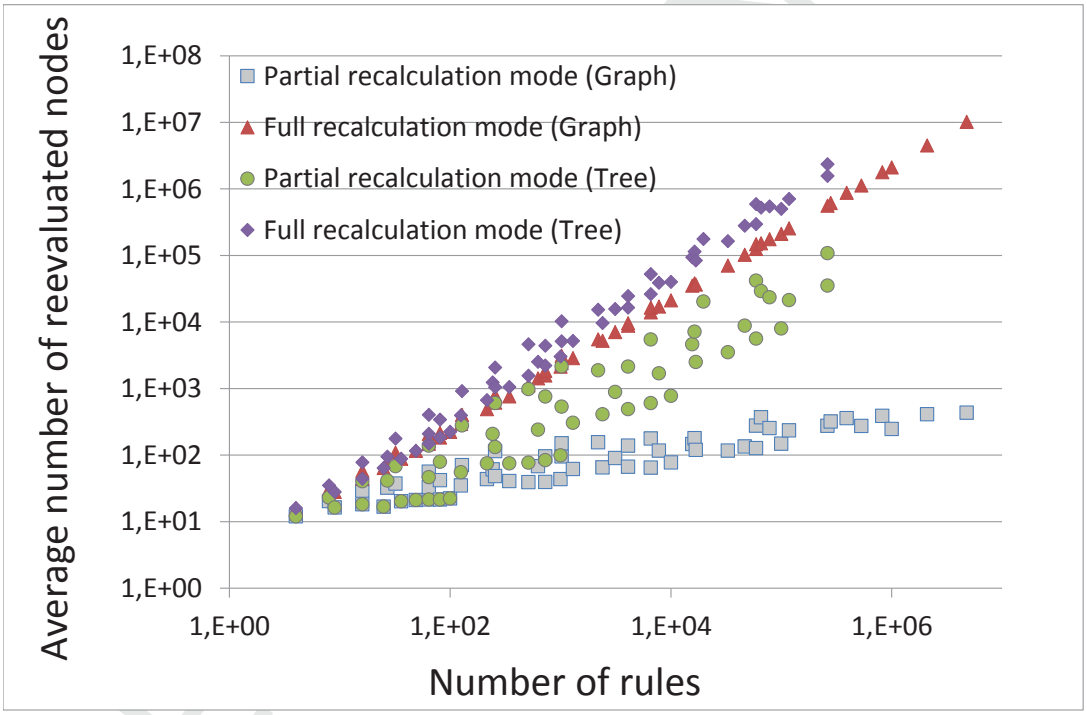


Figure 4: Average number of nodes to be evaluated for different rule base sizes in the all four modes (log-scale for both axes).

In particular, figure 5 shows the duration of the evaluation of the rule bases in full and partial modes with graph-based rule bases. With the same data streams as before, to evaluate 262144 rules, full recalculation mode needs almost 2s whereas the partial one needs only 300ms, i.e. the latter one is more than 6 times faster than the former one on this rule base structure.

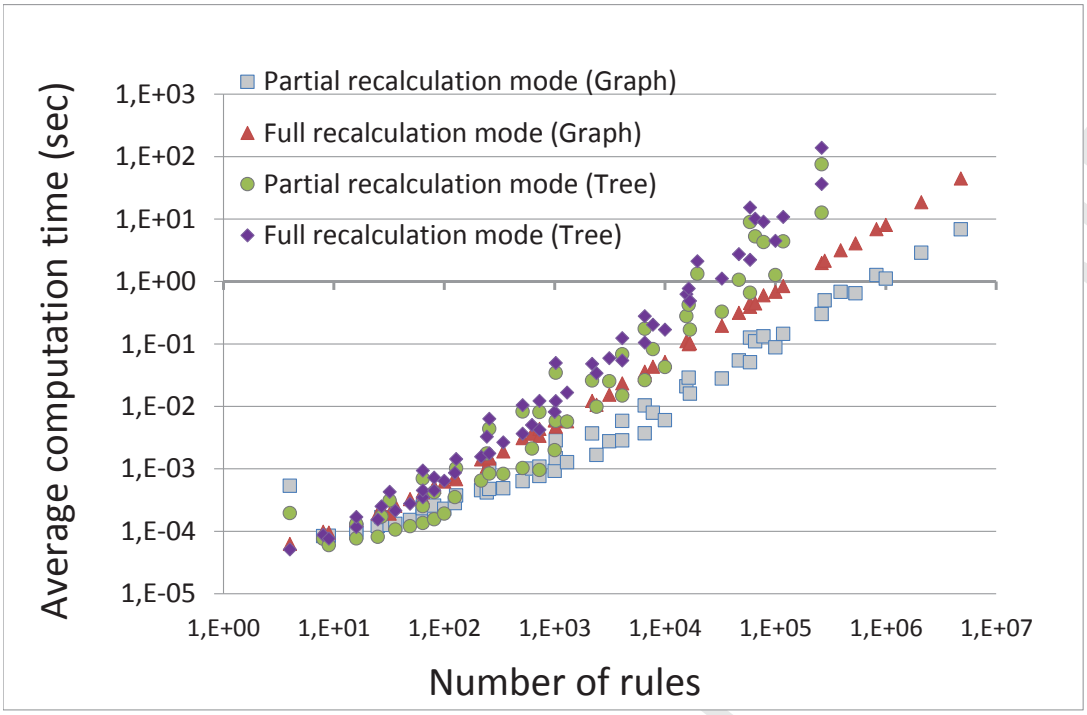


Figure 5: Average computation time in seconds for different rule base sizes in all four modes (log-scale for both axes).

4.1.3. Discusssion

The drastic reduction of the number of nodes to be evaluated can be explained by a theoretical analysis. The fuzzy partitions used to create the terms of the linguistic variables explain why, at each time, for each variable, at most 2 terms out of p are activated. Thus, at most $N_g(n, 2)$ nodes (equ. 2) have to be evaluated: for $n = 6$, at most 208 nodes will be activated. But a large number of them are null because of the conjunctive combination of the inputs. Now, to count the number of needed reevaluations, we should consider the worst case : all the active elementary propositions become null, and the same number of propositions get a non-null value. This gives $2 \times N_g(n, 2)$ as a pessimistic upper bound of the number of nodes that need to be reevaluated.

It seems that saved computational time is not as high as we could expect considering the saved computations shown just before. Indeed, saved computations correspond to the evaluation of a null value by a quite simple function (mainly either by the membership function evaluation or by a conjunctive combination of two expressions) and its affectation to nodes that were already null.

We will see in section 4.2 that partial recalculation mode is really helpful in event stream processing.

These tests are good stress tests because all the inputs change at the same time. For rule bases of conventional sizes, for instance 343 rules, the engine needs less than 0.5ms in the partial recalculation mode. Thus, we can handle inputs which change more than 2000 times per second on only one core.

4.2. Event stream experiment

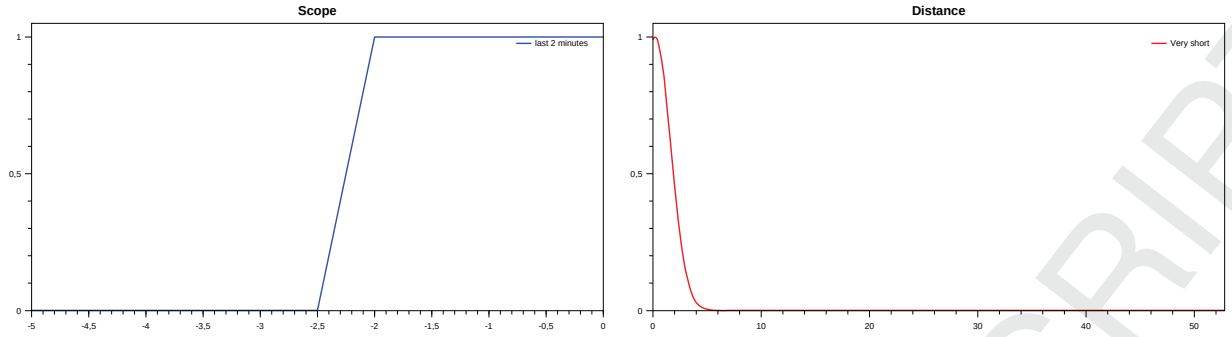
Streaming events poses some challenges to the design of a benchmark. As described in [41], the authors introduce "Linear Road", a benchmark for event stream processing systems which matches with many requirements of such benchmarks. For instance, we cite the two most important requirements in our case:

- the generated data must have a semantic validity,
- the benchmark must be verifiable (even if the streams may vary depending on the moment they are generated).

The linear road benchmark is inspired from the problem of variable tolling (also known as congestion pricing), i.e. the computation of tolls that vary according to different factors such as congestion levels and accident proximity [42]. The input data are generated by the MIT Traffic Simulator (MITSIM) [43]. The benchmark consists in different challenges for stream data management systems, from historical queries to tolls assessment. In our case, we have only implemented one challenge out five: the detection of car accidents. The other challenges are not appropriate for a fuzzy expert system and need the capability to query historical data or an aggregated view of the expressway.

4.2.1. Previous work

In the following experiment, we used the "strictly persists" temporal operator described in [23] and which evaluates how much a phenomenon persists during a given scope. In our case, the scope is fuzzy since some moments in



(a) Membership function for the fuzzy scope "the last 2 minutes"

(b) Membership function of a very short distance

Figure 6: Parameters used for the linear road benchmark.

the past are considered as more important. Figure 6(a) shows the membership function used to define the scope: it considers the values from 2.5 minutes to 2 minutes before the present as more and more important, and moments from 2 minutes to the present as important. This fuzzy scope allows testing a representative case for the temporal operator in terms of computational cost.

4.2.2. Protocol

We generated data with the MITSIM simulator [43] in a flat file which represents a simulation of 6 minutes and 40 seconds, involving 5953 cars. We then select only a given number of cars inside this simulation. The data consist in cars positions, emitted every 30 seconds on a bidirectional expressway. Each direction is composed of 3 travel lanes and one exit and one entrance ramps. The position is given by the direction, the lane number and an integer that represents a 1-D coordinate on a specific lane. In the original benchmark, a car accident is detected if at least two cars report the same position for at least four times. More details can be found in [41] and [43]. We developed a software to play the simulation in real time and another one to check if the accident notifications are true regarding the simulation. To use the expiration, we send the position of a car only if it changes from its previous position.

To match with a fuzzy problem, we changed the benchmark rules. Firstly, we used a fuzzy definition of the distance, to take into account the GPS inaccuracy

and the size of the car: figure 6(b) shows a narrow gaussian membership function which defines a short distance. Then, we decided to use the "strictly persists" temporal operator, described in section 4.2.1, and to develop two new nodes for the rule graph:

- a distance node which simply computes the absolute value of the difference between two inputs values,
- a "same lane" crisp node, which is boolean and indicates if two cars are on the same lane, same direction.

For each unique pair of distinct cars (car_i, car_j), we wrote the two following rules :

- IF (car_i is on the same lane as car_j AND distance between car_i and car_j IS very short) STRICTLY PERSISTS during at least the 2 last minutes THEN $Accident(i, j)$ IS true,
- IF NOT ((car_i is on the same lane as car_j AND distance between car_i and car_j IS very short) STRICTLY PERSISTS during at least the 2 last minutes) THEN $Accident(i, j)$ IS false.

For n cars on the expressway, we thus have $n^2 - n$ pairs of rules. When one car emits a position, it affects $2 \times (n - 1)$ rules which may be reevaluated. In particular, for the "strictly persists" operator, the different values of operand are kept in memory during the last 2.5min as a compressed signal : i.e., as we receive at most a position every 30s for each car, the signal for each pair of cars contains at most 10 values. However, we do not store the past values of the car positions.

During this test, we used the following configuration :

- Input queue groups inputs by timestamps with no delay,
- Scheduler processes on one core,
- Evaluator accepts temporal operators and use Zadeh norm and conorm,

Number of cars	Simultaneous events min	avg	max	Number of rules	Response time (ms)	Total computation time (s)
50	1	4.7	9	4900	9.38	1.8
100	2	5.6	9	19800	19.22	5.8
250	2	8	20	124500	317.57	5.8
400	2	12.4	24	319200	806.51	21.9
500	2	15.2	24	499000	836.56	40.1
750	2	21.7	37	1123500	847.53	40.5
1000	2	27.2	43	1998000	862.6	41.5
1100	2	29.2	48	2417800	912.37	41.9
1200	2	31	50	2877600	880.11	41.4
1300	2	32.5	53	3377400	808.65	37.2

Table 1: Results of the partial linear road benchmark performed on a scenario of 6'40"

- Broadcaster indicates the value of an output only when it changes,
- The persistence expires every ten seconds.

When an output changes from false to true, we record the time and the value to indicate that an accident occurs. Note that in the case more than two cars are involved in the accident, more than one output will change: for instance, if an accident occurred between car_1, car_2 and car_3, we will be noticed that an accident happened between car_1 and car_2, car_1 and car_3, car_2 and car_3 (thus, 3 notifications for the same accident).

4.2.3. Results

Table 1 shows the results of the car accident detection in the linear road benchmark. We iterate the tests with different number of cars, from 50 to 1300. The table characterizes the number of simultaneous inputs that change during the simulation by the minimum, the average and the maximum values. It then shows the number of involved rules, the response time, i.e. the average delay to evaluate the rule base in milliseconds, and finally the total time of computations during the simulation.

For instance, with 1300 cars, in average 32.5 cars report their positions simultaneously. 808.65ms were necessary to tell if an accident happened or not.

The total computation time lasts only 37.2s over 401s of simulation.Thus, the system was evaluating the rules during less than 10% of the total duration of the simulation.

The results show that, thanks to partial recalculation mode, the number of rules does not impact so much on the response time. Naturally, the latter is more impacted by the number of simultaneous events.

4.2.4. Discussion

In this benchmark, regarding the current state of the software, we were limited by two main factors. First, the number of simultaneous events is always low because of the nature of the simulation. Indeed, to the best of our comprehension, to increase the number of simultaneous events in MITSIM, we have to increase the number of cars. However, the number of rules grows too fast regarding the number of cars. This leads to the second limitation : the construction of such large rule bases. We have not been able to deal with more than 1300 cars because of the time needed to create the rule base. Indeed, in our architecture, to apply common subexpression elimination on rule bases, we need to check if a subexpression has been created before. Even with hashing functions, this step implies a certain amount of computations. Despite this cost, we have to remind that without CSE, we could not evaluate such large rule bases.

To better handle the linear road benchmark, systems may consider dynamic rule bases (when a new car appears on the express way, a new set of rules is added, and they are removed whenever the car disappears) or to filter the rules to evaluate (e.g. the rules concerning two cars are created only if the two cars are close enough).

The goal of this experiment was to study the behavior of the system on large rule bases for event streams. In the case we would want to address this benchmark fully, we should use a multi-core scheduler and higher level rules, like "*FOR ALL unique pair of cars* (x, y), IF ... THEN ...": we will thus have only two rules in memory for all the pair of cars, while keeping the performances of the inference engine.

5. Conclusion

In this paper, we have presented a modular architecture for a fuzzy expert system designed to handle information streams (data streams or event streams). The architecture relies on two aspects. Firstly, the graph representation of the rule base indicates the dependency between inputs, expressions, rules and outputs. More generally, it indicates what must be computed and in which order. Secondly, the use of four cooperating modules permits to filter and to decide when it is possible to process a set of inputs. The introduction of policies in the four modules allows to customize their behaviors regarding the addressed projects or issues. Moreover, the flexibility of the rule base representation has been shown by the addition of two ad-hoc types of node in the graph ("distance" node and "same lane and direction" node).

The described architecture has been implemented and used in several industrial projects in different domains: home automation, decision making in industry and home care services. All projects needed to process either data stream or event stream, sometimes both of them at the same time.

Uncertainty and imprecision are real-world challenges, but others emerge. The different experiments that have been presented in this paper show some limitations of the current system: the usage of only one core of the processor and the necessity to load a potentially huge number of rules. Considering CEP and several thousands of inputs per second, we should parallelize the computations. We should also consider higher level rules that could be applied efficiently to a lot of inputs while keeping the performances of the inference engine. Moreover, users need more fuzzy relations to be able to describe their scenarios or to characterize what they want to extract from the streams. Finally, online rule base optimization will allow users to sketch first rules and then let the system evolve.

References

[1] G. Cugola, A. Margara, Processing flows of information: From data stream to complex event processing, ACM Comput. Surv. 44 (3) (2012) 15:1–15:62.

[2] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, J. a. Gama, Data stream clustering: A survey, ACM Comput. Surv. 46 (1) (2013) 13:1–13:31.

[3] M. R. Henzinger, P. Raghavan, S. Rajagopalan, Computing on data streams, in: J. M. Abello, J. S. Vitter (Eds.), External Memory Algorithms, American Mathematical Society, Boston, MA, USA, 1999, Ch. Computing on Data Streams, pp. 107–118.

[4] S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams: Theory and practice, IEEE Trans. on Knowl. and Data Eng. 15 (3) (2003) 515–528.

[5] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom, Stream: The stanford data stream management system, Technical Report 2004-20, Stanford InfoLab (2004).

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, M. A. Shah, Telegraphcq: Continuous dataflow processing, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD '03, ACM, New York, NY, USA, 2003, pp. 668–668.

[7] M. Khalilian, M. Norwati, Data stream clustering: Challenges and issues, Proceedings of International Multi Conference of Engineers and Computer Scientists (2010) 566–569.

[8] D. Cardoso, M. De Gregorio, P. Lima, J. Gama, F. França, A weightless neural network-based approach for stream data clustering, in: H. Yin, J. Costa, G. Barreto (Eds.), Intelligent Data Engineering and Automated

Learning - IDEAL 2012, Vol. 7435 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 328–335.

[9] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, J. Stefanowski, Open challenges for data stream mining research, SIGKDD Explor. Newsl. 16 (1) (2014) 1–10.

[10] S. Muthukrishnan, Data streams: Algorithms and applications, in: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '03, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003, pp. 413–413.

[11] D. C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[12] E. Wu, Y. Diao, S. Rizvi, High-performance complex event processing over streams, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2006, pp. 407–418.

[13] M. Stonebraker, U. Çetintemel, S. Zdonik, The 8 requirements of real-time stream processing, SIGMOD Rec. 34 (4) (2005) 42–47.

[14] E. Alevizos, A. Skarlatidis, A. Artikis, G. Paliouras, Complex event processing under uncertainty: A short survey, in: P. M. Fischer, G. Alonso, M. Arenas, F. Geerts (Eds.), Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference, Vol. 1330 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 97–103.

[15] A. Artikis, C. Baber, P. Bizarro, C. Canudas-de Wit, O. Etzion, F. Fournier, P. Goulart, A. Howes, J. Lygeros, G. Paliouras, A. Schuster, I. Sharfman, Scalable proactive event-driven decision making, Technology and Society Magazine, IEEE 33 (3) (2014) 35–41.

[16] L. Zadeh, Fuzzy sets, Information and Control 8 (3) (1965) 338 – 353.

[17] J. M. Garibaldi, Do Smart Adaptive Systems Exist? Best Practice for Selection and Combination of Intelligent Methods, Springer, 2005, Ch. Fuzzy expert systems, pp. 105–132.

[18] W. Siler, J. Buckley, Fuzzy expert systems and fuzzy reasoning, Wiley-Interscience, 2005.

[19] K. Basterretxea, J. M. Tarela, del Campo I, G. Bosque, An experimental study on nonlinear function computation for neural/fuzzy hardware design, IEEE Transactions on Neural Networks 18 (1) (2007) 266–283.

[20] A. Laurent, M. J. Lesot (Eds.), Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design, Information Science Reference, 2010.

[21] G. Acampora, V. Loia, Fuzzy control interoperability and scalability for adaptive domotic framework, IEEE Transactions on Industrial Informatics 1 (2) (2005) 97–111.

[22] P. Cariñena, A. Bugarín, M. Mucientes, S. Barro, A language for expressing fuzzy temporal rules, Mathware and Soft Computing 7 (2-3) (2000) 213–227.

[23] J.-P. Poli, L. Boudet, D. Mercier, Online temporal reasoning for event and data streams processing, IEEE Conference on Fuzzy Systems, FUZZ-IEEE (2016) 2257–2264.

[24] I. Bloch, Fuzzy spatial relationships for image processing and interpretation: a review, Image and Vision Computing 23 (2) (2005) 89 – 110.

[25] S. Schockaert, M. D. Cock, E. Kerre, Reasoning about fuzzy temporal and spatial information from the web, World Scientific, 2010.

[26] J.-M. Le Yaouanc, J.-P. Poli, A fuzzy spatio-temporal-based approach for activity recognition, in: Advances in Conceptual Modeling, Vol. 7518 of Lecture Notes in Computer Science, 2012, pp. 314–323.

[27] W. E. Combs, J. E. Andrews, Combinatorial rule explosion eliminated by a fuzzy rule configuration, Trans. Fuz Sys. 6 (1) (1998) 1–11.

[28] J. Mendel, Q. Liang, Comments on "combinatorial rule explosion eliminated by a fuzzy rule configuration", IEEE Transactions on Fuzzy Systems 7 (3) (1999) 369–373.

[29] M. Sugeno, K. Tanaka, Successive identification of a fuzzy model and its applications to prediction of a complex system, Fuzzy sets and systems 42 (3) (1991) 315–334.

[30] C.-T. Sun, Rule-base structure identification in an adaptive-network-based inference systems, in: IEEE Transaction on Fuzzy Systems, Vol. 2, 1994, pp. 64–73.

[31] Y. Jin, Fuzzy modeling of high-dimensional systems: Complexity reduction and interpretability improvement, Trans. Fuz Sys. 8 (2) (2000) 212–221.

[32] B. Kosko, Optimal fuzzy rules cover extrema, International Journal of Intelligent Systems 10 (2) (1995) 249–255.

[33] I. Rodríguez-Fdez, M. Mucientes, A. Bugarín, S-FRULER: Scalable fuzzy rule learning through evolution for regression, Knowledge-Based Systems 110 (2016) 255–266.

[34] L. Reznik, Fuzzy Controllers Handbook, Newnes, 1997.

[35] K. Basterretxea, I. Del Campo, Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design, Information Science Reference, 2010, Ch. Electronic Hardware for Fuzzy Computation, pp. 1–30.

[36] N. Harvey, R. H. Luke, J. M. Keller, D. Anderson, Speedup of fuzzy logic through stream processing on graphics processing units, IEEE Congress on Evolutionary Computation (2008) 3809–3815.

[37] B. R. Preiss, Data Structures and Algorithms with Object-Oriented Design Patterns in Java, Worldwide Series in Computer Science, Wiley, 2000.

[38] J. Hopcroft, R. Tarjan, Algorithm 447: Efficient algorithms for graph manipulation, Commun. ACM 16 (6) (1973) 372–378.

[39] J.-P. Poli, L. Boudet, Une architecture moderne de système expert flou pour le traitement des flux d'information, Rencontres Francophones sur la Logique Floue et ses Applications, 2015.

[40] J.-P. Poli, L. Boudet, A modular fuzzy expert system architecture for data and event streams processing, in: IPMU, 2016, pp. 717–728.

[41] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbetts, Linear road: A stream data management benchmark, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04, VLDB Endowment, 2004, pp. 480–491.

[42] A. D. Palma, R. Lindsey, Traffic congestion pricing methods and technologies, Tech. rep., Ecole Polytechnique (2009).

[43] Q. Yang, H. N. Koutsopoulos, A microscopic traffic simulator for evaluation of dynamic traffic management systems, Transportation Research Part C: Emerging Technologies 4 (3) (1996) 113 – 129.