Applied Intelligence (2022) 52:15143–15163
https://doi.org/10.1007/s10489-022-03256-2

Bat4CEP: a bat algorithm for mining of complex event processing rules

Ralf Bruns[1] · **Jürgen Dunkel**[1]

Accepted: 15 January 2022 / Published online: 11 March 2022

Abstract
Complex Event Processing (CEP) is a modern software technology for the dynamic analysis of continuous data streams. CEP is able of searching extremely large data streams in real time for the presence of event patterns. So far, specifying event patterns of CEP rules is still a manual task based on the expertise of domain experts. This paper presents a novel bat-inspired swarm algorithm for automatically mining CEP rule patterns that express the relevant causal and temporal relations hidden in data streams. The basic suitability and performance of the approach is proven by extensive evaluation with both synthetically generated data and real data from the traffic domain.

Keywords Bat algorithm · Swarm algorithm · Rule learning · Complex event processing

1 Introduction

Today, companies as well as public and private organizations face the challenge of dealing with huge amounts of data, the volume of which has grown enormously in the last decade and will continue to rise rapidly in the future. Smartphones, sensor networks, industry 4.0, the internet of things, or information from social networks are causing an increasing flood of data. One particular challenge is the increasing need to evaluate *streams* of continuously arriving data.

The future viability of an institution strongly depends on its ability to extract the significant information value for its core business from the volume of data and to derive qualified decisions from it.

Complex Event Processing (CEP) is a software technology for the dynamic analysis of massive data streams in real-time [1, 2]. CEP allows the specification of *situations of interest* in terms of *event patterns* to express causal, temporal, spatial and other relationships between data elements (called *events* in CEP). The data streams must be continuously examined for these event patterns (*event pattern matching*) in order to detect an occurrence of a relevant situation. In CEP, event patterns are expressed by means of user-defined rules, formulated in a so-called *Event Processing Language* (EPL). An EPL is a Domain-Specific Language (DSL) [3, 4] tailored to model CEP rules on a user-friendly level of abstraction.

So far, specifying rule patterns is still a manual task based on the expertise of domain experts. But defining event patterns requires a very deep understanding of the application domain. In particular, knowledge about the relevant events, their temporal dependencies, and the relations between selected event attributes must be taken into account. When an application domain is characterized by highly dynamic changes then user-defined rules may be even impossible. Machine learning approaches for automatic rule pattern extraction is still an open research problem [5, 6].

Bat4CEP is a novel approach for mining CEP rules that express the relevant causal and temporal relations hidden in data streams. We propose an innovative bat-inspired swarm algorithm for automatically defining CEP rules. The algorithm utilizes an advanced syntax tree encoding of candidate rules, taking into account the specific language constructs of common EPLs. Based on previous solution candidates, new candidate rules are derived in a probabilistic and swarm-based manner. In the course of numerous

✉ Ralf Bruns
ralf.bruns@hs-hannover.de

Jürgen Dunkel
juergen.dunkel@hs-hannover.de

[1] Department of Computer Science, Hannover University of Applied Sciences and Arts, Ricklinger Stadtweg 120, 30459 Hannover, Germany

iterations, better and better candidate rules are to be generated by adaptating bat-specific random flight and local search operators. The performance of the approach is proven in extensive experiments with synthetically generated data as well as with real data from the traffic domain.

The rest of the paper is organized as follows: The following two sections define the problem of learning CEP rules and discuss the related work. The fourth section introduces the basic concepts of bat algorithms. Section 5 presents the Bat4CEP approach with all its building blocks. Then, Section 6 provides the results of experimental evaluation and discusses some implementation issues. The last section summarizes the most significant features of the approach and provides a brief outlook to future research.

2 Problem of mining CEP rules

The objective of our approach is to reveal the hidden causal and temporal relationships between the events in a data stream that lead to a specific situation of interest [7].

In CEP, the meaning of events is expressed in so-called *event processing rules* (CEP rules) specified in a declarative *Event Processing Language* (EPL).

A rule consists of two parts: (i) the condition part specifies declaratively an event pattern that corresponds to a *situation of interest*; (ii) the action part determines the operation to be performed in this particular situation (e.g. simply notifying the user) [8].

Well-known professional open source CEP rule engines are Esper[1] and Siddhi[2].

For example, let us assume that a production machine permanently emits data about its operational status. A situation of interest would be a machine breakdown that should desirably be predicted before it happens. What we are looking for is a CEP rule that describes the relation between the observations (e.g. sensor measurements, operational data) and the situation that should be detected (e.g. machine breakdown).

CEP rules are usually formulated manually by domain experts. However, domain experts often do not know precisely the underlying dependencies in the data. Moreover, in many application domains, there are no experts available who could formulate a rule.

The objective of our approach is to automatically learn CEP rules out of historical data streams, depicted in phase 1 of Fig. 1. A machine breakdown could be concluded by the operational data provided by the machine. For instance, if a temperature warning event is emitted from a machine part ($ValveX3$) with a temperature higher than 70 and, followed within 1min, a water supply problem event is detected for the same machine part then a machine breakdown event might be imminent. A prerequisite for the learning approach is that the situation of interest, here the machine breakdown, is marked as a *label* at the appropriate position in the historical data.

Once an event pattern has been learned, a corresponding CEP rule can be derived to analyze a live data stream in order to predict the occurrence of a situation of interest before it actually happens (phase 2 in Fig. 1). Automatic rule learning is desirable and can lead to new and unexpected insights when it comes to finding implicit dependencies in the data that are still unknown to experts.

Although the various CEP engines and their EPLs differ considerably, common features can be identified in the event models used and the operators applicable for formulating the event patterns [8, 9]:

- *Event types and attributes:* A data stream consists of a temporal sequence of *event instances* each belonging to a certain *event type*. Each event type defines the set of *attributes* allowed for all its instances. E.g., event type $TemperatureWarning$ contains the two attributes $name$ and $temp$. Alias names are defined with the keyword as in order to distinguish several event instances of the same type.
- *Operators on event types:* The events in an event pattern can be correlated by different operators: $\wedge$, $\vee$, $\rightarrow$, $\neg$. In Fig. 1, the event condition ($TemperatureWarning \rightarrow WaterSupplyEvent$) is formulated (i.e. an event of type $TemperatureWarning$ is followed in time by an event of type $WaterSupplyEvent$).
- *Operators on attributes:* Conditional expressions on attributes can be specified by common logical ($\wedge$, $\vee$, $\neg$) and numerical comparison and arithmetic operators: $>$, $<$, $=$, $+$, $-$, etc. In Fig. 1, the attribute $t.name$ can be dereferenced to the instance of an event of type $TemperatureWarning$ using its alias name t. The attribute condition defines that the $name$ attribute of event instance t and the $name$ attribute of event instance w must have the same value and, in addition, the $temp$ attribute of t must have a value greater than the threshold 70.
- *Aggregation functions:* Aggregation functions combine the attribute values of a defined set of events, e.g. all events that occurred over a certain time period, such as all $TemperatureWarning$ events in the last 5 minutes. Typical aggregation functions are: $avg()$ calculates the average value of an attribute value, $sum()$ calculates the sum of an attribute value, $min()$ and $max()$ return the minimum and maximum of the attribute value, respectively.

[1] http://www.espertech.com/esper/

[2] https://github.com/siddhi-io/

- *Sliding windows:* A sliding window restricts the number of events to excerpts from the event stream. The size of a sliding window is defined by its *type* and *value*. Two different window types can be distinguished: A *length window* of length *n* considers only the last *n* events of the stream. A *time window* considers only those events that occurred in the defined time period. For instance, in Fig. 1, a time window is defined to include all events that occurred in the last 1 minute, specifying that a *WaterSupplyEvent* must follow a *TempWarning* within 1 minute.

The Bat4CEP approach presented in the following is designed to learn CEP rules comprising the most common language constructs for event processing rules.

3 Related work

This section discusses previous swarm-based metaheuristics for learning association rules as well as the state of the art in learning CEP rules.

Association analysis and association rule mining (ARM) is a well-established approach in machine learning. An association rule describes correlations between a set of items that occur together. Thus, the objective of an association analysis is to determine items that imply the occurrence of other items in a transaction.

Mining CEP rules can be considered as a more complex form of association rule mining. CEP rules offer a more complex rule structure than association rules, because they rely on a more expressive EPL (see discussion in Section 2).

Moreover, with grammatical inference another problem related to the problem of rule learning is discussed.

3.1 Swarm-based metaheuristics learning of association rules

Next to conventional techniques like *Apriori* [10], *FP-Growth* [11], or *C4.5* [12] several proposals attempt to tackle the problem of learning association rules by swarm-based optimization algorithms. Within these algorithms the rule classification problem is formulated as an objective function and the swarm-based algorithm is used to optimize this function, i.e. to generate as good association rules as possible.

Several approaches apply particle swarm optimization (PSO) [13–15] where each particle represents a solution candidate (either a single association rule or a rule set).

Ant colony algorithms belong to the most popular swarm algorithms. The *AntMiner* system [16] applies the ant algorithm for ARM, where each path traversed by an ant represents a solution candidate (= an association rule). Some extensions of the AntMiner approach have been published [17, 18].

The positive learning results achieved by the different swarm algorithms prove that swarm-based metaheuristics are a promising technique for advanced rule learning tasks.

3.2 Bat algorithm for classification and association rule learning

Although bat-inspired algorithms are a rather new metaheuristic, they have already been applied in early approaches to classification problems and rule mining. Khan and Sahai [19] propose a bat algorithm in combination with fuzzy logic for clustering. Damodaram and Valarmathis [20] use bat metaheuristic for the classification of phishing website by association rules. Mining association

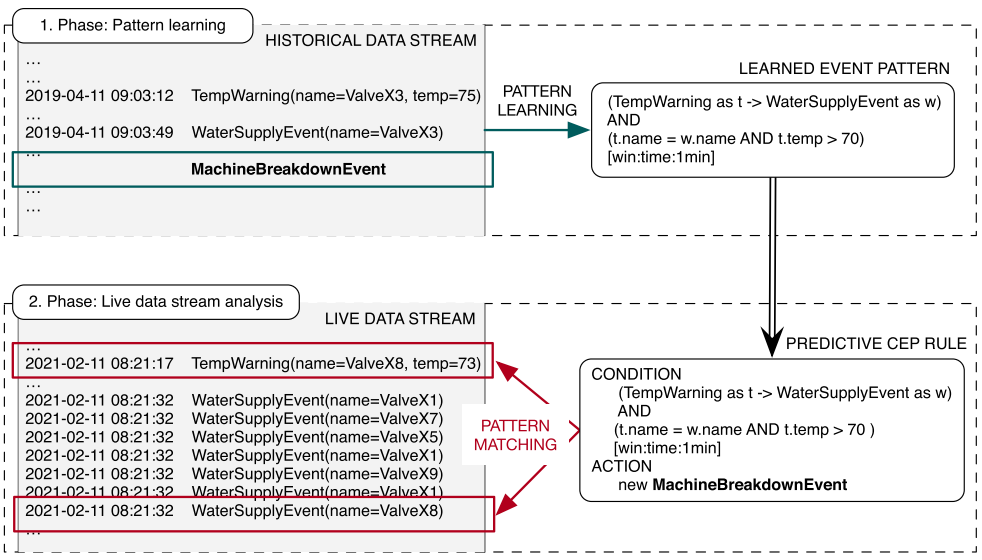

Fig. 1 Learning problem of CEP rules

rules by bat algorithm is investigated in the work of Song et al. [21] and Heraguemi et al. [22, 23]. The positions of the bats represent the candidate rules on which specialized actualization operators are applied.

3.3 Grammatical inference

Grammatical inference is a problem related to the problem of rule learning. The objective of grammatical inference is to infer a grammatical structure from a set of examples, i.e., to identify the correct syntax of a language or, with its extension semantic inference, also to derive the semantics of a language [24]. Different machine learning methods [24] as well as evolutionary algorithms/ genetic programming [25] or memetic algorithms [26] have been applied to this problem. However, the CEP rule learning problem has a different objective. In CEP rule learning, the grammar of the rule language is known in advance (see the EPL in Section 2), and instead a particular rule (or, more precisely, an event pattern) is searched for in order to detect a particular situation of interest.

3.4 Machine learning of CEP rules

Recently, some approaches have been published that are specifically designed for automatically mining CEP rules in data streams [27] and explore machine learning for CEP [28].

Frömmgen et al. [29] and Weiss and Hirsh [30] learn event patterns with restricted expressivity. They examine simplified languages for specifying patterns that do not comprise the language scope of common CEP event processing languages.

Semi-automatic machine learning approaches for rule pattern detection are proposed by [31, 32]. Both approaches combine manual intervention of a domain expert and automatic learning techniques. The drawback of semi-automatic approaches is that the learning outcome heavily relies on human expertise. In addition to semi-automated rule mining, Pielmeier et al. [33] proposes to apply mathematical optimization methods to optimize the values of rule parameters.

Several proposals rely on *rule-based classifiers* for rule extraction. The applicability of the established rule-based classifiers One-R, RIPPER, PART, DTNB, Ridor, and NNGE are investigated by Mehdiyev et al. [34]. The investigated classifiers classify the data elements in the data streams into different classes. Petersen et al. [35] propose a hybrid approach of rule-based classifiers and k-means algorithm that can derive labels for unlabelled data on the one hand and automatically generate CEP rules on the other. Similar to [35], the ARECEP framework [27] tackles data labeling and automatic rule extraction together. For this task, ARECEP integrates several deep learning methods, such as LSTM, CNN, or RNN, for data stream labeling, as well as several rule-based classifiers, such as PART, DT, FURIA and others, for rule extraction. The good experimental results of the above mentioned approaches show that CEP rule patterns can in principle be learned by rule-based classifiers.

The iCEP framework by Margara et al. [5] aims to automatically learn predictive CEP rules from historical traces. The main idea is to decompose the rule learning problem into sub-problems each handled by a dedicated submodule. The different constituents of a rule pattern are then learned by ad-hoc learning algorithms.

Mousheimish et al. [7, 36] implemented the autoCEP system to automatically generate CEP rules. The algorithm learns so-called shapelets, that build patterns of minimum possible length to classify the data. In a second step, these shapelets are transformed into CEP rules. The system uses a brute-force shapelet extraction algorithm [6]. The autoCEP approach aims to learn a rule set for all types of classes in order to predict their future occurrence.

In [37], we presented a genetic programming (GP) approach for rule learning. The GP approach evolves a population of CEP rules, represented as advanced syntax trees, over multiple generations and applies particular genetic crossover and mutation operators to them. Our Bat4CEP, described below, shares similar rule representation and rule manipulation operators with the GP-based approach, but Bat4CEP is based on a completely different metaheuristic using flight and search operators controlled by loudness and pulse rate. Bat4CEP contributes a substantial extension of our work in [37], because it supports a larger solution space due to a more complex CEP rule language and outperforms the solution quality.

Although first approaches have been published, automatic learning of CEP rules continues to be an open research problem [5, 34]. Until now, proactive CEP by means of automatically generated rules still remains a vision [7]. As far as we know, swarm algorithms have not yet been applied to the CEP rule extraction problem.

4 Basics of bat algorithm

In 2010, Xin-She Yang developed a new bat-inspired algorithm [38]. The proposed Bat Algorithm (BA) is a metaheuristic algorithm based on the echolocation behavior of bats, which simulates bat abilities to detect and hunt preys and to avoid obstacles even at complete darkness.

The bats emit a sound pulse and listen for the echo bouncing back from surrounding objects. Each pulse has a very short duration and is sent with a predefined frequency. Bats adjust automatically loudness and pulse rate to control

the search for prey. The closer the bat position to the prey the better. For a given optimization problem, the bat position represents a solution of the problem under investigation.

4.1 Basic bat algorithm in pseudo code

Each bat i is specified at time t by its position x_i^t, its velocity v_i^t, its frequency f_i^t, its pulse rate r_i^t and its loudness a_i^t in a d-dimensional continuous search space [38].

For each bat, its position is updated and attribute values are adjusted individually at each time step. The basic steps of the BA are shown in pseudo code in Algorithm 1 (according to [38]).

Algorithm 1 Bat algorithm.

1: **procedure** BA
2: Define objective function $F_{opt}(x)$;
3: **for all** Bat i **do**
4: Initialize position x_i^0, frequency f_i^0, velocity v_i^0,
5: pulse rate r_i^0 and loudness a_i^0;
6: **end for**
7: **for all** time step t & abort criterion not fulfilled **do**
8: **for all** Bat i **do**
9: Generate a new solution by random flight:
10: adjust frequency f_i^t and velocity v_i^t (eqs. 1 and 2);
11: calculate new position/solution x_i^t (eq. 3);
12: **if** rand $> r_i^t$ **then**
13: Generate a new solution by local search:
14: calculate a new x_i^t around a selected best solution (eq. 4);
15: **end if**
16: **if** (rand $< a_i^t$) & ($F_{opt}(x_i^t) < F_{opt}(x^*)$) **then**
17: Accept x_i^t as new global best solution: $x^* = x_i^t$;
18: Increase r_i^t and reduce a_i^t (eqs. 5 and 6);
19: **end if**
20: **end for**
21: **end for**
22: Final result is best position (= solution) for F_{opt};
23: **end procedure**

4.2 Generation of new solutions: random flight and local search

Two different ways of changing the position of bats (equivalent to creating a new candidate solution) can be distinguished: Either *flying randomly* around its current position or performing a *local search* around one of the best solutions so far.

4.2.1 Random flight operator

The random flight explores the search space around the current bat position. At time step t, the new position of bat x_i^t is the movement from its previous position x_i^{t-1} with its current velocity v_i^t (based on its frequency f_i^t).

$$f_i^t = f_{min} + (f_{max} - f_{min})\beta \tag{1}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x^*) f_i^t \tag{2}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{3}$$

Frequency f_i^t is calculated for each bat i in each time step t using the equation (1). It influences the calculation of the velocity v_i^t in the same time step t. Its initial value must be in the range of the problem domain and $\beta \in [0, 1]$ indicates a random number.

Velocity is calculated from the bat's previous velocity as well as the distance between the bat's previous position and the current global best position x^* of all n bats and its frequency (2).

The new position of a bat is its old position shifted by the new velocity (3).

Thus, the movements of the bats offer some similarity to the movement of the particles in particle swarm optimization (PSO) metaheuristic, where frequency controls the pace and range of the particle movements [13].

4.2.2 Local search operator

In order to increase the diversity of possible solutions, for each bat that meets the requirements in rows 12-15 in Algorithm 1, a new candidate solution is generated by a random walk around the current global best solution of all n bats (or a solution selected among the current best solutions).

$$x_i^t = x^* + \epsilon a^t \tag{4}$$

With x^* is the global best position, $\epsilon \in$ [-1, 1] is a random number and a^t is the average loudness over all bats in iteration t.

In local search, badly positioned bats or bats that have not been able to improve their position over several iterations perform a local search to find the prey near the swarm's best bat.

4.3 Control of search: loudness and pulse rate

Pulse rate and loudness of a bat i serve as indicators for the quality of the position/solution x_i^t. They guide the relation

between exploitation and exploration of the search space as a function of the bat's distance from the target.

The pulse rate r_i^t guides the local search. The greater the pulse rate is, the closer the bat is to the prey. Thus, the pulse rate of a bat increases with a better solution, which results in a smaller probability of a local search (rows 12-15 in Algorithm 1).

The loudness a_i^t decreases with increasing proximity to the target. It is used for the acceptance of the position as the new global best position.

If a new global best solution is found, for bat i the pulse rate is increased and the loudness is decreased as follows:

$$r_i^{t+1} = r_i^0[1 - exp(-\gamma t)] \tag{5}$$

$$a_i^{t+1} = \alpha a_i^t \tag{6}$$

where α and γ are constants.

At the initialization step of the algorithm (rows 3-6 in Algorithm 1), each bat should have different, randomly generated values of loudness and pulse rate (see [38] for more details).

Only if a new solution is better than the previous one, its loudness (it is decreased) and its pulse rate (it is increased) will be updated.

By adjusting the loudness and pulse rate in each iteration and for each bat individually, the bat algorithm balances between exploration (global search for searching new regions) and exploitation (local search in the neighborhood of promising previous solutions) during the search process. For this reason, the pulse rate emission rates and the loudness essentially provide a mechanism for automatic control and auto zooming in the region with promising solutions [39].

4.4 Application areas of bat algorithm

The BA metaheuristic has already been successfully applied to several problem domains. On the one hand, to continuous optimization problems like economic load dispatch problem [40], parameter estimation [41], diagnosing diseases [42]. On the other hand, to discrete/combinatorial optimization problems like traveling salesman problem [43], job shop scheduling [44], patient bed assignment problem [45], or human pose estimation [46]. Moreover, first approaches exist for learning association rules, see Chapter 3. A comprehensive overview of application areas can be found in [39, 47].

5 Bat4CEP: Applying bat algorithm to learning of CEP rules

5.1 Bat algorithm for rule learning: general approach

In this paper, we present Bat4CEP, a bat algorithm for automatically learning CEP rules by exploiting historical data streams. Our goal is to learn causal and temporal relationships between primitive events, which correspond to a certain *situation* of interest. In CEP terms, such a situation is represented by generating a new *complex event*, for instance a machine breakdown, fraud incident, or emergency situation event.

The core idea of our approach is to map of the CEP rule learning problem described in Section 2 to an optimization problem: That is, finding the best CEP rule that detects a particular situation of interest in a data stream out of the search space of all possible CEP rules. Through this mapping of problems, an optimization algorithm such as bat algorithm can be used to find a solution to the learning problem.

For learning CEP rules with BA we have to adapt the bat-inspired metaheuristics to the specifics of CEP rule learning. We try to keep the main features of the canonical bat algorithm (as introduced on the previous section) and to limit our changes to the adjustments absolutely necessary for the tackled discrete problem. The essential adjustments affect the bat attributes as well as all equations and operations, which we will describe in the following section in detail. In [48] a simplified initial version of the approach was developed.

5.2 Solution space: bat positions

The solution space in our problem is the set of all possible CEP rules meaningful for the given set of primitive events emitted in the event stream. Therefore, a bat position in the solution space corresponds to a certain CEP rule. This means that in Bat4CEP, a certain CEP rule is considered as the position x_i^t of bat i at time t. [3] The size of the solution space is determined by the operators of the EPL (see Section 2) and by the particular event types in the data stream under investigation.

A key issue is how to represent solution candidates, i.e. CEP rules in Bat4CEP. There are various ways to encode a certain CEP rule: e.g. as a string or as a number. However, we have decided to encode CEP rules as

[3] In the case of CEP rule learning, a solution is always represented by a single, individual rule.

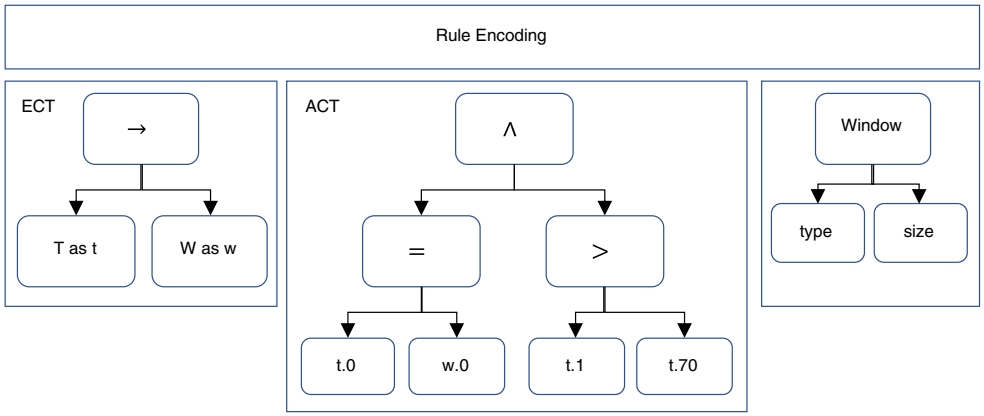

Fig. 2 Example of rule encoding with ECT, ACT and Window

syntax trees. The syntax tree encoding is well-established in Genetic Programming [49] and provides us with more semantic knowledge about the structure of a rule, which is helpful when modifying the rule in a meaningful way. In [37], we presented a different approach to rule learning based on genetic programming that also uses a syntax tree representation of candidate rules.

As an example for the syntax tree encoding, we consider the rule introduced in Fig. 1 indicating a machine breakdown. This sample rule can be written in a pseudo CEP language as follows:

CONDITION
(T as t → W as w) ∧ (t.0 = w.0 ∧ t.1 > 70)
ACTION new CE

The rule considers two events, one of event type T and the other of type W, and assigns the alias names t and w to them.[4] The rule condition is fulfilled, if an event instance of type T occurs followed in time by an event of type W and the first attribute of both events carry an equal value as well as the second attribute of t has a value greater than the constant value 70. If the rule fires then a new complex event of type CE is generated, which here is the $MachineBreakdownEvent$ in Fig. 1.

Figure 2 shows the *general tree representation* of a CEP rule, which represents the above sample rule. Each rule component is mapped to a particular tree node. The encoding of an entire CEP rule is represented by three syntax trees: (1) condition part 1: event types, (2) condition part 2: attribute conditions, and (3) sliding window:[5] [6]

1. *Event Condition Tree (ECT)*: The condition part of an event pattern consists of a combination of several event types and event instances, e.g. $(T \rightarrow W)$. The ECT encodes event conditions as rule subtrees. The possible event algebra operators ($\wedge$, $\vee$, $\rightarrow$, $\neg$) are represented as inner nodes and the event types as leaves of the syntax tree. Figure 2 depicts the ECT of the above sample rule with the event condition *(T as t → W as w)*. The inner node is the temporal sequence operator ($\rightarrow$), which has two operands as child nodes, one event instance of type T and the other of type W. Alias names are assigned either to distinguish two event instances of the same type or to refer the ACT conditions to instances in the ECT.
2. *Attribute Condition Tree (ACT)*: The event pattern may contain further conditions to restrict *attribute values*. The ACT also encodes attribute conditions as a rule subtree. Attribute conditions constrain the state of event instances, e.g. the equivalence of attribute values or that an attribute value exceeds a threshold. The unique relation of an attribute to the event instance it belongs to is achieved by alias names. In the example, the attributes $t.0$ and $t.1$ in the ACT can be dereferenced to the event instance of type T in the corresponding ECT. An ACT can only refer to aliases already defined in the ECT. The common logical and numerical comparison/arithmetic operators ($\wedge$, $\vee$, $\neg$, $>$, $<$, $=$, $+$, -, etc.) are available to formulate attribute conditions. Moreover, certain attribute values can be aggregated by common aggregation functions like $avg()$, $sum()$, etc. The root node of the ACT must either be an operator or an aggregation function.
3. *Sliding Window*: A sliding windows is defined by its *type*, which is either a *length window* or a *time window*, and its *size*.

[4] Event types are marked by capital letters, alias names by lower case letters (numbered if necessary). Attributes of an event instance t are either numbered (starting with 0): $t.0$, $t.1$, $t.2$, etc., or have domain-specific names.

[5] The action part of a CEP rule is not part of the encoding. This is not necessary, because we restrict our approach to the generation of a single complex event as the only kind of action clause.

[6] The inner nodes of the tree are the common EPL operators applicable to formulate the event pattern, similar to the set of functions F in GP. The leaves of the syntax tree are the event types, attributes and constants, similar to the set of terminals T in GP.

The possible operators of ECT and ACT subtrees are determined by the EPL as described in Section 2. An ECT is a mandatory component of every rule, whereas an ACT is optional. In the pseudo code representation, the ECT and the ACT parts of a rule condition are implicitly connected via a logical *and*-operator ($\wedge$). In case no ACT exists, its part in the (implicit) *and*-operator is assumed as true. The syntax tree encoding has to respect various constraints to ensure that a tree can only be mapped to a valid and executable CEP rule.

5.3 Preprocessing step

The Bat4CEP approach requires a preprocessing step to extract meta information about the event data from the training data stream. This data is used for generating or modifying the ECT and ACT parts of a CEP rule. For constructing ECT and ACT, we have to determine the set of all event types and all event attributes with their observed value ranges. For creating or modifying sliding windows, the time interval covered by the entire event stream and the total number of events is determined. This necessary meta information is obtained by parsing the training data stream with the historical trace.[7]

5.4 Flying bats

Obviously, the movement of a bat i between two positions x_i^t and x_i^{t+1} is a crucial part of BA. As presented in Section 5.2, a bat position in Bat4CEP is a certain CEP rule. As described in the original work [38], a bat flight changes the location of a particular bat. Thus, in Bat4CEP, a bat movement can be considered as changing one CEP rule (starting position) to another CEP rule (target position).

For changing a CEP rule represented by a syntax tree, a randomly selected tree node is modified. According to the type of the selected node and its position in the syntax tree, one of the following modification steps can be processed:[8]

- *Changing a window*: Given a window node, the modification operation changes with a certain probability the type of the window (from length to time or vice versa) and generates randomly a new window size.
- *Changing an ECT*: An ECT node is replaced by either (i) a random *event type* and, as a consequence, the previous subtree under the replaced node is erased, or by (ii) a new random *event operator* while retaining the child nodes of the previous operator. If an event type node has been replaced by an operator, missing operands are drawn from the set of all event types. Note that changing event types in the ECT can cause incorrect ACT conditions: for instance, if the ACT uses an event alias name, which has disappeared due to an ECT modification. In such a case, an ACT repair step takes place replacing invalid alias names by new and randomly chosen valid ones.

  Figure 3 gives an example of an ECT modification. The selected node is the root node of the ECT subtree (marked in dark grey) representing the temporal sequence operator ($\rightarrow$), which is replaced by the randomly chosen $\wedge$-operator. The original operands (here the event types A and B) are preserved.
- *Changing an ACT*: For changing a certain ACT node we have to distinguish the following ACT node types: logical operators, comparison operators and operands.
  - If the selected node represents a *logical operator* ($\wedge$, $\vee$ and $\neg$) it could be replaced by (i) another randomly drawn logical operator. In this case, the original operands are retained. (ii) As another possibility, the original logical operator is replaced by a comparison operator ($<$, $>$, $\leq$, $\geq$, $=$) or aggregate function ($avg()$, $sum()$, etc.) along with new randomly drawn operands (event attributes or numerical constants). In case the number of operands of the new operator differs from that of the replaced operator, a repair step is performed: either one of the original operands is deleted or a missing operand is randomly created.
  - A node with a *comparison operator* ($<$, $>$, $\leq$, $\geq$, $=$) can just be replaced by another randomly chosen comparison operator. Aggregate functions are treated in the same way.
  - An *operand* node can be changed to another event attribute matching the ECT part of the rule. If the selected operand node is on the right-hand side of a comparison operator, it can also be changed to a constant value (out of the value range of the left-hand side operand). Changing operands can cause inconsistencies, if the numerical value of constant value lies outside the value range of the corresponding attribute. In this case, a new constant value is drawn randomly out of the valid value range.

In Bat4CEP, the *distance of a bat flight* can be considered as the number of modification steps that has been processed. A long flight means that the the target position is far away, i.e. the rule has to be changed in several parts or the above steps are applied several times, respectively.

[7] While the function set F is implemented fix, the set of terminals T is automatically extracted from the training data stream.

[8] The rule manipulation operators of the GP approach in [37] had to be adapted to the Bat metaheuristic.

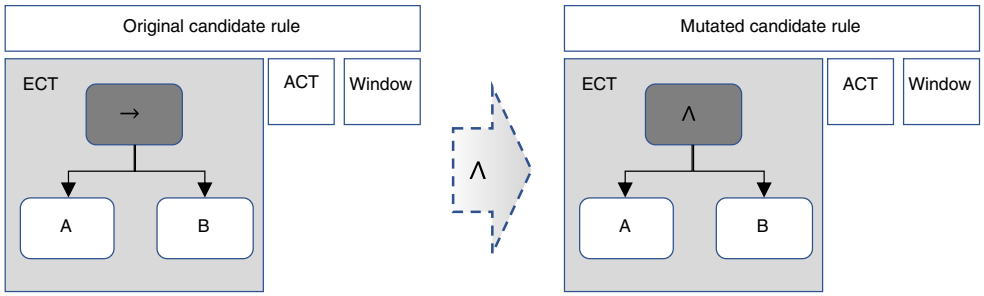

Fig. 3 Node modifying on event condition tree

5.5 Evaluating bat positions

During the Bat4CEP learning process, each bat position must be evaluated, i.e. we want to know, how far it is from an optimal solution. Because in Bat4CEP each bat position corresponds to a particular CEP rule, we have to quantify the quality (or fitness) of this rule.

A CEP rule can be considered as good, if it detects all situations of interests (and not others) in a data stream. As described in Section 2 and depicted in Fig. 1, the training data contains labels (or complex events) representing the occurrences of situations of interest. These complex events indicate the exact time instants at which the CEP rule to be learned should fire. The more often it fires at the correct (and only at the correct) positions in the training data stream, the better.

In order to quantify the *quality* of a given rule, it must be applied to the training data stream to identify the times at which the rule fires. By comparing the firing times with the positions of the labels, we compute the quality values *recall* and *precision*:[9]

1. $recall = TP/(TP + FN)$ specifies the proportion of complex events in the examined data stream that are successfully found by the candidate rule. A recall of 1.0 is optimal because the candidate rule finds all complex events.
2. $precision = TP/(TP + FP)$ specifies the proportion of complex events retrieved by the candidate rule that are correct, i.e. they relate to a complex event in the examined data. A precision of 1.0 is optimal because the candidate rule fires only at correct times.

The calculation of the rule quality uses the common *F1-score* to blend the different values of recall and precision providing an aggregated measure of the rule quality. A perfect rule yields a F1-score of 1.0, when precision and recall both are 1.0.

[9] *TP (True Positive rate)*: measures how often the rule does fire at the right positions, *TN (True Negative rate)*: measures how often the rule does not fire when it should not, *FP (False Positive rate)*: measures how often the rule does fire although it should not, *FN (False Negative rate)*: measures how often the rule does not fire although it should

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{7}$$

5.6 Bat4CEP: steps and operations

The Bat4CEP algorithm follows the general idea of the original BA and controls the movement of a swarm of bats by the parameters: frequency f_i, velocity v_i, pulse rate r_i and loudness a_i, which are all together defining the behavior of bat i. The movement of a certain bat i is determined by the causal chain: $f_i \rightarrow v_i \rightarrow x_i$ where x_i notes the new position of bat i. It is assumed that all bats change their positions in fixed time steps t (as in the standard BA).

Frequency: The frequency f_i^t controls how the velocity of a bat i is changed. It is calculated by simplifying (1). As suggested in [38], we set $f_{min} = 0$ and $\beta \in [0, 1]$ as a random number. The maximum frequency f_{max} can be chosen arbitrarily.

$$f_i^t = f_{max} \cdot \beta \tag{8}$$

Velocity For calculating the bat velocity v_i we adapt (2). In particular, the distance between the current bat position x_i^{t-1} and the position x^* of the best located bat in (1), is substituted by the difference between the corresponding F1-scores (given by (7)):

$$v_i^t = v_i^{t-1} + (F1(x_i^{t-1}) - F1(x^*)) \cdot f_i^t \tag{9}$$

Note that in each step, the bat velocity is decreased, because the F1-score of the best bat $F1(x^*)$ is always greater than that of the current bat position $F1(x_i^{t-1})$. This means that a bat starts flying fast but gets slower, when moving towards a better solution.

Controlling the flight – Loudness and Pulse Rate As described in the original bat algorithm [38], loudness a_i^t and pulse rate r_i^t control the bats behavior. Note that only if a new bat position proves to be the new best global position x^*, loudness and pulse rate are recalculated.

- The *pulse rate* r_i^t controls the decision whether a bat i performs a *local search*. According to (5), the pulse

rate r_i^t is calculated by $r_i^{t+1} = r_i^0[1 - exp(-\gamma t)]$. As suggested in [38], we choose $r_i^0 \in [0, 1]$ and $\gamma = 0.9$.[10] In every time step, a random number $rand \in [0, 1]$ is drawn. Only if $rand > r_i^t$ is valid, a local search is performed. Because r_i^t is increasing over time and converging to r_i^0, the probability of a local search is decreasing over time. (This makes sense, because the bat is approaching a better position.)

- The *loudness* a_i^t is calculated from the average loudness of the former time step by (6) as: $a_i^{t+1} = \alpha \cdot a_i^t$ with $\alpha = 0.9$ as suggested in [38] and $a_i^0 > 2$. First, the loudness determines the extend to which the global best CEP rule is changed during a local search.[11] Because a_i^t is continuously decreasing and converging to 0, the number of modification steps is getting smaller over time.

  Secondly, the acceptance of a flight destination as new best global position depends on loudness. For this purpose a_i^t is compared with a random number drawn from $rand \in [0, a^0]$. Only if $rand < a_i^t$ holds, a new bat position x_i^t with better F1-score than $F1(x^*)$ is chosen as the new global best solution x^*. This means that the probability of changing the global best solution is decreasing over time.

Random flight As in the original bat algorithm [38], the velocity is used for deriving the destination of a bat's flight, i.e. how much the position of the bat is changed (see (3)). In Bat4CEP, we consider Δ_v^f the truncated difference between the current bat's i frequency f_i^t and its velocity v_i^t. Then the value of Δ_v^f determines the number of modifications steps (see Section 5.4) that are processed on the rule that represents the bat's location of origin:

$$\Delta_v^f = \lfloor f_i^t - v_i^t \rfloor \tag{10}$$

For instance, with frequency $f_i^t = 5.2$ and velocity $v_i^t = 2.1$ the truncated difference $\Delta_v^f = 3$ means that 3 modification operations have to be processed on the rule corresponding to the current bat position x_i^t.

Local search A local search starts the bat flight from the so far best global solution. Using (4), the flight's target position is given by $x_i^t = x^* + \epsilon \cdot a^t$ with the best global position x^*, the average loudness of all bats a^t and a random number $\epsilon \in [0, 1]$. If we transfer this operation to Bat4CEP, then the global best CEP rule (corresponding with x^*) is changed by a number of modification steps (see Section 5.4), which is given by the truncated value of $\lfloor \epsilon \cdot a^t \rfloor$.

[10] In our experiments, we achieved best results with a pulse rate between 0.1 and 0.3.

[11] as explained in the subsequent subparagraph

Note that applying a number of modification steps to a candidate rule corresponds to a flight as described in Section 5.4.

5.7 Initializing bat swarm

In an initial step, a bat swarm must be generated, i.e. a set of bats located at positions representing valid and executable CEP rules well distributed over the entire problem space. The generated candidate CEP rules rely on the meta information gathered during the preprocessing step described in Section 5.3. The mandatory and optional rule components are randomly generated for each bat:

1. *Window generation*: Sliding windows are created at random: At first, the initialization procedure randomly chooses a window type assuming that length and time windows appear with the same probability. At second, the window size is randomly determined taking the boundaries observed in the preprocessing step into account.
2. *Event Condition Tree generation*: The ECT subtree is generated by applying the common *ramped half-and-half* method [50]. One half of the bat swarm is initialized by fully filled ECTs with a predefined maximum depth of the syntax tree. As long as the maximum depth has not been reached, the procedure draws uniformly an operator from the set of event condition operators and continues to generate the operands as subtrees (event types or again event condition operators). The other half of the swarm is created by the *grow method*, which produces partially filled trees with a lower depth, i.e. the generated rules are shorter. Ramped half-and-half initialization produces a swarm representing a set of initial CEP rules of great diversity that will converge to more promising results during the Bat4CEP learning process.
3. *Attribute Condition Tree generation*: Not every rule has to specify attribute conditions. Thus, the ACT subtree is optional and may be omitted during swarm initialization. Bat4CEP uses a fixed rate to determine how many bats are equipped with an ACT. Attribute conditions are created at random by selection of a comparison operator or an aggregate function and its suitable operands. Also simple arithmetic expressions are possible.[12] The first operand must be an attribute of an event type occurring in the ECT, referenced by an alias already defined in the ECT. The second operand can either be an event attribute or a (numerical) constant. The concrete value of a constant is randomly

[12] As an example, we implemented simple arithmetic expressions with addition and subtraction operators so far.

chosen out of the value range of the first operand gathered during the preprocessing step.

The initialization randomly selects possible operators of the EPL and the current event types of the problem at hand and generates only valid CEP rules.

5.8 Bat4CEP overview

The steps of the Bat4CEP algorithm are summarized in pseudo code in Algorithm 2. This algorithm retains the basic structure of the canonical bat algorithm presented in Section 4 in Algorithm 1. The necessary adjustments and extensions for CEP rule learning are incorporated at the affected statements.

Algorithm 2 Bat4CEP.

```
1:  procedure BAT4CEP
2:      Define objective function F_F1score;
3:      for all bat i do
4:          Initialize position x_i^0, frequency f_i^0, velocity v_i^0,
5:          pulse rate r_i^0 and loudness a_i^0;
6:          Calculate the F1-score: F_F1score(x_i^0);
7:      end for
8:      Remember the best x* ;
9:      for all time step t & abort criterion not fulfilled do
10:         for all bat i do
11:             Adjust f_i^t and v_i^t (eqs. 8 and 9);
12:             Generate a new CEP rule by random flight
                (eq. 10):
13:             Calculate a new rule x_i^t by modifying rule
                x_i^{t-1};
14:             if rand > r_i^t then                ▷ rand ∈ [0, 1]
15:                 Generate a new CEP rule by local search
                    (eq. 4):
16:                 Calculate a new rule x_i^t modifying a
                    selected best rule x*;
17:             end if
18:             if rand < a_i^t & F_F1score(x_i^t) >
                  F_F1score(x*) then
19:                                          ▷ rand ∈ [0, a^0]
20:                 Accept x_i^t as new best solution: x* = x_i^t;
21:                 Increase r_i^t and decrease a_i^t (eqs. 5 and 6);
22:             end if
23:         end for
24:     end for
25:     Final solution is best position x* (= best CEP rule)
        for F_F1score;
26: end procedure
```

In local search in row 16, a best rule is selected among the current n% best rules. Please note that the design of Bat4CEP ensures that only syntactically correct and executable CEP candidate rules can be generated during the learning process.

Yang identifies the following three key features as the reasons for the success of bat algorithms [39]:

- Parameter control: In contrast to other metaheuristic algorithms, Bat algorithm adjusts the values of parameters from iteration to iteration. This automatically guides the search process from exploration to exploitation.
- Automatic zooming: Bat algorithm has the inbuilt mechanisms to zoom into promising regions of the search space. Supported by the automatic move from exploratory search to local exploitation.
- Change control: Bat algorithm applies the frequency parameter to influence the extent of changes in a current solution. This is similar to other swarm algorithms.

The above-mentioned key features also hold for Bat4CEP, so do the advantages. Bat4CEP exhibits the main characteristics of the canonical bat algorithm. The very good performance of Bat4CEP is due to the specific advantages of the underlying metaheuristic.

6 Experiments and results

We have conducted an extensive number of experiments to prove the performance of Bat4CEP. In this section, the obtained experimental results are presented in detail. First, in Section 6.2, we consider synthetic data streams in order to systematically investigate the learning of CEP rules of different complexity. Second, in Section 6.3, we evaluate our approach on a real-word scenario with traffic data originating from road sensors in Madrid, Spain. The section starts with basic details on the setup of the experiments and concludes with information on key implementation issues.[13]

6.1 Experiment setting

As already mentioned in Section 2, our approach requires a training data stream containing labels indicating that a complex event has occurred, e.g. the historical data stream with the *MachineBreakdownEvent* shown in first phase: pattern learning of Fig. 1. The Bat4CEP algorithm described in the previous section uses the training data stream to determine the quality of each candidate rule (= evaluating bat positions by means of the F1-score in Section 5.5) in the course of the algorithmic procedure presented in Algorithm 2.

[13] The Bat4CEP system is implemented in Java. The source code and the data sets used in the experiments are available at http://sw-architecture.inform.hs-hannover.de/files/bat4cep.zip

Labeling For labeling an unlabeled training data stream, a *target rule* R^{T} is predefined to be learned by Bat4CEP. This target rule is executed on the training data stream and inserts a *complex event* CE at the appropriate position in the data stream whenever the event pattern of the target rule matches. In this way, considering the complex events as labels, we obtain a labeled training data stream.[14]

By using the predefined target rule, it is possible to systematically investigate rules with different levels of complexity. Thus, it enables us to increase the complexity of the rule to be learned in a step-by-step manner.

Evaluation process The labeled data stream is used for performing the experiments and evaluating the quality of each of the candidate rules that are generated by Bat4CEP. For this purpose, each candidate rule R^{learn} is executed on the training data stream, comparing its firing times with the firing times of the target CEP rule R^{T}. By measuring the TP, TF, FP and FN values as described in Section 5.5, we can calculate *F1-score*, *precision*, and *recall* values of rule R^{learn}. In this way, the problem of learning a CEP rule is mapped to an optimization problem: Find a rule R^{learn} with the maximum *F1-score* for a labelled data stream. Every independent labelled data stream poses a separate optimization problem.

For evaluation purpose, we conduct *hold-out validation*. Two independent data streams are randomly generated: a *training data set* used to run the Bat4CEP learning process and a *test data set* used to evaluate how well the best learned rule R^{learn} performs on a different and independent second data set. The values given for recall and precision in the tables below report the results obtained on the test data stream.

Each experiment setting has been run ten times with different random values. The presented results of an experiment setting are calculated as the average values over the ten runs using the recall and precision of the best rule learned R^{learn} in each run.

Bat algorithm control parameters In all experiments reported in this section, we used the Bat4CEP control parameters as listed in Table 1. Our algorithm has the same set of control parameters as the standard BA. The Bat4CEP algorithm, as shown in Algorithm 2, works with a swarm size of 500 candidate rules and iterates over 500 time steps at most. We select 5% of the bats in a swarm as the set of best bats. The concrete values of the parameters have been determined manually by experimentation. No automatic parameter tuning methods have been applied, because manual calibration of the algorithm was sufficient to achieve very good results.

Note that for other experimental settings or different application scenarios, the parameters of the bat algorithm may have to be adjusted. Finetuning of model hyperparameters is a common step in all machine learning approaches.

Table 1 Control parameters of Bat4CEP algorithm

Control parameter	Value
Swarm size	500
Best bats x^*	5%
Time steps	500
Frequency	2.5
Pulse rate	0.1
Loudness	1
α	0.9
γ	0.9

6.2 Performance on synthetic data

Now, we want to investigate the performance of our approach, in particular for exploring the limits and capabilities of Bat4CEP. How complex may the rules be that Bat4CEP is able to find?

Synthetic data For a systematic investigation, we first produce synthetic training data streams that contain randomly generated primitive events. Each primitive event is described by the following information: a timestamp, an event type and a set of key-value pairs for each attribute. For instance, [2021-01-21 14:19:31 C:: C.0=4711; C.1=1704] describes an event of type C with the two attributes: $C.0$ of value 4711 and $C.1$ of value 1704.

The default values of the parameters used for generating synthetic training and test data streams can be found in Table 2.

Table 2 Default values of parameters for generating synthetic training and test data

Data stream parameter	Value
Number of event types	10
Distribution of types	Uniform
Number of attributes per event	3
Number of primitive events	15,000
Distance between events (in sec)	[1,10]
Attribute range	[1, 10] or [1, 100]
Number of complex events / labels	> 250

[14] In a real-world scenario, of course, no target rule is known in advance. In real-world data streams, data labeling can be done in a variety of ways. For example, manually based on human observations or automatically by any kind of monitoring device. The performance of Bat4CEP is completely independent of the way the labeling was done.

Thus, the artificial training and test data streams each consist of 15,000 primitive events that are instances of 10 different event types. The temporal distance between successive events is a random value from [1,10] (in seconds), so the time interval between two consecutive events is 5 seconds on average. A stream contains at least 250 labels (= corresponding to the situation of interest for which the relevant event patterns have to be learned).

Learning target rules For each experimental scenario, we specify a CEP target rule R^{T} of a certain complexity, which should be learned by Bat4CEP.

This rule R^{T} is then executed on our synthetic data stream: each time when R^{T} fires, it inserts a label, a complex event CE, into the training data stream. BatCEP should now learn a rule that fires at exactly the same places as R^{T} did. Each individual target rule forms its own separate optimization problem and requires a specific labelled training data stream.

Table 3 shows some of our results achieved for rules of increasing complexity.[15] Depending on its complexity, a rule fires between 250 and 850 times on the training data. The more complex or specific a CEP rule is, the less it will fire.[16]

As target CEP rules we have selected a set of typical event patterns according to the experiments in [37] (rules 1 to 13) supplemented by further operators: The ECT parts of the rules contain between 2 and 6 different event types which are combined either by the sequence operator ($\rightarrow$), by the logical conjunction operator ($\wedge$), or by the *not* operator.

The *not* operator in ECT specifies the absence of one event in a matched pattern. This means, for example, that in rule no. 19 the pattern matches if an event of type A and one of type C occur in any order, but none of type B between them. Or in rule no. 20, first an A followed by a C followed by a D occur and no B between them.

In the ACT parts, we investigated various attribute conditions:

(i) comparing attribute values between two events with numerical comparison operators,
(ii) constraining attribute values with numerical thresholds (constant values),
(iii) calculating simple arithmetic expressions (addition and subtraction operators) and
(iv) combining the corresponding boolean expressions with logical operators ($\wedge$, $\vee$).

In addition, the rule no. 4 contains a negation operator, e.g. to state that two attributes of different events must not have the same value (for instance their IDs must be different).

In total, Table 3 exhibits excellent results. On average over 10 runs the best rule found by Bat4CEP achieved always recall and precision values either close to one or it is one. This means that the learned rule R^{learn} finds nearly all complex events produced by the target rule R^{T} (expressed by a recall near 1), and that R^{learn} nearly never fires at wrong places, i.e. there is no complex event created by R^{T} (expressed by a precision near 1).

Even rules with six different event types and six different attributes in the condition parts (row 11) can be learned with perfect results. The rule with the worst result is row 13 of Table 3. It consists of a rather complex ECT-ACT combination, but still has recall and precision values of 0.965 and 0.928, respectively.

To estimate the stability with which our approach delivers the results over multiple experimental runs, we determined the 95% confidence intervals. For all rules in Table 3 with excellent results, we got confidence interval widths smaller than 1% for the recall and precision values. For the rules for which Bat4CEP yields slightly worse results (rules 9, 10, 13), we obtained a confidence interval widths between 5% and 9% of the measured values. Rule 8 even had a confidence width of 16%. This means, that if Bat4CEP has problems to find a very good rule, the quality of the learned rules varies more between the different runs. But overall we can state, that our approach delivers all results with a high degree of confidence.

Learning aggregation functions A special feature of event processing languages are the aggregation functions: For all events of a certain type, which occur within a sliding window, they aggregate the values of a specific attribute. In Bat4CEP, we implemented four of the most common aggregation functions: $avg()$, $sum()$, $min()$, and $max()$. None of these functions cause any problem for Bat4CEP as Table 4 shows. Again Bat4CEP learns nearly perfect rules with very high recall and precision values.

Learning sliding windows Another important concept of a CEP rule language is *sliding windows*. The experiments discussed above learned rules with a fixed-size sliding time window of 60 seconds (except rules no. 3 and 4 of Table 4). Table 5 lists the results of experiments with varying window sizes. We chose two sample rules of different complexity and varied the window size to 60, 600, 3000, and 6000 seconds: the first rule of Table 4 as well as the sixth rule of Table 3.

[15] In all reported experiments, the window size for each rule is also learned by Bat4CEP. For the sake of clarity, each rule uses a time window of 60 seconds not shown in the table.

[16] Note that it is important that the training data stream contains a sufficient number of complex events. In our experiment setting, we achieved good results with more than 250 complex events inserted by the target rule into the stream.

Table 3 Bat4CEP results for rules of various complexity

No.	ECT	ACT	Recall	Precision
1	$A\ as\ a \rightarrow B\ as\ b$	$a.0 = b.0$	1.0	0.998
2	$A\ as\ a \rightarrow B\ as\ b$	$a.1 > b.1$	0.999	1.0
3	$A\ as\ a \rightarrow B\ as\ b$	$a.1 < b.1$	0.998	1.0
4	$A\ as\ a \rightarrow B\ as\ b$	$not(a.0 = b.0)$	1.0	0.997
5	$A\ as\ a \rightarrow B\ as\ b$	$a.0 = b.0 \wedge a.1 > b.1$	1.0	1.0
6	$A\ as\ a \rightarrow B\ as\ b \rightarrow C$	$a.0 = b.0 \wedge a.1 > b.1$	0.998	0.997
7	$A\ as\ a \rightarrow B\ as\ b \rightarrow C$	$a.0 = b.0 \wedge a.0 = c.0$	1.0	0.995
8	$(A\ as\ a \rightarrow B\ as\ b) \wedge (C\ as\ c \rightarrow D\ as\ d)$	$a.0 = b.0 \wedge c.0 = d.0$	0.945	0.931
9	$(A\ as\ a \rightarrow B\ as\ b) \wedge (C\ as\ c \rightarrow D\ as\ d$	$a.0 = b.0 \wedge c.1 > d.1$	0.964	0.954
10	$(A\ as\ a \rightarrow B\ as\ b) \wedge (C\ as\ c \rightarrow D\ as\ d)$	$a.1 > b.1 \wedge b.1 > c.1 \wedge c.1 > d.1$	0.972	0.967
11	$A\ as\ a \wedge B\ as\ b \wedge C\ as\ c \wedge D\ as\ d \wedge E\ as\ e \wedge F\ as\ f$	$a.1 > b.1 \wedge c.1 > d.1 \wedge e.0 = f.0$	1.0	1.0
12	$A\ as\ a \rightarrow B\ as\ b$	$a.0 = b.0 \wedge a.1 > b.1 \wedge a.1 > 80$	1.0	0.997
13	$(A\ as\ a \rightarrow B\ as\ b) \wedge (C\ as\ c \rightarrow D\ as\ d)$	$a.1 > 60 \wedge b.1 > 60 \wedge c.1 > 60 \wedge d.1 > 60$	0.965	0.928
14	$A\ as\ a \rightarrow B\ as\ b$	$a.0 = b.0 \vee a.1 > b.1$	1.0	1.0
15	$A\ as\ a \rightarrow B\ as\ b$	$a.1 - b.1 = b.0$	1.0	1.0
16	$A\ as\ a \rightarrow B\ as\ b$	$a.1 + b.1 > 110$	1.0	1.0
17	$A\ as\ a \rightarrow B\ as\ b$	$a.1 - b.1 > a.0 + b.0$	1.0	0.997
18	$A\ as\ a \rightarrow not\ B \rightarrow C\ as\ c$	$a.0 = c.0$	1.0	1.0
19	$(A\ as\ a \wedge C\ as\ c) \wedge not\ B$	$a.0 = c.0$	1.0	1.0
20	$(A\ as\ a \rightarrow C\ as\ c \rightarrow D\ as\ d) \wedge not\ B$	$a.0 = c.0$	0.998	0.987
21	$A\ as\ a \rightarrow not\ B \rightarrow C\ as\ c \rightarrow B\ as\ b$	$a.0 = c.0$	1.0	0.997

As larger windows contain more events, the fifth column of the table lists the average number of events per window.

We conducted various experiments with different window sizes without any significant impact on the excellent performance of Bat4CEP. Fortunately, the quality of the results is not affected by changing window sizes or the number of events in a window, respectively. The larger the window, the more events it contains (see column $Evts\ in\ Win$). Once again Bat4CEP yields very good results close to one.

Increasing the solution space The experiments reported above rely on synthetic training data streams randomly generated with the default parameters given in Table 2.

In order to investigate the effect of the size of the solution space on the performance of the Bat4CEP algorithm, the data stream generation parameters are varied in the following. For these experiments we selected two rules of Table 3 as R^{T} and run different test series:

- rule no. 5: $(A\ as\ a \rightarrow B\ as\ b \wedge a.0 = b.0 \wedge a.1 > b.1\ )$
- rule no. 6: $(A\ as\ a \rightarrow B\ as\ b \rightarrow C\ as\ c \wedge a.0 = b.0 \wedge a.1 > b.1)$

On the one hand, we varied the number of event types from 10 to 80. The results are reported in Table 6. On the other hand, we varied the number of attributes from 3 to 40. The results of this test series are reported in Table 7.

In general, increasing the number of event types and the number of attributes leads to a larger search space in which it is harder to find a good solution. Considering the rule no. 5, the enlargement of the search space has only little influence to the learning result. Bat4CEP stills learns the rule almost perfectly even with 60 different event types or 40 attributes per event in the training data set.

Table 4 Bat4CEP results for rules with aggregation operators

No.	ECT	ACT	Window	Recall	Precision
1	$A\ as\ a$	$a.1 > avg(a.1)$	60	1.0	1.0
2	$A\ as\ a \rightarrow B\ as\ b \rightarrow C\ as\ c$	$a.0 = b.0 \wedge a.1 > avg(b.1)$	60	0.993	1.0
3	$A\ as\ a$	$sum(a.0) > 500$	3000	0.996	0.994
4	$A\ as\ a$	$sum(a.1) > 5000$	3000	0.998	0.910
5	$A\ as\ a \rightarrow B\ as\ b$	$a.1 = min(a.1) \wedge b.1 = max(b.1)$	60	0.998	0.935

Table 5 Bat4CEP results for rules with various window sizes

No.	ECT	ACT	Window	Evts in Win	Recall	Precision
1	$A\ as\ a$	$a.1 > avg(a.1)$	60	12	1.0	1.0
2	$A\ as\ a$	$a.1 > avg(a.1)$	600	120	0.998	0.999
3	$A\ as\ a$	$a.1 > avg(a.1)$	3000	600	0.991	0.996
4	$A\ as\ a$	$a.1 > avg(a.1)$	6000	1200	0.995	0.997
5	$A\ as\ a \rightarrow B\ as\ b \rightarrow C$	$a.0 = b.0 \wedge a.1 > b.1$	60	12	1.0	0.996
6	$A\ as\ a \rightarrow B\ as\ b \rightarrow C$	$a.0 = b.0 \wedge a.1 > b.1$	600	120	1.0	1.0
7	$A\ as\ a \rightarrow B\ as\ b \rightarrow C$	$a.0 = b.0 \wedge a.1 > b.1$	3000	600	1.0	1.0
8	$A\ as\ a \rightarrow B\ as\ b \rightarrow C$	$a.0 = b.0 \wedge a.1 > b.1$	6000	1200	1.0	1.0

The rule no. 6 is more complex because it contains one more event type condition in the ECT. For this rule one can observe that the larger the search space, the harder it is to learn the target rule. Up to 40 different event types, the rule can be learned very well, but if the training data stream contains 60 or more event types, the values for recall and precision deteriorate. One reason for deterioration of the results is that when the number of event types is increased, the solution space for the ECT grows significantly. The number of attributes has less impact on rule learning, here a smaller deterioration of the precision value can be observed.

Examples of learned rule patterns An interesting question is how the learned rules look like. Of course, in a real live learning problem, the rule patterns are not known in advance and have to be extracted by Bat4CEP out of a labeled training data stream. Therefore, the major quality criterion of a learning algorithm is how precisely it detects the relevant patterns in the data stream: the recall and precision of the learning outcome.

Because each experiment setting has been run ten times with different random values, also ten different rules have been found in each experiment. For the sake of clarity, in the following we selected a typical rule out of the ten runs for each experiment. Table 8 lists the learned rules R^{learn} for some selected target rules R^{T} of Tables 3, 4, and 5.

The column *Occur* depicts how often the same rule was learned in the ten test runs. 100% means that all ten runs yielded the same rule, while 60% means that six out of ten runs found the given rule. Especially for the rules 3-4 and 4-5 no particular learned rule really dominated the result.

Moreover, in the table we combined learned rules whose windows and constants differ only slightly. These rules are marked by intervals.

It can be stated, that Bat4CEP usually finds learned rules rather similar to the given target rule. Especially the ECT part of the pattern is learned in most runs almost exactly. For simpler rules, e.g. 3-1 to 3-9, also the ACT is learned exactly. If the rules become more complex, some differences in the ACT part can be observed. Only for rules 3-4 and 4-5 no typical learning result emerged, because the results showed several slightly different ACTs. Note that also in these cases, Bat4CEP learns a rule, which yields perfect results for precision and recall. This means that for this training data stream, there are several slightly different rules that fire in the right places and produce almost the same results as the target rule.

When learning constants in the ACT, slightly different values can be observed, marked by using the intervals in the table. It is not possible to learn the exact threshold values of event attributes or window sizes used in the target rule if the observed event stream does not match these values. For instance, if the target rule contains the condition $a.1 > 60$, but only attribute values between [0, 55] and [62, 100] occur in the stream, then any threshold between 55 and 62 would yield perfect results.

Table 6 Bat4CEP results with different number of event types in data stream

No.	Event Types	Rule #5		Rule #6	
		Recall	Precision	Recall	Precision
1	10	1.0	1.0	1.0	0.998
2	20	1.0	1.0	0.999	0.994
3	40	0.999	0.990	0.975	0.973
4	60	0.990	0.990	0.902	0.845
5	80	0.964	0.938	0.789	0.685

Table 7 Bat4CEP results with different number of attributes in data stream

No.	Attributes	Rule #5		Rule #6	
		Recall	Precision	Recall	Precision
1	3	1.0	1.0	1.0	0.998
2	6	1.0	1.0	1.0	1.0
3	10	1.0	1.0	0.992	0.981
4	20	0.995	0.919	0.997	0.879
5	40	0.997	0.919	0.988	0.823

Table 8 Example rule patterns learned by Bat4CEP

Tab-No.	Learned ECT	Learned ACT	Learned window	Occur in %
3-1	$A\ as\ a \rightarrow B\ as\ b$	$a.0 = b.0$	60	100
3-2	$A\ as\ a \rightarrow B\ as\ b$	$a.1 > b.1$	60	80
3-4	$A\ as\ a \rightarrow B\ as\ b$	$not(a.0 = b.0 \wedge a.0 > b.3\ )$	60	30
3-6	$A\ as\ a \rightarrow B\ as\ b \rightarrow C$	$a.0 = b.0 \wedge a.1 > b.1$	60	70
3-9	$(A\ as\ a \rightarrow B\ as\ b) \wedge (C\ as\ c \rightarrow D\ as\ d)$	$a.0 = b.0 \wedge c.1 > d.1$	60	50
3-11	$A\ as\ a \wedge B\ as\ b \wedge C\ as\ c \wedge D\ as\ d \wedge E\ as\ e \wedge F\ as\ f$	$a.1 > b.1 \wedge c.1 > d.1 \wedge e.0 = f.0$	60	80
3-12	$A\ as\ a \rightarrow B\ as\ b$	$a.1 > [70, 80] \wedge a.0 = b.0$	60	50
3-13	$(A\ as\ a \rightarrow B\ as\ b) \wedge (C\ as\ c \rightarrow D\ as\ d)$	$a.1 > [50,60] \wedge b.1 > [50,60] \wedge c.1 > [50,60] \wedge d.1 > [50,60]$	60	60
4-3	$A\ as\ a$	$sum(a.0) > [499, 501]$	[2996,3002]	100
4-4	$A\ as\ a$	$sum(a.1) > [4952, 5002]$	[3000,3004]	80
4-5	$A\ as\ a \rightarrow B\ as\ b$	$min(a.1) = a.1 \wedge max(b.1) = b.1$	60	30
5-1	$A\ as\ a$	$a.1 > avg(a.1)$	60	90
5-3	$A\ as\ a$	$a.1 > avg(a.1)$	[2940,3173]	90
5-4	$A\ as\ a$	$a.1 > avg(a.1)$	[4843,6217]	100

It can be seen that very often the exact window size is found. Especially, the small 60 seconds window used in the rules of Table 3 is learned exactly in most runs. Large window sizes are more difficult to learn: they do not form such a tight boundary, and can be changed more without affecting when a rule fires, i.e. precision and recall do not change.

6.3 Performance on real-world traffic data

All experiments presented in the previous section have been conducted on artificial data under optimal conditions, e.g. without any noise, missing values, outliers, or incorrect timestamps. In this section, we would like to investigate how our approach performs on non-perfect real data.

As a real-world scenario, we applied Bat4CEP on traffic data collected by the city of Madrid, Spain.[17] Loop detector sensors are distributed in roads all around the city and measure the traffic flow every 15 minutes. Our goal is to learn rules that classify the traffic state on a road segment. In traffic management three different states can be distinguished: *free traffic*, *dense traffic* and *congested traffic*.

Among other data, each traffic sensor collects the following information:

(i) an ID identifying the sensor,
(ii) a timestamp specifying when the data was measured,
(iii) traffic intensity (number of vehicles per hour),
(iv) traffic occupation (percentage to which the loop detector area is occupied by a vehicle), and
(v) average velocity over all vehicles.

Because the data provided by the sensors is not labeled with the corresponding traffic state, labeling has to be done manually. The traffic state can be determined according to the fundamental diagram of traffic flow theory by rule-based assignment of traffic classes (see [51]). The traffic states can be derived as follows:

- $free = (occupation < 10 \wedge intensity < 6000 \wedge velocity > 60)\ OR\ (occupation < 2 \wedge intensity < 500)$
- $dense$ = not (congested or free)
- $congested = occupation > 30 \wedge intensity < 6000 \wedge velocity < 40$

The above formulae are now used to insert labels into the sensor data stream that indicate the current state of the traffic.

The traffic data stream contains only one type of event, which carries all traffic measurements at a certain time

[17] Open source traffic data of the city of Madrid can be found in CSV format on the public data portal: https://datos.madrid.es/portal/site/egob

instance. One measurement event contains three attributes: (i) $inten \in [0, 9400]$ indicating traffic intensity, (ii) $occ \in [1, 100]$ indicating traffic occupation, and (iii) $veloc \in [0, 110]$ indicating the average velocity.

For our experiments, we selected data from a loop detector sensor located on the M-30 ring expressway in Madrid. We run two experiments: The first with the sensor data measured in the period 01.01.-30.06.2018 (containing 17,338 events) and the second with the data in the period 01.07.-31.12.2018 (containing 16,851 events).

The learning process takes place in three steps, with three separate rules being learned, one for each traffic state. Consequently, we need three separate training data streams. One with labels representing the *free state* (about 66% of traffic events), another with labels representing the *dense state* (about 30% of traffic events) and a third data set with *congested labels* (only 4% of traffic events).

Again, we conduct hold-out validation. We split the data stream into a training data set and a separate test data set of almost equal size. Table 9 contains the results of the Bat4CEP learning process. The presented results are the results obtained by the best learned rules when applied to the test data (on average over ten runs).

Since the training data stream contains only events of a single event type, a learned rule R^{learn} has to represent the traffic flow state via a complex inequation in the ACT. We receive perfect results for both time periods with very high F1-score, recall and precision values. The very different numbers of labels in the training data streams, high for *free traffic* and low for *congested traffic*, have no notable impact on the learning process. Bat4CEP is able to learn the appropriate conjunctions of threshold values for the attributes identifying a certain state.

6.4 Discussion of experimental results

In rows 1 to 13 of Table 3, we have selected the same set of typical event patterns that has been investigated with a Genetic Programming (GP) based learning algorithm in [37]. Although the GP-based approach already obtained very good learning results, Bat4CEP even outperforms the GP results for the same rule set.

Table 9 Bat4CEP results of learning traffic flow states

No.	Data set	State	F1-score	Recall	Precision
1	1st half 2018	free	0.999	0.999	0.999
2		dense	0.985	0.985	0.985
3		congested	0.993	0.990	0.997
4	2nd half 2018	free	0.999	1.0	0.998
5		dense	0.983	0.970	0.996
6		congested	0.995	1.0	0.990

The results are not exactly comparable because slightly different training data sets with different number of labels have been used. Nevertheless, it can be stated that the GP approach performs very well for most rule patterns. However, there are some critical rule patterns for which the GP-approach could only achieve recall/ precision values below 0.9, which is still good but not perfect.

In contrast, Bat4CEP was able to deliver near perfect results even for the most complex rules with recall/precision values above 0.928. Thus, Bat4CEP seems to be a very reliable learning algorithm, regardless the complexity of the investigated target rules. Moreover, Bat4CEP offers a higher expressiveness of rules compared to the GP algorithm. In addition to the standard CEP operators, Bat4CEP is also capable of learning aggregation functions ($avg()$, $sum()$, $min()$, and $max()$), simple arithmetic operators ($+$ and $-$), the OR operator in ACT, and the NOT operator in ECT which makes the learning task even more challenging. Overall, Bat4CEP results in a very robust learning algorithm as it shows no problems with individual rules.

Figure 4 visualizes the convergence of Bat4CEP and GP-based approach while learning the sample rule no. 12 of Table 3. The graph shows the results of the F1-score of the best candidate rule per timestep/generation (calculated as an average over ten runs each).[18]

Starting from the best randomly generated candidate rule of the initial population, both algorithms improve the quality of the generated rules from timestep to timestep or from generation to generation, respectively. With Bat4CEP, a strong increase in solution quality can be observed already in the first timesteps. An optimal F1-score is reached after approximately 80 timesteps. In contrast, the GP-based approach shows a much slower increase in rule quality, needs much more generations to converge, and, in addition, achieves a good, but not optimal solution quality in the end.

6.5 Implementation issues

Bat4CEP is implemented in Java and uses the common open source CEP engine Esper.[19] The reported experiments were run on a cloud infrastructure with 32 processor units and 64 GB RAM. The available processor power is the most important factor for the runtime of the algorithm.

Due to the characteristics of the underlying bat algorithm, the number of candidate rules generated during a single experimental run differs significantly. In every run, the Bat4CEP algorithm evolves a swarm of 500 bats over (at most) 500 time steps. It is important to notice that each

[18] Please note that both algorithms produce a different number of solutions per timestep/generation.

[19] https://www.espertech.com/esper/

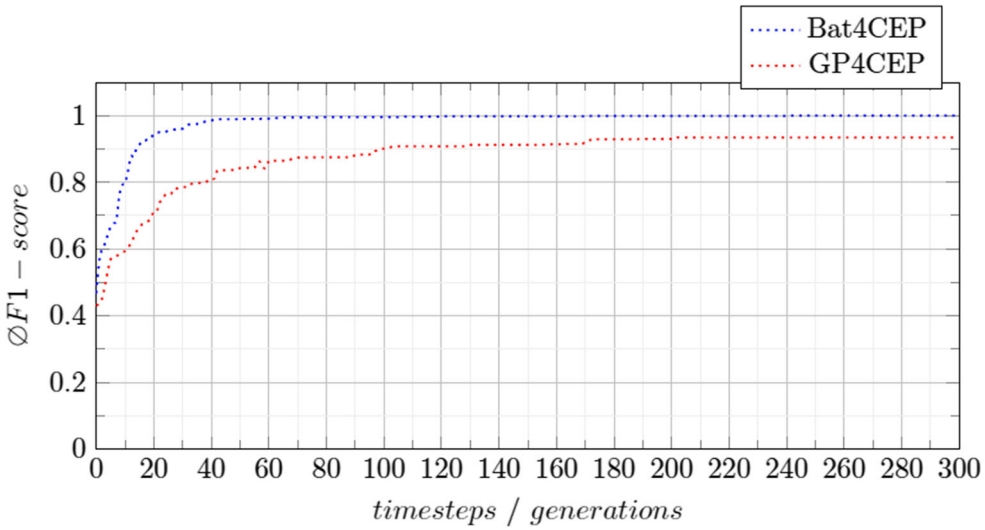

Fig. 4 Comparison of convergence behavior

generated candidate rule has to be deployed in the Esper engine and then the training data stream has to be analyzed to determine the F1-score.

For the synthetic data stream, learning the target rules listed in Table 3 requires the generation of 166,150 to 883,415 candidate rules with a processing time from 5 to 18 minutes. For the real-world traffic scenario, learning the target rules in Table 9 requires the generation of 45,000 to 1,8261,101 rules with a runtime between 6 to 126 minutes.

Obviously, the learning process requires considerable computational effort. The more complex a rule the more candidate rules have to be generated. However, if using a powerful cloud infrastructure, Bat4CEP is able to learn even complex rules in acceptable time although in some cases more than one million candidate rules have to be generated and evaluated.

The greatest computational effort is caused by the calculation of fitness: each candidate rule must be run against the training data stream to calculate TP, TF, FP and FN values. Therefore, re-evaluation of a rule that had already been evaluated in the search history should be avoided. During the evolutionary process, duplicate rules are especially likely to be generated when the rule trees are still small. LTMA (Long Term Memory Assistant for Evolutionary Algorithms) introduced in [52] is an approach to discover re-visited individuals and reuse already calculated fitness values. It can significantly improve the runtime behavior and should be integrated in next versions of Bat4CEP.

Surrogate models are another well-established approach to speeding up evolutionary computation [53]. Expensive exact evaluations are replaced by computationally inexpensive surrogate models. For Bat4CEP, a suitable surrogate model must approximate the fitness of a candidate rule for a given training data stream. Finding such an approximate and computationally cost-effective model is challenging and beyond the scope of this paper.

7 Conclusion

In this paper, we have successfully developed a new bat-inspired metaheuristic for the problem of learning CEP rules. The canonical bat algorithm has been adjusted and extended for mining CEP rule patterns hidden in massive data streams. The Bat4CEP algorithm applies the standard bat algorithm procedure on a swarm of candidate CEP rules (= bats) and derives new candidate solutions by advanced modification operators.

The experimental results with synthetic data and real-world data have proven the feasibility and performance of Bat4CEP. Even complex rule patterns can be learned nearly perfectly with Bat4CEP.

In future work, we will focus on investigating the performance of Bat4CEP on more real-world application scenarios. And, if necessary, to further extend the expressiveness of the rule language that can be learned.

Because each bat behaves autonomously to a great extend, the Bat4CEP algorithm has a high potential to be parallelized. In some approaches, multiple swarms that work concurrently have already been successfully applied in bat algorithms [23, 54]. They propose different approaches how independently acting bat swarms cooperate for finding an optimal solution. A further improvement of the runtime behavior of Bat4CEP could be achieved by using LTMA (Long Term Memory Assistant for Evolutionary Algorithms) [52].

Another exciting future line of work would be to investigate some other modern metaheuristics for the rule learning problem and compare them with Bat4CEP. Some of the latest population-based metaheuristics which have received considerable attention and have successfully solved several real-world problems include whale optimization algorithm (WOA) for the design of silicon-on-insulator FinFET [55], bacterial foraging optimization (BFO) for machine learning [56], and slime mould algorithm (SMA) for tuning of fuzzy controllers [57], just to name of few.

In addition, hybrid approaches attempt to mitigate the disadvantages of a single metaheuristic and combine the advantages of two metaheuristics, e.g. [58].

Acknowledgements We would like to thank our master students Mr. Rene Knop and Mr. Serif Seremet for their support. Mr. Knop developed a first version of the approach in his master thesis and Mr. Seremet provided most of the implementation and experimentation work.

Author Contributions All authors contributed to the study conception and design. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Availability of Data and Materials The datasets generated during and/or analysed during the current study are available at http://sw-architecture.inform.hs-hannover.de/files/bat4cep.zip

Code Availability The source code of the system is available at http://sw-architecture.inform.hs-hannover.de/files/bat4cep.zip

Ethics Approval Not applicable

Consent to Participate Not applicable

Consent for Publication Not applicable

Conflict of Interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Etzion O, Niblett P (2010) Event processing in action. Manning, USA
2. Luckham D (2002) The power of events: An introduction to complex event processing in distributed enterprise systems. Reading, MA
3. Mernik M, Heering J, Sloane AM (2005) When and how to develop domain-specific languages. ACM Comput Surv 37(4):316–344. https://doi.org/10.1145/1118890.1118892
4. Kosar T, Bohra S, Mernik M (2016) Domain-specific languages: a systematic mapping study. Inf Softw Technol 71:77–91. https://doi.org/10.1016/j.infsof.2015.11.001
5. Margara A, Cugola G, Tamburrelli G (2014) Learning from the past: Automated rule generation for complex event processing. In: Proceedings of the 8th ACM international conference on distributed event-based systems. ACM, pp 47–58
6. Mousheimish R, Taher Y, Zeitouni K (2016) Complex event processing for the non-expert with autoCEP: Demo. In: Proceedings of the 10th ACM international conference on distributed and event-based systems, DEBS '16. ACM, New York, pp 340–343
7. Mousheimish R, Taher Y, Zeitouni K (2017) Automatic learning of predictive CEP rules: Bridging the gat between data mining and complex event processing. In: Proceedings of the 11th ACM international conference on distributed and event-based systems, DEBS '17. ACM, New York, pp 158–169
8. Cugola G, Margara A (2012) Processing flows of information: From data stream to complex event processing. ACM Comput Surv 44(3):15:1–15:62. https://doi.org/10.1145/2187671.2187677
9. Bruns R, Dunkel J (2015) Complex Event Processing: Komplexe Analyse von massiven Datenströmen mit CEP. Springer Vieweg, Wiesbaden. in German
10. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th international conference on very large data bases, VLDB '94. Morgan Kaufmann Publishers Inc., San Francisco, pp 487–499
11. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. SIGMOD Rec 29(2):1–12. https://doi.org/10.1145/335191.335372
12. Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco
13. Chen M, Ludwig SA (2012) Discrete particle swarm optimization with local search strategy for rule classification. In: Fourth world congress on Nature and Biologically Inspired Computing (NaBIC), pp 162–167
14. Gupta M, Ram S (2012) Application of weighted particle swarm optimization in association rule mining. Int J Comput Sci Inf 1
15. Hassani K, Lee WS (2013) An incremental parallel particle swarm approach for classification rule discovery from dynamic data. In: 12th international conference on machine learning and applications, vol 1, pp 430–435
16. Parpinelli RS, Lopes HS, Freitas AA (2002) Data mining with an ant colony optimization algorithm. IEEE Trans Evol Comput 6(4):321–332. https://doi.org/10.1109/TEVC.2002.802452
17. Liu B, Abbas HA, McKay B (2003) Classification rule discovery with ant colony optimization. In: IEEE/WIC international conference on intelligent agent technology, 2003. IAT 2003, pp 83–88
18. Smaldon J, Freitas AA (2006) A new version of the ant-miner algorithm discovering unordered rule sets. In: Keijzer M (ed) 2006 Genetic and Evolutionary Computation Conference, vol 1. ACM Press, New York, pp 43–50
19. Khan K, Sahai A (2012) A fuzzy c-means bi-sonar-based metaheuristic optimization algorithm. Int J Artif Intell Interact Multimed 1(7):26–32. https://doi.org/10.9781/ijimai.2012.173

20. Damodaram R, Valarmathi ML (2012) Phishing website detection and optimization using modified bat algorithm. Int J Eng Res Appl 2:870–876
21. Song A, Ding X, Chen J, Li M, Cao W, Pu K (2016) Multi-objective association rule mining with binary bat algorithm. Intell Data Anal 20(1):105–128
22. Heraguemi KE, Kamel N, Drias H (2014) Association rule mining based on bat algorithm. In: Pan L (ed) Bio-Inspired Computing - Theories and Applications: 9th International Conference, BIC-TA 2014, Wuhan, China, October 16-19, 2014. Proceedings. Springer, Berlin, pp 182–186
23. Heraguemi KE, Kamel N, Drias H (2016) Multi-swarm bat algorithm for association rule mining using multiple cooperative strategies. Appl Intell 45(4):1021–1033. https://doi.org/10.1007/s10489-016-0806-y
24. DeLaHiguera C (2010) Grammatical inference: Learning automata and grammars. Cambridge University Press, Cambridge
25. Kovačević Ž, Mernik M, Ravber M, Črepinšek M (2020) From grammar inference to semantic inference—an evolutionary approach. Mathematics,(8), 5. https://doi.org/10.3390/math8050816
26. Hrnčič D, Mernik M, Bryant BR, Javed F (2012) A memetic grammar inference algorithm for language learning. Appl Soft Comput 12(3):1006–1020. https://doi.org/10.1016/j.asoc.2011.11.024
27. Simsek MU, YildirimOkay F, Ozdemir S (2021) A deep learning-based cep rule extraction framework for IoT data. The Journal of Supercomputing. https://doi.org/10.1007/s11227-020-03603-5
28. Wanner J, Wissuchek C, Janiesch C (2020) Machine learning and complex event processing - a review of real-time data analytics for the industrial internet of things. Enterprise Model Inf Syst Architect (EMISAJ) - Int J Conceptual Model 15:1–27. https://doi.org/10.18417/emisa.15.1
29. Frömmgen A, Rehner R, Lehn M, Buchmann A (2015) Fossa: Learning ECA rules for adaptive distributed systems. In: Proceedings of the IEEE International Conference on Autonomic Computing (ICAC). IEEE, pp 207–210
30. Weiss GM, Hirsh H (1998) Learning to predict rare events in event sequences. In: Proceedings of the 4th international conference on knowledge discovery and data mining. AAAI Press, New York, pp 359–363
31. Sen S, Stojanovic N, Stojanovic L (2010) An approach for iterative event pattern recommendation. In: Proceedings of the Fourth ACM international conference on distributed event-based systems, DEBS '10. ACM, New York, pp 196–205
32. Turchin Y, Gal A, Wasserkrug S (2009) Tuning complex event processing rules using the prediction-correction paradigm. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS '09. ACM, New York, pp 10:1–10:12
33. Pielmeier J, Braunreuther S, Reinhart G (2018) Approach for defining rules in the context of complex event processing. Procedia CIRP 67:8–12. 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 19-21 July 2017, Gulf of Naples, Italy
34. Mehdiyev N, Krumeich J, Enke D, Werth D, Loos P (2015) Determination of rule patterns in complex event processing using machine learning techniques. Procedia Comput Sci 61:395–401
35. Petersen E, Antonio To M, Maag S, Yamga T (2018) An unsupervised rule generation approach for online complex event processing. In: IEEE 17th international symposium on network computing and applications (NCA). IEEE, Cambridge, pp 1–8
36. Mousheimish R, Taher Y, Zeitouni K (2016) Automatic learning of predictive rules for complex event processing: Doctoral symposium. In: Proceedings of the 10th ACM international conference on distributed and event-based systems, DEBS '16. ACM, New York, pp 414–417
37. Bruns R, Dunkel J, Offel N (2019) Learning of complex event processing rules with genetic programming. Expert Syst Appl 129:186–199. https://doi.org/10.1016/j.eswa.2019.04.007
38. Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: González JR, Pelta DA, Cruz C, Terrazas G, Krasnogor N (eds) Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), studies in computational intelligence. Springer, Berlin, pp 65–74
39. Yang X-S, He X (2013) Bat algorithm: Literature review and applications. Int J Bio-Inspired Comput 5(3):141–149. https://doi.org/10.1504/IJBIC.2013.055093
40. Al-Betar MA, Awadallah MA (2018) Island bat algorithm for optimization. Expert Syst Appl 107:126–145. https://doi.org/10.1016/j.eswa.2018.04.024
41. Alsalibi B, Abualigah L, Khader AT (2020) A novel bat algorithm with dynamic membrane structure for optimization problems. Appl Intell 51:1992–2017. https://doi.org/10.1007/s10489-020-01898-8
42. Soliman OS, Elhamd EA (2015) A chaotic levy flights bat algorithm for diagnosing diabetes mellitus. Int J Comput Appl 111(1):36–42
43. Saji Y, Barkatou M (2021) A discrete bat algorithm based on levy flights for euclidean traveling salesman problem. Expert Syst Appl 172:114639. https://doi.org/10.1016/j.eswa.2021.114639
44. Dao T-K, Pan T-S, Nguyen T-T, Pan J-S (2018) Parallel bat algorithm for optimizing makespan in job shop scheduling problems. J Intell Manuf 29(2):451–462. https://doi.org/10.1007/s10845-015-1121-x
45. Taramasco C, Olivares R, Munoz R, Soto R, Villar M, de Albuquerque VHC (2019) The patient bed assignment problem solved by autonomous bat algorithm. Appl Soft Comput 81:105484. https://doi.org/10.1016/j.asoc.2019.105484
46. Akhtar S, Ahmad AR, Abdel-Rahman EM (2012) A metaheuristic bat-inspired algorithm for full body human pose estimation. In: Ninth conference on computer and robot vision, pp 369–375
47. Jayabarathi T, Raghunathan T, Gandomi AH (2018) The bat algorithm, variants and some practical engineering applications: A review. In: Yang X-S (ed) nature-inspired algorithms and applied optimization. Springer International Publishing, Cham, pp 313–330
48. Knop R (2017) Ein Bat-Algorithmus zum Lernen von Complex Event Processing Regeln aus Ereignisdaten. Master's Thesis, Hochschule Hannover, Department of Computer Science
49. Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, UK. With contributions by J. R. Koza
50. Koza JR (1992) Genetic programming: On the programming of computers by means of natural selection. MIT Press, Cambridge
51. Helmers M (2018) Analyse von Verkehrssensordaten mit dem Datamining-Tool WEKA am Beispiel von Realdaten der Stadt Madrid. Master's Thesis, Hochschule Hannover, Department of Computer Science
52. Črepinšek M, Liu S-H, Mernik M, Ravber M (2019) Long term memory assistance for evolutionary algorithms. Mathematics 7. https://doi.org/10.3390/math7111129
53. Jin Y (2011) Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm Evol Comput 1(2):61–70. https://doi.org/10.1016/j.swevo.2011.05.001
54. Wang GG, Chang B, Zhang Z (2015) A multi-swarm bat algorithm for global optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp 480–485
55. Kaur G, Gill SS, Rattan M (2020) Whale optimization algorithm for performance improvement of silicon-on-insulator FinFET. Int J Artif Intell 18:63–81

56. Chen H, Zhang Q, Luo J, Xu Y, Zhang X (2020) An enhanced bacterial foraging optimization and its application for training kernel extreme learning machine. Appl Soft Comput 86
57. Precup R-E, David R-C, Roman R-C, Szedlak-Stinean A-I, Petriu EM (2021) Optimal tuning of interval type-2 fuzzy controllers for nonlinear servo systems using slime mould algorithm. Int J Syst Sci:1–16. https://doi.org/10.1080/00207721.2021.1927236
58. Elaziz MA, Heidari AA, Fujita H, Moayedi H (2020) A competitive chain-based harris hawks optimizer for global optimization and multi-level image thresholding problems. Appl Soft Comput 95:106347. https://doi.org/10.1016/j.asoc.2020.106347

Ralf Bruns is a professor of computer science at the Hochschule Hannover (University of Applied Sciences and Arts). He received a Diploma degree and a Doctoral degree in computer science from University of Oldenburg (Germany). His research interests include software architecture, complex event processing, evolutionary and swarm algorithms. Prof. Bruns has authored several research papers that focus on the application of artificial intelligence techniques to real-world problems.

Jürgen Dunkel is a professor of computer science at the Hochschule Hannover (University of Applied Sciences and Arts). He received a Diploma degree in computer science from University of Dortmund and a Doctoral degree from University of Hagen (Germany). His research interests include data stream processing, recommender systems and swarm algorithms. He is a member of the German Computer Science Society.