Article

Towards Bio-Inspired Anomaly Detection Using the Cursory Dendritic Cell Algorithm

Carlos Pinto ^{1,*}, Rui Pinto ^{1,2,*} and Gil Gonçalves ^{1,2} ¹ Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal; gil@fe.up.pt² SYSTEC—Research Center for Systems and Technologies, Rua Dr. Roberto Frias, s/n, office i219, 4200-465 Porto, Portugal

* Correspondence: up201605439@edu.fe.up.pt (C.P.); rpinto@fe.up.pt (R.P.)

Abstract: The autonomous and adaptable identification of anomalies in industrial contexts, particularly in the physical processes of Cyber-Physical Production Systems (CPPS), requires using critical technologies to identify failures correctly. Most of the existing solutions in the anomaly detection research area do not consider such systems' dynamics. Due to the complexity and multidimensionality of CPPS, a scalable, adaptable, and rapid anomaly detection system is needed, considering the new design specifications of Industry 4.0 solutions. Immune-based models, such as the Dendritic Cell Algorithm (DCA), may provide a rich source of inspiration for detecting anomalies, since the anomaly detection problem in CPPS greatly resembles the functionality of the biological dendritic cells in defending the human body from hazardous pathogens. This paper tackles DCA limitations that may compromise its usage in anomaly detection applications, such as the manual characterization of safe and danger signals, data analysis not suitable for online classification, and the lack of an object-oriented implementation of the algorithm. The proposed approach, the Cursory Dendritic Cell Algorithm (CDCA), is a novel variation of the DCA, developed to be flexible and monitor physical industrial processes continually while detecting anomalies in an online fashion. This work's contribution is threefold. First, it provides a comprehensive review of Artificial Immune Systems (AIS), focusing on AIS applied to the anomaly detection problem. Then, a new object-oriented architecture for the DCA implementation is described, enabling the modularity and abstraction of the algorithm stages into different classes (modules). Finally, the CDCA for the anomaly detection problem is proposed. The CDCA was successfully validated in two industrial-oriented dataset benchmarks for physical anomaly and network intrusion detection, the *Skoltech Anomaly Benchmark (SKAB)* and *M2M using OPC UA*. When compared to other algorithms, the proposed approach exhibits promising classification results. It was placed fourth on the *SKAB* scoreboard and presented a competitive performance with the incremental Dendritic Cell Algorithm (iDCA).

Keywords: Artificial Immune Systems; Dendritic Cell Algorithm; Cyber-Physical Production Systems; anomaly detection; fault detection; Industry 4.0; smart manufacturing systems

Citation: Pinto, C.; Pinto, R.; Gonçalves, G. Towards Bio-Inspired Anomaly Detection Using the Cursory Dendritic Cell Algorithm. *Algorithms* **2022**, *15*, 1. <https://doi.org/10.3390/a15010001>

Academic Editor: Jie Zhang

Received: 26 November 2021

Accepted: 17 December 2021

Published: 21 December 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Industry 4.0 (I4.0) is a highly appealing paradigm established originally in Germany and spread throughout Europe. It is characterized as the merger of the Internet and adaptable items ("smart" [1] machinery and goods) with existing digitized industry. Access to the Internet through Cyber-Physical Production Systems (CPPS) is one of the primary facilitators of I4.0, and this combination is commonly referred to as the Industrial Internet of Things (IIoT). The use of the Internet of Things (IoT) in I4.0 enables devices to be more flexible in how they interface and behave, as well as the integration of significantly more complex architectures in smart and autonomic industrial scenarios, such as components that process massive amounts of data (Big Data), and distributed computation in the Cloud/Edge layer.

This synergy among technologies enables CPPS components in I4.0 to be autonomic since components can organize themselves in a network based on the context in which they are inserted, can monitor their environment, assess potential problems, and adapt, resulting in a decentralized system [1]. As time passes, CPPS tend to have more and more devices linked in a network, greater Internet access, and less human involvement [2], as CPPS is often viewed as a vital element in the notion of IIoT and the Fourth Industrial Revolution.

At their foundation, CPPS are a combination of computing and physical processes that are aligned with communication networks [3]. While typical embedded systems integrate computing with real-world interactions, CPPS place a particular emphasis on inter-device communication, according to Jazdi [3]. They are frequently integrated into feedback loops in which the cyber component (control, communication and computation in general) and physical component (the physical process, measurements by sensors and actuator impacts) are tightly coupled, just as with embedded systems, but with the added component of subsystem interactions due to their ability to communicate with each other. This increases the complexity and dynamism of such systems, which are frequently challenging to manage in centralized and prescribed techniques.

A CPPS is thought to be a safety-critical system in many of its application scenarios, where a variation in conventional behavior might result in substantial economic losses and material damage, if not human lives. Over the years, several attacks have tackled Industrial Control Systems (ICS), which had a considerable impact on the physical processes. Some known examples are the *Stuxnet worm* [4], the *Ukraine SCADA attack* [5], the *Mirai attack* [6], Maroochy Water Services in Australia (2000), German Steel Mill Thyssenkrupp (2014), and Norsk Hydro Aluminium (2019) [7].

To address this problem, designing robust CPPS is critical, mainly when under equipment faults, caused by attacks or not. Furthermore, due to the rise of automation and system dynamics, increased complexity makes assaults and naturally occurring errors more difficult to detect, analyze, and resolve by a human operator. This motivates the need for CPPS to identify such anomalies and take relevant action. Furthermore, since the reaction to mitigate the impact of anomalies should be fast, CPPS have real-time needs [8].

This work proposes a method for implementing anomaly detection as a tool for detecting anomalies. The underlying physical process at the shop floor is the focus of anomaly detection, as it is frequently disregarded in typical anomaly detection problems, which consider the computer security domain [9,10]. These objectives are further narrowed down with a solution proposal in the field of Artificial Immune Systems (AIS). The field of AIS delivers immunology-based solutions and demonstrates potential qualities such as adaptivity, scalability, and lightweight solutions. It is, however, mostly ignored by the technical community. As a result, it is critical to evaluate its suitability for addressing the difficulties outlined in anomaly detection problems.

So, considering that immune analogies are an excellent fit for the anomaly detection problem, with the potential for lightweight, yet effective solutions, this work has three main goals: (i) review AIS in terms of biological background and algorithm characterization; (ii) develop a real-time process-based anomaly detection, using the Dendritic Cell Algorithm (DCA) as immune inspiration, with CPPS resource and time constraints in mind; (iii) validate the developed technique using data that are representative of a real-world scenario of CPPS operation.

Thus, considering these goals, the main contributions presented in this work are:

- A comprehensive characterization of the significant immune-inspired techniques, namely Negative Selection Algorithms (NSA), Clonal Selection Algorithms (CSA), Artificial Immune Networks (AIN), and DCA.
- Abstraction of the DCA, considering an object-oriented approach.
- Proposal of the Cursory Dendritic Cell Algorithm (CDCA) to tackle DCA limitations, such as online classification and strategy for problem modeling.

- Validation of the CDCA suitability in anomaly detection testbeds, namely the *Skoltech Anomaly Benchmark (SKAB)* benchmark (equipment's physical anomalies) and *M2M using OPC UA* (network-based anomalies).

The paper is organized into five more sections. Section 2 provides a comprehensive characterization of AIS. In Section 3, a description of the anomaly detection problem is provided while presenting existing solutions found in the literature regarding immune-based anomaly detection. Section 4 describes in detail the proposed algorithm, the CDCA. Section 5 describes the methodology for testing, by characterizing the *SKAB* and *M2M using OPC UA* datasets, the deployment process of the CDCA in the datasets while presenting the main results for performance validation. In Section 6, an analysis of the proposed approach based on the experimental results is provided. Finally, Section 7 concludes the paper, stating final remarks about the anomaly detection algorithm and providing orientations for future work.

2. Artificial Immune Systems

AIS are algorithms based on immunology that have been applied in computer science and engineering problems. Their principal purpose is to solve engineering problems using recognized immunology theories to establish a correlation between a problem and the immune system. Its development as a discipline is still in its early stages, and research regarding immune-based anomaly detection solutions is still in its infancy. Because it draws inspiration from biological functions, AIS is bio-inspired, and advancement in this subject is intimately linked to the biological study of the Human Immune System (HIS).

One of the core principles of any AIS algorithm is that the HIS may be used as inspiration to create functional algorithms for various applications, since it is a system that has been tried and conclusively demonstrated biologically over time. However, because the primary purpose is to solve a problem rather than provide an exact reproduction of the immune system, abstractions are more oriented to problem solving and not HIS model correlation. Furthermore, using immunology as a source of inspiration might be problematic, as incorrect ideas or an inability to describe very complex immunological events can lead to poor algorithm abstractions.

Computer security, anomaly detection, fault-tolerant control, pattern recognition, clustering, and optimization are applications of AIS in the existing literature. Because AIS is a relatively new scientific topic, there is no commonly accepted categorization into sub-fields. However, an AIS taxonomy is represented in Figure 1, which is used in this work. Each sub-field relates to a distinct biological model of the immune system, which will be briefly presented, with specific scientific terminology of biological phenomena and corresponding computer science algorithms.

Figure 1. Artificial Immune System (AIS) taxonomy used in this work [11].

Thus, the four prominent families of AIS are introduced in this section, including those based on self/nonself theory, clonal selection theory, immune network theory, and danger theory. We review the biological background of the immune models and the original algorithm, such as NSA, CSA, AIN, and DCA. Note that several variations of the existing algorithms were proposed to improve one or more algorithm limitations or adapt the algorithm to be used in different contexts or application scenarios. These variations consist of updating the algorithm's overall functionality, usually by proposing a new algorithm architecture with edited software components, or by combining the algorithm with others

(resulting in ensembles). In the end, these updates may result in added features to the algorithm or with improved performance.

2.1. Negative Selection Algorithm

Forrest et al. [12] suggested the NSA for the first time in 1994. It was initially utilized in computer virus detection using anomaly detection and is based on the biological self–nonself principle.

2.1.1. Biological Background: Self/Nonself

According to Matzinger [13], the primary premise of the self–nonself theory is that the immune system recognizes organisms that are foreign to the body (nonself) and that their presence triggers an immune response. In the HIS, this is achieved through a process of negative selection in the thymus, where immature *T cells* become mature (step *a* in Figure 2). It is exceedingly unlikely that intruder organisms will infiltrate the thymus, which is a protected area where, ideally, only “self” *antigens* reside. *T cells* that detect “self” *antigens*, i.e., the body’s own distinctive *antigens*, receive an apoptotic signal and are destroyed (step *b* in Figure 2). This way, developed *T cells* leaving the thymus in a matured state can only identify “nonself” *antigens*, resulting in a widespread system of agents that can tolerate the system and detect invaders (step *c* in Figure 2).

Figure 2. Simplified diagram of self–nonself theory. Note that self and also nonself *antigens* are immensely varied. For illustrative purposes only, in this figure all self *antigens* are identical. It consists of 3 main steps: (a) Immature *T cells* with various *antigen* receptors are produced in the bone marrow and migrate to the thymus; (b) In the thymus, *T cells* that recognize self *antigens* undergo apoptosis. On the other hand, *T cells* that don’t recognize any self *antigen* are allowed to live; (c) Mature *T cells* that don’t recognize self *antigens* are now used to recognize any non self *antigen* such as virus and bacteria.

2.1.2. Algorithm Overview

NSA is based on Burnet's self-nonsel self methodology [14], particularly the negative selection of *T cells*. For the goal of anomaly-based intrusion detection, the analogy is straightforward, resulting in this application being the same as how Forrest et al. [12] utilized the approach for the first time. The technique operates by creating "detectors", which are *T cell*-like structures. These detectors can be created in various ways, but they are usually randomly produced within a problem-specific search area. Then, an appropriate measure must be created to compare these detectors to data points from known baseline behavior.

Detectors that identify data points from this expected behavior are deleted, while those that do not detect normal behavior are preserved for further use. No more detectors are created after a stop condition is met, and the now "mature" detector population is utilized for the anomaly classification of new data points. This way, if these detectors match any data point, these data are most likely an anomaly. The NSA's detectors are created according to Algorithm 1.

Algorithm 1: General Negative Selection Algorithm (NSA) Detector Set Generation.

```

Input:  $S_{self}$  set of points that are normal
Output:  $D$  detector set
while not stop_condition() do
    self_detected = True;
    while self_detected do
         $d = \text{generate\_new\_detector}();$ 
        foreach  $s_i$  in  $S_{self}$  do
            if detect( $d, s_i$ ) then
                break;
            end
        self_detected = False;
    end
     $D \leftarrow d$ 
end
return  $D$ 

```

According to the desired application scenario, the following must be defined in order to implement NSA:

- *Detectors*: How should they be represented? They could be represented by real-valued arrays and strings, for example. The nature of the detectors is also linked to the classification of the input data. The following must also be defined, depending on the kind of detector.
 - *Detection*: This is a method that produces a binary classification of detection when given a detector and a point to categorize ("detected" or "not detected"). For example, common n-contiguous bits/elements and Euclidean distance vs. threshold.
 - *Detector generation*: Generation is a process for creating new detectors—for example, random detector generation.
- *Stop criterion*: When to cease creating detectors during the training phase is defined by the stop criterion. For example, the predicted search space coverage and the number of detectors reached.

2.2. Clonal Selection Algorithm

In 1999, De Castro et al. [15] suggested the first instance of a CSA. It was originally employed in pattern recognition and is based on the clonal selection hypothesis, which explains acquired immunity.

2.2.1. Biological Background: Clonal Selection

Clonal selection theory [14] tries to explain how *B cells* are chosen and preferred based on their *antigen* affinity. *T cells* regulate the function of *B cells*, which secrete *antibodies* that can fight harmful organisms. Foreign *antigens* are gathered and delivered to *B cells*. *B cells* with receptors that detect these *antigens* (step *b* in Figure 3) will be divided into two groups (step *c* in Figure 3). *B cells* react to *antigens* in two ways: (i) on the one hand, they differentiate into short-lived *plasma cells* that secrete *antibodies* required to start an immune response (step *d.1* in Figure 3); (ii) the *plasma cells*, on the other hand, clone themselves, resulting in long-lasting *memory cells* (step *d.2* in Figure 3). The *memory cells* are clones of their parents, but they have been exposed to hypermutation (rapid mutation) in order to produce children with stronger affinity than their parents. Due to the hypermutations they were exposed to, this results in a pool of memory *B cells* that are proportionate to their affinity for the most common *antigens* that the body has encountered in the past, and these *memory cells* can readily respond to future occurrences of the *antigen* that lead to its proliferation, with even higher efficacy than their parents.

Figure 3. Simplified diagram of clonal selection theory. A foreign organism contains specific *antigen* that can be recognized by *B cells'* receptors with high affinity. *B cells* that recognize those *antigen* are cloned, resulting in *plasma cells* and *memory cells*. Short-lived *plasma cells* release *antibodies* that bond to the considered *antigen* and inhibit the foreign organism. Long-lived *memory cells* will react quickly in future occurrences of same *antigens*.

2.2.2. Algorithm Overview

The clonal selection hypothesis is the inspiration of the CSA, especially the selection of *B cells* based on *antigen* affinity, selective cloning of chosen *B cells*, and subsequent mutation and differentiation of selected *B cells* into *memory cells*. Although relevant in the context of clonal selection theory, differentiation into *plasma cells* is not implemented in this subgroup's algorithms.

The main premise of these algorithms is to choose the greatest (or lowest) affinity, store them in a pool, and develop new candidate solutions based on modest modifications to the prior best solutions. Candidate solutions are analogous to *B cells* or, more directly, the *antibodies* they generate. The poorest solutions are eliminated, leaving just the best possibilities. However, additional candidates can be produced to allow the algorithm to explore more options. CSA is similar to Genetic Algorithms (GA) in that they both follow Darwin's "survival of the fittest" principle, in which the most robust solutions survive to the next generation. However, they differ since CSA clones are mutated solutions between generations. Algorithms from this branch are generally related to function optimization, and pattern recognition since the method seeks to maximize/minimize the entire population's affinity [16].

Algorithm 2 describes how the population of the CSA is created and chosen over time.

Algorithm 2: General Clonal Selection Algorithm (CSA), adapted from [17].

```

Input:  $N$  population size
 $n$  number of antibodies to clone
 $d$  number of new antibodies to be generated each iteration
Output:  $P$  best population
 $P \leftarrow \text{generate\_antibodies}(N)$ 
while not stop_condition() do
    foreach  $p$  in  $P$  do
        | update_affinity( $p$ )
    end
     $P' \leftarrow \text{select}(P, n)$  foreach  $p'$  in  $P'$  do
        |  $C \leftarrow \text{clone}(p')$ 
    end
    foreach  $c$  in  $C$  do
        | hypermutate( $c$ )
    end
    foreach  $c$  in  $C$  do
        | update_affinity( $c$ )
    end
     $P \leftarrow \text{insert}(C, n)$ 
     $P_{\text{new}} \leftarrow \text{generate\_antibodies}(d)$ 
     $P \leftarrow \text{replace}(P, P_{\text{new}})$ 
end

```

According to the desired application scenario, the *antibody* concept must be defined in order to implement CSA, i.e., how *antibodies* symbolize potential solutions in the realm of the problem. Because CSA has many uses, defining what a "candidate solution" means is difficult—for example, a bit-string to match patterns, or a collection of parameters to apply to a problem (such as hyperparameters for a machine learning model, or scheduling times for scheduling problems). Given the structure of the *antibody*, the following must also be defined:

- *Affinity*: This considers how to rank candidate *antibodies* for selection based on affinity. In general, stronger affinity leads to better solutions, yet in problems involving minimizing a cost function, higher affinity might lead to worse solutions. For exam-

- ple, the number of matches of a certain bit-string to a target set of bit-strings, or the accuracy of a machine learning model utilizing a particular set of hyperparameters.
- *Mutation*: This is related to how to change *antibodies* slightly through the mutation of bit-flips in bit-strings, for example, or adding/multiplying a random real-valued integer to a single random hyperparameter of a machine learning model are examples of a “mutation”.
 - *Generation*: Making new *antibodies* is referred to as “generation”. Random bit-strings, for example, or hyperparameter sampling at random from a parameter sweep (also known as grid search) search space.
 - *Selection*: This is related to how to choose which *antibodies* will survive. In the algorithm, selection can take several forms, including which *antibodies* are chosen to be cloned, which are chosen to survive, and which are replaced with newly created *antibodies*. The selection process is usually performed by simply selecting the highest-ranking *antibodies* based on affinity. However, novel *antibodies* can be left to live or cloned to potentially identify additional optimum solutions and prevent converging to local optima.
 - *Cloning*: This is related to how to distribute clones in cloning. The cloning process involves duplicating, but the number of copies to make and then modify may be controlled in various ways. Create clones proportionate to affinity rankings, clones of x best *antibodies*, and clones of the total population, for example.
 - *Stop criterion*: When to cease producing solutions is determined by the stop criterion, e.g., an onerous restriction on the number of iterations or a decrease in affinity across generations.

2.3. Artificial Immune Network

Early work in AIN was proposed independently in 2000 by De Castro et al. [18] and Timmis et al. [19] for data-clustering applications. It is based on the immunological network theory, which explains that acquired immunity is highly dynamic.

2.3.1. Biological Background

The immune network hypothesis, also known as the idiotypic network theory [20], suggests that the immune system is more dynamic than previously thought. Instead of waiting for exogenous organisms to infiltrate and provoke a reaction, the immune system continually stimulates itself. According to this notion, immune cells not only identify foreign *antigens*, but also recognize each other (through receptors or free *antibodies*), resulting in a loose and dynamic network of immune cells. The basic assumption is that the immune system learns through time, not just through the *antigen* activation of individual cells, but rather through the network of immune cells as a whole.

This theory implies that tolerating an *antigen* does not necessarily imply that the *antigen* was not recognized, but rather that it was recognized by other immune cells. These cells were then recognized by other cells, which suppressed the former cells’ response or even eliminated some of these cells. In contrast, an *antigen* may be detected by specific immune cells, and this might be amplified by other immune cells that identify the former ones. In this view, every cell may stimulate or inhibit matching cells that it recognizes, influencing how the system responds to *antigens* indirectly, and therefore the immunological network theory can be viewed as an extension of the clonal select theory.

The immune network theory is not as evident among immunologists as the other theories discussed in this study. According to Hoffman [21], the “proposal initially seemed to many immunologists to make a complex subject even more complex, in fact unmanageably so”. Several models attempt to describe the dynamics of the network, e.g., Figure 4 depicts the Richter model [22], more particularly, a situation that shows how many *B cells* (and their *antibodies*) might react to *antigens*, even if the majority of the cells do not directly react to the *antigens* that initiated the response.

Figure 4. Richter’s model [22] of an *antibody* chain. Each *antibody* is labeled *Abx* and the *antigens* are pictured as dark squares. This example showcases how an *antigen* can trigger a cascade of reactions. The presence of the *antigen* is initially recognized by *Ab1*, which increases in number. This leads to *Ab2* to start recognizing *Ab1* that prompts *Ab2* to increase in number and so on. As the *antibodies* increase due to stimulation, they also inhibit the *antibodies* lower on the chain, regulating the response. An increase in *Ab4* will lead to a reverse cascade to gradually suppress the subsequent *antibodies*.

2.3.2. Algorithm Overview

AIN investigates immunological cell interconnection, namely their reciprocal suppression/stimulation mechanism. These algorithms’ primary goal is to generate dynamically connected populations over generations. AIN differs from CSA in that the population’s development is no longer personalized, and there is always some interaction between the population’s participants. Algorithms under this category are vastly different in interpreting the biological theory to solve problems, and attention is given to clustering problems.

In clustering, populations are typically represented as graphs, with nodes representing cells and edges representing a specific cell’s stimulation/suppression of other cells it “recognizes”. This recognition is abstracted as affinity, generally by some measure of similarity, such as Euclidean distance [18]. Each cell will preferentially clone itself through this affinity and may mutate or die off, resulting in clusters of cells representing the original data clusters. Each of these clusters is made up of numerous *B cells*. Algorithm 3 shortly introduces Artificial Immune Network (AINE) [23], where each *B cell* cluster includes a data representation, number of *B cells*, and stimulation values.

2.4. Danger-Inspired Algorithms

Danger-inspired algorithms refer to an algorithm that is inspired by the danger theory [13]. This subfield is very new, with the most notable method being DCA, which was initially proposed by Greensmith et al. [24], and was implemented for anomaly-based intrusion detection.

2.4.1. Biological Background

Danger theory [13] addresses significant flaws noted in the NSA. Its goal is to explain how the HIS may initiate an immune response to self cells, such as cancer cells, while avoiding immune responses to the nonself but useful or harmless things, such as gut microbiota. According to the hypothesis, while the immune system can distinguish between

the self and nonself, it is not this process that causes immunological reactions. Instead, a unique type of cell, the *Dendritic Cell* (DC), acts as a biological detective, navigating through human tissue and looking for situations representing danger. These DCs begin their life cycle as immature cells (*iDCs*), collecting whatever *antigens* they discover, as well as recognizing cell-mediated signals from the tissue itself and *Pathogen-associated molecular patterns* (PAMP)—chemicals that operate as markers of pathogenic presence.

Algorithm 3: A specific Artificial Immune Network (AIN) implementation for generating an *B cells* cluster population, based on Artificial Immune Network (AINE) algorithm [23].

Input: A input data (*antigens*)
 N population size
 NAT network affinity threshold
 R_m mutation rate
 N_c number of clones on each ARB
Output: S_{ARB} set of all ARBs
 $S_{ARB} \leftarrow \text{initialize_population}()$
 $C_{ARB} = \{\}$
while True **do**
 foreach arb in S_{ARB} **do**
 foreach a in A **do**
 $\text{present_antigen}(p)$
 end
 end
 foreach arb in S_{ARB} **do**
 $\text{update_stimulation}(arb)$
 end
 foreach arb in S_{ARB} **do**
 $\text{allocate_b_cells}(arb)$
 end
 if $\text{stop_condition}()$ **then**
 break
 end
 $C_{ARB} = \text{clone_and_mutate}(S_{ARB})$ **foreach** arb in C_{ARB} **do**
 $S_{ARB} \leftarrow arb$
 end
end
return S_{ARB}

The origin of these signals is unknown, but the idea behind the DC functionality is that tissue cells can communicate whether they are in distress (e.g., when cells are destroyed in unnatural ways or the presence of PAMP) or if they are safe (e.g., the cell died naturally through programmed cell death). As the immature DC gathers *antigens* and recognizes tissue signals, it eventually develops into either a mature (*mDC*) or semi-mature (*smDC*) states, depending on whether it comes from a “threatening” or a “safe” context. They then travel to a neighboring lymph node, where *T cells* are present, and deliver the *antigens* they have gathered to those *T cells*. *T cells* that detect *mDC antigens* may initiate an immunological response. Conversely, *T cells* that recognize *smDC antigens* will tolerate the *antigens* given. Figure 5 depicts the interaction between DCs, the organism’s tissue, and *T cells* from the standpoint of the danger theory.

Figure 5. Simplified model of the danger theory. Note that immature *dendritic cells* (*iDCs*) collect a vast variety of *antigens*, not just one specific type, as illustrated for simplicity purposes. Initially, *iDCs* navigate throughout the body, collecting encountered *antigens* without discriminating between self or non-self. While navigating, *iDCs* also collect signals emitted by dying tissue cells. Damaged cells release chaotically their contents, which are identified as *danger signals* by neighboring *iDCs*. On the other hand, when cell death is programmed, *iDCs* still collect *antigens* while recognizing *safe signals*. After collecting a sufficient amount of signals, a *iDC* evolves into either mature *dendritic cells* *mDC* or semi-mature *dendritic cells* *smDC*, depending if the overall context during *antigens* collection was dangerous or safe. In both cases, *DC* migrate and present their collected *antigens* to *T cells*. *T cells* that recognize the *antigens* presented by *mDCs* proceed to initiate an immune response, or they can tolerate *antigens* if presented by *smDCs*.

2.4.2. Algorithm Overview

Danger theory algorithms are inspired by the notion of cells emitting signals that characterize their state (normal or in distress), which shows that the immune system is activated by the pathogen's destructive effects on tissue rather than by explicitly recognizing the pathogen. Considering the DCA [24], four signals can be defined, such as *safe*, *danger*, *PAMP*, and *inflammatory cytokines*. There are other algorithms, such as the Toll-like receptor (TLR) algorithm [25], which specifies what *PAMP* is and focuses on the definition of a *T cell* matching measure during the *DC* presenting stage.

Greensmith's thesis [26] suggested DCA, inspired by the danger theory, for anomaly detection applied to intrusion detection. This technique is directly inspired by *DCs*, which are cells that gather data indiscriminately (*antigens*), and it is the most important danger-inspired algorithm currently. Algorithm 4 represents the DCA algorithm. When using DCA, there is no training phase. Instead, input signals, such as *danger*, *safe*, *PAMP*, and *inflammatory signals*, must be retrieved from the dataset. Currently, the extraction of these signals is dependent on expert knowledge since there is no universal process suitable to every application scenario. Each signal describes the degree of hazard in the problem domain (as a non-negative real number or integer). So, input signals may be:

- *PAMP* signals indicate the presence of hazard with high certainty.

- With intermediate confidence, *safe* signals imply the absence of danger.
- *Danger* signals indicate the presence of danger with a lower degree of certainty compared with *PAMP*.
- *Inflammatory cytokines* magnify all prior signals, amplifying their consequences.

Algorithm 4: Dendritic Cell Algorithm (DCA), adapted from Greensmith [26].

Input: *Ag* input data (*antigens*)
 N DC population size
 c number of times any *antigen* will be presented to distinct DCs
 $t_{migration}$ migration threshold after which DC matures
 t_{mcav} MCAV threshold above which the *antigen* is considered anomalous
Output: Classification of *antigens* as normal or anomalous

```

foreach ag in Ag do
    signals = compute_signals(ag)
     $DC_{sample}$  = sample_random( $c$ )
    foreach dc in  $DC_{sample}$  do
        /* update csm and k of each cell and store the antigen in
           their repertoire */
        expose(dc, ag, signals)
        if dc.csm >  $t_{migration}$  then
            if dc.k > 0 then
                /* this will increment the count of mDC for all antigens
                   that this cell had */
                mature(dc, MATURE)
            else
                mature(dc, SEMIMATURE)
            end
            end
            reset(dc)
        end
    end
end

foreach ag in Ag do
     $mcav = \frac{mDC\_count(ag)}{c}$ 
    if  $mcav > t_{mcav}$  then
        classify(ag, OUTLIER)
    else
        classify(ag, NORMAL)
    end
end
end

```

As previously stated, the modeling of these signals is inextricably linked to the problem. In intrusion detection, for example, the *PAMP* signal can be connected with an increase in incoming connections [27] and error messages [26], and *danger* signals can be packet counts, message size, or a fusion of various features [28]. Based on the data points, the way the signals are calculated depends significantly on the considered problem. In addition to signals, there is also the concept of *antigen*, which are unique identifiers of data points, which will eventually be the target for the detection. As a result, the algorithm treats each data point as an *antigen*–signal pair, since *antigens* are what we wish to categorize, while input signals are what we utilize to do so.

These data points are collected throughout the period by the DCA, resulting in classifications that consider the context of current and future data. If surrounded by *safe*-signaling data points, a single data point with a significant *danger* signal will likely be regarded as safe. The categorization process is carried out by spreading data points among *iDCs*. Each

iDC will store *antigens* (i.e., the identifier) and use each *antigen* signal to update its own state (output signals) in two ways:

- It will update its context value, k , indicating danger (positive values) or safety (negative values) sampling contexts. k increases with *PAMP* and *danger* signals, while decreasing with *safe* signals.
- It will update its costimulation value, csm , which indicates how many signals it has sampled, rising with *PAMP* *danger* and *safe* signals.

Inflammatory cytokines boost the strength of all three input signals, effectively boosting the pace at which k and csm change. Note that there is no need to use all the input signals mentioned in some application scenarios. Equation (1) is a matrix adaptation of Green-smith's method for calculating k and csm signals from *safe*, *danger*, *PAMP*, and inflammatory signals, whereas Equation (2) is similar, but expressed in an iterative form.

$$\begin{bmatrix} csm \\ k \end{bmatrix} = \begin{bmatrix} w_{P,csm} & w_{D,csm} & w_{S,csm} \\ w_{P,k} & w_{D,k} & w_{S,k} \end{bmatrix} \begin{bmatrix} \sum_i P_i * (1 + I_i) \\ \sum_i D_i * (1 + I_i) \\ \sum_i S_i * (1 + I_i) \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} csm_i \\ k_i \end{bmatrix} = \begin{bmatrix} csm_{i-1} \\ k_{i-1} \end{bmatrix} + \begin{bmatrix} w_{P,csm} & w_{D,csm} & w_{S,csm} \\ w_{P,k} & w_{D,k} & w_{S,k} \end{bmatrix} \begin{bmatrix} P_i \\ D_i \\ S_i \end{bmatrix} * (1 + I_i) \quad (2)$$

When the *iDC*'s csm exceeds a certain threshold, the *iDC* develops into either an *mDC* or a *smDC*, depending on whether k is greater than 0 or not, and starts the migration process. The migration process is the last stage of the algorithm and consists of the actual *antigen* classification based on the *Mature Context Antigen Value* (MCAV) calculation. The MCAV value is calculated using Equation (3), where $\#mDC_{Ag_i}$ represents the number of times a particular *antigen*, Ag_i , was discovered in an *mDC* and $\#DC_{Ag_i}$ represents the number of times a given *antigen*, Ag_i , was collected by any *DC* that migrated. According to the MCAV, an *antigen* can be classed as normal or abnormal. If the MCAV value exceeds a predetermined threshold, the *antigen* is regarded as abnormal; otherwise, it is considered normal.

$$MCAV(Ag_i) = \frac{\#mDC_{Ag_i}}{\#DC_{Ag_i}} \quad (3)$$

3. Anomaly Detection

Anomaly detection refers to algorithms that analyze data to identify anomalies or outliers. An anomaly may appear gradually in manufacturing processes, such as tool/wheel wear. It might be sudden, such as tool breakage, or it could be avoidable, such as excessive vibration/chatter. So, process anomaly detection is critical since knowledge of tool wear is required for scheduling tool changes, detection of tool breakage is required for salvaging the workpiece and the machine, and recognizing chatter is required for initiating remedial action [29].

An anomaly detection algorithm can be trained using previous information or typical operating circumstances. This is why process monitoring is so important, which refers to the capture, manipulation, and analysis of sensor readings to identify the condition of a process. In either instance, a profile of what is to be regarded as usual is constructed. The algorithm then accepts input data and determines whether it is consistent with its normal behavior or an outlier. A process set of variables (e.g., force, speed, motion, temperature, pressure, power, acoustic emission, feed motor current, and so on) is measured, processed online, and compared to their predicted values for this purpose. Any variations from predicted values are blamed on process anomalies [10].

A process anomaly is a deviation from normal process behavior linked to various sources. It is defined as an unallowable divergence of at least one system characteristic

property (feature) from the accepted or normal operating state. Facing an anomaly, if no recovery decisions are made, it may result in malfunctions or failures, which are long-term disruptions in a system's capacity or performance to fulfill a necessary function under specified operating parameters.

Anomalies can express themselves in three ways, according to Chandola et al. [30]:

- Point anomaly: this is the most basic situation in which a single data instance is abnormal compared to the rest—for example, if a data instance has a greater value than those of all other data instances.
- Contextual anomaly: a data instance that is out of place in its context. The definition of context varies depending on the situation. A recorded average yearly temperature of 9 °C in the Amazon rainforest, for example, can be considered as a contextual anomaly because it is abnormal in the context in which it is recorded (here, the context is longitude and latitude). However, such an average yearly temperature would not be anomalous in all contexts (e.g., Europe).
- Collective anomaly: this occurs when a connected data group is deviant compared to the total dataset. For example, suppose a periodic behavior that repeats for every constant time interval abruptly changes for a single period. In this case, that period alone is not anomalous, nor is any point in the context deviating from the sequence. However, the entire period behaves inconsistently when compared to surrounding data points, despite possibly not having any significantly high values or a high rate of change.

As previously stated, anomaly detection seeks to find data items that are out of the ordinary compared to “normal” ones. The first issue is defining what is normal and abnormal in a way that is objective and calculable. This definition changes according to the technique adopted by a particular algorithm. There are three primary techniques to establishing what defines an anomaly: (i) distance based: outliers are defined by their distance from other data points; (ii) density based: outliers are data points that are found in lower density locations; (iii) rank based: outliers are data points whose nearest neighbors have other data points as nearest neighbors.

Furthermore, anomaly detection algorithms can be further classified as hard computing based and soft computing based. Hard computing includes probabilistic learning, knowledge, and ML-based techniques, while soft computing includes branches such as fuzzy logic, genetic algorithms, ant colony optimization, and artificial immune systems. Finally, there are combination learners, which include hybrid and ensemble-based algorithms, as represented in Figure 6 [31].

Data have a known distribution and parameters in parametric anomaly detection (e.g., normal distribution characterized by mean and standard deviation). On the other hand, prior knowledge of data may be known in non-parametric techniques, but their distribution is unknown, so we must learn it. This can be accomplished using: (i) a supervised algorithm, in which all data points used have a known classification through the use of labels (e.g., “anomalous” and “normal”); (ii) an unsupervised algorithm, in which data points have no labeling and data patterns and relationships are learned; (iii) a semi-supervised algorithm, by employing a small amount of labeled data and applying that knowledge to attempt learning on additional, unlabeled data, where the latter constitutes the majority of all data, which can be accomplished by applying what was learned in a supervised setting, for example, to label the unlabeled data.

Finally, the application environment in which anomaly detection happens can be online or offline. Offline anomaly detection refers to the examination of comprehensive and collected data at any point in the past, with loose limits on the methods employed since they can be more complicated and do not need to provide rapid reaction times. On the other hand, online anomaly detection is either real-time or near real time and is frequently coupled with data streaming, resulting in a continuous process of data gathering and anomaly detection.

Figure 6. Types of network anomaly detection methods.

3.1. Anomaly Detection in CPPS

As previously said, the I4.0 paradigm emergence has some difficulties. Using anomaly detection in this context entails identifying essential components of these issues and the standard anomaly detection in time series problems. These issues vary depending on the unique context of industrial application, but a good anomaly detection technique must include the following features:

- **Generalization:** It is impossible to forecast the kinds of anomalies that will arise and how they will emerge in a time series. For example, if a time series is highly periodic, techniques that account for seasonal trends will do very well in identifying deviations. Nevertheless, the same strategy would perform poorly in unstable time series. A decent anomaly detection technique must be consistently successful in an agnostic way to time series typical behavior and presume that the intended behavior is dynamic.
- **Time performance:** In industrial applications, anomaly detection must be discovered within usable time, since late detection can result in massive losses and jeopardize safety.
- **Resource efficiency:** According to the I4.0 concept, CPPS components are anticipated to be self-monitoring. As a result, components must execute anomaly detection decentralized, which raises issues about resource utilization. These components are frequently resource constrained, both in themselves and in their use of network infrastructures (i.e., communication itself must be efficient).
- **Little need for labeled data:** There is a scarcity of publicly accessible datasets for industrial settings under assault. Many techniques to appropriate learning need massive volumes of data, which are not feasible in this scenario. Furthermore, depending on the labeled data, techniques may eventually have difficulties adapting effectively if

the system since the model would overfit the data from which it has learned, which would be contrary to the feature indicated in the first bullet point.

- Robustness and safety: Components must be able to adapt to unfavorable situations by continuing regular operation without jeopardizing the equipment's or human life's safety.

These difficulties are not trivial, and to the best of our knowledge, no solution seeks to handle all of them simultaneously. Instead, they focus on a subset of them most of the time. Some of these challenges motivate the design and development of AIS applied to anomaly detection.

3.2. AIS for Anomaly Detection

According to Tokarev et al. [32], AIS is one of the most promising current solutions to the problem of anomaly detection. These approaches offer a great degree of generality, are usually computationally efficient, and consume fewer resources. Costa Silva et al. [33] defend that the use of immune-inspired approaches for anomaly detection has been adopted in the literature because of its analogy with body resistance in the human immune system provided against agents which causes diseases. According to them, the applicability of AIS in anomaly detection consists mainly of the self–nonself discrimination and the danger models, which refer to the NSA [34] and DCA [35,36].

On the one hand, the NSA is a straightforward and intuitive algorithm for anomaly detection that analyzes the feature space and generates detectors in the nonself region. Because the self data are utilized as a reference, the detectors are positioned outside the self zone. This means that the overall process is similar to supervised and semi-supervised machine learning approaches. NSA has significant problems concerning the system context. Additionally, it scales poorly with data size, as detectors frequently overlap and leave gaps between them, failing to identify several anomalies. Finally, the process to measure the similarity between detectors and data can be pretty expensive.

On the other hand, the DCA is the most notorious danger model-based algorithm, and it relies significantly on the correlation mechanism between the *antigen* (what we want to classify, i.e., a system process) and input signals (behavior of the system). Compared with NSA, the DCA is super lightweight and eliminates the need to define normal (self data). However, it requires prior knowledge or modeling of the problem. Otherwise, input signals and *antigens* are not modeled correctly. Consequently, the pre-processing data stage is critical. In some contexts, to calculate the *safe* and *danger* signals, it is necessary to distinguish between self and nonself data. So, despite resembling an unsupervised machine learning approach, there is a strong dependency on the existing expert knowledge about the context of the application. Moreover, in the classification stage, the algorithm requires the entire *DC* population for cell maturation and overall *antigen* classification, gather then having only a particular data point to mature before categorizing said data point. This makes the algorithm unsuitable for online anomaly detection as it is.

In their work, Costa Silva et al. [33] compared both models using the *UCI Breast Cancer Database*, and reached the conclusion that there is no better immune-inspired model than another in terms of anomaly detection. Depending on the context of application and the problem definition, there are cases in which the DCA may overcome the NSA and vice versa. So, considering the features about anomaly detection in CPPS mentioned before, in theory, DCA is more suitable compared with NSA, since it is more lightweight, and in consequence, resource efficient, and was less dependent on labeled data. However, the limitations regarding the overall algorithm's data structures and procedural operations understanding, online application, and initial problem modeling leave room for improvement if this algorithm is intended to be used for anomaly detection in industrial scenarios.

Regarding the overall algorithm's understanding, Gu et al. [37] proposed a formal mathematical definition of the deterministic Dendritic Cell Algorithm (dDCA). This work helped the research community understand the algorithm and avoid ambiguities since previous investigations focused mainly on empirical aspects. Later, Greensmith and

Gale [38] proposed a specification of the dDCA using functional programming (in Haskell programming language), which on the contrary to the previous work, is targeted to a computer science audience.

Considering the limitation of applying DCA in online classification problems, such as anomaly/intrusion detection, Gu et al. [39] proposed a real-time or near real-time analysis component by applying segmentation to the current output of the DCA into slices. The authors tested the proposed approach in the *SYN scan* dataset, a large real-world dataset based on a medium-scale port-scan of a university computer network. The results show that segmentation applies to the DCA for real-time analysis since, in some cases, segmentation produces better results when compared with the standard algorithm. Later, Yuan and Chen [40] also proposed a real-time analysis to the DCA, where an *antigen* presented by sufficient dendritic cells is immediately assessed. The authors validated the proposed approach in the *UCI Wisconsin Breast Cancer* dataset. The results are promising, showing that the algorithm achieves considerable accuracy.

More recently, Pinto et al. proposed the incremental Dendritic Cell Algorithm (iDCA) [41] as a method to detect network intrusions in a real-time industrial Machine-to-Machine (M2M) scenario. For validation of the iDCA, two network intrusion detection datasets were used in an online classification scenario, namely, the *KDD Cup99* and the *M2M using OPC UA* datasets. The results show that the approach is a viable solution to detect anomalies in (near) real-time, especially in environments with little a priori system knowledge for intrusion detection.

As for problem modeling in the DCA, as mentioned before, the DCA avoids a model training step, but it requires domain or expert knowledge for data pre-processing. Gu et al. [42] attempted to use the Principal Component Analysis (PCA) technique to categorize input data in DCA automatically. This way, it is possible to avoid manually over-fit the data to the algorithm every time a new problem context is considered, which is undesirable. Experimental results were performed in the *Stress Recognition in Automobile Drivers* dataset and have shown that the application of PCA to the DCA for automated data pre-processing is successful. Later, Chelly and Elouedi [43] reviewed the pre-processing data phase of the DCA while making a comparative study of data reduction techniques within the DCA, based mainly on Rough Set Theory (RST) usage in the pre-processing phase. The authors used multiple real-valued and binary-classification datasets. The results show that different DCA versions outperform known Machine Learning classifiers in the literature, namely Support Vector Machine (SVM), Artificial Neural Network (ANN), and Decision Tree (DTree), in terms of the overall classification accuracy.

4. Cursory Dendritic Cell Algorithm

The development of the CDCA aims at solving or reducing the impact of the main limitations of DCA identified earlier. Thus, in this work, three different contributions are proposed. Firstly, we propose an object-oriented specification (using Python) to abstract the algorithm into its main mechanisms and decompose it into a more modular structure, such that specific components of the algorithm can be updated more independently. Secondly, we propose a near-real-time analysis component, which allows the algorithm to present intermediate classification results to data online, in contrast with its application to entire sets of data. Thirdly, we enable data-driven approaches to extract signals in the pre-processing stage of the DCA. The problem modeling process will not require as much expert knowledge as currently.

4.1. Modular Dendritic Cell Algorithm

DCA specification and implementation, using an object-oriented approach, was motivated by the need to adjust and update the algorithm's source code. With an object-oriented implementation, experimentation and changes to the original algorithm become easier to develop and test. This modular approach is illustrated in Figure 7.

Figure 7. Modular approach for DCA. Each modules consists on a step in the data-flow of the algorithm.

The decomposition of the DCA presented was carried out by identifying the main steps of the algorithm and abstracting those steps into separate classes (modules). The rationale for each class is according to the following steps: (i) the *antigens* and values are received by the system; (ii) their signals are extracted; (iii) the extracted signals are distributed into the DC population; (iv) each DC decides how it processes these *antigens* and signals; and (v) the *antigens* are somehow impacted by this processing. Considering this rationale, the following classes are defined:

- **Interface:** to keep the usage of the algorithm simple, a single interface is used to interact with the system as a whole. Therefore, the interface can receive *antigens*, and output *antigen* classification results. Indeed, the *Interface*, as implemented, orchestrates the remaining modules.
- **Signal Extractor:** defines how signals are extracted from data. It receives *antigens* and associated data, while outputs *safe*, *danger*, *PAMP*, and inflammation signals from received data and the *antigen* they are associated with.
- **Sampler:** decides on a strategy for signals and *antigen* distribution in the DC population, i.e., which DCs are attributed to which *antigens* and signals. It receives the *antigens* and signals from the *Signal Extractor* and chooses specific DCs to receive them.
- **DC Population:** the DCs themselves can receive *antigens* and signals. Received *antigens* are stored locally by each DC. Each DC also has access to the *antigen* repertoire (see next module), allowing the collected *antigen's* state to be updated by the DC that collects it.
- **Antigen Repertoire:** a database structure holding all *Antigens* that are given to the system. It might be relevant to store in the *Antigen Repertoire* changes in the *antigens'* state, for instance, when a DC migrates, or store information about which mature/semi-mature DC is associated with a given *antigen* collected. The *Interface* can query the *Antigen Repertoire* to achieve classification.

Each component can be updated separately with this modular approach, and different strategies in different modules can be implemented.

4.2. Online Dendritic Cell Algorithm

As previously mentioned, DCA does not support an online anomaly detection capability. The DCA can internally process input data in real-time, but it requires an analysis stage,

which is performed offline. To reduce the impact of this limitation, the DCA is updated to allow intermediate classification of data and thus, enable near real-time anomaly detection.

The classification of *antigens* is accomplished at the end of the algorithm's data-flow, by calculating the MCAV value, which consists of the ratio of matured DCs that collect a given *antigen*, to the total number of times an *antigen* is collected. Thus, the classification method was updated in two ways: (i) each time a DC matures, all *antigens* it has collected have their MCAV values derived; (ii) each *antigen* has three possible states: inconclusive, safe, or dangerous. The finite state diagram of the *antigens* is depicted in Figure 8, alongside the DC states and their interaction.

Figure 8. Antigen and dendritic cell state changes. As dendritic cells mature, they increment dendritic cell count in collected antigens, and update their Multi Context Antigen Value (MCAV).

All *antigens* begin as inconclusive. If their MCAV value increases beyond the predetermined threshold, the *antigen* is immediately classified as dangerous, i.e., anomalous. If all DCs that collected the *antigen* have matured, but the MCAV value has not yet increased beyond the predetermined threshold, the *antigen* is classified as safe. The introduction of an “inconclusive” state leads to a lack of immediate classification. Classification will only occur after a sufficient number of DCs have migrated. On the other hand, classification of “dangerous” *antigens* is achieved faster, as this classification is attributed to the *antigen* as soon as the MCAV increases above the threshold, rather than waiting for the entire data set to be processed. The classification of “safe” *antigens* is also faster than waiting for the entire dataset but is slower than the classification of “dangerous” *antigens*.

4.3. Data-Driven Dendritic Cell Algorithm

Regarding the difficulty of problem modeling, considering the dependency of previous expert knowledge about the problem and application context, the pre-processing stage of the DCA was updated to become more data-driven. This difficulty pertains mainly to the signal extraction procedure, as the DCA does not define how signals are extracted. Thus,

a data-driven approach to signal extraction is suitable for solving this limitation. For this, it was considered K-Means, as a scalable and inductive clustering algorithm, since it checks essential requirements, given the context of CPPS, such as:

- Suitable for multi-dimensional data processing;
- Able to process new, unseen, data;
- Usable in an incremental, per-point basis;
- Provides a real-valued score suitable for usage as signals;
- Scalable with data dimension and size;
- Provides real-time processing, such that its usage for online anomaly detection is not prohibitive.

K-Means is an algorithm that finds centroids for each cluster of data such that the variance in each cluster is minimized. The number of clusters is user defined. Therefore, each cluster can be represented in a single point which constitutes the centroid, and future points can be included in clusters according to their distance to each cluster's centroid.

In order to learn what is normal, K-Means first computes the centroids of normal data only. Once this process is completed, the centroids of data are stored, and the maximum distance for each centroid to the farthest data point that belongs to its cluster is also stored. These distances will be used as radii, forming a hypersphere per cluster. Future data points to be classified are compared to these hyperspheres, measuring their distance to the hypersphere's surface. If the data point is inside the hypersphere, the distance to its surface constitutes a *safe* signal. Otherwise, its distance constitutes a *danger* signal. Furthermore, if new normal behavior occurs, more hyperspheres can be produced. Figure 9 illustrates an example of a heat map obtained through K-Means with different numbers of clusters.

Figure 9. Heat map of K-Means for different numbers of clusters on the same data.

4.4. Proposal Overview

Both the online adaptation described in Section 4.2 and the data-driven adaptation described in Section 4.3 were implemented upon the modular structure described in Section 4.1. The online adaptation was implemented in the *DC Population* and in the *Antigen Repertoire* modules. In the former, *DCs* directly notify *antigens* of their collection and maturation, and in the latter, *MCAV* values are updated incrementally. Regarding the

data-driven adaptation, the *Signal Extractor* module is addressed, such that the K-Means approach is used for the extraction of *danger* and *safe* signals from data.

Considering the contributions described previously, the overall CDCA works as such:

1. Training phase:
 - (i) The *Interface* receives a predefined number of normal data points, one by one, which will be used as a baseline for normal behavior.
 - (ii) The *Signal Extractor* receives these data points, one by one, and applies a standard K-Means approach for clustering the data points. Note that, at this stage, a classification is not possible since there is not yet a normal baseline.
 - (iii) Eventually, when sufficient training points are received, the *Signal Extractor* computes each cluster's maximum distance from its centroid to each point considered in the cluster, which will serve as radii for each cluster's hypersphere. The centroids and radii are stored, but the training data points are not.
2. Anomaly Detection phase:
 - (i) The *Interface* receives each data point, i.e., an *antigen*, one at a time.
 - (ii) The *Signal Extractor* receives each point, and outputs its *danger* and *safe* signals, depending on whether they are inside a hypersphere or not, and how far they are from the closest hypersphere surface. *PAMP* and *Inflammation* signals are not used in CDCA.
 - (iii) Upon receiving the *antigen* and resulting signals, the *Sampler* samples *DCs* in a sliding window approach (n -contiguous cells).
 - (iv) Each *DC* receives the signals and computes their costimulatory value (used for migration) and k value (used to decide whether they migrate as mature or semi-mature). Upon receiving an *antigen*, each *DC* notifies the respective *antigen* in the *Antigen Repertoire* of its collection, incrementing its record of the number of times collected.
 - (v) Over time, *DCs* mature, and notify their *antigens* of the times they matured as mature *DC*. *DCs* that mature as semi-mature also notify *antigens*.
 - (vi) When *antigens* in the *Antigen Repertoire* achieve high *MCAVs*, they notify the *Interface* of their state change, from "inconclusive" to "dangerous", i.e., an anomaly has been detected. When *antigens* in the *Antigen Repertoire* have the sum of *mDC* and *smDC* to be equal to the times they have been collected, their state changes from "inconclusive" to "safe", i.e., the data point is considered normal. *Antigens* that have notified the *Interface* are removed from the *Antigen Repertoire*.

5. Validation and Results

For performance validation of the CDCA, the authors used two anomaly detection datasets, namely *SKAB* and *M2M using OPC UA* datasets. Both datasets were created considering an industrial context for physical equipment and network anomaly detection. For each one, a characterizing of the dataset is provided, along with the deployment process of the CDCA in the dataset and the main results achieved. All tests were executed using Python, on an Intel(R) Core(TM) i7-6700HQ CPU with four physical cores and eight logical cores at 2.60 GHz, on a 16 GB RAM system, using a Windows 10 operating system.

5.1. Skoltech Anomaly Benchmark Dataset

With the goal of comparing CDCA with other anomaly detection algorithms, the dataset (*SKAB*) [44] was used. The development of *SKAB* is based on sensor readings deployed in a water circulation system from an IoT testbed, located in the Skolkovo Institute of Science and Technology (Skoltech). The testbed used to build the dataset is shown in Figure 10.

Figure 10. Skoltech Anomaly Benchmark (SKAB) testbed. Figure obtained from [44]. 1,2—solenoid valve; 3—a tank with water; 4—a water pump; 5—emergency stop button; 6—electric motor; 7—inverter; 8—compactRIO; 9—a mechanical level for shaft misalignment.

Anomalies are introduced by inserting different physical faults in the system, such as closing valves, increases in temperature, and rotor imbalances. The testbed is composed of IoT sensors that communicate using the OPC Unified Architecture (OPC UA) protocol and whose data is stored in a database (from which the dataset is built). The dataset comprises 34 different files, each representing a specific anomaly. Each data point is real valued and consists of a sensor reading, such as current, pressure, temperature, and volume flow rates of the circulating fluid. All anomalies are collective, meaning the anomalous points are contiguous.

There are ten original features in the dataset. However, since we are treating this problem as a time series, the authors derived other features, such as moving average, derivative and absolute derivative, the difference from moving average and squared distance from moving average, in a total of 26 features.

SKAB Results

The authors of *SKAB* also made available a scoreboard, i.e., a public score table that includes results of different techniques used for anomaly detection, and it uses as scoring metrics: F_1 score, False Alarm Rate (FAR), and Missed Alarm Rate (MAR). Thus, by using the metrics that are suggested in *SKAB*, a comparison can be made between CDCA and other anomaly detection algorithms. Table 1 shows a comparison, using the metrics mentioned, between CDCA (with different parameter combinations) and existing algorithms validated in *SKAB*.

The nomenclature $CDCA_N_S_k$ is used, which corresponds to CDCA using N cells, from which S is sampled for each data point and k clusters for K-Means. Furthermore, “_original” denotes that no extra features (moving average and moving average of derivative) were added. Only the original features are used.

CDCA shows competitive results with most algorithms currently in the *SKAB* table. Its F_1 score is 0.07 below the highest contender—a convolutional autoencoder—and 0.01 below the second best. The F_1 score for CDCA is reasonably similar for multiple parameter variations, never changing more than 0.01. Using K-Means alone, without CDCA, leads to a decrease in F_1 score of 0.11. Using no additional features results in a decrease in the F_1 score of 0.01, concerning CDCA with the moving average and moving average of derivative. Regarding FAR, CDCA is consistently worse (higher value) than all other current implementations, with MAR values being consistently better (lower value). As CDCA has

the limitation that classification is, in the majority of the cases, not immediately possible, the cycle delays and timing delays are also presented in Table 2. Cycle delays correspond to how many additional data points were needed to be received before classification could be achieved.

Table 1. Results for all existing algorithms in *SKAB* and the Cursory Dendritic Cell Algorithm (DCDA) with different parameter's range, as well as the adapted version of K-Means. Results for CDCA and K-Means are an average of the results of 10 repeated test instances.

Algorithm	F1	FAR [%]	MAR [%]
Perfect detector	1	0	0
Conv-AE	0.79	13.69	17.77
MSET	0.73	20.82	20.08
CDCA_100_10_20	0.72	37.95	5.71
CDCA_100_5_20	0.72	38.27	5.75
CDCA_50_10_20	0.72	38.33	5.80
CDCA_20_10_20	0.72	37.74	6.41
CDCA_100_1_20	0.72	38.76	5.57
CDCA_100_10_5	0.72	29.18	15.24
CDCA_100_10_20_original	0.71	28.27	15.61
LSTM-AE	0.68	14.24	35.56
T-squared+Q (PCA)	0.67	13.95	36.32
MSCRED	0.64	13.56	41.16
LSTM	0.64	15.40	39.93
K-Means w/ <i>danger</i> signals	0.61	25.63	35.76
LSTM-VAE	0.56	9.13	55.03
T-squared	0.56	12.14	52.56
Autoencoder	0.45	7.56	66.57
Isolation forest	0.40	6.86	72.09
Null detector	0	0	100

Table 2. Performance results for CDCA with multiple parameters. These results pertain only to outliers and are an average of the results of 10 repeated test instances.

Test Instance	Classification Cycle Delay		Classification Delay [ms]	
	Mean	Standard Deviation	Mean	Standard Deviation
CDCA_100_1_20	7.684	10.290	1.537	2.097
CDCA_20_10_20	7.758	8.797	2.124	3.242
CDCA_100_10_20	7.818	8.654	2.112	2.288
CDCA_100_10_5	10.081	10.889	2.662	2.854
CDCA_100_10_20_original	14.055	15.185	3.801	4.450

5.2. M2M Using OPC UA Dataset

The *M2M using OPC UA* dataset [45] was developed by simulating known network attacks, and was designed to assess intrusion detection solutions in industrial scenarios [28].

The dataset was obtained from a lab CPPS testbed (see Figure 11), where multiple nodes implement the OPC UA protocol for communication.

Figure 11. Illustration of the testbed used for the generation of the *M2M using OPC UA* dataset. Figure obtained from [28].

All traffic is recorded and saved, and significant features, such as message size and source and destination addresses are extracted to form the dataset. Different attacks, such as Denial of Service (DoS), Message Spoofing, and Man-in-the-middle (MITM), are injected in the same time series and spaced with normal system operation moments. The dataset contains a single continuous time series of 107,633 records, each with 29 network-related features. The real-valued features were standardized such that K-Means hyperspheres have the same significance in each dimension. Additionally, because our implementation is not compatible with categorical data, all categorical data was ignored, as well as real-valued data and integers that do not have meaningful information from Euclidean distance (e.g., destination or source ports).

M2M Using OPC UA Results

This dataset was used previously to validate the iDCA [41], which also enables online classification. So, using this dataset with CDCA makes it possible to compare both proposals in terms of classification performance. Additionally, we took this opportunity to assess how the CDCA behaves with no expert knowledge compared to the iDCA, which implied a considerable problem and feature analysis. In this case, the authors used the Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) as an evaluation metric for classification performance.

We tested the CDCA using a population size of 100, with ten copies of each data point and using 20 clusters for K-Means. The obtained AUC is 0.98 compared with the value of 0.97 obtained by Pinto et al. [41] using the iDCA. In order to define what is normal behavior, we defined a training window as the first 8000 data points (the first attack happens in the 8012th data point).

6. Discussion

Firstly, the CDCA modular approach proved helpful for experimentation and development, allowing the effortless replacement of modules. Compared with related work focusing on the formal specification of the algorithm using Haskell (Functional Programming), an object-oriented approach enables problem abstraction into classes, and data can be easily stored in objects. This approach provides an easy way to add new data and functions. Additionally, by using Python as the coding language, we have the most recent data science and machine learning packages available, which can be very useful to extend the algorithm to a hybrid or ensemble approach, or replace specific algorithm components' functionality with statistical and machine learning techniques. However,

an object-oriented approach does not easily describe the algorithm's processes and data flow, since it focuses on their implementation. Additionally, the object-oriented approach can be used as a formal specification of the algorithm since there is no definition of actual mathematical functions.

Secondly, using a clustering approach compared to PCA or RST may present advantages regarding the problem modeling limitation. We know that the bulk of the effort in the pre-processing stage is in deciding how to calculate the input signals (*danger* and *safe*). With clustering, the main advantage is creating a normal baseline, which is used to derive the input signals when new unseen data are processed. The creation of data clusters achieves this baseline. Since PCA and RST focus on an automatic feature selection process, no expert knowledge is required. However, they cannot decide how to map input data to *safe* and *danger* signals and, consequently, define the normal baseline. Another advantage of clustering in the pre-processing stage is the possibility of continuous adaptation of the normal baseline. However, the K-Means technique used in this work does not contemplate this functionality.

Thirdly, considering the online application of the algorithm, CDCA presents a strategy of enabling the classification of *antigens* as soon as the *DC* migrates. This strategy is similar to previous work regarding segmentation, but with the difference of not defining a fixed segmentation window (based on time or number of *antigens*). Dynamic segmentation is achieved, which can deal with online dynamics during real-time analysis. However, the proposed approach does not provide real-time analysis since the classification has time delays. Additionally, compared with the iDCA, the proposed approach is not ready to be applied in a streaming problem, since an *antigen* classification requires processing a mini-batch of *antigens* instead of classifying data points as soon as they become available.

Fourthly, based on the achieved results after testing the CDCA in industrial context scenarios (based on datasets) and comparing the performance with existing classifiers, the proposed approach is competitive with other anomaly detection approaches. In the *SKAB* dataset, CDCA outperforms state-of-the-art approaches, such as deep neural networks and regression approaches. This performance is mainly assessed by the F_1 score, false alarm, and missed alarm rates. The algorithm seems to be very good in identifying anomalies, given its consistently low MAR values. However, on the other hand, it also consistently misclassified normal occurrences surrounding the anomalous region as anomalous, as indicated by its high FAR value.

Furthermore, the CDCA is tested in terms of parameter variation, where the role of *DC* population and sample sizes is residual, i.e., the variables do not significantly influence the algorithm's performance. However, there is a slight impact on performance (indicated by lower MAR values at the cost of higher FAR values), with higher sample sizes and a slight increase in performance (lower FAR) with an increasing population size. On the other hand, a small number of clusters leads to worse MAR values, which can be justified because a smaller number of clusters implies that each cluster is bigger in radius. This leads to more data being considered normal, as it is easier for new data to lie inside the learned clusters. Additionally, the added features allowed for higher dimensional centroids, and more information to be used, which resulted in a slightly better F_1 score (a 0.01 increase) with a decrease in MAR at the expense of an increase in FAR.

The timing results for CDCA were also analyzed in *SKAB*, which leads us to the conclusion that an online application is possible, but the real-time classification is not. Despite presenting a fast classification, it is not suitable for a streaming problem. CDCA cannot immediately classify data points, with each point requiring on average 7–14 more data points to be received in the classification of any given point. Despite this limitation, the millisecond delay is promising, in that, in the worst-case scenario, the algorithm requires, on average, 3.8 ms to classify a point for an average of 14 points delay. This timing is obtained by considering only processing time, i.e., discarding the time between processing in one data point and receiving the next.

For the *M2M using OPC UA* dataset, the results are promising, given that the obtained AUC of ROC is slightly better when compared to the iDCA [41]. Furthermore, CDCA required no expert knowledge when compared with the iDCA. However, as mentioned before, CDCA requires an initial training phase to define what is a standard system behavior, whereas iDCA does not. This is a limitation since, during training, the algorithm does not perform any detection. Additionally, it may be challenging to have only normal data available for training purposes in a real case scenario.

7. Conclusions

A new immune-based algorithm for anomaly detection was developed, called CDCA, which expands the existing DCA by enabling the original algorithm to be applied in real time as an online anomaly detection tool. CDCA uses the same principles of the original DCA version but iteratively updates the classification metric and keeps track of said metric for individual data points. This leads to near real-time classification. This development was implemented on top of the algorithm's proposed modular approach, which allows the replacing of components, namely, how data are collected, how they are pre-processed, how values of abnormality are extracted, how these values are mapped to the algorithm's inherent population, and how the population behaves. Consequentially, this modular approach allows faster development and experimentation.

Additionally, an adaptation of K-Means, an unsupervised clustering algorithm, was implemented to allow us to ease the implementation process of the DCA, namely by making it substantially easier to model any given real-valued problem. The hypothesis is that by using K-Means to extract abnormality values, both algorithms can help each other. DCA can be more flexible and adapt to different problem contexts, while K-Means' classification is more robust for detecting collective anomalies, which are closely located in time. In the end, it is possible to define a normal baseline behavior of the system, which is highly desired in anomaly detection applications.

CDCA was validated on two different datasets representative of industrial IoT scenarios, namely *SKAB* and *M2M using OPC UA*. The results revealed some solid advantages and a few shortcomings in the implemented algorithm. As an advantage, the usage of signals from K-Means effectively increases the algorithm's performance by leveraging anomalies as phenomena that tend to happen with the proximity in time, corroborating the hypothesis that the algorithms benefit each other. The current implementation also outperforms a significant amount of existing techniques found in the *SKAB* scoreboard and is competitive with the iDCA implementation in the *M2M using OPC UA* dataset. However, shortcomings of the algorithm are also identified. First, it suffers from a high rate of false positives. Secondly, there is a classification delay inherent to the voting mechanism of the DC population. Finally, single-point anomalies are harder to detect as the algorithm leverages time locality, which is suitable for collective anomalies.

The main drawback of the algorithm is its delay in classification, which needs to be further refined to offer immediate classification. The association with other scalable techniques for generating signals, other than K-Means, is also a good point to explore. Moreover, the algorithm was not built with time optimization in mind and, thus, optimizing the algorithm for performance is also a promising avenue for future work. Furthermore, the application of CDCA in a real industrial test case is critical before being validated as a real-time implementation. Finally, validation of the CDCA in a much larger set of datasets outside the industrial scope is needed. This way, we will assess the feasibility of using the approach in different application contexts and its sensibility when generalizing to different anomaly detection domains.

Author Contributions: The contributions to this work are the following: Conceptualization, C.P. and R.P.; methodology, C.P.; software, C.P.; validation, C.P. and R.P.; formal analysis, C.P.; investigation, C.P.; resources, R.P. and G.G.; data curation, C.P.; writing—original draft preparation, C.P. and R.P.; writing—review and editing, R.P. and C.P.; visualization, C.P.; supervision, R.P. and G.G.; project

administration, R.P. and G.G.; funding acquisition, G.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Two publicly available datasets were analyzed in this study, namely the *Skoltech Anomaly Benchmark (SKAB)* and the *M2M using OPC UA*. *SKAB* is openly available in kaggle at 10.34740/KAGGLE/DSV/1693952 and can be found here: <https://www.kaggle.com/yuriykatser/skoltech-anomaly-benchmark-skab> accessed on 21 November 2021. *M2M using OPC UA* is openly available in IEEE Dataport at 10.21227/ychv-6c68 and can be found here: <https://ieee-dataport.org/open-access/m2m-using-opc-ua> accessed on 21 November 2021.

Acknowledgments: This work was financially supported by: INDTECH 4.0 (SP4)—POCI-01-0247-FEDER-026653, co-funded by European Regional Development Fund (FEDER), through Competitiveness and Internationalization Operational Program (POCI) and Base Funding—UIDB/00147/2020 of the Systems and Technologies Center—SYSTEC—funded by national funds through the FCT/MCTES, and was part of the project “RECLAIM- RE-manufaCturing and Refurbishment LARge Industrial equipMent” and received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 869884.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lasi, H.; Fettke, P.; Kemper, H.G.; Feld, T.; Hoffmann, M. Industry 4.0. *Bus. Inf. Syst. Eng.* **2014**, *6*, 239–242. [CrossRef]
2. Xu, H.; Yu, W.; Griffith, D.; Golmie, N. A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective. *IEEE Access* **2018**, *6*, 78238–78259. [CrossRef]
3. Jazdi, N. Cyber physical systems in the context of Industry 4.0. In Proceedings of the 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, Romania, 22–24 May 2014; pp. 1–4. [CrossRef]
4. Sharma, M.; Elmiligi, H.; Gebali, F., Network Security and Privacy Evaluation Scheme for Cyber Physical Systems (CPS). In *Handbook of Big Data Privacy*; Springer International Publishing: Cham, Switzerland, 2020; pp. 191–217. [CrossRef]
5. Langner, R. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Secur. Priv.* **2011**, *9*, 49–51. [CrossRef]
6. Sinanović, H.; Mrdovic, S. Analysis of Mirai malicious software. In Proceedings of the 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 21–23 September 2017; pp. 1–5. [CrossRef]
7. Oueslati, N.E.; Mrabet, H.; Jemai, A.; Alhomoud, A. Comparative Study of the Common Cyber-physical Attacks in Industry 4.0. In Proceedings of the 2019 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC), Tunis, Tunisia, 20–22 December 2019; pp. 1–7. [CrossRef]
8. Mitchell, R.; Chen, I.R. A Survey of Intrusion Detection Techniques for Cyber-Physical Systems. *ACM Comput. Surv.* **2014**, *46*, 55:1–55:29. [CrossRef]
9. Yasaei, R.; Hernandez, F.; Al Faruque, M.A. IoT-CAD: Context-Aware Adaptive Anomaly Detection in IoT Systems Through Sensor Association. In Proceedings of the 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), San Diego, CA, USA, 2–5 November 2020; pp. 1–9.
10. Bayar, N.; Darmoul, S.; Hajri-Gabouj, S.; Pierreh, H. Fault detection, diagnosis and recovery using Artificial Immune Systems: A review. *Eng. Appl. Artif. Intell.* **2015**, *46*, 43–57. [CrossRef]
11. Dasgupta, D.; Yu, S.; Nino, F. Recent Advances in Artificial Immune Systems: Models and Applications. *Appl. Soft Comput.* **2011**, *11*, 1574–1587. [CrossRef]
12. Forrest, S.; Perelson, A.; Allen, L.; Cherukuri, R. Self-nonsel self discrimination in a computer. In Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, 16–18 May 1994; pp. 202–212. [CrossRef]
13. Matzinger, P. The Danger Model: A Renewed Sense of Self. *Science* **2002**, *296*, 301–305. [CrossRef] [PubMed]
14. Burnet, F.M. A modification of jerne’s theory of antibody production using the concept of clonal selection. *CA A Cancer J. Clin.* **1976**, *26*, 119–121. [CrossRef] [PubMed]
15. De Castro, L.N.; Von Zuben, F.J. *Artificial Immune Systems: Part I—Basic Theory and Applications*; Dezenbro de, Tech. Report; Universidade Estadual de Campinas: Campinas, Brazil, 1999; Volume 210.
16. Haktanirlar Ulutas, B.; Kulturel-Konak, S. A review of clonal selection algorithm and its applications. *Artif. Intell. Rev.* **2011**, *36*, 117–138. [CrossRef]
17. Brownlee, J. *Clonal Selection Algorithms*; Technical Report 070209A; Complex Intelligent Systems Laboratory, Swinburne University of Technology: Victoria, Australia, 2007; pp. 1–13.
18. Nunes de Casto, L.; Von Zuben, F. An evolutionary immune network for data clustering. In Proceedings of the Vol.1. Sixth Brazilian Symposium on Neural Networks, Rio de Janeiro, Brazil, 25 November 2000; pp. 84–89. [CrossRef]

19. Timmis, J.; Neal, M.; Hunt, J. An artificial immune system for data analysis. *Biosystems* **2000**, *55*, 143–150. [CrossRef]
20. Jerne, N.K. Towards a network theory of the immune system. *Ann. Immunol.* **1974**, *125C*, 373–389.
21. Hoffmann, G.W. Immune Network Theory. Monograph. 2008. Available online: <https://phas.ubc.ca/~hoffmann/book/Immune%20Network%20Theory.pdf> (accessed on 21 November 2021).
22. Richter, P.H. A network theory of the immune system. *Eur. J. Immunol.* **1975**, *5*, 350–354. [CrossRef] [PubMed]
23. Knight, T.; Timmis, J. AINE: An immunological approach to data mining. In Proceedings of the 2001 IEEE International Conference on Data Mining, San Jose, CA, USA, 29 November–2 December 2001; IEEE Computer Society: Los Alamitos, CA, USA, 2001; pp. 297–304. [CrossRef]
24. Greensmith, J.; Aickelin, U.; Twycross, J. Articulation and Clarification of the Dendritic Cell Algorithm. In *Artificial Immune Systems*; Bersini, H., Carneiro, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 404–417.
25. Twycross, J.P. Integrated Innate and Adaptive Artificial Immune Systems Applied to Process Anomaly Detection. Ph.D. Thesis, University of Nottingham, Nottingham, UK, 2007.
26. Greensmith, J. The Dendritic Cell Algorithm. Ph.D. Thesis, University of Nottingham, Nottingham, UK, 2007.
27. Al-Dabagh, N.B.I.; Ali, I.A. Design and implementation of artificial immune system for detecting flooding attacks. In Proceedings of the 2011 International Conference on High Performance Computing Simulation, Istanbul, Turkey, 4–8 July 2011; pp. 381–390. [CrossRef]
28. Pinto, R.; Gonçalves, G.; Tovar, E.; Delsing, J. Attack Detection in Cyber-Physical Production Systems using the Deterministic Dendritic Cell Algorithm. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 1552–1559. [CrossRef]
29. Ulsoy, A.G., Monitoring and Control of Machining. In *Condition Monitoring and Control for Intelligent Manufacturing*; Springer: London, UK, 2006; pp. 1–32. [CrossRef]
30. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly Detection: A Survey. *ACM Comput. Surv.* **2009**, *41*, 1–58. [CrossRef]
31. Bhattacharyya, D.K.; Kalita, J.K. *Network Anomaly Detection: A Machine Learning Perspective*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2019.
32. Tokarev, V.; Sychugov, A.; Anchishkin, A. Detection of Anomalies in the Information Networks of Industrial Automation Systems Based on Artificial Immune Detectors. In Proceedings of the 2019 International Russian Automation Conference (RusAutoCon), Sochi, Russia, 8–14 September 2019; pp. 1–5. [CrossRef]
33. Costa Silva, G.; Palhares, R.M.; Caminhas, W.M. A Transitional View of Immune Inspired Techniques for Anomaly Detection. In *Intelligent Data Engineering and Automated Learning—IDEAL 2012*; Yin, H., Costa, J.A.F., Barreto, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 568–577.
34. Textor, J. A Comparative Study of Negative Selection Based Anomaly Detection in Sequence Data. In *Artificial Immune Systems*; Coello Coello, C.A., Greensmith, J., Krasnogor, N., Liò, P., Nicosia, G., Pavone, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 28–41.
35. Greensmith, J.; Twycross, J.; Aickelin, U. Dendritic Cells for Anomaly Detection. In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 664–671. [CrossRef]
36. Greensmith, J.; Aickelin, U.; Tedesco, G. Information fusion for anomaly detection with the dendritic cell algorithm. *Inf. Fusion* **2010**, *11*, 21–34. [CrossRef]
37. Gu, F.; Greensmith, J.; Aickelin, U. Theoretical formulation and analysis of the deterministic dendritic cell algorithm. *Biosystems* **2013**, *111*, 127–135. [CrossRef] [PubMed]
38. Greensmith, J.; Gale, M.B. The Functional Dendritic Cell Algorithm: A formal specification with Haskell. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 5–8 June 2017; pp. 1787–1794. [CrossRef]
39. Gu, F.; Greensmith, J.; Aickelin, U. Integrating Real-Time Analysis with the Dendritic Cell Algorithm through Segmentation. In Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, Montreal, QC, Canada, 8–12 July 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 1203–1210. [CrossRef]
40. Yuan, S.; Chen, Q.j. A Dendritic Cell Algorithm for real-time anomaly detection. In Proceedings of the 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), Zhangjiajie, China, 25–27 May 2012; Volume 1, pp. 448–451. [CrossRef]
41. Pinto, R.; Gonçalves, G.; Delsing, J.; Tovar, E. Incremental Dendritic Cell Algorithm for Intrusion Detection in Cyber-Physical Production Systems. In *Intelligent Computing*; Arai, K., Ed.; Springer International Publishing: Cham, Switzerland, 2021; pp. 664–680.
42. Gu, F.; Greensmith, J.; Oates, R.; Aickelin, U. Pca 4 dca: The application of principal component analysis to the dendritic cell algorithm. *SSRN Electron. J.* **2009**. [CrossRef]
43. Chelly, Z.; Elouedi, Z. A study of the data pre-processing module of the dendritic cell evolutionary algorithm. In Proceedings of the 2014 International Conference on Control, Decision and Information Technologies (CoDIT), Metz, France, 3–5 November 2014; pp. 634–639. [CrossRef]
44. Katser, I.D.; Kozitsin, V.O. Skoltech Anomaly Benchmark (SKAB). 2020. Available online: <https://www.kaggle.com/dsv/1693952> (accessed on 21 November 2021). [CrossRef]
45. Pinto, R. M2M Using OPC UA; IEEE Dataport; 2020. Available online: <https://iee-dataport.org/open-access/m2m-using-opc-ua> (accessed on 21 November 2021).