

Environment Modeling and Abstraction of Network States for Cognitive Functions

Stephen S. Mwanje*, Márton Kajó[†], Sayantini Majumdar[‡], Georg Carle[†]

*Nokia Bell Labs

Email: stephen.mwanje@nokia-bell-labs.com

[†]Technical University of Munich, Department of Informatics

Email: {kajo, carle}@net.in.tum.de

[‡]Technical University of Munich, Department of Electrical and Computer Engineering

Email: sayantini.majumdar@tum.de

Abstract—Cognitive Autonomous Networks (CANs) promise to overcome the shortcomings of current Self-Organizing Network (SON) implementations, i.e., the limited flexibility and adaptability to changing environments, by applying cognition. In CAN, intelligent network automation functions, herein called Cognitive Functions (CFs), apply machine learning techniques to learn context-specific behavioral policies with which to automate network operations. For proper operation, the CAN system needs to learn the environment in which the functions are operating and to abstract the environment and performance observations into states to which the CFs must respond. This paper proposes a design and implementation of an Environmental-state Modeling and Abstraction (EMA) engine that could be tasked to learn the required abstract states in a consistent way across multiple CFs.

Index Terms—Cognitive Network Management, Cognitive Autonomous Network, Network State Modeling

I. INTRODUCTION

The concept of applying cognition to Network Management (NM) has been advanced in several publications, including [1], [2], which propose to replace the hard-coded logic in Self-Organizing Network (SON) functions with highly intelligent, adaptive Cognitive Functions (CFs). The CFs should be capable of learning optimal behavior based on their actions and the observed impact on the network, network environment and other CFs, by accepting and processing various kinds of data (such as configuration, performance, failure, or user/service-related data). If these CFs are in place, the evolved system would result in what has been labeled in our nomenclature as a Cognitive Autonomous Network (CAN). In effect, CAN extends SON to: 1) be able to infer higher level network and environment states from a multitude of data sources instead of the current low-level basic states recovered from Key Performance Indicators (KPIs) values, 2) allow for adaptive selection and changes of Network Configuration Parameters (NCPs) depending on previous actions and operator goals.

The CAN framework proposed in [3] highlights where and how cognition should be introduced into the NM system. The framework, shown in Fig. 1, proposes five major components:

- 1) a Network Objectives Manager (NOM) to interpret operator service and application goals for the CAN so that the CFs adjust their behavior in line with those goals;

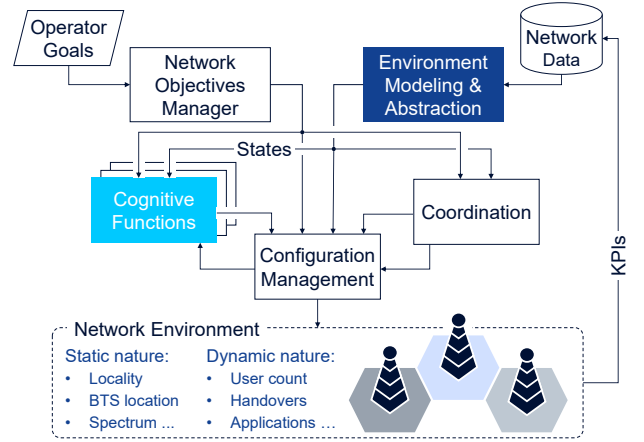


Fig. 1. The components of a Cognitive Autonomous Network

- 2) a set of CFs which learn the appropriate network configuration for each the network and environmental state. These ‘active configurations’ are selected from a set of legal/acceptable candidate network configurations;
- 3) a Coordination Engine (CE) coordinates the actions of multiple CFs (even amid the non-deterministic behavior of the CFs resulting from their learning nature) to ensure acceptable system-wide performance;
- 4) a Configuration Management Engine (CME) learns to (re-)define the permissible candidate network and function configurations for the different contexts of the CFs;
- 5) an Environmental-state Modeling and Abstraction (EMA) engine learns to abstract the environment into states which are used for subsequent decision making in the other components

This paper focuses on the realization of the EMA engine, herein simply called “the EMA”. The EMA is critical to the realization of CANs as CFs need to respond to specific states of the network, which distill varied network information into simpler, quantized values defined uniformly across CFs.

The paper is structured as follows: Section II presents the design objectives of the EMA while section III summarizes the simulation and development environment where the EMA has been tested. Sections IV, V and VI respectively discuss the im-

plementation and performance of the three core components of the EMA - feature extraction, quantization and state mapping. Finally, Section VII summarizes the related works and we end with conclusions and possible extensions in section VIII.

II. ENVIRONMENT-STATE MODELING AND ABSTRACTION

The EMA engine needs to abstract data from the combination of configurations, experienced environmental conditions and the subsequently observed KPIs into states to which the CFs respond. Specifically, the EMA defines abstract states built from combinations of quantitative KPIs, abstract (semantic) state labels and operational contexts, such as the current configurations of the network or its elements. The abstraction must be consistent across multiple CFs within a CAN instance be it in one or more network elements, domains or sub-networks. Within a single CAN instance, multiple modules need to work together for the system to eventually learn the optimal network configurations. For example, a CME needs to configure CFs according to its learning about their behavior while the CE may want to inform a CF about the effects of its action to other CFs. All these modules must reference the same abstract states when communicating with one another, and so they require a separate, common function that defines these states. Meanwhile, the creation of such states should be flexible enough to allow for their online adjustment during operations, i.e., the EMA should be able to modify/split/aggregate/delete states as may be required by the subsequent CFs.

The input to the EMA at a given time is a vector of continuous-valued environmental parameters, network configuration and KPI values. The output of the EMA is a network-state-vector S of dimension m , representing the (quasi-orthogonal) aspects of interest or optimization for the specific CFs active in the CAN system. Such dimensions could be for example those for which the operator expects some action to be taken e.g. user mobility, cell load, energy consumption level, etc. These are either defined by the operator or by the NOM as configuration for the EMA. These output state vectors populate an *external state-space*.

The proposed implementation of the EMA can be seen in Fig. 2. Here the EMA is made up of 3 distinct components - feature extraction, quantization and state mapping. The main goal of the EMA is to create an *internal state-space* through the Environmental Modeling (EM) module. The internal state-space serves as an all-encompassing and objective, but also simplified and structured view of the network and its environment. Although this view contains the behavioral patterns hidden in the observations, it does not by itself contain contextual information required for the functioning of the CFs. However, the quantized nature of the state-space makes it easy to append this contextual information later, which is the task of the State Abstraction (SA) module.

At run-time, the EM module maps incoming input vectors to one of the internal states, transforming the n -dimensional continuous-space input into k discrete volumes through quantization. Since it can be expected that some of the input

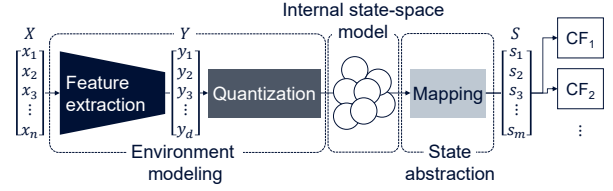


Fig. 2. The components and input-output vectors of the EMA

dimensions contain noise or redundant information, it is beneficial to precede the quantization step with a feature extractor, which removes these interfering parts of the data. Because both feature extraction and quantization only aims to model structure that is already present in the data (no additional expert knowledge or context need to be added), both functions can be trained through unsupervised means. The internal state-space model is not modifiable after training, and tries to encompass all behavioral aspects of the network elements.

The SA module creates flexible mappings modifiable at run-time to fit the CFs' needs. This bridges the gap between the objective internal state-space representation and the subjective external states that are specific to the active CFs. This is done by storing a unique labeling (coloration) of the internal states for each output feature, and possibly also for each attached CF. This way, each output dimension can only take at most k individual values, where k is the number of internal network states defined in the EMA. The coloration contains expert knowledge or context information required for the functioning of the CFs that consume this information.

For the internal state-space model to be able to map the network's behavior regardless of user bias, the EM functions need to be trainable in an unsupervised manner, without the use of labeled training data. Usually, most unsupervised learning algorithms do require a handful of meta-parameters, which must be set prior to training by the user, or, by the implementer. As it stands currently, the internal state-space is not meant to be re-configurable after training, and so should be trained with as much data as possible from the network to be able to form a comprehensive view that is valid for all network elements and CFs. On the other hand, State Mapping is trained in a supervised or semi-supervised way, owing mainly to the need for feedback about the utility of the different dimensions for the CFs, information which is not present in the input data for the EMA. For the same reason, State Abstraction is also meant to be re-configurable, as it is highly likely that requirements or settings change during a CF's lifetime.

It is important to note that since the State Abstraction can be influenced by re-configurations of the constraints for the specific dimensions, the EMA needs to have a fine-grain internal representation of the state-space which it uses to abstract into the output states. This way, even with reconfiguration of constraints, it does not need to re-learn the underlying state-space model, but to only adjust the mapping between internal and external states. The rest of the paper explores the extent to which the concept is valid and feasible.

III. SIMULATION ENVIRONMENT AND DATA

The EMA prototype was validated with simulation data generated from a system level simulator that uses and extends the emulator in [4]. The radio access network can utilize multiple Radio Access Technologies (RATs), including directional 3-sector macro (2 GHz), and omnidirectional micro (2.6 GHz) and WLAN (2.4 GHz) transceivers, with each RAT simulated using realistic radio propagation models (specifically, the WINNER+ model [5]). The environment where the simulations took place is a faithful recreation of the inner city of Helsinki, where simulated automotive and pedestrian users moved along a street graph mobility pattern. Although we only used radio related measurements in this evaluation, the simulator implements full flow-level simulation, such as FTP or video traffic, and could also provide flow related data if needed.

To create a scenario with a sufficiently diverse network landscape, we deployed 36 access points/cells with the above mentioned 3 RAT types. To add further complexity, 12 different cell configurations (classes) were defined by varying RAT type, Reference Signal Received Power (RSRP) offset, Time-To-Trigger (TTT) and Transmission Power (TXP) in the deployed cells, as shown in Table I. Note that although the simulator configuration allows for inclusion of TTT for Wifi cells, these are not used for the handover between Wifi cells. Otherwise, the aim of these 12 parameter combinations is to introduce a high degree of heterogeneity in the network, which, together with the unpredictability of the UE mobility tries to emulate a complex heterogeneous network deployment.

The output of the simulator is made up of measurements at the cell transceiver level. Generally, three categories of output measurements were observed, an overview of which can be found in Table II with examples. These measurements were used in subsequent data preprocessing steps to bring the raw simulation data into a form intelligible to machine learning algorithms (specifically neural networks). The data pertaining to a single cell was enriched by appending the aggregated properties (KPIs) of its neighbors, essentially considering *shared* characteristics of the cells in the network. The simulator worked with a simulation step of 0.1 ms, outputting these measurements too frequently for network management use. To rectify this, all measurements were aggregated in time to 15 second intervals. Finally, the resulting dataset was normalized per feature across all timestamps and cells. The end result of the preprocessing is a clean, standardized dataset consisting of two kinds of information - identifiers (in integer form) and KPIs (in floating point values). The dataset, which serves as input to the EMA, did not include network configuration details in any quantity. With all this, the final dataset contained 34 features altogether.

IV. FEATURE EXTRACTION, USING AN AUTOENCODER

The EMA input is made up of many features (dimensions), each of which embodies an aspect of the network environment that the input represents. Since it is highly challenging to process or visualize high-dimensional datasets [6], feature

TABLE I
CELL CONFIGURATIONS

RAT type	RSRP offset (dB)	TTT(s)	TXP (dBm)
WLAN ₁ *	-	0.512	20
WLAN ₂ *	-	0.64	20
LTE micro ₁	2	0.512	20
LTE micro ₂	2	0.64	20
LTE macro ₁₁	2	0.512	46
LTE macro ₁₂	3	0.512	46
LTE macro ₁₃	3	0.512	40
LTE macro ₁₄	2	0.512	40
LTE macro ₂₁	2	0.64	46
LTE macro ₂₂	3	0.64	46
LTE macro ₂₃	3	0.64	40
LTE macro ₂₄	2	0.64	40

TABLE II
CATEGORIES OF CELL EVENTS AND MEASURES

Type	Description	Example
Cell counters	Local events or those in neighbor cells	Radio Link Failures (RLFs), Handovers (HOs)
Shared counters	Events for specific cell pairs	RLF due to late handover (HO) from cell 1 to 2
KPIs	KPIs aggregated to every simulation step	Total number of active UEs, cell load

extraction is often used to try to find a better representation to which later machine learning algorithms can be applied effectively. Specifically, in our setting, distance-based quantization is susceptible of breaking down in high-dimensional spaces, which warrants the use of feature extraction.

The task of the feature extraction module is to compress the input to a lower-dimensional representation, while also possibly removing redundant information and noise. The number of extracted features (d) is usually significantly smaller than the number of input features ($d \ll n$), but using more dimensions ($d \gg n$) with sparsity enforced could also be a viable alternative. The reduction of features or the enforcement of sparsity usually means a tradeoff between the level of compression and loss of valuable information. It is up to the designer to figure out a good tradeoff, which can be specific to the dataset used, or to the later processing steps.

For the implementation here, a neural network type called an Autoencoder is used as feature extractor. Autoencoders are feedforward neural networks that, by nature of their topology, are forced to learn a lower dimensional representation of the data during training. Autoencoders can be split into two parts: an encoder network, and a decoder, with the low-dimensional encoded representation being communicated between the two halves. During training, the encoder learns an effective compression of the input to the designated dimensionality, while the decoder learns to reconstruct the original information from this compressed representation. Autoencoders are trained in an unsupervised manner, their target being the minimization of the reconstruction loss measured against the original data. In our case, at inference, only the encoder part of the network was utilized.

As stated previously, arriving at a good tradeoff between

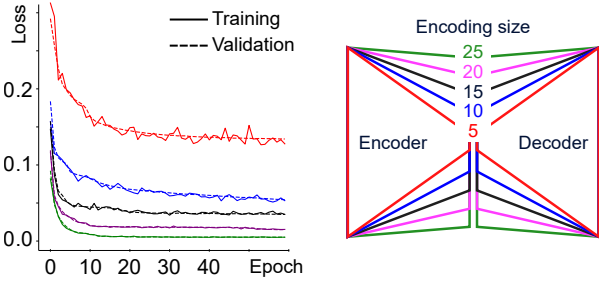


Fig. 3. Plot of training and validation losses for different AE topologies

compression and loss of information is up to the designer. Autoencoders excel in generalizing underlying logic and structure in the data. The more compression is enforced, the more the autoencoder is forced to extract generalization, removing noise and redundancies in the process. However, as compression is increased, more and more nuance is lost in the reconstructed data. The level of compression is set by the number of neurons in the middle layers (encoding size), as such, different compression levels equate to different topologies. Figure 3) shows the reconstruction loss measured on a few autoencoder topologies, with the encoding size highlighted. The used autoencoders also utilized a single hidden layer in both the encoder and the decoder, the size of which were set to the average of the number of original and compressed dimensions (n and d), so that the progression of compression and decompression in the network is roughly linear. In the end, we chose to use a encoding size of 10 a topology that we think is a good tradeoff between compression and loss of information, equating roughly to a $3\times$ compression factor.

Figure 4 shows a scatter plot of the encoded observations with the chosen autoencoder topology, using the first 2 dimensions in the 10-dimensional encoded space. The activations are colored according to the parameter combination of the origin cell. Even though the 2-dimensional scatter plot is incapable of capturing the complexity of the 10-dimensional space, clustering/grouping of the colors is observable even in this basic visualization. As one example, according to Table I, WLAN (and micro) cells could be easily distinguished from the macro cells in terms of transmit power. Although transmit power settings were not communicated to the autoencoder, the difference in behavior still makes these cells stand apart from the other measurements. This distinction is also captured in the encoding, as the WLAN observations are mostly clustered along the top and left part of the plot, as highlighted by the two respectively colored curves.

V. QUANTIZATION WITH k -MEANS AND BSQ

Quantization divides the input space into finite volumes. In the case of the EMA, this is used to form the internal state-space model on the output of the feature extraction. We chose to evaluate two unsupervised clustering algorithms for this task, the traditional Lloyd's k -Means algorithm [7], and a modification of it called Bounding Sphere Quantization (BSQ) [8]. Instead of using existing packages, both algorithms were

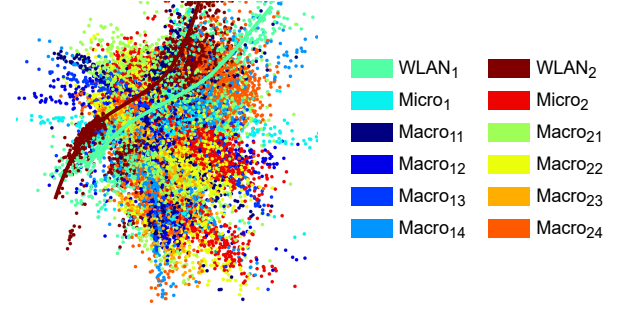


Fig. 4. Distribution of encoded measurements taken from cells with the 12 parameter classes

re-implemented in a neural network-style framework. This implementation synergizes well with the autoencoder neural network implementation, so that the whole processing pipeline of the EMA can use the same tools (python¹, PyTorch²) and utilize hardware acceleration.

A. k -Means vs. BSQ

Lloyd's k -Means is a simple and effective quantization algorithm that is often used in data-preprocessing, data-mining or machine learning tasks. The algorithm's goal is to find k quantum centerpoint (centroid) locations, so that the average of distances between training points and their closest centroids is minimal on the whole training dataset. This overall minimum can be seen as a *global* optimization target. To reach this goal, the original k -Means algorithm employs the Expectation-Maximization framework, where two steps are alternated; the assignment of training points to the closest centroid, and the update of the centroids to best fit the assigned training points. For each centroid, the update moves the centroid to where the average distance to the assigned points is minimal, which can be seen as the fulfillment of the *local* (per quantum) optimization target. The algorithm does not have an explicit mechanism to reach the global target, however, the interactions between the centroids and the individual strives towards the local target do move the quantization towards the global goal. The distance measure used is usually the L2 (Euclidean) distance, in which case the local optimum for each centroid is the mean of the assigned points, hence the name k -Means.

BSQ is a modification of k -Means, where the local optimization target is not to find the smallest average distance in a quantum, but the smallest maximum distance. This can be achieved by moving each centroid to the center of the minimal bounding sphere of the assigned points in each update. Changing the local optimization target affects the global optimization, which then strives to reach a quantization with the smallest maximum distance (across quanta and all training points). This produces quanta that have roughly equal assigned volumes. It has been shown in [8] that this quality is preferable in quantization tasks where the goal is to map the input space, and not necessarily to follow the density distribution of the training points, such as anomaly detection. Perceptibly,

¹<https://www.python.org/>

²<https://www.pytorch.org/>

BSQ produces a quantization where both densely and sparsely populated regions of the training data are quantized with equal resolution, whereas k -Means produces a quantization that has a finer resolution in dense regions, and a coarser resolution in sparse areas of the training data.

B. Neural-Network-based k -Means and BSQ

The neural-network-based implementation for both k -Means and BSQ differs from the above cited implementations, as it uses gradient descent for optimization instead of the expectation-maximization framework, and uses *matrix-matrix operations* organized into layers. This structure can be seen on Fig. 5. For k -Means, the network comprises of two layers, the distance and the masking layers. BSQ expands on this by adding a third layer (illustrated in Fig. 6) that is responsible for selecting the points with the maximum quantization error:

- 1) *Distance layer*: This layer computes the set-to-set distances between the input points and the currently stored quantum centroids. This is implemented using PyTorch's `norm` operation on a 3 dimensional tensor that contains every possible pairwise difference vector between the two sets of points. The norm is the full Euclidean norm according to:

$$\|x\|_2 := \left(\sum_{i=1}^n x_i^2 \right)^{1/2}. \quad (1)$$

The output of the distance layer is a matrix of distances between every observation and quantum centroid, the size of which is thus $p * k$, where p is the number of points fed to the network.

- 2) *Masking layer*: The role of the masking layer is to mask irrelevant distances from the distance matrix that are between points and their non-closest quantum centroids. This is done by creating a mask of equal shape as the distance matrix, where in each row only a single position contains the value 1, the rest is filled with 0s. The masking matrix is element-wise multiplied with the distance matrix, turning off distance values from contributing to the overall loss by multiplying them with the masking factor of 0. The output of this layer is the masked distance matrix with the same shape as the distance matrix.
- 3) *Aggregation layer*: At this point, in case of k -Means the masked distances matrix can be used to create a loss, by column-wise averaging the non-zero values of the distance matrix. For BSQ, only the largest distance is relevant (per quantum), so the aggregation layer picks the largest distances from the masked distance matrix, and only lets these contribute to the final loss value.

In this implementation, the quantum centroids are stored as trainable weights. At the end of the forward propagation phase, the masked distances are summed to create a single loss value. This loss is then backpropagated to calculate a gradient on the centroids, and update them accordingly.

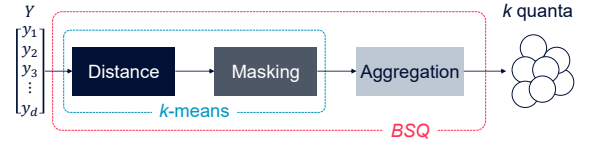


Fig. 5. Layers of neural-network-based implementation of k -Means and BSQ

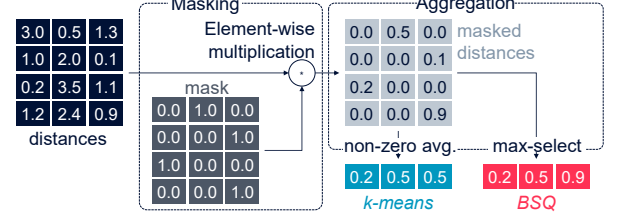


Fig. 6. Example computation in neural-network-based quantization

C. Quantization outcomes

Fig. 7 depicts the results of running the two algorithms on the autoencoder activations from Section IV, by plotting the first 2 dimensions from the 10-dimensional space. Although these plots lose information that is contained in the remaining eight dimensions when projected on 2D, they provide a general idea of how the clustering behaves in higher dimensions. It is visible that with k -Means, quanta are more concentrated around the denser regions of the dataset, while they are more spread out when using BSQ.

The two algorithms differ in their end goal when fitting the quantization. Quantization being an unsupervised learning problem, it is hard to deduct which algorithm is "better" without use-case specific (supervised) metrics. This evaluation will be shown in the next section (Sec. VI). However, it is still beneficial to look at unsupervised metrics to get a better understanding how the two algorithms behave. We considered two evaluation metrics, which cover the respective global optimization goals of the two algorithms:

- Average distances are computed by taking the mean of the nonzero masked distances first within each cluster (i.e. along each column), and then for the resulting one-dimensional tensor over all.
- Maximum distances are computed by taking the maximum of the nonzero masked distances in each quanta, and afterwards averaging them across quanta.

The number of fitted quanta (the value k) is a user-defined parameter for both algorithms. In other clustering tasks, where the goal is to find a logical grouping in the data (logical to a human user), there are theoretically correct k value(s) for each task. In these cases, finding the correct k value is crucial to achieving good results, even more so, if the user has a very strong (biased) expectations of how the formed clusters should look. Fortunately, this is not the case for the quantization in the EMA, because the goal is not to find a logical (to the human supervisor) grouping of the observations. In our case, it is sufficient to quantize the input space with a "fine enough"

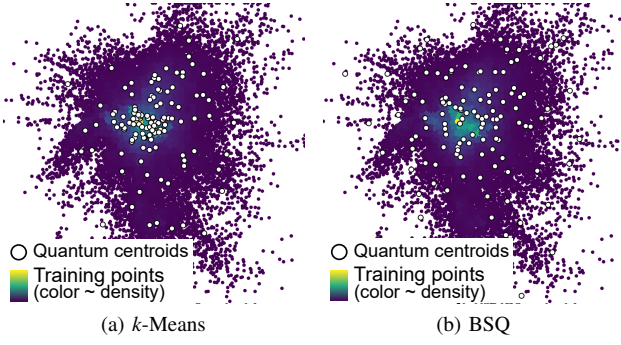


Fig. 7. k -Means and BSQ quantization on the simulator dataset for $k = 120$. The fitted quantum centroids are depicted as white dots, while the training points are colored according to density.

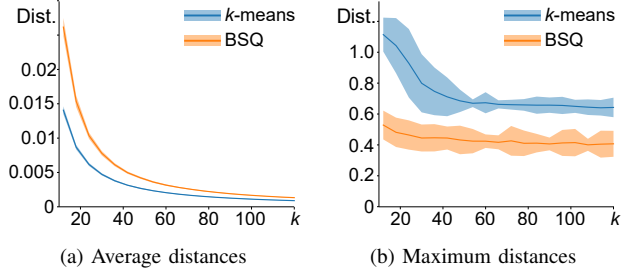


Fig. 8. Average and maximum distances measured for multiple k values for k -Means and BSQ. Shaded regions depict the ± 1 standard deviation, solid lines depict the mean.

resolution so that the mapping stage can work with reasonable precision. This criterium does not really pose a hard upper-limit to the number of quanta that can be used. The number of possible quanta can be softly limited by:

- The number of available training points, so that each quantum stays reasonably populated. This restriction coincides with the avoidance of overfitting, because underpopulated quanta can easily overfit the data, so that the quantization loses generalization power.
- The available computational resources, such as memory. More quanta used in the computation requires more memory, as well as more computational power, or takes longer to compute.

Performance for both algorithms was evaluated for $k = 12, 18, 24, \dots, 120$. For each value of k , both algorithms were run 100 times starting from random initial quantum locations, and the previously explained two evaluation metrics were measured (both metrics for both algorithms). The results can be seen on Figure 8.

As expected, increasing k achieves a smooth reduction of the average distance for both algorithms, with BSQ constantly having 60% larger averages, and both having minimal variance. On the other side, BSQ achieves 30% smaller average maximum distance at larger values of k , and almost 50% smaller average maximum at small values of k . Generally, the maximum distances for both algorithms show a large variance across all values of k .

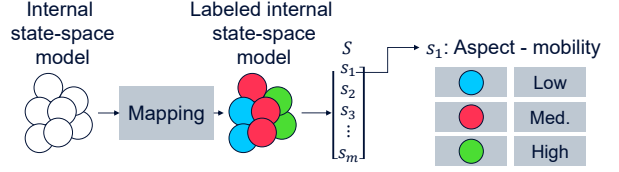


Fig. 9. Mapping operation for a single dimension of the output states - each internal state is labeled with one of a finite number of label of the dimension (here one of three values for the mobility dimension).

VI. STATE MAPPING

Mapping of states, the third component of the EMA, enables translation from the objective but abstract internal states to CF-specific subjective external states. State mapping is essential to convert the output of the quantization, which is the internal state-space model, to well-defined labels, which are understandable by both the human operator and the CFs in the CAN subsequent stages. State-mapping is envisioned as a simple labeling task, where a specific "coloration" (labeling) of the internal state-space is stored for each dimension/aspect of the output external states (Fig. 9). Each "color" (label) represents a different level of the quantized output dimension.

A. The state-mapping assumption

Ideally, the entirely-unsupervised feature reduction and quantization process produces quanta that are *well-aligned* to any sensible labeling of the dataset if it is to be done by the human operator. "Well-aligned", in this case, implies that each cluster contains mostly observations of a single class/label. In practice, this is not necessarily true due to the unsupervised nature of the feature reduction and quantization algorithms, which can produce results that are unfitting to the biased supervised labels produced by the system designer. In this study, the class labels are the different settings for the simulated cells. As mentioned before, there are only soft upper limits to the number of quanta that can be used in the internal state-space model. This is important, because it is assumed that by increasing k , the internal states become more and more aligned to any outside labeling, following a sensible curve. However, this is not necessarily true, depending generally on the distribution of the different classes, but also specifically on the feature reduction and the fitted states. This section tries to answer the question as to whether or not the assumption is correct that increasing k leads to better alignment of the quanta.

B. State-mapping outcomes

We implemented state mapping by assigning the label of the class with the majority population to each internal state. If the above assumption is true, by increasing the resolution of the internal state-space (increasing k), this alignment should constantly improve until reaching an acceptable level. To evaluate the alignment quality, we defined the following state purity measure:

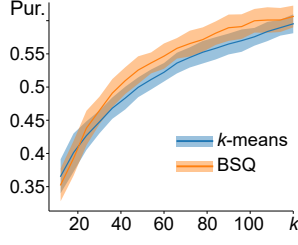


Fig. 10. k -Means and BSQ overall purity for different cluster numbers. Shaded regions depict the ± 1 standard deviation, solid lines depict the mean.

State purity (P), is an external evaluation of quantum alignment, measuring the percentage of the majority class (largest population) against the total population within an internal state, i.e.,

$$P = \sum_i \frac{\max_j r_{ij}}{\sum_j r_{ij}} / k, \quad (2)$$

where r_{ij} is the number of points of class j in state i , and k is the number of internal states. For 100% purity, each state would contain observations from only one class.

Between the two extremes of k , it is not clear how purity changes. On one end, using a single state for the whole dataset would yield a purity that is inherent in the class populations in the data. On the other end, using a state for every single observation would produce a purity value of a 100%. However, between these two values, if the change is not gradual, state mapping would be hard to use, as a well-functioning k would be hard to find, or computationally infeasible to use.

Figure 10 shows how state purity changes with k . For both algorithms, the average overall purity values steadily increase with increasing k , indicating that quantization resolution is well correlated to mapping accuracy. Although starting at a lower value, BSQ also seems to have a steeper slope in the lower ranges of k compared to k -Means. The two averages cross over at $k = 24$, after which point BSQ has a constant lead up to $k = 100$, where this lead starts to diminish. The variance seems to keep constant for all the different k values.

Fig. 11 provides a more insightful look into the distribution of state purity values for both algorithms, for 4 different k values. In the plots, the horizontal axis gives the percentage of purity. We see that for low values of k (blue curve, $k = 12$), k -Means has very strong peak densities of purity around 0.25 and 0.55, whereas BSQ does not have such defined peaks. At a high value of k (red curve, $k = 120$), both distributions are very similar, however, k -Means seems to have more states which are close to a 100% purity. These differences in the distributions reflect the ones that we have already seen for the average state purity previously.

Fig. 12 shows the share average and variance for each class separately for 4 different values of k . The horizontal axis gives the class labels, while the vertical axis shows the normalized number of clusters with majority for the specific class, so that 1.0 equals to having $1/12^{th}$ of the clusters as majority. Each observation for a specific value of k and class has been plotted on the horizontal axis with a small offset. The main takeaway here is that as k increases, the normalized average share

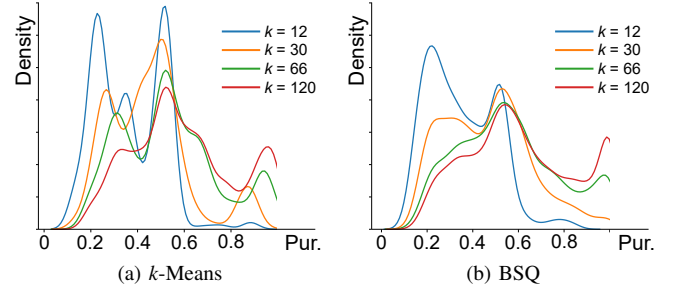


Fig. 11. State purity distribution for $k = 12, 30, 66, 120$

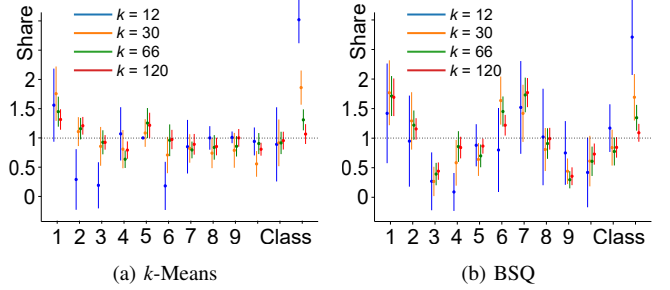


Fig. 12. Share of clusters for each class for $k = 12, 30, 66, 120$ quanta

for each class tends towards 1.0, meaning that the majority class labels are more and more evenly distributed between the clusters, with no class having a tendency to not be assigned to any of the clusters (to be "left out" of the mapping). This is warranted, as originally the class labels occupy a roughly even amount of observations in the dataset.

In general, the original assumption of being able to find a good enough mapping accuracy to any sensible labeling of the state-space seems warranted. With a larger number of k , the simple mapping process of assigning the label of the majority population to the cluster can give an acceptable level of cluster purity. When the value of k is constrained, BSQ is recommended, otherwise k -Means is the better choice.

VII. RELATED WORK

The modeling of a system's states is of great interest in studies on many complex systems. System-state modeling has been studied in generic autonomous systems, e.g., in [9], and in cyber-security systems, e.g., to characterize the behavior of attack systems as in [10], [11]. The most widely studied systems-states are in robotic control systems where a controller may wish to identify the different states of the underlying system and how to respond to such states [12], [13]. The typical challenge here is to derive a closed form description of the different features of the robotic system, e.g. to characterize the angular position, angular velocity and armature current in the case of motion control of a DC motor [14].

There has, however, been little work on modeling states in communication networks. Typically, theoretical system models are developed to study the behavior of the wireless network in particular conditions, the most widely documented models being in wireless sensor networks [15], [16]. These however do not model the different states in which the system may be

or their interaction with the environment or control conditions. This is also the case for communication network management automation. To our knowledge, there exists no generic evaluation of network states for multiple cognitive functions - all that is available is specific to use cases such as anomaly detection.

The concept of network states and their inference in self-organizing networks for enabling 5G have been treated in [17]. Similar to our work, the authors proposed a framework responsible for selecting a set of metrics to characterize the network states, reduced the dimensionality of the data using Principal Component Analysis and applied a semi-supervised classification algorithm, suited to deal with unambiguous knowledge about the classes. Although this paper adopts a similar algorithmic framework in the sequence of dimension reduction and clustering using semi-supervised Fuzzy C-Means, it is solely applied to the anomaly detection use case.

The authors in [18] also proposed an improved anomaly detection framework by dynamically learning mobile network cell states using the Merge Growing Neural Gas (MGNG) algorithm. The baseline was set to be the KPIs of individual cells fed to the MGNG algorithm in order to determine shared or common behavior of the cells, an approach which the data processing steps in our study follow.

VIII. CONCLUSION AND FUTURE STUDIES

This paper has studied the problem of network state abstraction for the creation of internal states required by the CFs as part of a CAN framework. The components of the EMA have been implemented and validated using system level simulations of an LTE network simulator. We considered the problem of feature extraction using autoencoders to reduce dimensionality of the input space. The next step, quantization, dealt with the problem of approximating the input space with a varying number of states, using a modified implementation of the k -Means and the Bounding Sphere Quantization (BSQ) algorithms, in order to generate an accurate, fine-grained internal state-space model. Finally, the key challenge in state mapping was to produce states of acceptable class purity, such that every state contained points with a single label.

We observed that increasing cluster resolution increased the mapping accuracy. In general, we concluded that the BSQ algorithm is suitable when the number of clusters is a constraint and k -Means otherwise.

A potential future task would be the implementation of a sub-setting block, an additional challenge after state mapping, to filter out external dimensions specific to the CFs based on CF feedback. Apart from that, by using a dataset with more variance with varying UE mobility patterns and dynamic assignment of radio network settings, more unpredictability would be expected in the network to reflect realistic scenarios. A second future goal is the inclusion of feedback from the CFs to the mapping. This goal will be pursued as more advanced CFs are developed for 5G scenarios.

REFERENCES

- [1] S. Mwanje, C. Schmelz, and A. Mitschele-Thiel, "Cognitive cellular networks: A q-learning framework for self-organizing networks," *IEEE Transactions on Network and Service Management*, vol. 13, no. Issue 1, pp. 85 – 98, January 2016.
- [2] S. Mwanje, G. Decarreau, C. Mannweiler, M. Naseer-ul-Islam, and L. C. Schmelz, "Network management automation in 5g: Challenges and opportunities," in *Proceedings of the 27th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2016, pp. 1 – 6.
- [3] S. S. Mwanje and C. Mannweiler, "Towards cognitive autonomous networks in 5g," in *Proceedings of ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*. IEEE, 2018.
- [4] Nokia Siemens Networks, "Self-organizing network (son): Introducing the nokia siemens networks son suite-an efficient, future-proof platform for son," *White Paper*, October, 2009.
- [5] J. Meinilä, P. Kyösti, L. Hentilä, T. Jämsä, E. E. Suikkanen, and M. Narandzic, "Wireless world initiative new radio winner+, d5. 3: Winner+ final channel models," *CELTIC Telecommunication Solutions, Tech. Rep.*, 2010.
- [6] L. Chen, "Curse of dimensionality," in *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009, pp. 545–546.
- [7] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.
- [8] M. Kajó, B. Schultz, J. Ali-Tolppa, and G. Carle, "Equal-volume quantization of mobile network data using bounding spheres and boxes," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.
- [9] C. G. Downes, "An investigation into hazard-centric analysis of complex autonomous systems," 2013.
- [10] H. A. Hamadani, Ali Zeinal; Khorshidi, "System reliability optimization using time value of money," *The International Journal of Advanced Manufacturing Technology*, vol. 66, no. 1–4, pp. 97–106, 2013.
- [11] R. Hassan, Sharif; Guha, "Modelling of the state of systems with defensive deception," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2016, pp. 1031–1036.
- [12] B. Triggs, "Model-based sonar localisation for mobile robots," *Robotics and Autonomous Systems*, vol. 12, no. 3-4, pp. 173–186, 1994.
- [13] B. Mahler and J. Haase, "Mathematical model and control strategy of a two-wheeled self-balancing robot," in *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2013, pp. 4198–4203.
- [14] S. Ghosh, R. K. Barai, S. Bhattacharya, P. Bhattacharyya, S. Rudra, A. Dutta, and R. Pyne, "An fpga based implementation of a flexible digital pid controller for a motion control system," in *2013 International Conference on Computer Communication and Informatics*. IEEE, 2013, pp. 1–6.
- [15] B. Krishnamachari, D. Estrin, and S. Wicker, "Modelling data-centric routing in wireless sensor networks," in *IEEE infocom*, vol. 2, 2002, pp. 39–44.
- [16] A. Fehnker, L. Van Hoesel, and A. Mader, "Modelling and verification of the lmac protocol for wireless sensor networks," in *International Conference on Integrated Formal Methods*. Springer, 2007, pp. 253–272.
- [17] Q. Liao and S. Stanczak, "Network state awareness and proactive anomaly detection in self-organizing networks," in *2015 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2015, pp. 1–6.
- [18] B. Gajic, S. Nováczki, and S. Mwanje, "An improved anomaly detection in mobile networks by using incremental time-aware clustering," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 1286–1291.