Game theoretic Conflict Resolution Mechanism for Cognitive Autonomous Networks

Anubhab Banerjee[1,2], Stephen S. Mwanje[1], and Georg Carle [2]
[1]Nokia Bell Labs, Munich, Germany
[2]Dept. of Informatics, Technical University of Munich, Germany
Email:*anubhab.banerjee@tum.de, stephen.mwanje@nokia-bell-labs.com, carle@net.in.tum.de*

***Abstract*—Cognitive Autonomous Networks (CAN) advance network automation by using Cognitive Functions (CFs) which learn optimal behavior through interaction with the network. However, as in self Organizing Networks (SON), CFs encounter conflicts due to overlap in parameters or objectives. Owing to the non-deterministic behavior of CFs, their conflicts cannot be resolved using SON-style rule-based approaches. This paper proposes the Cognitive Bargaining Mechanism (CBM) as the optimal generic way for resolving - any type of conflict among CFs, conflict among any number of CFs and any number of simultaneously existing conflicts among CFs. With the CAN modeled as a multi-agent system (MAS), CBM uses Nash's Social Welfare Function (NSWF) to compute a compromise among CFs that is fair and optimal for the collective interest of the system. To prove the feasibility of the approach, we model three different CAN scenarios in Python and show the resulting configurations when a CBM-enabled controller is used to resolve all the possible conflicts in the CAN.**

***Index Terms*—Cognitive Autonomous Networks, Conflict Resolution, Game Theory, Machine Learning, Nash's Social Welfare Function**

I. Introduction

Cognitive Autonomous Networks (CAN) [1] are expected to raise the degree of automation in mobile networks beyond what was achieved by Self-Organizing Networks (SON) [2]. SON proposed to deploy several closed-loop control-based functions, called SON Functions (SFs), each of which is responsible for managing an individual target. An SF calculates network configuration parameters following some predefined rules for a particular network state which marks the basis of limitations of SON. Using rules in automation implies that the system has several disadvantages, like, limited adaptability in a changing environment and difficulty in maintenance and upgrade. CAN overcomes these drawbacks by replacing SFs with Cognitive Functions (CFs). Each CF is a learning agent which adapts itself in a changing environment based on its experience and does not follow any predefined rules, which makes the maintenance and upgrade of the system easier, and thus, the disadvantages of SON are overcome.

However, making network automation functions (NAFs) cognitive raises new problems. Since NAFs work in parallel, conflicts of interest may arise among them due to overlap of parameters and objectives. In SON, to remove the conflicts, conventionally a rule-based controller atop the SFs is used. For example, in [3] authors proposed an external controller which guides the SON for faster convergence. However, a CF does not follow any pre-written rules like a SF does, it generates its own rules based on its previous learning experience, which makes its behavior unknown to the controller beforehand. As no external entity knows the rules a CF follows, a rule-based SON controller fails to resolve conflicts in a manner beneficial for the combined interest of all CFs and some new approach is needed. The idea of a controller for CFs in CAN already exists [1], but currently there does not exist any mechanism for such a controller which is beneficial for combined interest of all CFs in the CAN.

The main contribution of this paper is designing a technology, named Cognitive Bargaining Mechanism (CBM), which can be used in the controller in CAN and which is able to dynamically resolve - i) any kind of conflicts which may arise among the CFs, ii) any number of simultaneously existing conflicts between the CFs, and, iii) conflicts among any number of CFs. These properties of the proposed CBM shows that it is suitable for use in any real system with no restriction on number and type of automation functions. The CBM, proposed in this paper, is generic so that we do not need separate controllers to resolve separate types of conflicts and a single CBM implemented in a single controller can be used to resolve all types of conflicts, which reduces the design complexity of the controller and makes it easy to implement. Whenever any type of conflict arises between two or multiple CFs over a configuration, the controller recalculates the configuration, which is optimal for combined interest of all parties involved, using Nash's Social Welfare Function (NSWF) [4]. To prove the effectiveness of the solution proposed by the CBM, we model the CAN in Python and perform extensive numerical analysis. All the acronyms used in this paper are listed in Table I.

II. MAS model of CAN

In this section we provide a hierarchical overview of CAN. In the hierarchy, the controller acts one layer above the CFs as depicted in Fig. 1a. The input and output of CFs and the controller are discussed in Section IV in detail. To formulate the problem we study in this paper,

SummerSim-SPECTS 2020, July 20-22, Madrid, Spain; ©2020 Society for Modeling and Simulation (SCS) International

TABLE I: List of acronyms

Acronym	**Full name**
CAN	Cognitive Autonomous Networks
CBM	Cognitive Bargaining Mechanism
CF	Cognitive Function
MAS	Multi Agent System
MLM	Machine Learning Model
NAF	Network Automation Function
NSWF	Nash's Social Welfare Function
SF	SON Function
SON	Self Organizing Networks

we abstract CAN as a Multi-Agent System (MAS), where each CF acts as an agent of the system. All agents in the MAS, i.e., CFs, have the following four properties -

P1 Each agent can learn and decide what is the best action for it by itself in a dynamic environment.
P2 The agents do not communicate with each other and no one has a complete knowledge of the system.
P3 Some or all of these agents share the same resources and there exist conflicts of interests among them.
P4 The agents try to optimize its own target or goal simultaneously, and the concept of a common or team goal does not exist.

A detailed survey on existing research works on MAS, where agents have these properties, are covered in Section VI. This survey ensures that currently there does not exist any research work on a MAS which has all the four properties listed above.

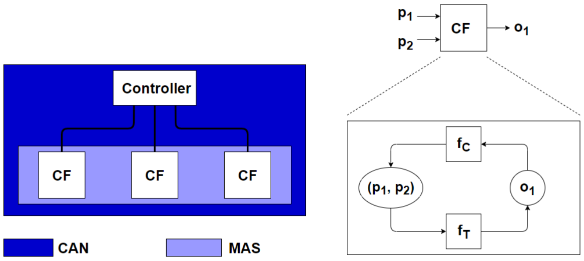
Each CF to consist of two parts - Control function part and Transfer function part. Let us consider a CF F_1 (shown in Fig. 1b) with its Control (f_C) and Transfer function (f_T) parts. The objective (or, output) of F_1 is o_1. o_1 is dependent on two configurations (or, parameters) p_1 and p_2. The relationship between o_1 and (p_1, p_2) can be written as $o_1 = f_T(p_1, p_2, s_N)$, where f_T is the transfer function and s_N is the corresponding network state. When s_N changes, o_1 also may change, so, f_T is not constant and it may vary over time.

The functionality of f_C is to imitate and model f_T which is not constant and unknown beforehand. If we can model f_C such that it is the same as f_T, we can - i) get the value of output o_1 for any (p_1, p_2) without trying (p_1, p_2) directly on the network, and, ii) determine in advance the values of (p_1, p_2) for which o_1 is optimum. To model f_T as accurately as possible, f_C always observes o_1 and figures out the dependence of o_1 on (p_1, p_2). Hereafter, whenever we mention CF, we refer to the Control function part (f_C) of the CF only.

III. Problem statement

In this section we elaborate the problem addressed in this paper. As already stated earlier, NAFs work in parallel often sharing the same resources and objectives and because of that, conflicts may arise among them. These NAFs exhibit three types of conflicts [2] -

- **Category A.** Configuration conflict which occurs on either, (A1) Input, or, (A2) Output parameter(s).



(a) MAS abstraction of hierarchical CAN

(b) CF model

Fig. 1: CAN and CF model

- **Category B.** Measurement conflict where action of one function influences measurement of output of another.
- **Category C.** Characteristic conflict which are of two types - (C1)Direct characteristic conflict and (C2)Logical dependency conflict.

To state the problem we address in this paper, we build three separate CAN models-

- **CAN Model 1** with six CFs (F_1, F_2, F_3, F_4, F_5, F_6) and a controller with in-built proposed CBM (as shown in Fig. 2a).
- **CAN Model 2** with four CFs (F_1', F_2', F_3', F_4') and a controller with in-built proposed CBM (Fig. 2b).
- **CAN Model 3** with two CFs (F_1'', F_2'') and a controller with in-built proposed CBM (Fig. 2c).

From CAN Model 1 we see that - both F_1 and F_2 share the same input parameter (p_1), so if they have an input parameter conflict (A1). As actions of F_3 affects the measurement of output of F_4, it is a measurement conflict (B1). Also, changing p_7 affects o_5 and o_6 is dependent on o_5, hence it is a logical dependency conflict (C2). Thus, this model exhibits all possible types of conflicts. Similarly, if we analyze CAN Model 2, we see that four CFs (F_1', F_2', F_3', F_4') have input parameter conflict over input parameter p_1'. Finally, when we analyze CAN Model 3, we see that F_1'' and F_2'' have - input parameter conflict over p_1'', measurement conflict (as action of F_1'' impacts the measurement of o_2''), and, characteristic conflict (logical dependency conflict, $o_2'' \rightarrow o_1'' \rightarrow p_2''$), i.e., all three types of conflicts are existing simultaneously. In all three CAN models, design and working principle of the controller is always the same.

Our target is to design a single controller to resolve -

- any kind of conflicts which may arise among the CFs,
- any number of simultaneously existing conflicts between the CFs,
- conflicts among any number of CFs, in a dynamic way.

If the controller can resolve all the conflicts in CAN Model 1, we can say that it is able to resolve any kind of conflicts which may arise among the CFs. Similarly, if the controller is able to resolve the conflicts in CAN Model 2 and Model 3, we can say that it is able to resolve any number of simultaneously existing conflicts and conflicts among any number of CFs simultaneously. To summarize, if we can show that the proposed CBM enabled controller can resolve conflicts in all three CAN models, our claim will be justified. To have the proposed functionality, the controller has to hold the following properties -

- it has to be dynamic, i.e., the controller is automated and is able to resolve the conflict automatically the moment it arises, and,
- be generic, i.e., the solution mechanism is independent of the specific conflict of interest.

In the remaining sections we discuss about the design of CBM and its ability to resolve conflicts in detail.

IV. Design and model of proposed controller

In this section we discuss the design and workflow of the CBM proposed in this paper. We discuss the working principle of each component individually followed by a complete overview on the workflow of the whole system. We start the discussion with Nash's Social Welfare Function (NSWF), and the reason behind using NSWF in the CBM before going into details of the system design.

A. Nash's Social Welfare Function (NSWF)

Conflict resolution in CAN is similar to a Multi-Agent Resource Allocation (MARA) scenario in MAS [5] where the CFs act as the agents. A MARA scenario is defined as a triple $< \mathcal{A}, \mathcal{R}, \mathcal{U} >$. $\mathcal{A}$ is a finite set of n agents $\mathcal{A} = \{1, 2, .., n\}$. $\mathcal{R}$ is a finite set of m resources $\mathcal{R} = \{r_1, r_2, .., r_m\}$. $\mathcal{U}$ is a set of utility functions $\mathcal{U} = \{u_1, u_2, .., u_n\}$ one for each agent. Each $u_i \in \mathcal{U}$ is a mapping from set of resources to the objective of each agent. Utilities are often expressed in a same and predetermined scale to make comparison among them easier. A *utility profile* for a particular resource allocation combination J is a vector which contains utility of all agents for this particular allocation J and is defined by $u(J) = (u_1(J), u_2(J), .., u_n(J))$. A *Collective Utility Function* (CUF) is a function which operates on all the utility profiles and maps $u(J)$ to some real number. Most social welfare functions can be defined as a CUF [5].

The Nash CUF, or the Nash Social Welfare Function (NSWF), is defined as the product of the individual agent utilities for a particular resource allocation J:

$$NSWF(J) = \prod_{i \in \mathcal{A}} u_i(J) = u_1(J) \cdot u_2(J) \cdot u_3(J) ... u_n(J) \quad (1)$$

The reason behind using NSWF in a resource allocation problem is the solution provided by NSWF balances efficiency and fairness ([5], [6]), and it is sensitive to change in overall welfare. The fairness in allocation can be observed from the lowest difference in utilities obtained. For example, NSWF prefers (50,50) to (99,1) and (24,76) as (50,50) provides best and equal fairness to all. In the proposed CAN model, objective of each CF has equal weight in the system, that is why we use NSWF to obtain a solution which provides equal weight to each output and is optimal for their collective interest. How the proposed mechanism changes when different objectives have different weights in CAN is a part of our future research.

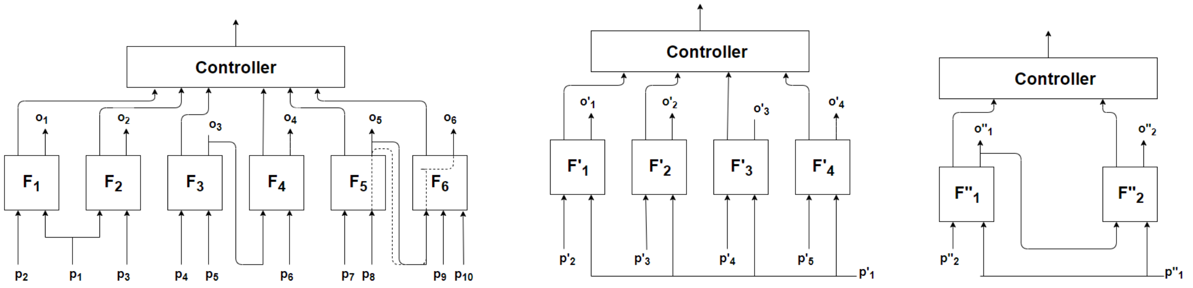
Now, in the CAN Model 1 described in section III, $\mathcal{A} = \{F_1, F_2, F_3, F_4, F_5, F_6\}$, $\mathcal{R} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ and $\mathcal{U} = \{u_1, u_2, u_3, u_4, u_5, u_6\}$. For a particular resource, usually multiple resource allocation combination (J) values are available to the controller. These J values are the aggregation of all the suggested values by all the CFs who share that particular resource (generation of J values are described in the next section). For example, as F_1 and F_2 share the same resource p_1, both of them suggest multiple J values to the controller. The functionality of the controller is to find the particular J_p, from the set of all J values, for which $\prod_{i=1}^{2} u_i(J)$ is maximum.

B. Workflow of CF

Each CF is an independent learning agent which makes decisions on its own based on its learning history. For a certain network state, the CF is able to generate its favorable set of values for a particular configuration, which is defined as *optimal-configuration-range set*. The values of optimal-configuration-range sets, proposed by different CFs, serve as the J values (mentioned in the last section) from which the optimal configuration (J_p) is to be found. When value of the parameter lies within this set, the output of the CF always lies within a certain percentage of its value. We denote this percentage as *cf-return-size* and this value can be set by the operator manually. An optimal-configuration-range set has the following structure - $[min_{config}, max_{config}]_{\mathrm{p}}$, where min_{config} denotes the minimum and max_{config} denotes the maximum value of the optimal-configuration-range set for p.

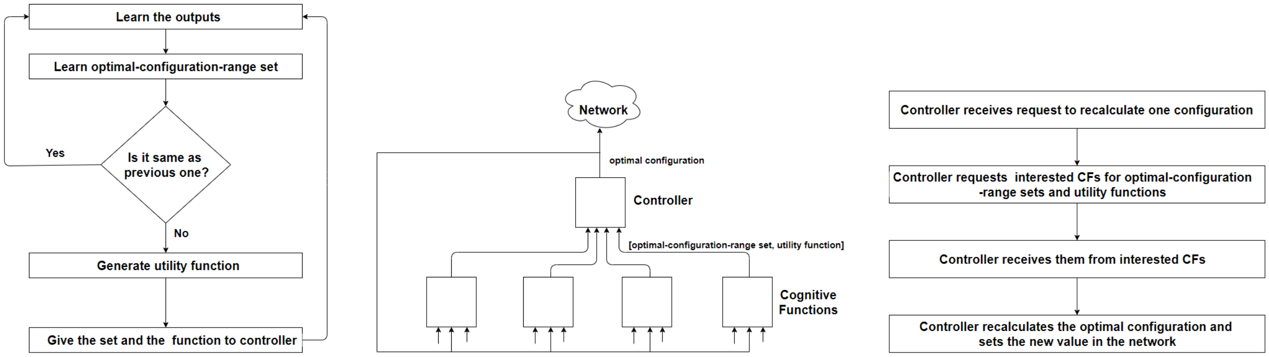
As each CF is a learning agent, it continually observes the output and studies it and based on its learning, after every pre-defined time interval, it calculates the optimal-configuration-range set. Thus, whenever the CF is asked to provide its optimal-configuration-range set, it is ready with the latest one. Every time the CF generates the optimal-configuration-range set, the CF checks if it is the same as the last one. If they are the same, the CF takes no action and continues with its normal workflow. If not, the CF sends a request to the controller to recalculate the configuration. This entire workflow of a CF is depicted in Fig. 3a.

Sending a request to the controller is a two-step process - i) CF generates the utility function, and, ii) send both the latest optimal-configuration-range set and the utility function to the controller (as can be seen from Fig. 3a). As different CFs have different objectives with different dimensions, it is convenient to convert them all into a certain pre-defined scale to make decision making by the controller easier. To do so, each CF maps the values of its objective (or, output) to a common utility scale defined a priori either by the network operator or provided by the controller. An example scale may be [0:10], where 0



(a) CAN model 1 (b) CAN model 2 (c) CAN model 3

Fig. 2: Different CAN models



Learn the outputs
Learn optimal-configuration-range set
Yes
Is it same as previous one?
No
Generate utility function
Give the set and the function to controller
Network
optimal configuration
Controller
[optimal-configuration-range set, utility function]
Cognitive Functions
Controller receives request to recalculate one configuration
Controller requests interested CFs for optimal-configuration -range sets and utility functions
Controller receives them from interested CFs
Controller recalculates the optimal configuration and sets the new value in the network

(a) Workflow of a CF (b) Input and output of proposed controller (c) Workflow of the controller

Fig. 3: Workflow and IO of CF and controller

indicates the lowest value and 10 indicates the highest possible value of its objective. The mapping function, which maps each output value between 0 and 10, is called *utility function*. A utility function is denoted by $f(p)$, where p is the configuration value and corresponding to a particular p, $f(p)$ provides the utility value in the predefined scale. Unlike optimal-configuration-range set, a utility function is not calculated after every time interval, it is recalculated only when optimal-configuration-range set changes. The CF delivers both optimal-configuration-range set and utility function to the controller in the following structure: $\{[min_{config},\, max_{config}]_{\mathrm{p}},\, f(p)\}$.

C. Workflow of Controller

The controller takes optimal-configuration-range sets and utility functions from CFs as its input and returns a single optimal configuration as its output (as shown in Fig. 3b). Workflow of the proposed controller consists of four steps as depicted in Fig. 3c. For several reasons (environmental change, change of other network configuration parameter), objective (output) value of a CF can change and with it optimal-configuration-range set for a particular parameter may change. Whenever a CF wants to change the configuration, it sends a request to the controller. Upon receiving the new request, the controller sends requests to interested CFs (all the CFs who share the same parameter or use the same configuration), asking them to send their corresponding individual optimal-configuration-range sets and utility functions. After receiving information from all interested CFs, the controller calculates the optimal configuration using NSWF and makes the necessary changes in the network. The detailed steps regarding the optimal configuration calculation are described in the next section.

D. Calculation of optimal configuration

Main purpose of the controller is to calculate an optimal configuration for the combined interest of all CFs from a set of proposed configurations. By doing so, the controller also resolves all existing conflicts among the CFs.

After receiving optimal-configuration-range sets from multiple CFs, the controller combines them into a single *combined-configuration-range set* by taking the lowest and highest values from all the optimal-configuration-range sets. As an example, let us assume that there are three CFs F_1, F_2 and F_3 interested in a certain configuration p and they send - $\{[p_1^{min,F1},\, p_1^{max,F1}]_{\mathrm{p}},\, f_1(p)\}$, $\{[p_1^{min,F2},\, p_1^{max,F2}]_{\mathrm{p}},\, f_2(p)\}$ and $\{[p_1^{min,F3},\, p_1^{max,F3}]_{\mathrm{p}},\, f_3(p)\}$ to the controller respectively. The controller takes the minimum value from $\{p_1^{min,F1},\, p_1^{min,F2},\, p_1^{min,F3}\}$, which is denoted by min_p, and takes the maximum value from $\{p_1^{max,F1},\, p_1^{max,F2},\, p_1^{max,F3}\}$, which is denoted by max_p, and generates the combined-configuration-range set which has the following structure - $[min_p,\, max_p]$.

In the next step, the controller samples as many values as possible between min_p and max_p. We denote the number of samples by *controller-sampling-size*. Following the NSWF, corresponding to each sample value, the controller calculates the product of the utilities of all three functions and selects the highest product. For example, if s_1, s_2 and s_3 are three sampled values, then the controller computes the product - $f_1(s) \cdot f_2(s) \cdot f_3(s)$ for each of them. If for s_2, the product $f_1(s) \cdot f_2(s) \cdot f_3(s)$ is maximum, then s_2 is selected as the optimal configuration.

Sometimes it may happen that instead of a single one, we get multiple configuration values as the optimal one. Under those circumstances, that sampled value is selected for which the outputs of the utility functions are closer to one another. For example, if we assume that s_1 and s_2 are two sampled values and also assume that $f_1(s_1) = 8$, $f_2(s_1) = 3$, $f_3(s_1) = 1$, and, $f_1(s_2) = 4$, $f_2(s_2) = 3$, $f_3(s_2) = 2$, we see that the product of the utilities are same in both case. However, standard deviation of the outputs for s_1 is 2.943 and for s_2 is 0.816. Thus, for s_2 the utility values are closer to one another than for s_1, and so, s_2 is selected. Standard deviation gives a measure of closeness of the values. The lower the standard deviation value, the closer are the values to one another.

E. *Special advantages of using CBM in controller*

A CBM enabled controller has two important characteristics which are important for managing CFs - i) Tailing coordination, and, ii) Guaranteed minimum service requirement.

1) Tailing coordination: To prevent undesired conflicts among network automation functions, two types of protection can be taken [2] - i) *active protection*, which resolves the conflicts in the run time, and, ii) *pro-active protection*, which prevents future conflicts. Heading coordination implements pro-active protection whereas Tailing coordination incorporates the active protection. Between these two, Tailing coordination is the more important as it has the ability to overcome any problem that exists in Heading coordination and it can change the incoming requests in order to harmonize requests from multiple automation functions concerning an identical target [2]. A CBM enabled controller provides Tailing coordination providing an active protection to the CFs and most beneficial to use in CAN.

2) Guaranteed minimum service requirement: Using a CBM enabled controller, the Mobile Network Operator (MNO) can guarantee that during and after conflict resolution, QoS will always remain above some predefined threshold value. Let us elaborate this with an example of CAN Model 1 (2a). Let us assume that, when the controller is resolving conflicts between F_1 and F_2 over p_1, to maintain the minimum QoS, user satisfaction or for some other reasons, o_1, o_2 should be always kept at or above c_1 and c_2 respectively, with $c_1, c_2 \in \mathbb{R}^+$. In this case, for a sample configuration s, the controller calculates the product $|f_1(s) - c_1| \cdot |f_2(s) - c_2|$ and chooses that configuration for which this product is maximum. If the maximum product is found to be zero, then it means that no compromise solution can be found with the constraints $o_1 \geq c_1$, $o_2 \geq c_2$. In that case, the MNO reevaluates the c_1 and c_2, and the controller recalculates the product with new constraint values.

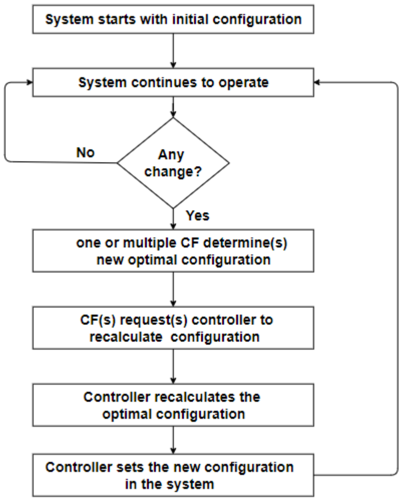

System starts with initial configuration
System continues to operate
No
Any change?
Yes
one or multiple CF determine(s) new optimal configuration
CF(s) request(s) controller to recalculate configuration
Controller recalculates the optimal configuration
Controller sets the new configuration in the system

Fig. 4: CAN System workflow

TABLE II: Dummy dataset structure

Row index	p_1	p_2	o_1
1	p_1^1	p_2^1	o_1^1
2	p_1^2	p_2^2	o_1^2
.	. .	. .	. .
training-size	$p_1^{training\text{-}size}$	$p_2^{training\text{-}size}$	$o_1^{training\text{-}size}$

F. *System workflow*

We assume that CAN initially starts with some preloaded configuration. After the system becomes operational, the CFs also start operating following its workflow (shown in Fig. 3a). After one or multiple CFs determine a new-optimal-configuration set for one configuration (or, parameter), those CFs send a request to the controller to recalculate the configuration. Upon receiving the request, the controller recalculates the configuration (as described in Sec IV-C and Sec IV-D) and the newly calculated configuration value is propagated to the CFs and set in the network by the controller. Every time a CF detects a new optimal-configuration set for one parameter, the whole process is repeated. This entire workflow is also depicted in Fig. 4.

V. SIMULATION & NUMERICAL ANALYSIS

A. *Example model*

To analyze the performance of the controller quantitatively, let us revisit the CAN models described in Section III. During numerical analysis, we assume that - $o_i \forall i \in \{1, 6\}, o'_j \forall j \in \{1, 4\}$ and $o''_k \forall k \in \{1, 2\}$, are all Gaussian distribution functions. The reason behind this assumption is that in a real life scenario, mobile network parameters resemble this distribution very often. For example, from [7] we see that both SNR and Latency on a loaded cellular network follow Gaussian distribution.

The outputs of the CFs are given by:

$$o_1 = e^{-\frac{(p_1+50)^2}{2p_2^2}} \quad (2)$$

$$o_3 = e^{-\frac{(p_4+60)^2}{2p_5^2}} \quad (3)$$

$$o_5 = e^{-\frac{(p_7+70)^2}{2p_8^2}} \quad (4)$$

$$o_1' = e^{-\frac{(p_1'+100)^2}{2p_2'^2}} \quad (5)$$

$$o_2' = e^{-\frac{(p_1'+50)^2}{2p_3'^2}} \quad (6)$$

$$o_1'' = e^{-\frac{(p_1''+50)^2}{2p_2''^2}} \quad (7)$$

$$o_2 = e^{-\frac{(p_1-50)^2}{2p_3^2}} \quad (8)$$

$$o_4 = e^{-\frac{(p_6-60)^2}{\frac{2}{o_3^2}}} \quad (9)$$

$$o_6 = e^{-\frac{(p_9-o_5)^2}{2p_{10}^2}} \quad (10)$$

$$o_3' = e^{-\frac{(p_1'-50)^2}{2p_4'^2}} \quad (11)$$

$$o_4' = e^{-\frac{(p_1'-100)^2}{2p_5'^2}} \quad (12)$$

$$o_2'' = e^{-\frac{(p_1''-50)^2}{\frac{2}{o_1''^2}}} \quad (13)$$

These equations have been formulated in such a way that between a pair of CFs, the conflict(s) as mentioned in Section III hold true. For example, when p_1 is -50, o_1 is maximum and when p_1 is 50, o_2 is maximum, and thus, F_1 and F_2 have a conflict over p_1. In the second case, o_4 does not have any direct dependency on p_5. However, increasing p_5 increases o_3 which decreases o_4, and thus, F_3 and F_4 have a measurement conflict over p_5. Lastly, o_6 is dependent on o_5 and o_5 is dependent on p_7 (and p_8), thus, F_6 has a logical dependency conflict with F_5 over p_7 (and p_8). Using similar analysis, input parameter conflict can be observed among (o_1', o_2', o_3', o_4') and all three types of conflicts can be observed between (o_1'', o_2''). Moreover, for better analysis and graphic visualization, we model all conflicts as parameterized conflicts without any loss of generality.

B. Analysis Setup

In reality, the underlying relationship between the outputs and network states (or, configurations) is not known beforehand. As mentioned previously, main functionality of a CF is to learn the variation of its output when the network state or configuration changes, so that it can predict its output and generate the utility function for any particular network state or configuration. To make the numerical analysis as close to reality as possible, we create dummy datasets, using which the CFs can learn the variation of output w.r.t. input parameters. For each individual CF F_i, we create a dummy dataset D_i following equation o_i. Next, we create a Machine Learning Model (MLM), and train the MLM using the dummy dataset. After the training is complete, the MLM, which now acts as a CF, is able to predict its output corresponding to a particular input configuration.

1) Dummy Dataset Generation: To create a dummy dataset(e.g., D_1 for F_1) we take a random combination of the input parameters (p_1, p_2), calculate the output o_1 (using Eq. 2), and store them in a table like Table II.

We select the value of p_1 and p_2 randomly from their predefined range (which can be found from Table III) and make sure they are unique. The total number of instances of (p_1, p_2), which is used for generating D_1, is denoted by *training-size.*

TABLE III: Parameters and their default values

Parameter	**Value**	**Parameter**	**Value**
p_1	[-150, 150]	p_2	20
p_3	60	p_4	0
p_5	[-50, 100]	p_6	100
p_7	[-150, 80]	p_8	80
p_9	40	p_{10}	60
p_1'	[-200, 200]	p_2'	35
p_3'	60	p_4'	85
p_5'	150	p_1''	[-100, 100]
p_2''	40		

2) ML Model Training: To model the CFs in Python, we use Polynomial Regression block of order 5 using Python package sklearn. The code to implement a CF in Python can be found in [8]. Each individual F_i is trained using dummy dataset D_i, so that, F_i can predict its output o_i for any combination of its input parameters.

3) Parameter Values: Relevant parameters and their values used in the numerical analysis are:

1. *cf-return-size*: it is already defined in Section IV-B. Default value is 25%, unless stated otherwise.

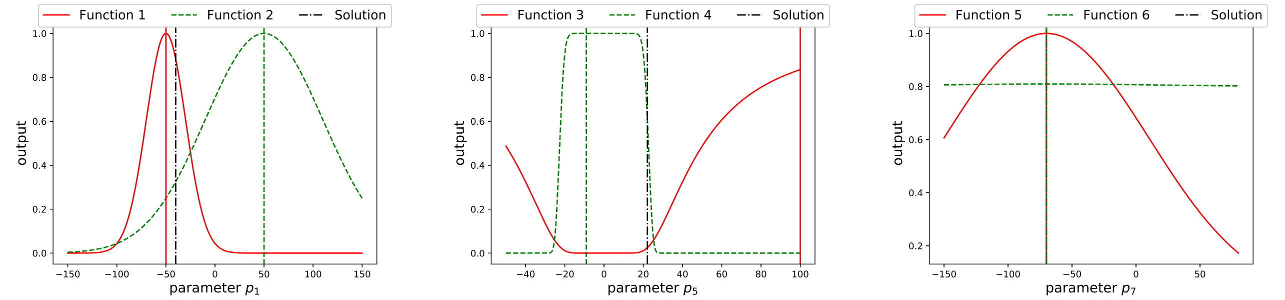
2. *controller-sampling-size*: it is already defined in Section IV-D. Default value is 2000 unless stated otherwise.

3. *training-size*: it has already been defined in Section V-B1. Unless stated otherwise, default value of this variable used in the analysis is 2000.

4. p_i, p_i', p_i'': The input parameters of the system are chosen in a way such that the outputs are distinct from one another and the impact of the proposed solution is clearly visible. The parameters with their default values or range of values are described in Table III.

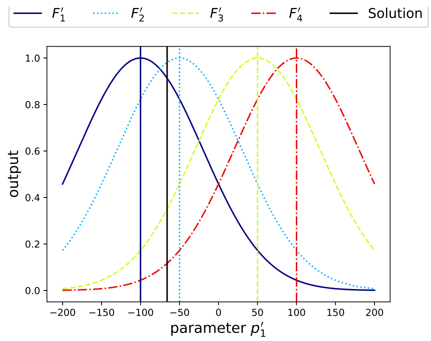
C. Numerical analysis of the controller performance

1) CAN Model 1: Let us start the analysis with the variations of the outputs (o_i) w.r.t. the parameter of conflict. For example, F_1 and F_2 have a conflict over parameter p_1, and in Fig. 5a we show the variations of o_1 and o_2 w.r.t. p_1. The black vertical line shows the output values corresponding to the calculated solution and each other vertical line shows the maximum output of the function with the same color. For example, in Fig. 5a, the red vertical line shows the optimal configuration for F_1, the green vertical line shows the optimal configuration for F_2, and the black vertical line corresponds to the calculated optimal configuration which lies between the other two vertical lines. From Fig. 5a we see that when the value of p_1 is as depicted by the black line, neither o_1 nor o_2 have their maximum values, but it is a good balance between these two outputs. In the same way we plot F_3 and F_4 in Fig. 5b and F_5 and F_6 in Fig. 5c. However, from Fig. 5c we see that, although F_5 and F_6 have a logical dependency conflict over p_7, o_6 remains almost constant w.r.t. p_7, i.e., maximum of o_6 is independent of p_7. In such cases, the ideal configuration is the point where the other function has its maximum output. Now from Fig. 5c we can see that

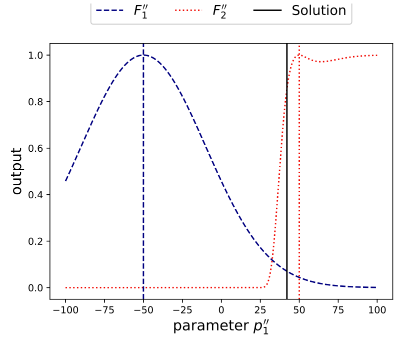


(a) Conflict resolution between F_1 and F_2 (b) Conflict resolution between F_3 and F_4 (c) Conflict resolution between F_5 and F_6

Fig. 5: Numerical analysis of CAN Model 1



(a) Numerical analysis of CAN Model 2



(b) Numerical analysis of CAN Model 3

Fig. 6: Numerical analysis of CAN Model 2 and 3

the proposed configuration in this case coincides with the point where o_5 has its maximum, proving that proposed CBM always provides the best configuration.

2) CAN Model 2: Using this model we show that the proposed controller can resolve conflict among any number of CFs. We consider a CAN model with 4 CFs - F_1', F_2', F_3', F_4', all of which share the same input parameter p_1 and have a conflict over it. To resolve this conflict, we use the same controller and plot the result in Fig. 6a to show the optimal configuration for combined interest of all four CFs. Although in this model we consider only 4 CFs, the solution idea can be extended and used for any finite number of CFs.

3) CAN Model 3: To show that the proposed controller can resolve any number of simultaneously existing conflicts among CFs, we consider CAN model 3 with 2 CFs - F_1'', F_2'' as shown in Fig. 2c. All three types of conflicts exist simultaneously between F_1'' and F_2'' - they have input parameter conflict (over p_1''), measurement conflict (as action of F_1'' influences measurement of o_2'') and logical dependency conflict ($o_2'' \rightarrow o_1'' \rightarrow p_2''$). To resolve all these conflicts simultaneously, we use the same controller and plot the result in Fig. 6b to show the optimal configuration. It proves that the proposed CBM enabled controller is able to resolve any number of conflicts which may exist simultaneously.

VI. RELATED WORKS

A. Related works in MAS

In this paper CAN has been abstracted as a MAS and the proposed controller works on removal of conflicts in the MAS. So, in this section we study already existing research works on MAS ([14], [22]) and removal of conflicts (or, reaching consensus) in a MAS ([18], [23]).

In the MAS model used in this paper, described in Section II, we already highlighted that each agent (CF) should have four properties. Based on agent characteristics, we divide existing research works on MAS into several categories so that a combination of these features are covered in each category. These categories are listed in Table IV. From Table IV we see that there are a number of prior research articles which encompass one or some combinations of those four features described above, but there does not exist any paper which covers all the four features (as shown in Table IV). Ours is the first one which considers a MAS with all of these four properties and proposes a solution for conflict removal (or, reaching consensus) in such a MAS.

B. Related works on controller

As mentioned earlier, the idea of a controller for network automation functions is not new and already exists in SON [3], [24], [25]. In these papers some external controllers have been proposed which work on top of the SON Functions and these controllers also coordinate and remove conflicts among SON Functions. SON coordination has extensively been researched in SOCRATES (2008) [26] and SOCRATES (2011) [27] also. But these coordination mechanisms or controllers cannot be used in CAN because these are rule based controllers which follow predefined

TABLE IV: Existing works on MAS features

P1	✓				✓	✓	✓				✓	✓	✓		✓
P2		✓			✓			✓	✓		✓	✓		✓	✓
P3			✓			✓		✓		✓	✓		✓	✓	✓
P4				✓			✓		✓	✓		✓	✓	✓	✓
	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	**X**

rules and rule-based coordination does not work in CAN. To remove conflicts among CFs in CAN, we need some new coordination mechanism which is dynamic and generic, like the one proposed in this paper.

VII. Conclusion and Future Direction

In this paper we provide design of a CBM enabled controller which coordinates among CFs in CAN and dynamically resolves - any type of conflict among CFs, any number of simultaneously existing conflicts among CFs, and, conflict among any number of CFs. The proposed mechanism is generic,i.e., to resolve conflicts in all possible cases a single mechanism is used, and it has the ability to resolve the conflict reactively, i.e., whenever the conflict arises. The proposed mechanism calculates configuration for the system using NSWF, so that the calculated configuration is optimal for the collective interest of the whole system. To prove the validity of the proposed model, we implement three separate CAN models in Python and perform numerical analysis on the performance of the controller. As a future direction from this work, we plan to implement some real life network automation functions (like, CCO, HO) from simulator generated dataset in 5G scenario and explore the implementation and performance of the proposed mechanism.

References

[1] S. S. Mwanje and C. Mannweiler, "Towards cognitive autonomous networks in 5g," in *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*, pp. 1–8, IEEE, 2018.

[2] S. Hämäläinen, H. Sanneck, and C. Sartori, *LTE self-organising networks (SON): network management automation for operational efficiency*. John Wiley & Sons, 2012.

[3] D. Kominami, M. Sugano, M. Murata, and T. Hatauchi, "Controlled and self-organized routingfor large-scale wireless sensor networks," *ACM Transactions on Sensor Netw. 10*, vol. 1, no. 13, pp. 1–27, 2013.

[4] J. F. Nash Jr, "The bargaining problem," *Econometrica: Journal of the Econometric Society*, pp. 155–162, 1950.

[5] S. Ramezani and U. Endriss, "Nash social welfare in multiagent resource allocation," in *Agent-mediated electronic commerce. Designing trading strategies and mechanisms for electronic markets*, pp. 117–131, Springer, 2009.

[6] E. Van Damme, "The nash bargaining solutions is optimal," *Journal of Economic Theory*, vol. 38, no. 78, p. 100, 1986.

[7] I. Marsh, B. Grönvall, and F. Hammer, "The design and implementation of a quality-based handover trigger," in *International Conference on Research in Networking*, Springer, 2006.

[8] https://github.com/AnubhabBanerjee/CF_in_CAN.

[9] S. Sarika and V. Paul, "Agenttab: An agent based approach to detect tabnabbing attack," *Procedia Computer Science*, vol. 46, pp. 574–581, 2015.

[10] M. Khayyat and A. Awasthi, "An intelligent multi-agent based model for collaborative logistics systems," *Transp. Res. Procedia*, vol. 12, pp. 325–338, 2016.

[11] F. Rahimzadeh, L. Khanli, and F. Mahan, "High reliable and efficient task allocation in networked multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 29, no. 6, pp. 1023–1040, 2015.

[12] A. Banerjee, N. Sastry, and C. Mas Machuca, "Sharing content at the edge of the network using game theoretic centrality," in *International Conference on Transparent Optical Networks ICTON*, 2019.

[13] J. Mano and P. Glize, "Self-adaptive network of cooperative neuro-agents," in *AISB'04 Symposium on Adaptive Multi-Agent Systems*, 2004.

[14] M. Gatti, P. Cavalin, S. Neto, C. Pinhanez, C. dos Santos, D. Gribel, and A. Appel, "Large-scale multi-agent-based modeling and simulation of microblogging-based online social network," in *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pp. 17–33, Springer, 2013.

[15] S. Manvi and M. Kakkasageri, "Multicast routing in mobile ad hoc networks by using a multiagent system," *Information Sciences*, vol. 178, no. 6, pp. 1611–1628, 2008.

[16] T. Morstyn, B. Hredzak, and V. Agelidis, "Cooperative multi-agent control of heterogeneous storage devices distributed in a dc microgrid," *IEEE Transactions on Power Systems*, vol. 31, no. 4, pp. 2974–2986, 2015.

[17] R. Bianchi, M. Martins, C. Ribeiro, and A. Costa, "Heuristically-accelerated multiagent reinforcement learning," *IEEE transactions on cybernetics*, vol. 44, no. 2, 2013.

[18] S. Resmerita and M. Heymann, "Conflict resolution in multi-agent systems," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 3, pp. 2537–2542, IEEE, 2003.

[19] E. Semsar-Kazerooni and K. Khorasani, "A game theory approach to multi-agent team cooperation," in *2009 American Control Conference*, pp. 4512–4518, IEEE, 2009.

[20] J. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*, pp. 66–83, Springer, 2017.

[21] H. Liu, P. Zhang, B. Hu, and P. Moore, "A novel approach to task assignment in a cooperative multi-agent design system," *Applied Intelligence*, vol. 43, no. 1, pp. 162–175, 2015.

[22] Q. Liu, X. Cui, and X. Hu, "Conflict resolution within multi-agent system in collaborative design," in *International Conference on Computer Science and Software Engineering*, IEEE, 2008.

[23] M. Genesereth, M. Ginsberg, and J. Rosenschein, "Cooperation without communication," in *Readings in distributed artificial Intelligence*, pp. 220–226, Elsevier, 1988.

[24] M. Lamba and M. Dagar, "Review on self organizing networks (son) in lte-advanced hetnets,"

[25] T. Bandh, H. Sanneck, and R. Romeikat, "An experimental system for son function coordination," in *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pp. 1–2, IEEE.

[26] S. Deliverable, "D2. 1: Use cases for self-organising networks," *EU STREP SOCRATES (INFSO-ICT-216284), Version*, vol. 1, 2008.

[27] F. S. INFSO-ICT216284, "Final report on self-organisation and its implications in wireless access networks," *D5*, vol. 9, p. v1, 2010.