Adaptive and Scalable Controller Placement in Software-Defined Networking

Adekoya Oladipupo Adewale
Birmingham City University

A DISSERTATION SUBMITTED TO THE SCHOOL OF COMPUTING AND DIGITAL TECHNOLOGY IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF DOCTOR OF PHILOSOPHY

Doctor of Philosophy

MAY 2023

Dedication

With profound humility, I dedicate this thesis to my exceptional wife, Moninuola Adekoya. Her unwavering dedication, steadfast support, and tireless efforts throughout my doctoral journey have been nothing short of remarkable. My gratitude towards her knows no limits. Equally, my heartfelt appreciation extends to Peter, Banjo, Kayode, and my cherished network of supporters. Your combined encouragement has been an invaluable source of strength, and I am profoundly thankful for the exceptional contributions you have made.

Acknowledgements

"Glory be to God.

Above all else, I wish to extend my heartfelt gratitude to the Almighty God, who holds ultimate responsibility for my present existence. Through His boundless grace, I embarked upon and successfully completed this thesis. Throughout this journey, my rock and unwavering support have been Jesus Christ, my Lord, and Redeemer.

To Prof. Adel Aneiba, I struggle to convey the depth of my appreciation for your presence as my supervisor. Your steadfast support and unwavering trust over the years have left me profoundly grateful. Serving under your guidance has been both an honor and a pleasure. May God richly bless your endeavors. Equally deserving is my esteemed professor, Mak Sharma. I am thankful for your enthusiasm, patience, and willingness to share your extensive management and administrative expertise. Your dedication to mentoring young individuals has been a true inspiration to me.

Special gratitude extends to Professor Gaber for his insightful contributions during the intelligence phase of my research. Your persistence, dedication, and guidance have significantly contributed to my journey thus far. I offer my heartfelt thanks.

Eternal gratitude is reserved for Peter, Banjo, and Kayode for their unwavering support and heartfelt care that sustained me. The support and prayers of my in-laws have been a true source of strength. My parents, I thank you for your prayers, guidance, and unwavering support throughout this remarkable journey. I also wish to extend my appreciation to all those who have contributed to my success during my pursuit of a PhD.

Lastly, my incredible wife, Moninuola, deserves my deepest gratitude. Your ceaseless words of encouragement and unwavering support have been my anchor. I hold you in the highest regard."

Abstract

Software-defined networking (SDN) revolutionizes network control by externalizing and centralizing the control plane. A critical aspect of SDN is Controller Placement (CP), which involves identifying the ideal number and location of controllers in a network to fulfill diverse objectives such as latency constraints (node-to-controller and controller-controller delay), fault tolerance, and controller load. Existing optimization techniques like Multi-Objective Particle Swarm Optimisation (MOPSO), Adapted Non-Dominating Sorting Genetic Algorithm-III (ANSGA-III), and Non-Dominating Sorting Genetic Algorithm-II (NSGA-II) struggle with scalability (except ANSGA-III), computational complexity, and inability to predict the required number of controllers. This thesis proposes two novel approaches to address these challenges. First, an enhanced version of NSGA-III with a repair operator-based approach (referred to as ANSGA-III) is introduced, enabling efficient CP in SD-WAN by optimizing multiple conflicting objectives simultaneously. Second, a Stochastic Computational Graph Model with Ensemble Learning (SCGMEL) is developed, overcoming scalability and computational inefficiency associated with existing methods. SCGMEL employs stochastic gradient descent with momentum, a learning rate decay, a computational graph model, a weighted sum approach, and the XGBoost algorithm for optimization and machine learning. The XGBoost predicts the number of controllers needed and a supervised classification algorithm called Learning Vector Quantization (LVQ) is used to predict the optimal locations of controllers. Additionally, this research introduces the Improved Switch Migration Decision Algorithm (ISMDA) as part of the holistic contribution. ISMDA is implemented on each controller to ensure even load distribution throughout the controllers. It functions as a plug-and-play module, periodically checking if the load surpasses a certain limit. ISMDA improves controller throughput by approximately 7.4% over CAMD and roughly 1.1% over DALB. ISMDA also outperforms DALB and CAMD with a decrease of 5.7% and 1%, respectively, in terms of controller response time. Additionally, ISMDA outperforms DALB and CAMD with a decrease of 1.7% and 5.6%, respectively, in terms of the average frequency of migrations. The established framework results in fewer switch migrations during controller load imbalance. Finally, ISMDA proves more efficient than DALB and CAMD, with an estimated 1% and 6.4% lower average packet loss, respectively. This efficiency is a result of the proposed migration efficiency strategy, allowing ISMDA to handle higher loads and reject fewer packets.

Real-world experiments were conducted using the Internet Zoo topology dataset to evaluate the proposed solutions. Six objective functions, including worst-case switch-to-controller delay, load balancing, reliability, average controller-to-controller latency, maximum controller-to-controller delay, and average switch-to-controller delay, were utilized for performance evaluation. Results demonstrated that ANSGA-III outperforms existing algorithms in terms of

hypervolume indicator, execution time, convergence, diversity, and scalability. SCGMEL exhibited exceptional computational efficiency, surpassing ANSGA-III, NSGA-II, and MOPSO by 99.983%, 99.985%, and 99.446% respectively. The XGBoost regression model performed significantly better in predicting the number of controllers with a mean absolute error of 1.855751 compared to 3.829268, 3.729883, and 1.883536 for KNN, linear regression, and random forest, respectively. The proposed LVQ-based classification method achieved a test accuracy of 84% and accurately predicted six of the seven controller locations.

To culminate, this study presents a refined and intelligent framework designed to optimize Controller Placement (CP) within the context of SD-WAN. The proposed solutions effectively tackle the shortcomings associated with existing algorithms, addressing challenges of scalability, intelligence (including the prediction of optimal controller numbers), and computational efficiency in the pursuit of simultaneous optimization of multiple conflicting objectives. The outcomes underscore the supremacy of the suggested methodologies and underscore their potential transformative influence on SDN deployments. Notably, the findings validate the efficacy of the proposed strategies, ANSGA-III and SCGMEL, in enhancing the optimization of controller placement within SD-WAN setups. The integration of the XGBoost regression model and LVQ-based classification technique yields precise predictions for both optimal controller quantities and their respective positions. Additionally, the ISMDA algorithm emerges as a pivotal enhancement, enhancing controller throughput, mitigating packet losses, and reducing switch migration frequency—collectively contributing to elevated standards in SDN deployments.

Publications

In this research, two journal articles had already been accepted and published, and one is under review.

Adekoya, O., Aneiba, A. and Patwary, M., 2020. An improved switch migration decision algorithm for SDN load balancing. IEEE Open Journal of the Communications Society, 1, (pp.1602-1613).

Adekoya, O. and Aneiba, A., 2022. An Adapted Nondominated Sorting Genetic Algorithm III (NSGA-III) With Repair-Based Operator for Solving Controller Placement Problem in Software-Defined Wide Area Networks. IEEE Open Journal of the Communications Society, 3, pp.(888-901).

Adekoya, O. and Aneiba, A., 2023. A stochastic computational graph with an ensemble learning model for solving the controller placement problem in software-defined wide area networks (under review).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software-defined networking (SDN) has emerged as a promising paradigm for constructing networks tailored to meet the evolving demands of the next generation of network technologies and services [1]. This innovative approach represents a substantial departure from traditional networking methods, marking a foundational shift within communication networks. This transformation revolves around the development of a logically centralized architecture that effectively separates the control and data planes. To elaborate, the data plane comprises simplified packet forwarding switches, while the control plane consists of specialized software-based controllers that serve as the intelligent core of the system. This segregation brings forth enhanced efficiency in terms of reconfigurability and programmability, resulting in several advantages, including streamlined network administration, improved network performance, and the facilitation of novel network advancements [1][2][3].

The predominant communication interface for SDN is OpenFlow [4], a widely adopted software communication protocol. This is accompanied by Netconf (Network Configuration Protocol) and Restconf (Restful Network Configuration Protocol), which serve as connectors between the control and data planes, facilitating connectivity via the southbound application programming interface (API) [5]. While the OpenFlow protocol initially assumes a single controller, scalability and performance issues can arise as networks expand. To address these concerns, multi-controller systems have been developed, exemplified by concepts like HyperFlow [6]. Such systems partition OpenFlow networks into distinct segments, each governed by an independent controller. Collaborative efforts in design have led to the establishment of this multi-controller framework.

The core architecture of SDN comprises three layers: the data plane, the control plane, and the application plane, as illustrated in Figure 1.1. The data plane encompasses packet-forwarding switches managed by control planes, or "controllers." These controllers, utilizing southbound APIs such as Netconf, Restconf, or OpenFlow, establish connectivity. Through northbound application programming interfaces like Representational State Transfer (REST API), the controllers interface with the application plane, facilitating network control and the provisioning of network services. As indicated by [2], a pivotal

hurdle arising from the deployment of multiple controllers lies in the strategic placement of these entities. This complex challenge, often denoted as the "controller placement challenge," involves making critical determinations regarding the optimal locations for deploying controllers and the appropriate quantity of controllers to allocate within a software-defined wide area network (SD-WAN). This intricate deliberation is geared toward fulfilling a spectrum of objectives encompassing various dimensions. **These**

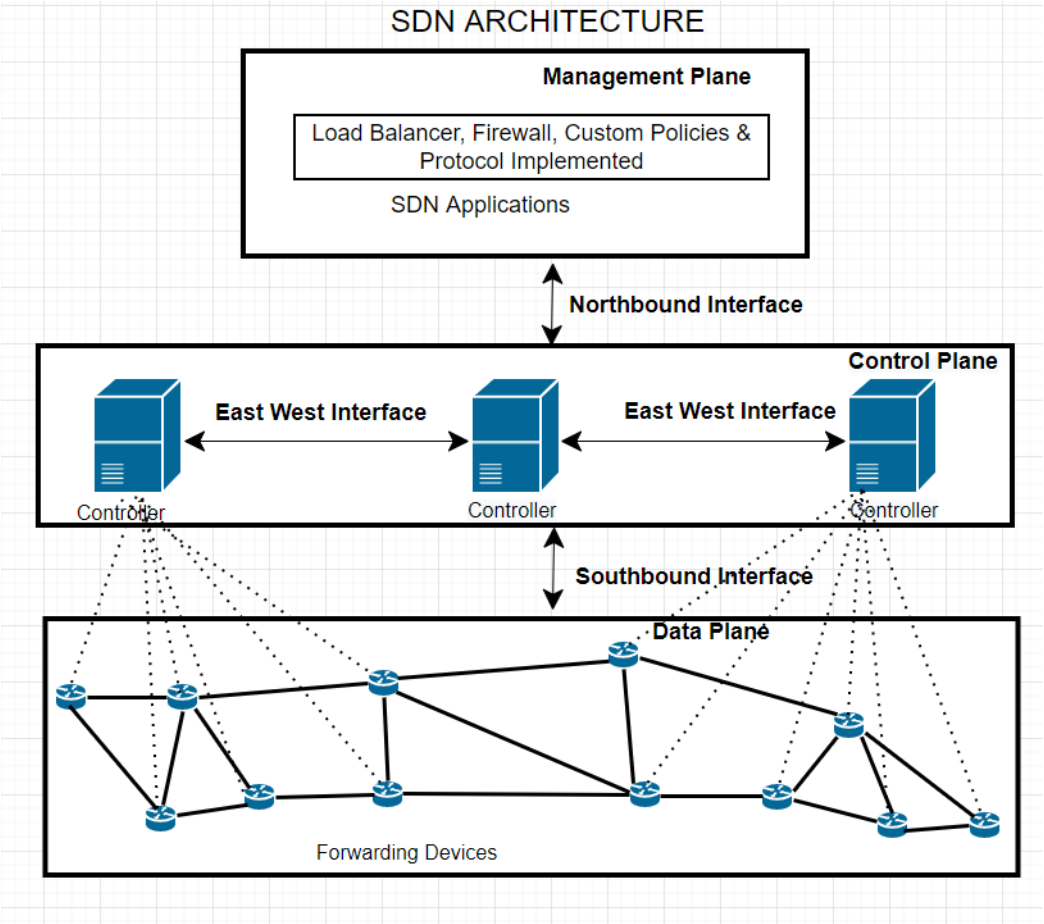

Figure 1.1: A simplified architecture for SDN layers

objectives encompass a wide range of goals, including reducing latency in both average and worst-case scenarios, minimizing controller-to-controller latency, achieving load balancing, enhancing network reliability, and delving into energy conservation. For a comprehensive visual representation and deeper understanding, please refer to Figure 1.1, which aptly captures the intricacies inherent in the controller placement quandary. The interconnections between switches and controllers can take on diverse configurations, as evidenced by the array of linkages demonstrated by the black dotted lines. However, it's important to acknowledge that this particular arrangement might not be the most optimal, particularly in scenarios where multiple conflicting objectives need to be concurrently optimized.

Within an SDN-enabled network, specifically in the context of SD-WAN, the endeavor of controller placement solutions revolves around identifying efficient methods to optimize the positioning of controllers while simultaneously catering to diverse performance metrics. It's noteworthy that in a substantial number of scenarios, several of these performance criteria inherently conflict with one another. Consequently, the pursuit of an unequivocally perfect placement is often unattainable. Instead, decision-makers are compelled to navigate a path of balanced trade-offs, strategically managing the intricate interplay between these competing factors.

The strategic placement of controllers within the context of an SD-WAN deployment, where multiple conflicting objectives are at play, has garnered significant attention. Notably, research has demonstrated

the efficacy of metaheuristic algorithms such as the Adapted Non-Dominated Sorting Genetic Algorithm III (ANSGA-III) [7], the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [8], and the Multi-Objective Particle Swarm Optimization (MOPSO) [9] in tackling the intricate Controller Placement Problem (CPP) inherent in SD-WAN. However, it's worth noting that while these metaheuristic algorithms offer promising solutions, some encountered challenges impact their applicability. Specifically, with the exception of ANSGA-III, the other algorithms faced issues related to scalability, particularly when dealing with more than three objective functions. Additionally, they grappled with high computational complexity and the inability to predict the optimal number of controllers required.

As the need to optimize multiple objectives simultaneously grows in complexity, there is a pressing demand for innovation and the development of efficient techniques capable of addressing challenges related to scalability, computational complexity, and intelligent decision-making. In this context, 'intelligence' pertains to deploying predictive models such as XGBoost for informed decision-making. These innovations are essential for enhancing the capabilities of the aforementioned metaheuristic algorithms, ensuring their suitability for addressing the intricate scenarios encountered in SD-WAN deployments.

1.1 Problem Statement

Despite the demonstrated advantages of SDN being realized in data centre networks (DCNs) and various local area networks (LANs) [10], the implementation of SDN controllers within operational Wide Area Networks (WANs) continues to present a range of intricate architectural challenges [11]. To effectively serve its pivotal role as the network's central processing unit (CPU), an SDN controller must be capable of promptly responding to control request messages, even when confronted with multiple conflicting objectives [12]. Moreover, the execution of control tasks, such as efficient monitoring of the forwarding plane, is essential for maintaining up-to-date status information. This underscores the necessity for enhancements in the southbound interface. The significant impact of switch-to-controller latency on WAN performance has accentuated the significance of controller placement as a critical design hurdle, profoundly influencing the southbound performance of SDN [13]. The strategic arrangement of SDN controllers plays a determining role in their positioning relative to forwarding plane components [11]. Furthermore, the quantity of controllers deployed within a given WAN introduces an additional layer of consideration in controller placement. Various objectives, encompassing elements like end-to-end latency, load balancing, and network reliability, are directly influenced by the deployment count of controllers [14]. It's essential to emphasize that in an SD-WAN, randomly selecting the number of controllers without a well-calculated mechanism to determine the optimal number can significantly impact various network performance metrics. These metrics encompass critical aspects such as end-to-end latency, load balancing, and network dependability. This research briefly explores how these factors can affect the performance of an SD-WAN below and is further examined in Chapter 5 of this thesis. In Chapter 5, this research delves into the stochastic computational graph approach and its utilization of the XGBoost model to gain a deeper understanding of these implications

- End-to-End Latency: Randomly deploying controllers may lead to uneven distribution across the network. Some areas may be overserved with controllers while others are underserved. This imbalance can result in variations in end-to-end latency. Nodes closer to heavily deployed controllers may experience lower latency, while those farther away may suffer from higher latency. The lack of optimization for controller placement can exacerbate latency issues, especially in real-time or latency-sensitive applications.

- Load Balancing: Random placement of controllers can result in uneven controller workloads. Some controllers may become overloaded with traffic and management tasks, while others remain underutilized. Load balancing, which is critical for efficient network operation, becomes challenging without an appropriate mechanism to calculate the optimal number of controllers. This imbalance can impact the overall network's ability to handle traffic efficiently and may lead to congestion in some parts of the network.

- Network Dependability: Random controller placement can also affect network dependability and resilience. Inadequate controller coverage may result in reduced fault tolerance and recovery capabilities. In case of controller failures or network disruptions, the lack of optimized controller placement may hinder the network's ability to reroute traffic and maintain service availability.

These attempts encompass mathematical models and multi-objective evolutionary algorithms [15]. Mathematical models, including quadratic programming, mixed integer programming, and linear programming, have been explored within the literature [10]. However, these methods are generally limited to small-scale network scenarios because they suffer from significant computational complexity and are prone to becoming trapped in local optima. In such situations, the algorithm reaches a point where it can no longer enhance the objective or fitness function it's attempting to optimize [7]. On the other hand, metaheuristic techniques such as ANSGA-III [7], NSGA-II [8], and MOPSO [9] have demonstrated effectiveness in resolving the CPP in SD-WAN. Nevertheless, the mentioned metaheuristic algorithms, with the exception of ANSGA-III, have grappled with challenges encompassing scalability (particularly with more than three objective functions), heightened computational complexity, and the inability to predict controller numbers. In summary, the absence of a well-calculated mechanism to determine the optimal number of controllers and their placement can lead to suboptimal network performance. It can cause latency disparities, load imbalances, and reduced network dependability. To mitigate these issues, it is essential to employ optimization techniques that consider various performance metrics and network objectives to determine the appropriate number and placement of controllers in an SDN. As a result, the central challenge to address is as follows: Within a functional SDN-enabled WAN, the challenge arises: What defines the optimal number of SDN controllers, and where should they be strategically positioned to align with customer requirements, adhere to constraints, and ensure both peak runtime performance (referring to computational efficiency) and accuracy (referring to algorithm precision and correctness)? [16]. This challenge presents a multi-objective optimization problem characterized by competing objectives, [17], necessitating a solution to enhance overall network performance. Numerous research efforts have aimed to concurrently optimize controllers while addressing conflicting objectives [8]. However, in the pursuit of simultaneously optimizing controller placement (CP) while managing conflicting objectives, a gap exists in terms of scalability (excluding ANSGA-III), intelligence, and the challenge of high computational complexity [7].

1.2 Research Problem

The landscape of controller placement methods is entangled with intricate challenges, specifically regarding scalability and intelligence. In this context, scalability denotes the algorithm's inefficiency when faced with the simultaneous optimization of more than three conflicting objectives. Intelligence, on the other hand, pertains to the existing optimization techniques' limitations in comprehending the heuristics of combinatorial optimization problems, like SD-WAN controller placement, and their incapability

to predict the optimal number of controllers. These complexities underscore the multifaceted nature of optimizing controller placement in SD-WAN networks At the same time, these methods grapple with formidable computational intricacies during CP within the SD-WAN environment. As CP plays a pivotal role in SD-WAN architectures, administrators must strategically position controllers to optimize multiple conflicting objectives. The overarching research problem revolves around optimizing the placement of controllers in SD-WANs while addressing key challenges:

Scalability and Computational Complexity: Existing controller placement algorithms, including NSGA-II, MOPSO, and even ANSGA-III (with the exception of the ANSGA-III), exhibit substantial computational burdens that restrict their applicability in large-scale SD-WAN deployments. This exposes a critical research gap: the need to develop optimization methods that adeptly handle the intricate computational demands arising from addressing multiple, clashing objectives. This research endeavor seeks to harness the power of established optimization paradigms to tackle these computational complexities head-on.

Predefined Controller Deployment: Present paradigms often assume a fixed number of controllers for SD-WAN deployments, potentially leading to suboptimal outcomes in varying network conditions and requirements. This highlights the need for predictive models, exemplified by techniques like XGBoost and learning vector quantization, capable of accurately determining the optimal number of controllers essential for effective SD-WAN deployment.

Statistical Analysis and Significance Testing: A distinct research gap emerges in conducting comprehensive statistical evaluations that compare the performance of proposed methodologies against existing controller placement algorithms (e.g., NSGA-II, MOPSO, and ANSGA-III). While the computational efficiency and scalability of the stochastic gradient descent approach are emphasized, ensuring the observed performance differences hold statistical significance is crucial. The absence of rigorous statistical analysis hampers the capacity to assert the superiority or effectiveness of proposed methodologies over existing alternatives.

Overall, the current academic landscape lacks an approach that adeptly optimizes multiple conflicting objectives (beyond three) while effectively managing computational complexities. Additionally, the absence of predictive intelligence for determining optimal controller numbers and placements underscores the need for innovative methodologies. Recognizing the significance of computational costs and performance for network operators, the research calls for novel strategies that address scalability, computational efficiency, and predictive intelligence—ultimately elevating SD-WAN deployments while prudently minimizing organizational costs.

1.3 Research Questions

The research problem at hand raises a significant concern for SD-WAN service providers and operators who seek to enhance the efficiency of their chosen algorithms while adapting controller placement strategies. In light of this challenge, it is imperative to formulate specific research questions that can guide the investigation and provide actionable insights. The research holds crucial implications due to the substantial impact that controller placement has on SD-WAN operators. Based on the underlying problem, the following research questions have been formulated:

RQ1. How can an enhanced migration decision algorithm for controller placement be formulated to effectively tackle the complexities of SDN load balancing?

RQ2. How can the optimization of SD-WAN controllers be achieved in the context of multiple conflicting objectives, surpassing the count of three)?

RQ3. How can machine learning methodologies be leveraged to facilitate the acquisition of heuristics for solving intricate combinatorial optimization problems, such as the placement of SD-WAN controllers?

1.4 Solution Overview

This thesis undertook a comprehensive approach to address challenges associated with existing switch migration techniques in SDN load balancing. The initial focus was on refining switch migration methods to improve SDN load balancing. While existing algorithms, such as controller adaptation [12], migration decision [18], and dynamic and adaptive load balancing [19], have been developed for SDN load balancing, they face issues like high packet loss, extended response times, inefficient switch selection, and low throughput in scenarios involving high-volume incoming elephant traffic flows. As a result, this study introduced an enhanced switch migration decision algorithm for more effective SDN load balancing.

The introduced mechanism, termed ISMDA, selects heavily loaded switches for migration from controllers with excessive loads. This migration aims to target the most suitable controller to optimally release clustered resources. The balancing module of the developed framework initiates during the migration process. The mechanism assesses both the variance and average load states of controllers to identify underutilized controller groups. Moreover, a migration model was developed to consider migration cost and load-balancing variance for selecting optimal controllers from a pool of unloaded ones. Building upon this, the thesis delves into two additional solutions, Solutions 2 and 3, focusing on SD-WAN controller placement without assuming an optimal initial placement. These solutions bridge gaps in the existing literature by addressing CP optimization algorithms within the SD-WAN context. These solutions specifically target the challenges of scalability, intelligence, and high computational complexity encountered while positioning controllers to optimize conflicting objectives.

The first approach incorporates a repair operator-based mechanism, reference points, normalization, association, and a niching algorithm, creating a scalable framework beneficial for WAN operators. The second method employs a stochastic computational graph with an ensemble learning model (SCGMEL) and a learning vector quantization classification algorithm. This dual approach effectively determines the optimal controller placement while predicting the number and location of controllers. These strategies enhance overall SD-WAN performance by promptly catering to service provider requirements.

To assess the classification model's effectiveness, a new metric called "mean accurate location" was developed. It was compared with existing classification algorithms, including XGBoost, CatBoost, Random Forest, k-NN, and Logistic Classification. Additionally, the proposed XGBoost regression model underwent evaluation and performance comparison with KNN, Random Forest, and linear regression

models using mean absolute error as the performance measure. The SCGMEL leveraged stochastic gradient descent with momentum, learning rate decay, a computational graph model, and the XGBoost algorithm. This hybrid approach predicted the required number of controllers for SD-WAN deployment and simultaneously optimized controller placement. A novel Learning Vector Quantization (LVQ) based classification algorithm was also introduced for SD-WAN CP prediction.

These innovative solutions collectively offer a comprehensive framework to enhance SDN load balancing and optimize controller placement in SD-WAN scenarios. The utilization of advanced algorithms and machine learning techniques sets the foundation for improved network performance and efficient resource allocation.

1.5 Motivation

Recognizing the gaps prevalent in existing literature pertaining to challenges associated with controller placement algorithms in SD-WAN, there arises a pressing need to address these gaps to cater to the requirements of service providers and network administrators. Algorithms lacking scalability, computational efficiency, and intelligence in controller placement are impractical and uneconomical for users and service providers utilizing SD-WAN. This research draws inspiration from these identified issues and their significance in the realm of SD-WAN architecture users.

The core differentiating aspect of SDN architectures lies in the segregation of the network's control plane and forwarding plane. This division of control and forwarding planes gives rise to the intricacies of controller placement. The CPP stands as a combinatorial optimization challenge akin to location analysis, classified under non-deterministic polynomial-time hardness. Developing an algorithm capable of delivering nearly optimal solutions within a short span for such scenarios demands substantial effort. The CPP's search space encompasses all possible combinations of 'k' (controller number) and 'n' (potential nodes) within the network architecture, where 'k' is less than 'n'. For instance, with 35 nodes, the exploration spans $\binom{n}{k}$, resulting in 52,360 distinct alternative placements when aiming to determine optimal locations for four controllers. Even for smaller 'k' values, the feasible locations escalate significantly in both magnitude and scope. In such contexts, meta-heuristic approaches offer an alternative by exploring a subset of the search space to locate a near-optimal solution. As demonstrated by [20], the position of controllers profoundly impacts network efficiency by shifting controllers across the network. Hence, in dynamic-changing architectures like SD-WAN, determining the positioning of network controllers becomes a formidable task. For this determination to hold value, the controller's placement must be strategically derived. The Pareto-Optimal Resilient Controller (POCO) approach [21] exhibits the capability to handle small and medium-sized topologies, delivering solutions swiftly. However, evaluating the location of controllers within large-scale networks demands extensive assessment, consuming substantial time and struggling to keep pace with the network's rapid changes.

Developing an algorithm capable of providing optimal solutions within seconds or minutes for complex challenges like SD-WAN is an arduous undertaking. Achieving optimal solutions within an acceptable time frame can be accomplished through the application of meta-heuristic techniques or, alternatively, through standalone machine-learning techniques. The popularity of meta-heuristic strategies has surged due to their capacity to optimize multiple conflicting objectives and yield Pareto-optimal solutions in a

reasonable time, surpassing mathematical approaches. Approaches like Simulated Annealing (SA), Iterated Local Search, and Guided Local Search exemplify single-solution meta-heuristic techniques. Each of these strategies focuses on individual potential solutions at a time. In contrast, population-based meta-heuristic algorithms such as ANSGA-III, NSGA-II, and MOPSO store numerous potential solutions, facilitating the attainment of non-dominated solutions through Pareto optimization. Single optimization techniques and mathematical approaches prove inadequate for extensive search spaces like SD-WAN due to the risk of local optima entrapment, extended computational time, and lack of comprehensive evaluation. Guided by these attributes, this thesis is propelled to leverage diverse meta-heuristics and machine-learning techniques for selecting optimal solutions from an extensive array of possibilities within a predefined time frame. The presented meta-heuristic algorithms and machine learning techniques systematically address the issue in a scalable manner, augmenting overall system performance. Meanwhile, the lack of intelligence in current optimization methods propels further exploration and provision of more intelligent and robust solutions tailored for SD-WAN operators.

1.6 Primary Research Aim and Objectives

This research aims to develop a collaborative and adaptive optimization learning-based framework to facilitate the placement of SD-WAN controllers, even in the presence of multiple conflicting objectives. The primary objectives to achieve this aim are as follows:

- **[Obj-1] To meticulously examine and assess the cutting-edge optimization and classification algorithms utilized in the realm of SD-WAN controller placement. This rigorous analysis is conducted to pinpoint any existing research gaps within the public domain pertaining to this field.**
  A comprehensive overview of various optimization and classification algorithms is provided and subjected to in-depth analysis. This critical examination delves into the limitations inherent in the current state-of-the-art optimization and classification algorithms found in the literature. The investigation particularly focuses on their effectiveness in concurrently optimizing multiple conflicting objectives. Moreover, the potential implications of these techniques on SD-WAN operators' Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) are explored, especially in the context of determining optimal positions for placing SD-WAN controllers.

- **[Obj-2] To formulate and actualize an improved switch migration decision model that ensures efficient load distribution among distributed controllers within the SDN architecture.**
  In light of the identified limitations within the existing literature, the imperative need arises to develop an advanced switch migration decision model that can ensure the efficient distribution of workloads among the dispersed SDN controllers. Consequently, an enhanced switch migration decision algorithm has been devised to address the challenge posed by the influx of substantial incoming data flows. During instances of controller load disparity, the balancing components of the switch migration model are initiated. To determine the ensemble of controllers operating below their capacity in the network, the enhanced framework incorporates parameters such as controller variance, controller threshold, and controller average load status.

- **[Obj-3] To design and develop a repair operator-based mechanism, integrating it into existing NSGA-III, for optimizing SD-WAN controller placement while addressing**

multiple conflicting objectives (more than three).
Considering various network performance metrics, SD-WAN operators strive to strategically position controllers to achieve diverse and sometimes conflicting goals. Existing optimization techniques like NSGA-II and MOPSO face scalability challenges when dealing with more than three objectives. These challenges include limited acquisition of non-dominated solutions. To address this, a solution that supports more than three objectives and covers the Pareto front comprehensively is imperative. To achieve this, a repair operator-based mechanism was integrated into ANSGA-III for optimal SD-WAN controller placement. The enhanced ANSGA-III utilizes a reference-based approach, association, normalization, and a niching algorithm to simultaneously optimize multiple competitive objectives in SD-WAN controller placement.

- **[Obj-4] To create an automated learning-based decision-making model for optimal controller placement, this research aims to leverage a stochastic computational graph combined with an ensemble learning model and a learning vector quantization.**
  The existing literature has proposed the utilization of meta-heuristic algorithms for SD-WAN controller placement, particularly when faced with conflicting objectives. However, a thorough review conducted within this study has revealed a common limitation across these developed meta-heuristic algorithms (ANSGA-III, NSGA-II, and MOPSO), primarily pertaining to their scalability challenge in scenarios involving more than three conflicting objectives. Additionally, these solutions are associated with high computational costs and lack the intrinsic capability to intelligently learn the heuristics required for solving combinatorial optimization problems, such as the intricate task of controller placement in SDN. Given these inherent issues within the current methodologies for controller placement, the urgency arises to devise a solution that is not only scalable and adaptable but also significantly computationally efficient for addressing SDN controller placement challenges.

- **[Obj-5] To test, verify, validate, and analyze the obtained results for critical evaluation.** The proposed intelligent and scalable framework will undergo rigorous testing, analysis, and validation to assess its performance and functionality. These tests aim to uncover the framework's strengths and weaknesses, providing valuable insights for a comprehensive comparison with existing frameworks designed to address similar challenges. This process is vital for ensuring the framework's effectiveness and suitability for practical implementation.

1.7 Contributions

In pursuit of bridging the identified gaps, this thesis presents a range of innovative contributions. The primary goal was to formulate and implement a collaborative and adaptive learning-based optimization solution tailored for controller placement in the presence of numerous concurrent optimization objectives. Aligned with the framework outlined in the objectives, these contributions culminate in the following achievements:

- **[OC-1] Introducing a novel approach for load balancing among controllers through the utilization of switch migration techniques.**
  Existing approaches for load balancing among distributed controllers in SDN architecture have been limited in their effectiveness. Previous authors addressing this issue often focused on scenarios

where incoming traffic comprised smaller "mice flows," and they optimized migration efficiency considering low-flow data planes. However, this research goes beyond these limitations by introducing enhanced switch migration techniques that address challenges associated with substantial incoming data traffic loads. The developed algorithm's balancing component, present in each controller, activates to achieve efficient load distribution when a controller's load exceeds a predefined threshold. In this novel approach, the algorithm identifies under-loaded controllers within the set by analyzing the variance and average load of the controllers. By effectively reallocating highly loaded switches from overloaded to under-loaded controllers, the method optimizes resource utilization. Additionally, the study includes a migration efficiency model that highlights the trade-off between migration cost variance and load balancing. This comprehensive methodology overcomes previous constraints and offers a more robust solution for load balancing across distributed SDN controllers.

- **[OC-2] This research introduces an innovative repair operator-based mechanism seamlessly integrated into the existing NSGA-III framework to achieve optimal controller placement within SD-WANs.**
  Current state-of-the-art optimization techniques face limitations when dealing with more than three objectives to be simultaneously optimized within the network architecture. This research addresses this challenge by introducing a novel repair operator-based mechanism, seamlessly integrated into the engineering-based NSGA-III framework for optimizing controller placement. The developed repair operator ensures the elimination of impractical solutions during the crossover and mutation process, and it replaces continuous optimization features in NSGA-III with discrete optimization characteristics. This approach enhances the convergence and diversity of solutions across the Pareto Front through techniques like normalization, association, reference points, and niching. Furthermore, this strategy proves to be highly effective, enabling the concurrent optimization of multiple conflicting objectives during SD-WAN controller placement.

- **[OC-3] Introduces an innovative automated learning-based decision-making model for achieving optimal controller placement. This model leverages a stochastic computational graph in conjunction with an ensemble learning approach and learning vector quantization.**
  Recent studies have revealed that traditional exhaustive and meta-heuristic approaches face significant challenges due to their high computational demands and their inability to learn heuristics crucial for solving complex combinatorial optimization tasks like SDN controller placement. In response to these limitations, this research introduces an innovative approach that combines a stochastic computational graph with an ensemble learning model and learning vector quantization. This novel solution aims to optimize controller placement while simultaneously addressing competing objectives and predicting the ideal number and positions of controllers within an SD-WAN topology. By utilizing a stochastic and dynamic computational graph and leveraging an ensemble learning model, specifically XGBoost, to predict optimal controller counts, this approach efficiently tackles the challenge posed by multiple conflicting goals during SD-WAN controller placement. Additionally, this study introduces a classification algorithm based on LVQ to accurately predict controller placements. The choice of XGBoost as an ensemble learning method is based on its proven ability to enhance predictive accuracy and overall model performance. XGBoost, classified as a boosting algorithm, falls under the category of ensemble methods. Unlike ensemble learning

methods that involve combining multiple independently trained models, XGBoost's ensemble model pertains to the collaborative operation of multiple decision trees within the algorithm.

1.8 Thesis Structure

The thesis is organized into the following sections:

- **Chapter 2 - Literature Review :** The literature review section conducts an extensive analysis of the challenges associated with SD-WAN architecture's controller placement problem. A comprehensive examination of the intricacies involved in positioning controllers within SD-WAN is presented, offering a thorough understanding. In Chapter 2, a critical assessment of methodologies, strategies, and approaches aimed at addressing the controller placement challenges in SD-WAN is undertaken. This analysis is rooted in the gaps identified in existing literature, highlighting the need for innovative solutions. Various frameworks contribute to resolving these placement challenges, significantly impacting the SD-WAN architecture. The review systematically evaluates the strengths and weaknesses of diverse techniques, serving as a guiding roadmap for implementing the proposed scalable and intelligent framework. The exploration of the requisite development environment and architecture is also conducted, essential prerequisites for constructing the envisioned scalable and intelligent framework.

- **Chapter 3 - Novel Approach for Controller Load Balancing through Switch Migration**
  This chapter presents a pioneering solution aimed at rectifying a prominent gap identified in the existing literature: Optimal load balancing across distributed controllers. Drawing from this identified void, a unique approach has been conceptualized and developed to tackle this challenge head-on. This chapter offers an insightful exploration into the intricate workings of this novel approach, detailing the synergy between its distinct modules: Load Judgment, Switch Selection, and Target Controller Selection. By elucidating the interplay of these modules, this chapter unveils the mechanics of the proposed approach. Moreover, the implementation of this approach is thoroughly elucidated, highlighting the incorporation of mathematical techniques such as variance, average load, migration cost, and load balancing rate. The comprehensive implementation and testing procedures undertaken in this study are meticulously outlined, providing a comprehensive view of the approach's viability and effectiveness. Through these in-depth insights, readers can gain a robust understanding of how this novel approach addresses the critical load-balancing challenge within the context of distributed controllers.

- **Chapter 4 - Introducing a Scalable Approach for Simultaneously Optimizing Competing Objectives in SD-WAN Controller Placement**
  Chapter 4 introduces an innovative meta-heuristics solution designed to revolutionize the way SD-WAN controllers are placed within the network. This approach takes a bold step forward by addressing the challenge of optimizing multiple, often conflicting, objectives simultaneously during controller placement. Within this chapter, readers will find an intricate exploration of the modified NSGA-III at the heart of this cutting-edge solution. The algorithm's various components, including the introduced mechanisms of the *repair-based operator, normalization, association*, and *niching*, are meticulously dissected and explained.
  Central to this endeavor is the replacement of continuous optimization characteristics in the existing NSGA-III with discrete optimization attributes through the innovative repair-operator mechanism.

This chapter delves deep into the mechanics of this substitution and offers a comprehensive understanding of how it enables SD-WAN operators to effectively balance multiple competing objectives when placing controllers. As the chapter unfolds, readers are guided through the intricate technical details, presenting a coherent narrative that culminates in a robust solution ready for implementation. From concept to execution, this chapter provides a holistic view of how the proposed approach empowers SD-WAN operators to make informed decisions while optimizing controller placement to align with diverse network objectives. Comprehensive implementation and rigorous testing are also presented, ensuring that the proposed approach is thoroughly evaluated for its effectiveness and real-world viability.

- **Chapter - 5 Innovative Learning-Based Decision-Making Model for Optimal SD-WAN Controller Placement**

  Chapter 5 introduces a groundbreaking approach that reshapes the landscape of SD-WAN controller placement. At its core is a novel framework that combines a stochastic computational graph with an ensemble learning model, resulting in a powerful solution to the complex challenges of controller placement in SD-WAN networks. This chapter provides an in-depth exploration of the proposed SCGMEL model. Through a meticulous description, readers gain a comprehensive understanding of how this innovative fusion of technologies works synergistically to tackle the intricate problem of controller positioning.

  The SCGMEL model is meticulously dissected, offering insights into its components, including stochastic gradient descent with momentum, a weighted sum approach, learning rate decay, and a dynamic computational graph. The ensemble learning model, a key player in the SCGMEL approach, is unveiled, illustrating its ability to harness the power of multiple predictive models to make informed decisions. Furthermore, this chapter demonstrates the practical application of the SCGMEL model in solving the controller placement challenge. By leveraging the dynamic computational graph and ensemble learning techniques, the SCGMEL model effectively determines the optimal placement of controllers within SD-WAN networks. Notably, the XGBoost decision tree model is incorporated to predict the number of controllers required for an efficient SD-WAN deployment. A crucial element of this chapter is the comprehensive implementation and testing of the proposed approach. Through rigorous assessment, the performance, accuracy, and scalability of the SCGMEL model are thoroughly evaluated, showcasing its potential to revolutionize SD-WAN controller placement strategies.

  To validate the significance of the offered solutions, this chapter presents a comparative analysis, pitting the proposed stochastic computational graph with an ensemble learning model (XGBoost) and learning vector quantization against existing state-of-the-art solutions. By providing a comprehensive context for the contributions made in this study, Chapter 5 establishes a foundation for elevating SD-WAN controller placement practices to new heights.

- **Chapter 6 - Conclusion and Future Work**

  In culmination, this chapter draws the curtain on the thesis, providing an insightful encapsulation of the research's discoveries and accomplishments. The path traversed has been one of innovation, aiming to bridge gaps and forge new frontiers in the realm of SD-WAN controller placement optimization. As the contributions unfold, it becomes evident that these novel solutions hold immense promise for both practical application and scholarly exploration.

Chapter 2

A Review of Controller Placement Techniques in SDN

This pivotal chapter delves into the intricate landscape of research concerning controller placement predicaments within the realm of SDN architecture. This exploration unfolds across three distinct sections, each contributing to a comprehensive understanding of the subject matter. The chapter commences by offering a comprehensive overview of SDN, dissecting its core elements and mechanisms. This unveiling encompasses the very essence of SDN's design, operation, and underlying architecture. Notably, the discussion encompasses key elements such as open-source SDN controllers, protocol intricacies, and the bedrock of standards that collectively shape the architecture. These layers, aptly termed *Application*, *Control*, and *Infrastructure*, set the stage for the subsequent exploration. Continuing its journey, the chapter seamlessly transitions into an exploration of related endeavors in the realm of controller placement. This comprehensive survey delves into prior works, illuminating the various facets of controller placement concerns. This meticulous investigation traverses through the existing literature, shedding light on the innovative strategies that have attempted to address this pressing challenge. The chapter's culmination witnesses a meticulous dissection of potential controller placement strategies, closely tethered to the all-encompassing objective function. This analytical endeavor navigates through diverse placement tactics, evaluating their alignment with the overarching objective. A systematic exploration of these strategies offers insights into the multifaceted nature of controller placement within the context of the objective function. Embarking on this literary journey, the chapter encapsulates a panoramic exploration of the research milieu pertaining to the intricate realm of controller placement predicaments, especially pronounced within SDN architecture and the SD-WAN domain. This thorough examination unfurls against the backdrop of state-of-the-art literature, with gaps and discrepancies pinpointed, setting the stage for this research's emergence. In summary, this chapter serves as a crucial bridge connecting the historical, current, and future dimensions of controller placement in SDN architecture. Its thorough analysis not only sheds light on the existing landscape but also lays the foundation for the study's pivotal role in addressing the identified research gaps. Navigating through this chapter, the research embark on a journey that traces the evolution of controller placement, captures its present complexities, and charts a course for impactful future advancements

2.1 Background

In the current networking landscape, the coupling of hardware and software constrains operators' ability to swiftly respond to evolving market needs. The intricate interplay between these components hampers

the introduction of new services on demand, often resulting in delayed responses to changing requirements. Notably, the substantial investment in specialized hardware further exacerbates this challenge, as operators must navigate intricate processes to extract maximum utility from hardware updates and releases [22]. Consequently, the rigid structure imposed by this hardware-software integration impedes the flexibility required for agile adaptation to emerging market trends.Furthermore, the absence of streamlined protocols for remote problem identification and resolution across diverse multi-vendor systems compounds the challenges of network maintenance. The prevailing practice of deploying specialized Systems Engineers to physically inspect and rectify network malfunctions [16] not only consumes valuable resources but also exposes operations to potential human fallibility. This heavy reliance on vendor-specific methodologies not only intensifies resource allocation but also undermines operational efficiency, ultimately eroding the overall quality of services provided.

SDN emerges as a revolutionary solution poised to revolutionize conventional networking paradigms. In essence, SDN serves as a potent response to the intricate challenges entrenched within traditional network configurations. The core premise of SDN revolves around a fundamental restructuring of network principles, making them open, programmable, and remarkably adaptable. This paradigm shift entails a profound architectural alteration whereby the control plane, responsible for pivotal network management decisions, is meticulously decoupled from the data plane, which constitutes the tangible path traversed by data packets [10]. Through this strategic division, SDN takes center stage by orchestrating a centralized control entity, commonly referred to as a controller, thereby unifying and streamlining the governance of the entire network's traffic dynamics.The defining hallmark of SDN lies in its extraordinary ability to abstract intricate networking functionalities into elevated constructs, effectively disentangled from the intricate underpinnings of hardware intricacies. This sophisticated orchestration of network behaviors not only underscores SDN's prowess but also underpins its ability to seamlessly harmonize and administer services across a diverse array of computing systems. Leveraging exposed APIs, network operators are empowered to configure and deploy specialized applications within the centralized controller. These dynamic applications encompass an extensive spectrum of functions, ranging from the realms of virtualization and load balancing to the intricacies of traffic engineering, the optimization of quality of service, the adept handling of faults, and the fortification of security protocols [23].
In this context, SDN unfolds a plethora of compelling benefits:

- Innovation Empowerment: SDN enables businesses to develop robust applications, innovate new products, and devise novel business models by abstracting the intricacies of underlying forwarding operations;
- Capital Expenditure (CapEx) Savings: SDN permits businesses to utilize "white box" switches and routers, facilitating the adaptation of their conventional hardware to SDN compatibility. This leads to cost reductions in CapEx;
- Operational Expenditure (OpEx) Savings: By enabling automated network administration and enhanced data plane programmability, SDN contributes to reduced OpEx;
- Enhanced Security: SDN enables the consistent deployment of security policies managed from a centralized control panel, resulting in improved security measures;
- Service Agility: SDN accelerates the deployment of new applications and services to adapt to changing traffic patterns, enhancing operational flexibility.

Table 2.1 provides a comprehensive comparison between conventional networks and software-defined networking. This table serves to elucidate the pivotal role that SDN plays in contemporary networks, underscoring the imperative for organizations and service providers to embrace software-defined networks in their infrastructure. A prime instance of SDN's advantage is its capacity to decouple the control plane from the data plane, streamlining management and enhancing overall network performance. Furthermore, when applied to the deployment of controller placement in SD-WAN, this architectural approach facilitates centralized monitoring of all connected devices, yielding cost-efficient maintenance in contrast to traditional WAN technology.

Table 2.1: Overall comparison between conventional networks and software-defined networks.

Criteria	**Conventional Networks**	**Software-Defined Networks**
Network configuration and management [24]	Needs to use vendor-specific instructions, which makes the program modifications complex and necessitates specialized experience.	Facilitates network programmability by offering interfaces that are independent of certain vendors.
Awareness of the status of the global network [25]	Complicated by the tight coupling of control and data planes	Simplified with a logically centralized decoupled controller
Cost of Maintenance [26]	Higher	Less
The time needed for updates and error handling [27]	Might take months	Due to centralized control logic, the process may be completed in only minutes
Load-balancing in the control plane [24]	Not Crucial	Crucial
The utilization of the control plane [28]	Irrelevant	Relevant
The availability of a control plane [24]	Not Crucial	Critical
Utilisation of resources. [28]	Less	High
The integrity of the flow table and the state information [29]	Important	Essential
Integrity, authenticity, and consistency of the control plane [29]	Irrelevant	Vital

2.1.1 SDN Architecture

The architecture of a software-defined network is structured into three functional planes: the data layer, the control layer, and the application layer (refer to Figure 2.1). Within the data layer, diverse network components like firewalls, switches, and routers form a cohesive network infrastructure. These components communicate their capabilities to the control layer via southbound programmable interfaces such as

RESTCONF, NETCONF, and *OpenFlow*. The control layer is embodied by a logically centralized controller that offers a comprehensive overview of the network's myriad constituents. The application layer encompasses a diverse array of software, encompassing software-defined networking applications, custom policies, cloud orchestration, and mobility management [30]. Through the utilization of a northbound programmable interface like *RESTful*, applications convey their requirements to the controller in the form of high-level directives. Subsequently, the controller translates these application needs into granular flow instructions, which configure the data layer accordingly. Resource utilization and state are abstracted for the application layer by the controller through its northbound interface. This abstraction ensures that the application layer is presented with relevant information while extraneous elements are concealed. To address scalability concerns, distributed controllers are often employed as an alternative to centralized control [31].

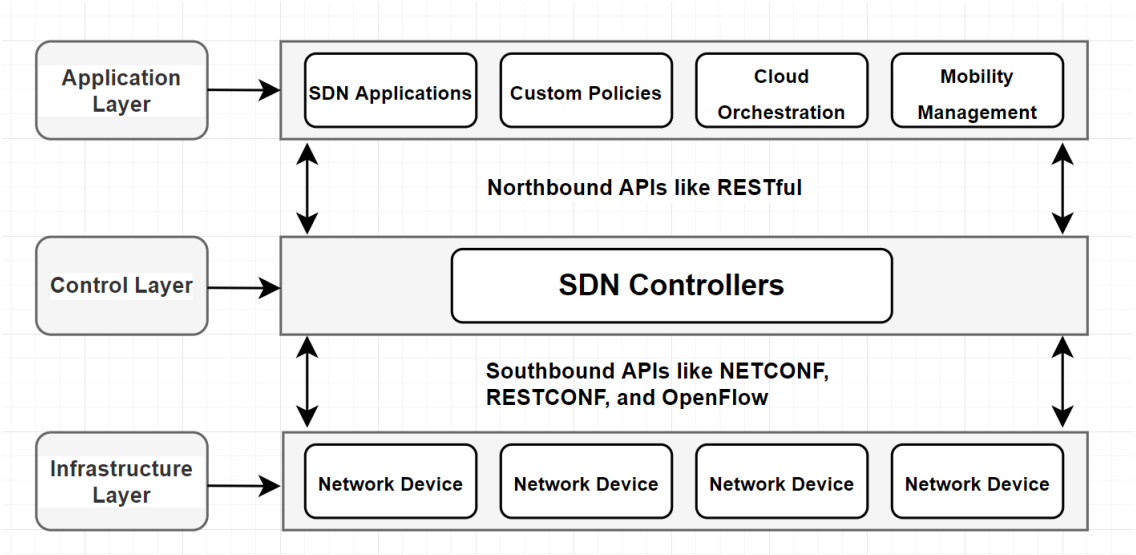

Figure 2.1: High level Software-Defined networks reference architecture

2.1.1.1 Protocols, Standards, and SDN Operations

Protocols, Standards, and SDN Operations are pivotal components of the SDN framework, collectively shaping its functionality and interoperability. This section delves into the intricate web of protocols and standards that underpin SDN's operation, ensuring seamless communication between its various layers and components. SDN relies on a variety of communication protocols to facilitate data exchange and coordination between its distinct layers: the data layer, control layer, and application layer. Notably, OpenFlow has emerged as a prominent protocol, acting as a conduit for communication between the control and data planes. Through OpenFlow, the control layer instructs network devices at the data layer on how to handle traffic flows. This separation of control and data planes allows for centralized control and programmability of network behavior. **"Although OpenFlow is not commonly employed as a standard component within SD-WAN, it's essential to distinguish SD-WAN's primary focus. SD-WAN technology is designed for the optimization and efficient management of wide-area network connections [32]. While OpenFlow may be utilized in specific SD-WAN implementations, the core attributes of SD-WAN encompass dynamic path selection, application-oriented routing, WAN optimization, and centralized administration**

of diverse network connections, including MPLS, broadband, and LTE [33]. While certain SD-WAN solutions may incorporate OpenFlow-like principles for traffic management within the WAN infrastructure, it's important to note that the SD-WAN architecture is fundamentally geared towards application-level control and optimization, rather than the granular control offered by OpenFlow." Through abstracting the complexities, SDN communication protocols facilitate the convergence of diverse network devices. Extensive efforts have been directed toward standardizing both southbound and northbound protocols, with a particular emphasis on the southbound interface [10]. Currently, a range of southbound and northbound protocols are available. Notable examples of southbound protocols include NETCONF [34], OVSDB (Open vSwitch Database Management) [35], and OF-CONFIG (OpenFlow Configuration) [36]. Northbound protocols encompass LISP (Locator ID Separation Protocol) [37], Path Computational Element Protocol (PCEP) [38], BGP Link-State (BGP-LS) [39], and I2RS (Interface to Routing System) [40]. Southbound protocols facilitate the implementation of forwarding plane activities, such as port allocation, IP assignment, and policy enforcement, while northbound protocols enable the configuration of packet flow activities on the forwarding layer [41]. Protocols like I2RS, PCEP, and BGP-LS, categorized as "Hybrid SDN" in Table 2.2, offer viable options for introducing SDN into traditional networks without requiring a complete infrastructure overhaul. This approach minimizes migration costs, particularly capital expenses.

Table 2.2: The comparison of southbound protocols at a high level

Standards	**Regulatory Body**	**Motive**	**Business Domain**	**SDN Interface**
OpenFlow [42]	Open Networking Foundation	Control	Software-Defined Networks	Southbound Interface
Interface to Routing System [43]	Internet Engineering Task Force	Control	Conventional Networking+SDN protocols	Southbound Interface
Locator ID Separation Protocol [44]	Internet Engineering Task Force	Control	Conventional Networking+SDN protocols	Southbound Interface
Network Configuration protocol [26]	Internet Engineering Task Force	Management	SDN Conventional Networking+SDN protocols	Northbound Interface, Southbound Interface, East/Westbound
Path Computational Element Protocol [42]	Internet Engineering Task Force	Control	Conventional Networking + SDN protocols	Southbound, East/ Westbound
Open vSwitch Database Management [45]	European Telecommunication standards institute	Management	Software-Defined Networks	Southbound, Southbound

Conversely, protocols like OF-CONFIG may not be compatible with existing networks and may demand substantial capital investments. BGP-LS and PCEP have gained prominence in carrier-grade SDN deployments due to their scalability features, while OpenFlow has become the standard for data center environments [46] [47]. A comparative overview of frequently used southbound SDN protocols is presented in Table 2.2.

OpenFlow remains a prominent choice for controlling SDN deployments, despite the proliferation of alternative SDN control protocols. Major networking equipment providers such as Dell, Cisco, HP, Arista, and Big Switch Networks have expressed interest in OpenFlow. Serving as a communication standard within SDN, OpenFlow enables the execution of flow instructions into the data layer via the Transmission Control Protocol. To ensure secure communication, OpenFlow recommends using the Transport Layer Security (TLS) protocol. Figure 2.2 illustrates the packet flow mechanism in OpenFlow [4]. Upon receiving user packets, switches examine the packet's match fields (including packet headers, ingress port, and metadata) against table entries. When a matching flow entry is found, the switch updates counters and executes the instructions from the instruction set [48]. These instructions can direct the traffic to a specific egress port or discard the packet. In cases where no matching entry is found (a table miss), the instruction set in the table miss flow entry defines how mismatched packets are handled. Options include discarding packets, routing them to another table for further matching, or forwarding them to the controller using a packet-In message through the southbound interface. Upon receiving packet-in messages, the controller decides on the course of action and utilizes packet-out messages to install flow entries on the switch.

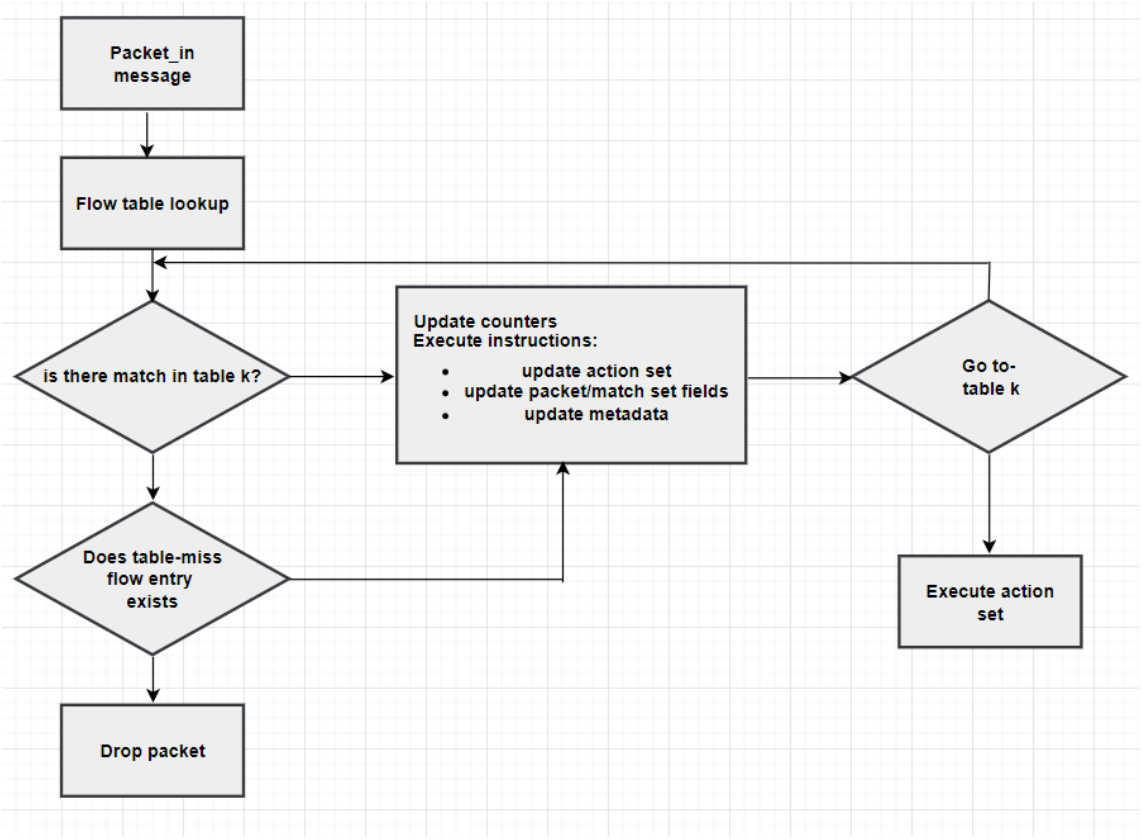

Figure 2.2: Processing of packets in an OpenFlow switch.

OpenFlow encompasses three distinct types of messages: controller-to-switch, asynchronous, and symmetric [49]. In the controller-to-switch message format, initiated by the controller, the conversation between the two entities commences. This message type includes role request messages, feature requests, and switch statuses, among others, and serves purposes like monitoring and discovery [50].

Conversely, in the asynchronous message type, the switch can transmit messages to the controller without any explicit request. Examples of such messages encompass switch state changes, packet-in notifications, port status updates, packet-out messages, and flow-removed notifications. These asynchronous messages facilitate real-time communication and event handling [51].

Lastly, symmetric messages [52] are those that can be initiated by either the controller or the switch, without prior solicitation. These messages promote bidirectional interaction. Notable examples of symmetric messages include Hello messages for connection establishment, Echo messages for connection verification, and Experimental messages for research and custom functionalities.

Table 2.3: Overview of OpenFlow Control Messages

Message Type	**Description**	**Examples**
Controller-to-Switch [50]	Message sent by the controller, which may or may not need a response from the switch depending on the circumstances.	Read-State, Features request, Barrier messages, and Send-Packets are some of the operations that are performed.
Asynchronous [51]	A message is transmitted by the switch regardless of whether or not it was requested by the controller.	Flow-removed, Packet-in, Error messages, as well as Port-Status
Symmetric [52]	The direction of an unsolicited message might be either way.	Hello, Echo, and Vendor messages

2.1.1.2 Open Source Software-Defined Networks Controllers

Open-source software-defined network (SDN) controllers play a pivotal role in the architecture, acting as the central processing unit, as highlighted in section 2.1.1. To expedite the development of SDN solutions, substantial efforts have been invested in crafting these open-source controllers. Among the most prominent ones are Ryu [53], OpenDaylight [54], and ONOS [55]. These controllers have surged in popularity and offer diverse attributes encompassing scalability, complexity, security, and interoperability. The subsequent section furnishes a concise overview of these controllers, coupled with a comprehensive feature-based juxtaposition. ONOS, a leading open-source software-defined network controller, was developed by ON.Lab with a primary objective of empowering service providers to establish practical SDN solutions. This controller, notably optimized through its distributed core, prioritizes crucial production network attributes such as high-performance metrics, scalability, and dependability [56]. ONOS employs two abstraction frameworks within its northbound interface: the global network topology view and the intent framework. Leveraging the intent framework, network applications can deploy services using policy-based instructions, abstracting the "what" over the "how" [10]. The global network view equips the application layer with the network's real-time status, streamlining resource utilization comprehension.

The southbound interface further enhances abstraction by representing hardware resources as objects, accommodating diverse infrastructural plane elements through protocol plugins like OpenFlow, OVSDB, and NETCONF. Eastbound and westbound communication across multiple ONOS controller instances is facilitated by an adapted version of the BGP protocol, allowing distributed control nodes to provide domain-specific status data [10]. ONOS finds its prime use case in CORD (Corporate Office Re-architected as a Data Centre), revolutionizing the domain through cloud computing, SDN, and network function virtualization [57]. This approach enables operators to achieve economies of scale and process optimization akin to cloud service providers, fostering rapid network service deployment and elastic scaling.

Recognized as the service provider controller due to its distributed core, ONOS boasts a series of version releases, including Velociraptor, Uguisu, Toucan, Sparrow, Raven, Quail, Peacock, Owl, Magpie, Nightingale, Loon, Kingfisher, Junco, Ibis, Hummingbird, Goldeneye, Falcon, Emu, Drake, Cardinal, Blackbird, and Avocet. These versions signify its substantial community support and widespread adoption [10].

2.1.1.3 OpenDaylight

OpenDaylight, a versatile Java-based open-source software-defined controller, is maintained by the Linux Foundation and serves as a configuration and coordination tool for networks of varying scales [58]. Leveraging the Model-Driven Software Engineering (MDSE) paradigm, OpenDaylight abstracts the lower-level capabilities of data layers, utilizing YANG as its data modeling syntax. The controller represents hardware resources as manageable objects through the Service Abstraction Layer (SAL), ensuring interoperability. OpenDaylight's modular design empowers users and technology providers to tailor traffic controls to their specific needs. Embedded in the control plane, the Model-Driven SAL (MD-SAL) functions as the SDN network's "brain," translating application layer rules into the data layer through its northbound and southbound interfaces, respectively. OpenDaylight supports an array of southbound protocols, including OpenFlow, OVSDB, NETCONF, PCEP, BGP-LS, among others. Notably, OpenDaylight occupies a central role in open-source supervision and automation frameworks like ONAP (Open Networking Automation Platform), OPNFV, OpenStack, and industry-specific groups such as MEF (Metro Ethernet Forum) [59]. An instance of this is the UNI Manager plugin version within OpenDaylight, offering APIs for MEF's LSO (Lifecycle Service Orchestration) project. Focused on interoperability, OpenDaylight has become the standard choice for hybrid SDN implementations. With 16 releases to date, including Sulfur, Phosphorus, Silicon, Aluminium, Magnesium, Sodium, Neon, Fluorine, Oxygen, Nitrogen, Carbon, Boron, Beryllium, Lithium, Helium, and Hydrogen, each version expands application instances, incorporates IoT support, integrates network function virtualization management, and enhances S3P (Scalability, Security, Stability, and Performance) through clustering and federation strategies.

2.1.1.4 Floodlight

Floodlight, an event-based SDN controller developed by Big Switch Networks, serves as a valuable choice for rapid prototyping in smaller environments. Notably, Floodlight is tailored for OpenFlow compatibility in its southbound, making it optimal for scenarios not requiring support for more intricate protocols like PCEP or BGP-LS. Its northbound REST API support streamlines the creation of diverse traffic engineering strategies for application developers. Furthermore, Floodlight boasts features like multi-threading, flexibility, and operates within an asynchronous architecture [60].

2.1.1.5 Ryu

Ryu [53], a Python-based SDN controller, finds its utility as a versatile software-defined network controller, commonly utilized in cloud orchestration applications. One of its distinctive advantages lies in its extensibility through the integration of modules written in various languages, offering the flexibility of a generic controller. Ryu employs REST API as its northbound abstraction interface and supports a spectrum of southbound interfaces including OVSDB, NETCONF, OFCONFIG, and OpenFlow [61]. However, its lack of support for broader protocols like BGP-LS renders it unsuitable for large-scale deployments, restricting its use primarily to rapid development in limited-scale scenarios. Among various open-source controllers, OpenDaylight and Ryu stand out for their extensive range of southbound connections. Floodlight, in contrast, is tailored for OpenFlow exclusively. This narrow focus limits its deployment to environments exclusively using SDN. Notably, OpenDaylight [26] garners significant community and vendor support, closely followed by ONOS [55]. Both Floodlight and Ryu provide developers with full control over their codebases. In terms of deployment, OpenDaylight and open network operating systems offer distributed control, rendering them suitable for data networks. Both systems also exhibit high modularity, simplifying the addition of new features. A distinguishing limitation of open network operating systems is their lack of support for cloud orchestration tools like OpenStack, which is crucial for managing cloud infrastructure. Decision-makers involved in SDN network design can leverage this feature-based comparison to match each controller's attributes against their requirements. For example, open network operating systems excel in scalability due to their distributed core, enhancing dependability and availability. On the other hand, OpenDaylight shines in its robust support for legacy southbound protocols, making it ideal for diverse enterprise contexts. For small-scale campus networks, Ryu's Python foundation streamlines installation and its centralized core minimizes inter-controller latency. However, it's vital to note that performance effectiveness can't solely rely on feature comparison; the controller selection depends on a blend of feature capability and performance aligned with the intended application.

2.2 Software-Defined Network Controller Placement Algorithms

The division between the control plane and data plane in software-defined networking introduces additional challenges. One pivotal question that emerges when constructing an SDN network revolves around the optimal placement of controllers. The subject of Controller Placement Problem (CPP) assumes great significance in the realm of SDN, and its exploration dates back to 2012 [62]. For smaller to medium-scale networks, CPP rarely poses a concern, as a single controller can capably oversee these networks. However, the landscape changes dramatically in expansive enterprise networks like SD-WAN, where multiple controllers must be strategically situated. Often referred to as the NP-hard problem, CPP mirrors the location analysis or facility locating dilemma. Arbitrarily placing controllers anywhere within the network is counterproductive. Such an approach not only escalates the overhead delay of network services but also undermines overall network performance [63]. The challenge lies in determining the most fitting locations for controllers to ensure efficient network management and optimal performance. The principal objective of this study is to determine the most optimal controller placement for SD-WAN infrastructure while addressing a multitude of conflicting objectives. To identify the existing gaps and contextualize the research within the domain, a comprehensive literature review on controller placement is undertaken. Moreover, the author aims to leverage insights from prior research for comparative purposes subsequent to the analytical assessment of the proposed solution. Ultimately, the intention is to showcase the potential outcomes of the present study through a juxtaposition with the findings of previous research.

In this study, the controller placement algorithms have been systematically categorized based on their optimized criteria. These criteria fall into five distinct categories: (a) Minimization of network latency; (b) Maximization of resilience and reliability; (c) Load balancing; (d) Reduction of infrastructure costs and energy consumption, and (e) Multi-objective strategy. Additionally, the enhancement of existing SDN controller placement algorithms through heuristic approaches is explored. Visual representations of the optimized criteria and the current placement method are depicted in Figures 2.3 and 2.4, respectively.

The ensuing subsections delve into the pertinent literature concerning controller placement algorithms, expounding on their respective optimized criteria. Moreover, the insights gained from these studies significantly contribute to shaping the trajectory of this thesis. A comprehensive summary and conclusion section will encapsulate these findings, as illustrated in Figure 2.3.

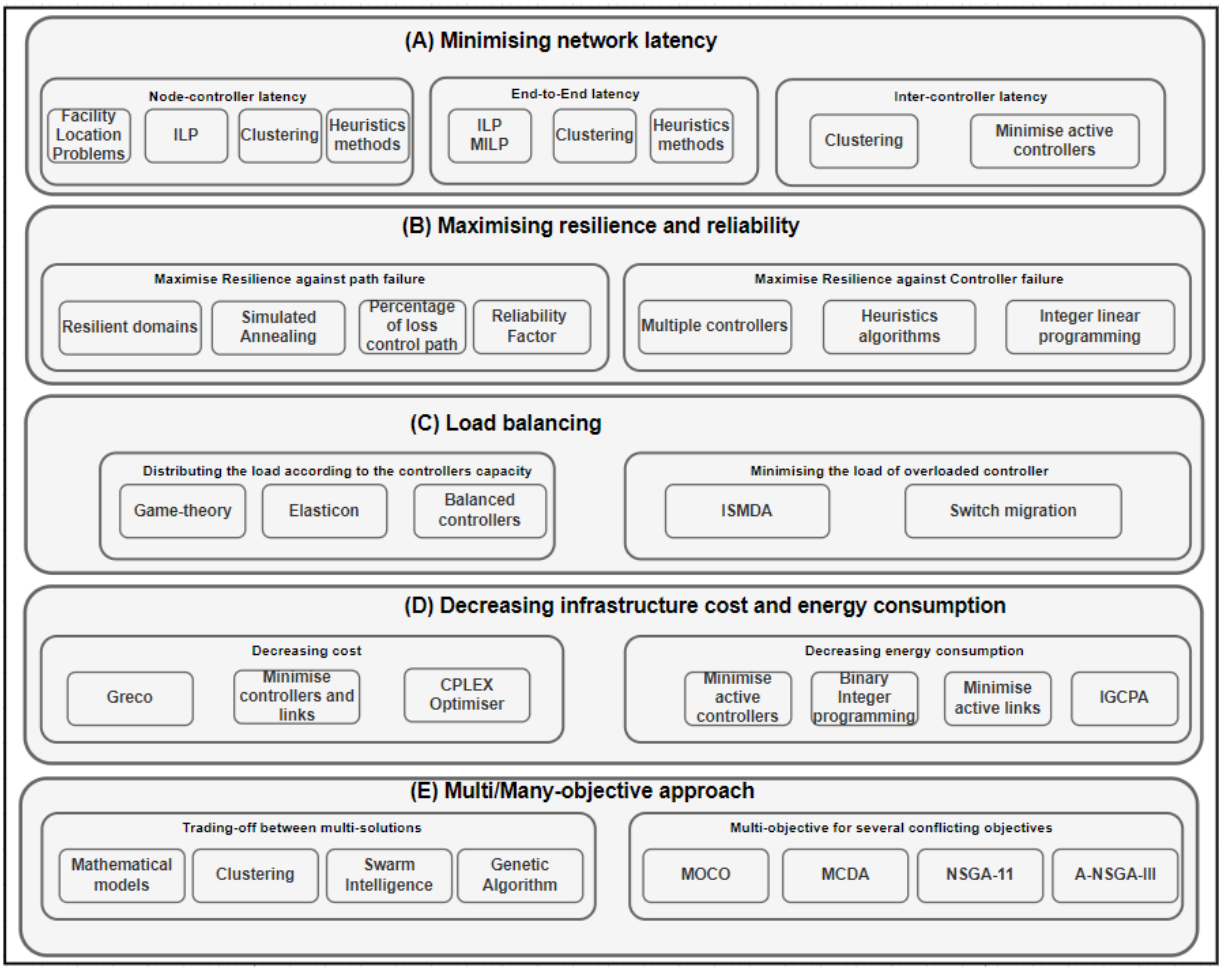

Figure 2.3: Overview of Existing Optimization Algorithms Based on Different Performance Metrics

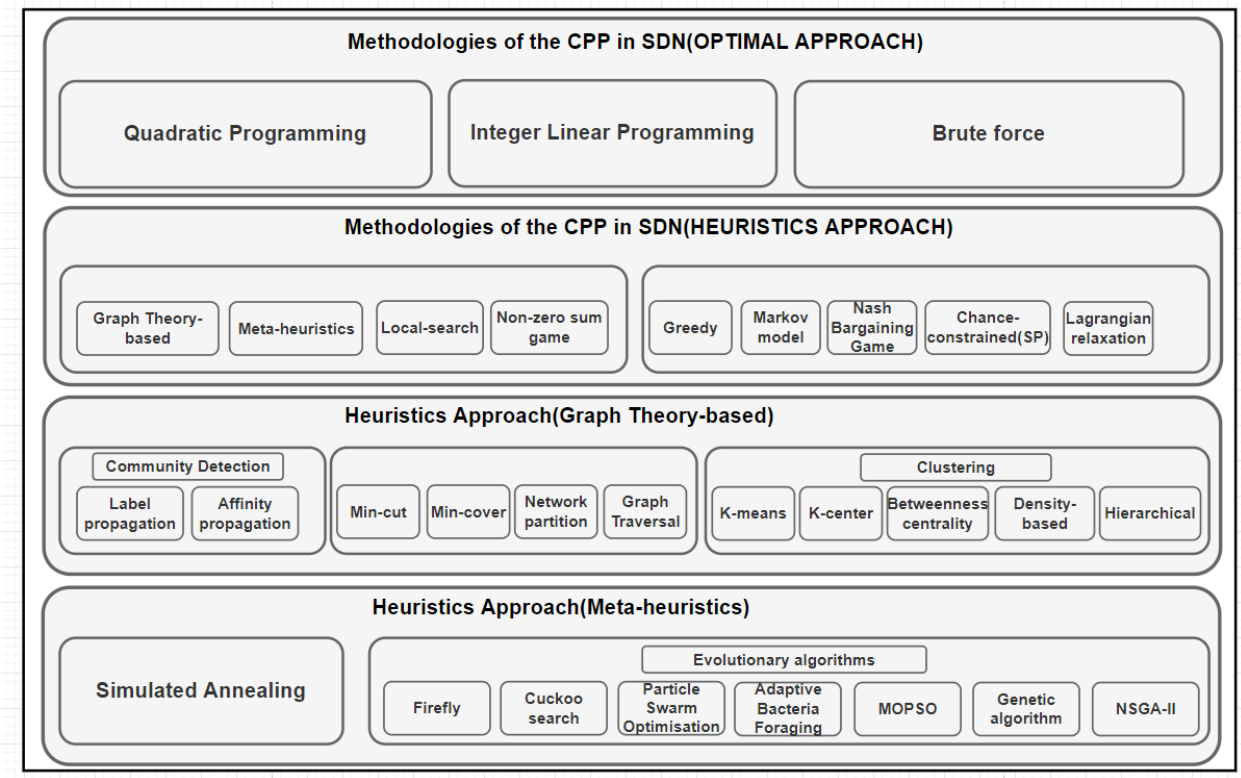

Figure 2.4: Existing controller placement algorithms

2.2.1 Minimising Network Latency

In an SDN-based architecture, controller placement doesn't have a direct impact on the latency of all packets generated by data plane devices. Instead, it notably influences the latency of the first packet, often termed as the 'final flow setup time.' This encompasses the comprehensive duration required for transferring the initial packet from source to destination, encompassing the time taken to initiate a request, the controller's duration to establish forwarding rules across switches along the source and destination paths, along with any network latency encountered during packet transmission [64]. The researchers who prioritize latency as their primary objective for controller placement can be categorized into three distinct groups, as illustrated in Table 2.4:

- Node-Controller Latency
- End-to-End Latency
- Inter-Controller Latency

These researchers operate under the assumption that the optimal approach to optimize controller placement is by minimizing the switch-to-controller latency. Notably, one of the pioneering researchers to investigate the impact of controller placement on average and worst-case node-to-controller latencies within private networks is cited in [20]. These researchers developed an algorithm that utilizes an exhaustive

approach to explore every conceivable controller combination for determining the optimal controller placement. The exhaustive method necessitates the algorithm to examine all possible controller combinations, potentially leading to increased computational complexity.

Table 2.4: The sub-classes of the publication that considers latency to optimize SD-WAN controllers

Class No	**Sub-Optimization criteria**	**Publication No**	**Reference**	**Algorithm**
1	Switch-Controller latency	1	(Heller et al. 2012)	Farthest-point clustering
		2	(Bari et al. 2013)	Integer Linear Programming
		3	(Penna et al. 2014)	Modifies K-clustering
		4	(Tuncer et al. 2015)	Clustering
		5	(Tanha et al. 2016)	Capacitated k-center
		6	(Wang et al. 2016)	Improved K-Means
		7	(Zhao et al. 2017)	Exempler-clustering
		8	(Sahoo et al. 2017)	PSO and Firefly
		9	(Sahoo et al. 2018)	CAMD
		10	(Mamushiane et al. 2021)	PAM and Gap Statistics
2	End-to-end latency	11	(Zeng et al. 2016)	Integer Linear Programming
		12	(Guodong et al. 2017)	Optimized K-mean algorithm
		13	(He et al. 2017)	Spectral clustering and MIP
		14	(Sood and Xiang, 2017)	Analytical model
3	End-to-end latency	15	(Nagano and Shinomiya, 2015)	Clustering
		16	(Zhang et al. 2016)	Analytical model
		17	(Han et al. 2016)	Exhaustive search algorithm
		18	(Zhu et al. 2017)	Adapted K-mean
		19	(Li et al. 2018)	Approximation algorithm

The primary finding of this study suggests that a single controller is satisfactory to fulfill network demands. Nonetheless, the author acknowledges that this falls short when considering fault tolerance requirements. The authors in [65] and [66] built upon the work initiated by [20]. In [67], the author introduced a clustering technique aimed at minimizing the number of hops between controllers and the controlled switches. Additionally, the authors of [68], [69], and [70] also employed clustering approaches to reduce latency between controllers and their controlled switches. In conclusion, the authors in [71] introduced an evolutionary-based approach to determine the optimal controller placement, focusing on

minimizing latency between controllers and their related switches. It's worth noting that the first group of researchers did not consider other types of latencies during the optimization of controller placement, even though these latencies also play a crucial role in the network's overall performance.
The second group of researchers focuses on minimizing end-to-end latency by considering various factors such as controller processing, queue latency, link propagation, and switch transmission [64]. Their work, including studies like [72], [73], and [64], takes a comprehensive approach by accounting for multiple latency types when optimizing controller placement and calculating end-to-end latency. Additionally, in [74], the authors introduced the concept of shifting the controller placement problem to a controller selection problem. They emphasized the importance of dynamically modifying logical controllers using a topology-independent, computationally efficient placement procedure. However, it's worth noting that while this second group of researchers proposed suitable positions for dedicated network zones, they overlooked inter-controller latency. This oversight rendered their placement strategy less suitable for SD-WAN controller placement when controllers need to communicate with each other.
The third group of researchers delved deeply into the realm of end-to-end latency. Beyond considering just switch-to-controller delay, they also factored in inter-controller latency. For instance, in [75], the authors illustrate the substantial impact of inter-controller communication on overall network performance. Moreover, the authors of [76] highlighted the following key insights: Firstly, the choice of consistency techniques employed among controllers, such as Zookeeper and the Raft consensus algorithm, along with the selected consistency level, significantly influences controller response times. Secondly, they emphasized the necessity of incorporating inter-controller latency into the calculation of end-to-end latency. These endeavors were complemented by three additional algorithms - switch-to-controller [77] [78] and [79], each striving to minimize latency between inter-controllers and switch-to-controller. Consequently, the contributions of this third group of researchers led to more empirically grounded insights, ultimately demonstrating an improved controller placement strategy in terms of latency performance.

2.2.2 Maximising Resilience and Reliability

The failure to install forwarding rules in switches can lead to network disruptions, posing a significant threat to network stability and reliability. Furthermore, delaying the installation of these forwarding rules can exacerbate the situation, making it even more critical to optimize the controller placement problem. This optimization is essential not only to enhance resilience in the event of control layer failures but also to significantly improve network reliability [3, 80]. Numerous research studies have been dedicated to optimizing the

placement of SD-WAN controllers with a particular emphasis on resilience and reliability measures. These investigations can be broadly categorized into two groups. The first group of researchers concentrates on bolstering resilience in scenarios where there's a failure in the connection between the controller and switches. Conversely, the second group of researchers is primarily concerned with enhancing reliability in situations where the controller itself experiences a failure. For further details, please refer to Table 2.5.
The first group of researchers explored the potential of clustering techniques to create resilient domains. In [80], the authors initially examined how controller placement impacts the resilience of connections between switches and controllers. Subsequently, [81] and [82] improved upon the concept of clustering the network into resilient domains. Interestingly, the authors noted that extending SD-WAN can inadvertently lead to inter-controller broadcast storms, even though this wasn't their intention. Building on this concept, [75] and [83] employed partitioning strategies to enhance network resilience. However, a drawback of the resilient clustering approach is its tendency to produce unbalanced clusters [84]. On

Table 2.5: The sub-classes of the publication that considers resilience and reliability to optimize SD-WAN controllers

Class No	**Sub-Optimization criteria**	**Publication No**	**Reference**	**Algorithm**
1	Maximize resilience against path failure	1	(Zhang et al. 2011)	Min-cut clustering
		2	(Hu et al. 2013)	Percentage of loss control
		3	(Guo and Bhattacharya, 2013)	Robust tree (clustering)
		4	(Jimenex et al. 2013)	Robust tree
		5	(Hu et al. 2014)	Simulated Annealing
		6	(Jimenex et al. 2014)	Improved K-critical
		7	(Xiao et al. 2014)	Spectral clustering
		8	(Liu et al. 2016)	K-Means
		9	(Aoki and Shinomiya, 2016)	Clustering
2	Maximize Resilience against Controller failure	10	(Muller et al. 2014)	Integer Linear Programming
		11	(Perrot and Reynaud, 2016)	Integer linear programming
		12	(Tanha et al. 2016)	Exhaustive search
		13	(Killi and Rao, 2017)	MILP and Simulated Annealing
		14	(Bannour et al. 2017)	NSGA-II and PAM
		15	(Tanha et al. 2018)	Clique-based
		16	(Killi et al. 2019)	Mathematical model
		17	(Calle et al. 2021)	Mixed integer programming

a different front, [85], [84], and [86] introduced a novel approach to tolerate path failures, proposing a robust control tree to address this issue. Nevertheless, due to the computational complexity of identifying the optimal tree, these methods might not be suitable for deployment in complex network structures. Another angle of research, exemplified by [87], examined controller placement concerning path failures. These authors introduced a new metric called the "percentage of loss control path" to maximize the control layer's reliability. According to their findings, controller placement could enhance the control plane's reliability without necessarily causing unacceptable latencies between the controller and its as-

sociated switches. Finally, [3] recognized the challenge of link outages and introduced the concept of the "Reliability Factor" (RF), which utilizes the average distance of multi-paths between controllers and their associated switches. However, all these investigations primarily focused on addressing link and node failures, while neglecting controller failures. In pursuit of greater reliability in controller placement, some researchers have explored methods to assist in recovering from control layer faults, particularly controller failures. A common approach followed by authors in studies such as [88], [89], [69], [90], [91], and [92] involves the use of a single backup controller. They employ a Mixed Integer Linear Programming (MILP) and simulated annealing approach, distributing a set of three controllers to each cluster. This controller placement strategy, focusing on resilience, proves to be more robust against network failures

2.2.3 Load Balancing

In the pursuit of maintaining optimal network performance, ensuring a balanced load among distributed controllers is crucial. Consequently, researchers aim to identify the most effective controller placement strategies for load balancing, with the objective of dynamically distributing the controller's workload. Load balancing can be achieved through two primary methods: assessing the controller's capacity and ensuring load distribution accordingly, or employing the switch migration approach to alleviate the burden on overloaded controllers. The publications that have explored controller placement with a focus on load balancing, utilizing both the controller's capacity and the switch migration approach, are summarized in Table 2.6.

Table 2.6: The sub-classes of the publication consider load balancing to optimise SD-WAN controllers using controller capacity and switch migration approach.

Class No	**Sub-Optimization criteria**	**Publication No**	**Reference**	**Algorithm**
1	Distribute load equally	1	(Rath et al. 2014)	Non-zero game theory
		2	(Yao et al. 2014)	Capacitated K-center
		3	(Aoki et al. 2015)	Minimum cut (clustering)
		4	(Aoki and Shinomiya, 2015)	Spectral clustering
		5	(Sanner et al. 2016)	Hierarchical clustering
2	Miminize the load of the overloaded controller (switch migration)	6	(Yao et al. 2015)	K-mean clustering
		7	(Hedge et al. 2017)	Exhaustive search
		8	(Adekoya et al. 2020)	ISMDA

The authors in [93] utilize a non-zero game theory method to dynamically adjust the placement of controllers. Similarly, the authors in [94], [95], and [96] explored graph partitioning techniques to create

balanced clusters. It's worth noting that the clustering methods investigated in previous papers can achieve controller load balancing in a static network environment. However, due to the high computational cost of these approaches, achieving load balancing of controllers in a dynamic network context is challenging. Other related works, such as [97], [98], and [99], opt to position the controllers in fixed locations and then migrate the switches among them to balance the controller load. However, it's important to note that while this approach is effective for minimizing changes in network load, it may not be sufficient for handling significant fluctuations in network traffic or topology. In particular, the last two works determine controller locations based on cluster shapes, which can change depending on the network's status.

2.2.4 Decreasing Infrastructure Cost and Energy Consumption

The goal in SDN is to minimize both Capital Expenditures (CAPEX) and Operating Expenses (OPEX) to ensure the financial viability of deployment and operations. This entails decreasing infrastructure and operating expenses, alongside efforts to minimize energy consumption, while simultaneously optimizing network resources to enhance efficiency without compromising performance [100]. Consequently, several researchers have explored the concept of cost-effective controller placement, aiming to maximize controller utilization while minimizing the number of active controllers (refer to Table 2.7).

Table 2.7: The sub-classes of publications that result in controller placement that lowers SD-WAN costs and power consumption.

Class No	**Sub-Optimization criteria**	**Publication No**	**Reference**	**Algorithm**
1	Trading-off between multi-objectives	1	(Sallahi and St-Hilaire, 2015)	Exhaustive search algorithm
		2	(Sallahi and St-Hilaire, 2017)	Exhaustive search algorithm
2	Combining multi-objective in single solution	3	(Auroux et al. 2014)	Minimize active controllers
		4	(Auroux et al. 2015)	Optimal placement among limited locations
		5	(Ruiz-Rivera et al. 2015)	Minimize active links
		6	(Hu et al. 2017)	Genetic algorithm

The authors of papers [101] and [102] have created a mathematical model for determining the most cost-effective controller placement in newly constructed or upgraded networks. While effective in reducing network costs, this model doesn't account for inter-controller delay during controller placement and has high computational complexity. Additionally, the authors of [103], [104], and [100] have developed

placement algorithms aimed at minimizing power consumption by reducing active controllers. However, these approaches may increase latency and do not consider inter-controller delay.

2.2.5 Combinatorial Optimization Approach

Combinatorial optimization problems come up in many different areas, such as decision-making, planning, telecommunications, transportation, routing, and scheduling [105]. A multi-objective approach refers to a problem-solving or decision-making strategy used to address scenarios where there are multiple, often conflicting, objectives or criteria that need to be simultaneously optimized. In such situations, the goal is to find a set of solutions that represents a trade-off between the different objectives because it's often impossible to optimize all objectives simultaneously due to their conflicting nature.

This approach is commonly used in various fields, including engineering, economics, operations research, and more [106]. It involves mathematical modeling and optimization techniques to find a set of solutions (known as the Pareto front) that are considered optimal with respect to the different objectives. Decision-makers can then choose from this set based on their preferences, balancing the trade-offs between the objectives to make informed decisions. This study is centered on addressing the intricate challenge of SD-WAN controller placement, which serves as a quintessential example of a combinatorial optimization problem. The crux of the controller placement problem lies within the domain of discrete optimization, characterized by the delineation of distinct and finite solutions [8]. Conversely, even cutting-edge algorithms like NSGA-III predominantly excel in tackling continuous optimization quandaries [29]. In the realm of continuous optimization, where variables possess continuous value ranges, the solution space extends infinitely. In stark contrast, discrete optimization grapples with variables that assume specific, discrete values, yielding finite and distinct solution sets. The selection of optimization paradigms is intrinsically tied to the inherent characteristics and constraints of the problem at hand. To ensure the derivation of pragmatic and viable solutions while guarding against in-feasibility during critical genetic algorithm operations such as cross-over and mutation, and to circumvent the unwarranted generation of redundant solutions, an innovative repair-operator-based mechanism has been meticulously developed. The pivotal roles of cross-over and mutation in catalyzing genetic diversity cannot be overstated. These operations perpetually explore uncharted solution territories while tenaciously safeguarding favorable traits inherited from preceding generations. This ingeniously crafted mechanism effectively supplants continuous optimization elements in the NSGA-III algorithm with discrete optimization attributes, thus rendering it exceptionally well-suited for the resolution of discrete controller placement challenges. This is accomplished while upholding its remarkable proficiency in addressing multifaceted objectives.. There are several different goals that need to be optimized at the same time to get the best controller placement. Prominent metaheuristic algorithms, including the ANSGA-III [7], NSGA-II [107], and MOPSO [108], have been proposed to seek solutions that approach optimality. Nonetheless, these methodologies exhibit certain limitations that warrant prompt attention. Existing techniques grapple with issues such as computational inefficiency, a lack of capacity to acquire combinatorial optimization heuristics, and suboptimal controller placement optimization, especially when dealing with more than three objectives (with the exception of ANSGA-III).

This study's findings underscore a common challenge across optimization algorithms, namely, the substantial computational demands associated with MOPSO, NSGA-II, and ANSGA-III (refer to Chapter 5, subsection 5.6.3, figure 5.19). Despite ANSGA-III's notable scalability in SD-WAN controller placement, none of the existing solutions, including MOPSO and NSGA-II, or even the adapted ANSGA-III, possess autonomous heuristic learning capabilities for tasks like predicting the optimal number and placement

of controllers in combinatorial optimization scenarios, such as SD-WAN controller placement. These advanced algorithms (ANSGA-III, NSGA-II, and MOPSO) rely on handcrafted heuristics to make complex decisions that are either computationally expensive or lack well-defined mathematical formulations. Thus, there's a pressing need for intelligent-based solutions that can swiftly compute both the optimal controller number and their strategic placement [109, 106, 110].

2.2.6 Multi-Objective Approach

This research now delves into an in-depth exploration of the background study concerning multi-objective approaches. The final group of researchers in Table 2.7 believes that focusing on a single goal (such as cost, energy, latency, load, or reliability) won't lead to an optimal controller placement. Instead, they consider multi-objectives in two ways: first, by trading off objectives, and second, by merging multi-objectives into a single placement with adaptive values (see Table 2.8 and 2.9). The authors in [21] tackle all forms of placement using the NP-hard brute-force search algorithm. They introduce POCO, also known as the Pareto-based Optimal Controller Placement tool. Pareto-based optimal controller placement refers to a strategic approach employed for determining the most favorable controller locations within a network, while simultaneously addressing multiple competing objectives [21]. Within this framework, 'local' solutions denote those that exhibit optimality or efficiency within specific problem space regions. These solutions excel within their localized domains but may not necessarily translate to global optimality.

The terms 'maximum minimum' and 'minimum maximum' encapsulate the essence of balancing diverse objectives [15]. In the realm of multi-objective optimization, the objective often revolves around optimizing one metric (maximum) while minimizing another (minimum). 'Maximum minimum' signifies the pursuit of solutions that maximize one objective while minimizing another, while 'minimum maximum' represents the converse—minimizing one objective while maximizing another. Notably, even objectives aimed at maximization are transformed into minimization objectives through the introduction of negative values. This alteration fosters a harmonized optimization process, wherein all objectives are uniformly minimized. This unification eliminates the need to differentiate between maximization and minimization objectives, streamlining the optimization algorithm.

In the context of Pareto-based optimal controller placement, the overarching objective is to identify a set of controller positions that offer an optimal compromise between conflicting objectives. This pursuit takes into consideration both localized and global optima, navigating the intricate interplay between maximizing and minimizing various performance metrics. This POCO Matlab framework is enhanced with a Visual Interface to facilitate the presentation of placement results in dynamic conditions [21]. In their subsequent study, [15] utilize POCO and PlanetLab [111] to optimize controller placement in dynamic environments. Additionally, [112] presents a heuristic strategy for using the POCO tool in their study. The POCO algorithm could be a viable option for small and medium-sized networks (not exceeding fifty nodes and seven controllers). However, due to its high resource and time requirements, it may not be suitable for large networks and dynamic implementations [8]. The proposed heuristic strategy simplifies the placement of POCOs, yielding acceptable but less precise outcomes [113].

Consequently, when employing the heuristic method, a greater number of controllers might be placed in larger networks (e.g., fifty nodes and fifteen controllers). This prior heuristic technique is further refined in a subsequent work by [113] to achieve more precise placement. However, it's important to

Table 2.8: The sub-classes of publications that result in Multi-objective controller placement.

Class No	**Sub-Optimization criteria**	**Publication No**	**Reference**	**Algorithm**
1	Trading-off between multi-objectives	1	(Hock et al. 2013)	K-centers and K-median algorithms
		2	(Gebert et al. 2014)	K-median algorithms
		3	(Lange et al. 2015)	Heuristics approach
		4	(Gebert et al. 2015)	Pareto capacitated K-medoids
		5	(Naning et al. 2016)	Analytical model
		6	(Hollinhurst et al. 2016)	Local search and Adapted K-means++
		7	(Xu et al. 2022)	Genetic Algorithm
		8	(Radam et al. 2022)	Harmony search and PSO Algorithm
		9	(Alouache et al. 2022)	Multi-Objective Genetic Algorithm
		10	(Aravind et al. 2022)	Simulated Annealing
		11	(Hemagowri et al. 2023)	Artificial fish algorithm
2	Combining multi-objective in single solution	12	(Jalili et al. 2015)	Adapted NSGA-II
		13	(Borcoci et al. 2015)	Multi-criteria Decision Algorithm
		14	(Gao et al. 2015)	Particle Swarm Optimization
		15	(Hu Bo et al. 2016)	MOGA
		16	(Liao et al. 2017)	Density-based clustering
		17	(Bannour et al. 2017)	NSGA-II and PAM
		18	(Zhang et al. 2018)	MOCP
		19	(Kuang et al. 2018)	Hierarchical K-means
		20	(Tanha et al. 2018)	Heuristics approach

Table 2.9: The sub-classes of publications that result in Multi-objective controller placement.

Class No	Sub-Optimization criteria	Publication No	Reference	Algorithm
2	Combining multi-objective in single solution	21	(Ahmadi et al. 2018)	Heuristics approach algorithms
		22	(Mohanty et al. 2019)	Metaheuristics techniques
		23	(Liao et al. 2021)	Genetic Algorithm and PSO
		24	(Adekoya et al. 2022)	ANSGA-III
		25	(Bagha et al. 2022)	Seagull Optimization Algorithm
		26	(Sapkota et al. 2022)	Naked Mole-Rat Algorithm
		27	(Thalapala et al. 2022)	Wisdom of Artificial crowds
		28	(Kazemian et al. 2022)	Antilon Optimization Algorithm
		29	(Qaffas et al. 2023)	Cuckoo search algorithm

note that the size of the networks and the number of controllers in the tests conducted are similar to those of the POCO tool's tests, which do not represent truly large networks. Additionally, analytical research employing the POCO algorithm is presented in the work by [114]. From this, it can be deduced that placing controllers based on their resilience can fulfill system needs such as load balancing, dependability, and latency. Most scientific studies utilizing the trade-off methodology suggest that local search and k-means++ approaches are more feasible for large organizations [114]. However, it's worth noting that some drawbacks are common among these works, including the fact that selecting between multiple placements may not be an appropriate solution, controller overload during reassignment of nodes to the closest controller in case of a failure is not considered, and the latency between the switch and controller is depicted solely by the propagation latency.

In their research, the authors of [115] have introduced a multi-controller load balancing model with three primary optimization objectives. The model aims to achieve controller load balancing, reduce communication latency between the control plane and forwarding plane, and minimize switch migration costs. This approach utilizes dynamic switch migration to balance the load among controllers, resulting in decreased network latency and migration costs. However, it's important to note that the genetic algorithm proposed by these authors has demonstrated challenges related to scalability and computational efficiency. Consequently, further testing is warranted to assess the practicality and effectiveness of this algorithm in real-world scenarios. Similarly, in a study focusing on enhancing network performance by addressing the

controller placement problem, researchers employed a multi-controller-based SDN approach [116]. This approach uses a hybrid strategy that combines the harmony search algorithm with the PSO algorithm to determine the optimal controller placement. While it successfully minimizes communication latency and optimizes controller placement, it does have drawbacks. The approach involves several heuristic strategies in the optimization process, which can lead to computational inefficiency. Additionally, it doesn't possess the capability to effectively learn the heuristics required for solving combinatorial optimization problems. Additionally, the authors in [117] focus on controller placement for Vehicular networks coupled with 5G. They employ a Multi-Objective Genetic Algorithm to optimize controller placement considering objectives like latency, load balancing, and robustness. While this method can compute controller placement and identify nearly optimal solutions, it faces challenges related to scalability, computational efficiency, and the ability to learn heuristics. The authors in [118] propose a simulated annealing algorithm to address controller placement issues. This approach aims to reduce execution time while optimizing controllers using simulated annealing. However, it may struggle to efficiently find diverse solutions when dealing with several conflicting objectives and could encounter computational inefficiency and scalability issues. Finally, the authors in [119] present a hybrid evolutionary algorithm optimized controller placement approach that utilizes artificial fish algorithms and chaotic Gaussian maps. This approach optimizes controller placement based on conflicting metrics but may face difficulties finding diverse solutions in scenarios with numerous conflicting objectives. It could also be computationally intensive and lacks the ability to learn the heuristics of combinatorial optimization problems. In summary, these approaches make significant contributions to optimizing controller placement. However, they often face challenges in terms of scalability, computational efficiency, and effectively addressing a wide range of objectives.

Transitioning to the second approach, which revolves around the fusion of multiple objectives, this research identifies notable studies like [107], [120], and [121]. These works employ a multi-objective placement strategy, skillfully adjusting the distribution based on the weight or threshold assigned to each objective. As articulated by [122], a pioneering density-based clustering technique emerges to orchestrate controllers in alignment with diverse objectives encompassing latency, load balancing, and reliability. However, it's imperative to note that this method predominantly centers on propagation latency, potentially overlooking the meticulous minimization of inter-controller latency. Furthermore, the application of a capacity-aware clustering technique occasionally begets clusters with fragmented nodes, as illuminated by [14], who accentuate the necessity of simulating traffic flows among controllers for ensuring consistent placement. Lastly, several studies place great emphasis on the optimization of factors like reliability, load balancing, and node-to-controller latency. This is exemplified in the scholarly contributions of [123], [124], and [125].

In the research detailed in [126], the authors explore the concurrent optimization of multiple objectives for the purpose of identifying optimal controller placements using their proposed metaheuristic approach. These objectives primarily pertain to load balancing and the minimization of node-to-controller latency. To achieve this objective, they employ heuristic techniques grounded in a genetic algorithm, which is a key component of their approach. This algorithm assists in partitioning the network topology based on a predefined number of controllers. The fitness function employed amalgamates the weights of the minimal spanning tree for each partition with the weights of the average controller load deviation. The strategic placement of controllers is carefully designed to shorten the shortest path to each switch within its designated partition, achieved by siting controllers at the centroids of their respective partitions. The

proposed approach undergoes rigorous testing on various topologies, including those from Internet2 and the Internet Topology Zoo.

In the research outlined in [127], the authors undertake the simultaneous optimization of multiple conflicting objective functions to ascertain the optimal controller placements in SD-WAN. These objectives encompass crucial factors like controller load balancing, network reliability, and maximum switch-controller latency. The overarching goal is to harmonize these diverse objectives into a single, unified optimization problem. To tackle this intricate challenge, the authors introduce the Adaptive Bacterial Foraging Optimization (ABFO) algorithm. A comprehensive comparative analysis is conducted, juxtaposing the newly introduced ABFO algorithm with a previous method elucidated in [87]. This evaluation spans three distinct network topologies.

In the research conducted by [9], they introduce a Particle Swarm Optimization (PSO) algorithm to address the challenge of determining optimal controller placements while simultaneously considering multiple concurrent objectives. The optimization goals revolve around the capacity limitations of controllers and include objectives like minimizing node-controller latency and inter-controller latency. To evaluate the effectiveness of their approach, the authors employ key performance metrics such as maximum latency and computational cost. In their assessment, they compare their PSO algorithm against various placement strategies, encompassing a greedy strategy [126], an integer linear programming approach [101], and a random placement strategy. In the research conducted by [128], they present an innovative approach known as Network Clustering Particle Swarm Optimization (NCPSO). This approach is designed to optimize controller load while simultaneously adhering to controller load balancing and node-controller latency constraints. Notably, the NCPSO technique outperforms other methods, including k-center [128] and capacitated k-center [20], across several critical factors such as the number of controllers, propagation latency, load balancing, and controller utilization.

The studies conducted by [11] and [107] introduced a genetic algorithm-based approach to simultaneously optimize multiple competitive objectives in controller placement. These objectives include load balancing, node-to-controller latency, and inter-controller latency, making it a multi-objective problem. To address this challenge, the researchers harnessed the power of NSGA-II. NSGA-II [129] is a widely acclaimed metaheuristic algorithm renowned for its ability to tackle complex optimization problems characterized by competing objectives. Its versatility extends seamlessly to the realm of SD-WAN, where strategically placing network controllers plays a pivotal role in enhancing a variety of performance metrics. Within the intricate landscape of SD-WAN controller placement, NSGA-II proves to be an invaluable tool. It excels at orchestrating solutions that harmonize conflicting objectives. NSGA-II accomplishes this feat through the skillful application of non-dominating sorting techniques and the utilization of crowding distance metrics to evaluate potential solutions. Their framework, configured with six predefined controllers, was compared against the POCO framework [21]. The evaluation was based on efficiency (run-time) and effectiveness (near-optimality), utilizing a dataset from the topology zoo. In subsequent work, [130] enhanced the approaches of [11] and [107] by introducing several improvement mechanisms. These included a greedy initialization approach, an enhanced performance evaluation metric, and IGD-inverted generated distance. Building on these foundations, [131] further refined the methodology based on the works of [130]. They introduced a multiple-objective capacity-aware controller placement problem and incorporated a constraint handling mechanism inspired by the work of [132]. The techniques

proposed in these studies demonstrated superior performance compared to the approaches proposed by [112] and [9], especially concerning the coverage parameter [133]. The study conducted by [120] identified a set of performance metrics that exhibited inherent conflicts. These metrics encompassed various aspects, including node-to-link controller failure, node-to-controller latency, load balancing, inter-controller latency, and multi-path node-controller connectivity. To tackle the complexity of optimizing these diverse and occasionally conflicting objectives, the authors introduced a Multi-Criteria Decision Algorithm (MCDA) framework. This approach was designed to enable the simultaneous optimization of these objectives. The MCDA framework was constructed based on a benchmark decision algorithm [134], and its efficacy was demonstrated through numerical examples. The study conducted by the authors in [135] delved into the intricate realm of the multi-objective controller placement problem, specifically exploring the intricate balance among multiple conflicting objectives. These objectives encompass load balancing, inter-controller latency, and node-to-controller latency. To address the challenge of optimizing these competing objectives, the authors introduced the Nash bargaining model. This model aimed to identify the Pareto optimal solution that strikes a harmonious compromise among the various objectives. The research presented different problem formulations, including those focusing solely on switch-to-controller objectives, inter-controller latency, and simultaneous optimization of multiple objectives. By juxtaposing the multi-objective approach against a comparable single-objective formulation, the study effectively highlighted the advantageous nature of their proposed solutions in achieving a judicious trade-off among the conflicting objectives.

In [14], the researchers conducted a comprehensive evaluation of scalability and reliability performance metrics, both acknowledged as significant challenges within the centralized control plane of software-defined networks. To address these concerns, the study compared two multi-objective optimization techniques that jointly optimize performance and reliability metrics. The first technique, a variant of a genetic algorithm known as NSGA-II (non-dominated sorting genetic algorithm-II), was considered, alongside the second technique which employed an unsupervised machine learning approach called Partitioning around medoids (PAM). The study aimed to concurrently optimize controller load imbalance, controller failure reliability, and switch-to-controller latency. Through experimentation, the researchers found that PAM consistently outperformed NSGA-II across various scenarios and random topologies of different sizes. This comparison demonstrated the superior performance of the PAM technique in optimizing the specified objectives.

In the study by [136], a heuristic algorithm employing simulated annealing was introduced to concurrently optimize conflicting objectives in controller placement problems. The primary objective was to minimize mean node-controller latency. This approach was compared against the k-centre and k-medoids algorithms using the zoo topology. Similarly, in [137], spectral clustering methods were evaluated along with k-median and k-centre algorithms for controller placement problems. The objective was to reduce both node-controller delay and inter-controller delay. Furthermore, in [138], multiple objectives in controller placement were addressed by examining two population-based evolutionary algorithms: Firefly and Particle Swarm Optimization (PSO). These algorithms aimed to optimize node-to-controller and inter-controller latencies. The study concluded that Firefly outperformed PSO based on analyses conducted on the zoo topology.

The Pareto Optimal Controller Placement framework, developed by [15] and [21] using a simulated annealing algorithm, extended its investigation into controller location parameters beyond network latency. It considered metrics such as controller-to-controller latency in both maximum and worst-case scenarios, switch-to-controller latency in similar scenarios, resilience in controller failure, and load balancing among

controllers. To enable the incorporation of multiple objectives into the optimization process, they introduced the Pareto Simulated Annealing meta-heuristic. The POCO framework employed an exhaustive approach, analyzing the entire solution space for a limited number of controller installations while considering potential link or node failures. While known for providing optimal solutions, it has drawbacks in terms of computational expense and suitability for large-scale networks. Additionally, the predetermined number of controllers may lead to either excessive deployment costs or decreased network performance.

Algorithmic solutions for SD-WAN controller placement have seen the emergence of several multi-objective evolutionary algorithms and metaheuristic algorithms. These approaches are effective in identifying multiple Pareto optimal solutions within a single simulation run, a key feature contributing to their effectiveness. One of the early EA algorithms adapted for multi-objective problems is Non-dominated Sorting Genetic Algorithm, introduced by [139]. Subsequent versions, such as NSGA-II and NSGA-III, introduced performance enhancements. In the same vein, [8] proposed a multi-start hybrid NSGA to address multi-objective controller placement challenges in large networks. This approach utilizes a greedy heuristic for generating an initial population and incorporates intelligent mechanisms to enhance diversity and intensification within the non-dominated set. However, it's important to note that this strategy relies heavily on computational power and lacks the ability to learn combinatorial optimization heuristics, which distinguishes it from the approach discussed in this research.

The Fast and Exclusive Multi-Objective NSGA-II, introduced by [129], was a pioneering method equipped with innovative operators such as fast non-dominated sorting, crowding distance, elitism, and a crowded comparison operator. NSGA-II proved to be a robust population-based algorithm for tackling multi-objective problems with two or three goals. By efficiently approximating Pareto fronts through a combination of crowding distance diversity and non-dominated sorting elitism, NSGA-II excelled, even in scenarios with non-convex and non-connective Pareto optima. However, it's important to note that NSGA-II's scalability is limited when dealing with more than three objectives. In response to this limitation, [140] introduced NSGA-III, a modified version of NSGA-II capable of handling a broader range of objective functions. NSGA-III built upon the foundation of NSGA-II and extended its capabilities. Leveraging these advancements, [7] proposed ANSGA-III to address the Controller Placement (CP) problem in SD-WAN. ANSGA-III simultaneously optimizes multiple competing objectives for SD-WAN controller placement. However, it's worth noting that this approach is computationally intensive and lacks the ability to learn combinatorial optimization heuristics, a characteristic essential for addressing challenges like SD-WAN controller placement. Additionally, like other methods, ANSGA-III predetermines the number of controllers, which can potentially result in either unnecessary costs or compromised network performance.

In the research conducted by [141], a seagull optimization algorithm (a metaheuristics algorithm) is introduced for controller placement, coupled with the use of fuzzy C-Means clustering, a well-known clustering algorithm, for network partitioning. This combined approach aims to determine suitable controller locations within each network partition. The optimization objectives considered in this study encompass end-to-end delay, inter-controller delay, node-to-controller delay, and controller load. The use of the C-Means clustering algorithm can be effective in grouping controllers that are spatially close to each other, potentially improving network performance. However, the practicality of employing fuzzy C-Means for partitioning the network, particularly in the context of addressing several conflicting objectives, remains

uncertain. Additionally, population-based metaheuristics, as utilized in this work, often encounter scalability issues when simultaneously optimizing multiple conflicting objectives. Furthermore, this approach, when compared to the recommended solution proposed in this thesis, lacks the capability to learn the heuristics of combinatorial optimization algorithms.

In the research conducted by [142], a population-based metaheuristic optimization technique known as Naked Mole-Rat is introduced for optimizing multiple conflicting objectives in the context of SD-WAN controller placement. The objectives considered include inter-controller delay, load balancing, and switch-controller delay. This study compares the performance of the Naked Mole-Rat algorithm with the Bat algorithm, focusing on their computational complexity as the number of controllers increases and their impact on minimizing switch-controller latency. The Naked Mole-Rat algorithm demonstrates the ability to efficiently find near-optimal solutions based on the specified objective functions within a reasonable time frame. However, its scalability remains uncertain. Furthermore, like other population-based approaches, the Naked Mole-Rat algorithm lacks the capability to predict the required number of controllers for deployment and does not possess the ability to learn the heuristics of combinatorial optimization problems. In a similar vein, the authors of [143] introduce the concept of the "wisdom of artificial crowds," which is an evolutionary algorithm designed for controller placement in SDN. The primary objective of this study is to enhance overall network performance by identifying optimal controller positions while minimizing the number of controllers used. The optimization process takes into account factors such as inter-controller delay, controller-switch delay, and reliability metrics. In this approach, artificial agents collaborate by merging their individual candidates to generate a single outcome that outperforms the other candidates in the population. The algorithm proposed in this study outperforms both the K-means algorithm and modified density peak clustering techniques, as reported in the work. While this metaheuristic approach demonstrates efficient performance in optimizing controller placement, it may face challenges when tasked with finding diverse solutions for artificial crowds in situations where multiple conflicting objectives, exceeding three, need to be simultaneously optimized. Additionally, it may be computationally intensive compared to the approach suggested in this thesis due to the involvement of several heuristic strategies (selection, crossover, and mutation) during the training process while finding the optimal locations. Finally, this approach lacks the ability to learn the heuristics of combinatorial optimization problems, such as SD-WAN controller placement, which may impact the time required for the algorithm to complete its execution. Moreover, the research presented in [144] tackles the controller placement problem (CPP) as a multi-objective optimization challenge and introduces the Antlion optimization algorithm to address it. The CPP involves optimizing three objective functions: switch-to-controller delay, controller-controller delay, and the existence of separate connectivity links between nodes and controllers. The Antlion optimization algorithm is a population-based evolutionary technique known for its effectiveness in optimizing conflicting objectives in SDN controller placement. However, this approach may encounter scalability issues when tasked with simultaneously optimizing more than three objectives. Additionally, it lacks the ability to learn the heuristics of combinatorial optimization problems. In comparison to the stochastic computational graph approach proposed in this thesis, the Antlion optimization algorithm, as mentioned in this reference, is recognized for its computational expense due to the various heuristic strategies involved during the training process of the algorithm.

Finally, the study in [145] introduces the challenge of positioning controllers as a multi-objective optimization and proposes the adaptive group-based cuckoo optimization algorithm to address the CPP.

The primary objective of this study is to utilize the adaptive group-based cuckoo optimization algorithm to optimize the efficiency of an SDN-enabled wireless sensor network. This reference considered three objective functions, such as reliability constraints, timing constraints, and the cost of controller installation, for the controller placement problem. This approach may find a near-optimal solution based on the considered objective functions in an efficient time; however, the practicality of this algorithm in terms of scalability is not guaranteed. Moreover, the group-based approach, like the Adaptive group-based cuckoo optimization algorithm suggested in this research, does not possess the capability to predict the required number of controllers for deployment and also lacks the ability to learn the heuristics of combinatorial optimization problems. Compared to the stochastic computational graph approach proposed in this thesis, the Adaptive group-based cuckoo optimization algorithm suggested in this reference is known to be computationally expensive due to the various heuristic strategies involved during the training process of the algorithm.

2.2.7 Artificial Intelligence and Machine Learning Usages in Software Defined Networking

Artificial Intelligence (AI) is a rapidly expanding field that encompasses various sub-fields such as reasoning, machine learning (ML), decision-making, planning, and evolutionary algorithms [146]. Alan Turing's [147] famous Turing Test defines intelligence for a computer as the ability to answer questions in a way that is indistinguishable from a human. To successfully pass this test, a computer must possess sophisticated abilities, such as automated reasoning, ML, computer vision, knowledge representation, and natural language processing [147]. AI investigation has its roots in the mid-1950s, when a summer program at Dartmouth College, planned by Claude Shannon and Martin Minsky, laid the foundation [148]. However, even earlier in 1943, Pitts and McCulloch introduced the initial model for artificial neural networks. Over the years, AI has witnessed notable advancements, giving rise to sub-fields like evolutionary algorithms, expert systems, and fuzzy logic. These advancements have played a crucial role in refining existing AI methods and introducing hybrid intelligent approaches [146].

Within the framework of SDN, AI techniques like machine learning and meta-heuristic algorithms play crucial roles in tasks such as network management, resource allocation, and traffic routing. Additionally, the integration of natural language processing and computer vision enhances the interaction between humans and machines in SDN systems [149]. In the modern era, owing to technological progress and the rapid expansion of mobile communication technologies and the Internet, communication networks have become progressively advanced and complex [149]. To efficiently handle, arranged, maximize, and sustain such communication networks, a substantial amount of information must be taken into account and utilized. Traditionally, using machine learning in closed networks was challenging. However, with the introduction of SDN, network flexibility, agility, and programmability have been revolutionized, offering researchers the chance to explore different facets of the network through software-driven solutions. Machine learning is a potent method that can be employed in SDN architecture to boost network capabilities and improve non-functional aspects like security and performance [150].

Using machine learning techniques to address the placement of controllers in SD-WAN is an emerging approach to optimize network performance and efficiency [146]. In SD-WAN, determining the optimal locations for SDN controllers to effectively manage the network is crucial. Various ML techniques, like particle swarm optimization, reinforcement learning, and genetic algorithm are applied to tackle the complex controller placement problem [151]. These algorithms analyze network data, traffic patterns, and other relevant parameters to identify the most suitable controller positions. By leveraging machine

learning, SD-WAN benefits from reduced communication latency, improved network responsiveness, and efficient resource utilization [151]. ML enables the system to adapt to dynamic network conditions and changing traffic patterns [146]. Stochastic Gradient Descent (SGD) is another optimization algorithm widely used in training machine learning models [152],[153]. In controller placement, SGD is applied to optimize controller locations based on network metrics and objectives. It iteratively updates controller positions to minimize the cost function, considering factors like latency, load balancing, and communication overhead. This leads to efficient controller placement and improved network performance. The computational graph approach [154] is a method that represents relationships between variables and operations in a graph structure. In SD-WAN controller placement, a computational graph can model interactions between network components, controller locations, and various performance metrics. This approach facilitates the analysis and optimization of controller placement decisions through techniques like back-propagation or graph-based optimization algorithms. The computational graph approach offers a systematic and flexible representation of the controller placement problem, making it easier to apply diverse machine learning techniques for finding optimal solutions. By integrating SGD and the Computational Graph Approach [155] with other machine learning techniques, SD-WAN can approach controller placement from multiple perspectives, enabling sophisticated optimization and better adaptation to varying network conditions. These techniques contribute to achieving load balancing, reducing latency, and enhancing overall SD-WAN efficiency. Ultimately, the integration of machine learning in controller placement strengthens network performance, optimizes resource allocation, and supports the successful implementation of SD-WAN.

In reinforcement learning (RL), the system acquires knowledge through feedback from its environment, which can be in the form of rewards or punishments, reflecting the system's performance. With each interaction with the environment, the system gathers information and uses it to improve its understanding and update its knowledge. The study in [156] tackles the complex challenge of optimizing controller placement within a multi-objective optimization framework. The study proposes a novel approach that leverages deep reinforcement learning to enhance network control and management, capitalizing on its ability to explore solution spaces and adapt to rapidly changing data flows. This novel approach incorporates historical network data learning into both the deployment of controllers and the real-time mapping of switches to controllers, effectively adjusting to the dynamic conditions of the network environment. The study focuses on addressing three fundamental objective functions: load balancing, flow fluctuations, and latency reduction, all aimed at achieving optimal controller placement. This study contributes by framing the CPP as a multi-objective optimization issue and introducing a dynamic flow data-driven methodology that harnesses the power of deep reinforcement learning techniques. The proposed approach demonstrates its effectiveness in improving latency and load balance within the dynamic network context characterized by flow fluctuations. This strategy holds the potential for refining controller placement and switch-controller mapping techniques within SDN networks. However, it is crucial to emphasize that this study primarily revolves around resolving the controller placement issue specific to SD-WAN. This solution leans heavily on historical network data, implying the necessity of having comprehensive operational data from a functioning SD-WAN network deployment. In contrast, this thesis advances a stochastic computational graph approach for determining the optimal number and placement of controllers, uniquely relying on coordinates-based datasets and predefined objective functions. The study in [157] introduced a multi-path routing technique for software-defined networking, leveraging reinforcement learning. The aim of this approach is to tackle the challenges linked to underutilized links and the inability to promptly

adapt to real-time network conditions, which are prevalent in current SDN routing methods. The agent adeptly selects the most advantageous route from a range of options, considering network metrics like bandwidth, loss rate, and delay, to optimize the desired outcome. This strategy proficiently manages varied data flows, judiciously allocating them based on quality-of-service priorities to achieve multi-path routing. As a result, it led to diminished loss rates and improved jitter values compared to the conventional SDN routing methods. While not designed to solve the controller placement issue central to this thesis, this approach exhibits potential relevance to addressing the CPP in SD-WAN. The stochastic computational graph approach, which forms a cornerstone of this thesis, shares similarities with reinforcement learning since both involve learning through interactions within an environment to attain specific goals. However, in this context, it pertains to the framework of CP in SD-WAN. Given this perspective, the present thesis harnesses the stochastic computational graph approach in conjunction with ensemble learning (XGBoost) to ascertain the optimal number and arrangement of SD-WAN controllers. Similarly, the work in [158] introduces a novel strategy for addressing the controller placement problem related to load imbalance in the distributed control plane, employing deep reinforcement learning techniques. This approach involves the creation of a deep reinforcement learning model that interacts with the network, acquiring the ability to migrate switches effectively to maximize rewards. Notably, this method relies on historical network data concerning switch migrations. Once adequately trained, the model can swiftly and accurately determine switch migration actions. The load balancing achieved through this deep reinforcement learning-based approach is recognized for its capability to effectively distribute controller load and significantly reduce migration frequency times. While the primary focus of this approach is not specifically on solving the controller placement challenge central to this thesis, it exhibits potential relevance to addressing the CPP in SD-WAN. Notably, the stochastic computational graph approach, a pivotal aspect of this thesis, shares similarities with reinforcement learning, both involving the acquisition of knowledge through interactions within an environment to achieve specific objectives. In the same manner, the study in [159] introduces an innovative SDN controller architecture tailored for drone management, deeply rooted in the principles of machine learning. The comprehensive scope of drone management encompasses vital tasks like migration, drone authentication, and regulation of communication. A notable stride in this work is the introduction of an application-based authentication method, strategically designed to expedite the authentication process for drones. Moreover, the researchers harness widely adopted ML techniques, including Logistic Regression, Decision Tree, Support Vector Machine, and Random Forest, to effectively categorize and classify diverse drone types. The integration of the SDN controller into this framework ushers in an era of efficient drone communication, tailored to the specific requirements of each drone's designated purpose. This approach is specifically crafted to tackle persistent challenges within drone management systems, such as time and energy consumption, intricate complexities, and a rigid operational paradigm. An intriguing facet of their proposal is the utilization of the inherent capabilities of the SDN controller to dynamically program the network, facilitating seamless information management and astute analysis. Through the utilization of real-time and historical data within the drone network, machine learning techniques contribute intelligence to the SDN controller. This empowerment results in automated provisioning of network services, data analysis, and network optimization. Although this work primarily revolves around the intricacies of drone management and does not explicitly delve into controller optimization, the deployment of machine learning techniques for application classification shares parallels with the utilization of similar techniques in predicting the optimal number of controllers required for an efficient SD-WAN environment. Furthermore, the work of [160] presents a software-defined optical transport network routing optimization procedure that relies on the

reinforcement learning model of the ensemble and a Deep Q-Network utilizing a message-passing neural network structure. This novel approach for optimizing optical network routing in software-defined optical transport networks is suggested to effectively improve the extrapolation capability of deep reinforcement learning decision-makers. Notably, a message-passing neural network is employed to enhance the agent's generalization capabilities by capturing the relationship between network topology and demand. Ensemble and Message passing neural network-based Deep Q-network algorithm effectively tackles challenges posed by traditional routing methods and mitigates limitations often observed in deep reinforcement learning agents. This method stands out by addressing the complex and computationally intensive nature of network problems through ensemble learning. The primary focus is on introducing the Ensemble and Message passing neural-network-based reinforcement learning model, that offers reliable decisions or predictions even when faced with new or previously unencountered scenarios. The integration of message-passing neural networks and ensemble learning contributes robustness, facilitating stable and efficient environment exploration. While not directly tailored to the controller placement issue discussed in this thesis, this method holds the potential for addressing controller placement challenges in SD-WAN.

2.3 Research Gaps

This section examines the identified gaps in the literature regarding the placement of controllers in SD-WAN. One notable gap revolves around the inefficiency of existing switch migration techniques for SDN load balancing, particularly when dealing with substantial incoming traffic flows. In response, the literature has seen the development of various switch migration techniques [12, 19, 18]. However, while it is assumed that the placement of controllers in these techniques has been optimized and appropriately positioned, these approaches have often been associated with drawbacks such as prolonged migration times, increased response times, or reduced throughput. For instance, the authors in [19] proposed a dynamic and adaptive load-balancing architecture for software-defined load balancing using switch migration techniques. This framework selects the closest controller from a group of unloaded controllers to facilitate load shifting during switch migration. Another framework, known as the Controller Adaption and Migration Decision, as suggested by [12], chooses the controller with the "lightest load" from the pool of unloaded controllers for participation in load shifting during switch migration. While these approaches prove efficient, especially when dealing with high traffic volumes, they are most effective with a specific type of traffic known as mice flow, characterized by data packets loading at rates between 1 p/s and 499 p/s. However, it's worth noting that the controller adaption and migration decision algorithm, which prioritizes the controller with the lightest load, has improved response time, throughput, and migration cost compared to dynamic and adaptive load balancing frameworks [18] and Elasticon [18]. Nevertheless, it can result in low load balancing rates (LBR) when the mechanism selects the switch with the lightest load. This method can also lead to issues such as high packet loss, increased response time, inefficiencies in switch selection, and reduced network efficiency, ultimately affecting throughput. Furthermore, when examining the target controller, the dynamic and adaptive load balancing approach of [19] is associated with increased congestion, while the controller adaption and migration decision technique by [12] can lead to higher packet loss, increased response time, inefficiencies in switch selection, low throughput, and network inefficiency. Most previous research on switch migration primarily considers unloaded controllers without focusing on migration efficiency. A more effective approach would involve migrating a switch to a new master controller with higher efficiency to prevent overload. In response to these challenges in existing frameworks for dynamic and adaptive load balancing and controller adaptation and migration

decision-making, this research introduces an enhanced switch migration decision algorithm for software-defined network load balancing (ISMDA). The ISMDA mechanism selects a switch with a high load for migration from a controller with excessive load, similar to dynamic and adaptive load balancing. However, it also migrates to the most suitable controller to free up the most clustered resources. During the migration step, the framework's balancing module, operating on each controller, is initiated. By considering both the variance and average load state of the controller, the developed mechanism identifies groups of controllers in the system that are underutilized. The research proceeds to establish a migration model that assesses migration cost and load-balancing variance for selecting optimal controllers from a group of unloaded controllers.

Additionally, this thesis identifies gaps in the literature regarding solutions developed for optimizing controller placement while dealing with multiple conflicting objectives. These objectives encompass switch-to-controller latency in both average and worst-case scenarios, controller-to-controller latency in both average and worst-case scenarios, load balancing, and maximum controller failure. Existing literature introduces metaheuristic algorithms, such as NSGA-II [8] and MOPSO [161] for simultaneous controller optimization. However, scalability becomes a challenge when more than three objectives need consideration [7]. These methods are also computationally expensive, hampering their practicality. State-of-the-art optimization techniques often lack support for overall network architecture as the number of objectives surpasses three [17], leading to high computational complexity. In terms of scalability, the investigated MOPSO and NSGA-II optimization methods prove ineffective when dealing with more than three objectives, as they rely on operators like crowding distance to preserve diversity and Pareto dominance techniques to rank solutions. The majority of solutions remain non-dominated even in early generations, making it difficult to maintain a strong selection pressure towards optimal solutions. Consequently, the quality of identifying the best solutions suffers, resulting in less efficient exploration for optimal solutions. In contrast, a guiding mechanism, such as a clustering operator with a well-distributed reference point, in NSGA-III has been shown to help maintain a diverse solution pool. To address these issues, this thesis proposes an enhanced version of NSGA-III, incorporating a repair operator-based mechanism, which allows it to handle the discrete optimization problem of controller placement in SD-WAN. This enhancement ensures impractical solutions are eliminated during crossover and mutation processes while preventing duplicate solutions. Alongside the repair operator mechanism, the developed strategy ensures convergence and diversity across the Pareto front through techniques like normalization, association, reference points, and niching approaches. This approach effectively enables the simultaneous optimization of multiple competing objectives during SD-WAN controller placement.

Moreover, existing optimization methods, including MOPSO, NSGA-II, and the adapted NSGA-III, are computationally intensive and lack the ability to automatically learn heuristics, such as predicting the number and placement of controllers, for combinatorial optimization tasks like SD-WAN controller placement. These algorithms rely on manually crafted heuristics to make decisions that are either computationally expensive or not well-defined mathematically. Therefore, a quick, intelligent solution is required to compute both the optimal number of controllers and their placement, considering the complex nature of these algorithms. To address these limitations, this thesis suggests the use of a stochastic computational graph model with an ensemble learning approach (XGBoost) to determine the optimal controller location and the number of controllers required for an SD-WAN deployment when various performance metrics conflict. The proposed solution combines a stochastic computational graph model with

an Extreme Gradient Boosting Machine Learning Regression Model (SCGMEL). The ensemble learning method, specifically XGBoost, predicts the optimal number of controllers needed for SD-WAN deployment, offering a more efficient alternative to existing SD-WAN controller placement techniques. The choice of XGBoost as an ensemble learning method is based on its proven ability to enhance predictive accuracy and overall model performance. XGBoost, classified as a boosting algorithm, falls under the category of ensemble methods. Unlike ensemble learning methods that involve combining multiple independently trained models, XGBoost's ensemble model pertains to the collaborative operation of multiple decision trees within the algorithm. Additionally, this study suggests the use of an artificial neural network, specifically learning vector quantization, for predicting controller placement in an evolving SD-WAN topology. However, this approach may require collaboration between administrators and service providers due to its usage implications.

2.4 Summary of Literature Review

This section provides a concise overview of the current state of controller placement in SD-WAN and identifies critical issues within existing methodologies. The core challenge lies in determining the optimal placement of controllers and the required quantity of controllers for that placement. However, many existing approaches used in the literature to simultaneously optimize multiple conflicting objectives have significant drawbacks. For instance, popular metaheuristic algorithms like NSGA-II, MOPSO, Adaptive Foraging, and Firefly Optimization often face scalability issues, particularly when dealing with more than three performance metrics that must be optimized simultaneously. Additionally, these approaches demand substantial computational resources. Both the business and academic communities share a keen interest in optimizing controller placement in software-defined wide-area networks, especially when multiple conflicting objectives are involved. A comprehensive analysis of existing efforts to tackle the controller placement problem in SD-WAN reveals that metaheuristic algorithms like ANSGA-III, MOPSO, and NSGA-II have addressed certain aspects of the problem. However, apart from ANSGA-III, these metaheuristic algorithms encounter scalability challenges when optimizing more than three objectives. Moreover, these solutions tend to be computationally intensive and lack the capability to learn heuristics for combinatorial optimization problems, such as determining the optimal controller placement and quantity. The aspect of the controller placement problem related to identifying the optimal number of controllers has received limited attention in the literature. Even in cases where some research has been conducted in this area, it typically focuses on a single performance metric and relies on clustering approaches to determine the ideal number of controllers. Notably, these studies often predefined the number of controllers without employing robust tools to ascertain the most suitable quantity, potentially affecting overall network performance. Inadequate controller numbers might degrade network performance, while an excessive number may result in unnecessary installation costs for service providers.

This review of related work and the examination of existing solutions in the literature have unequivocally revealed critical gaps, including scalability issues, high computational complexity, and the absence of predictive tools, associated with current controller placement methodologies. Consequently, there is a pressing need to develop scalable, adaptive, and computationally efficient frameworks to address controller placement challenges in software-defined networks.

Chapter 3

An Improved Switch Migration Decision Algorithm for SDN Load Balancing

3.1 Introduction

This chapter discusses the developed solution that addresses the first discovered gap in light of the gaps mentioned in section 2.4. This discovered gap results from the absence of an optimized switch migration technique for software-defined load balancing when an elephant-sized traffic flow is present. This is important for this research because, as the network scale continues to expand, the scalability of the centralized controller emerges as a major challenge in a software-defined network. A promising solution to the issue is the deployment of distributed controllers, each of which controls a subset of the network's switches. However, the switch and the controller mapping are static in nature. Fixed switch-controller mapping causes sub-optimal performance and load imbalances when inconsistent load distribution across the controllers. A dynamic switch migration strategy, like the one described in this research, is a potential method for efficient load balancing and flexible scalability. The first section of this chapter will provide an introduction to the subject matter. The section 3.2 of this chapter will look at the proposed developed algorithm and its components. This section will examine the various modules that these methods use to optimize switch migration for load balancing in software-defined networks. In the third section, the outcomes of the simulation approach will be discussed, as well as how it will be evaluated and validated against some existing methods using several performance metrics such as the number of migrations, response time, throughput, and packet loss.

SDN is an enabling technology that simplifies network management and fosters development and innovation by decoupling the control plane from the data plane. Software-defined intelligent networking is achieved by having a logically centralized controller monitor switches and provide them with instructions to govern how they handle packets[162]. A promising solution to the issue is the deployment of distributed controllers, each of which controls a subset of the network's switches. However, the switch and the controller mapping are static in nature. Fixed switch-controller mapping causes sub-optimal performance and load imbalances when the controllers are not equally distributed with their loads [163]. A dynamic switch migration strategy, like the one described in this study, is therefore a potential method suitable to address this challenge which will bring about an efficient load balancing and a flexible scalable network environment. In actual fact, the migration of switches might take place in three different scenarios. In

the first possible situation, new controllers will need to be installed if the current ones are unable to handle the overall traffic load and the switches can then be switched to the new controllers. In the second scenario, with the aim of diminishing both power consumption and cost incurred in communication, it is advisable to consider relocating a controller's switches to an alternative location whenever the controller is powered down or enters a sleep state. In contrast, in the third scenario, when the load on a single controller exceeds its capacity, the switch migration process must be carried out by migrating the chosen switch to other controllers regardless of whether the number of deployed controllers changes or not. This procedure of migrating switches from the controller that was overloaded to the controller that will ensure that the optimum amount of clustered resources is left free (minimum under-loaded controller suitable for the incoming load) is referred to as load balancing. The scalability and performance of distributed controllers may be significantly improved with real-time switch migration. However, the execution of such migrations necessitates a meticulously designed system that can effectively determine which switch to relocate and the optimal destination for relocation. The prevailing body of literature in this domain has established that switch migration decision algorithms typically consider either the switch with the maximum or minimum flow to the controller during migration. For example, in the research by [12], a switch migration algorithm was introduced that prioritizes relocating the lightest load switch when predefined load thresholds are exceeded. In contrast, our proposed solution favors migrating the maximum-loaded switch, ensuring that the network's load distribution is optimized. In essence, it highlights the importance of having a structured and intelligent process for managing switch migrations in a network. This issue is what is referred to as a switch migration problem (SMP).
According to [164], the switch migration problem is a resource maximization issue that necessitates approximate solutions to handle each distributed controller and find the optimal method for establishing load balancing in real-time. However, this method picks the recipient controller and the migrating switch at random, which could result in poor overall throughput of the network and a high response time.

According to the findings of some research, improving the control plane performance in software-defined networks requires a number of different considerations. These include optimizing the reliability of every physical controller [165][166], relieving the controller by delegating some tasks to the network elements [18][167], as well as permitting a group of controller nodes to realize distributed network control [168][164]. These are just a few examples of the types of factors that should be taken into consideration. These studies have suggested a network control that is logically centralized and has attempted to resolve the state's consistency and the global view of distributed control planes. This might enable greater scalability and resilience with distinct controllers, but it would result in a load imbalance among controllers if an unusually high volume of traffic arrived at these separated controllers. This is due to the fact that each controller would be accountable for their own traffic load.

Additionally, DALB - a Dynamic and Adaptive load-balancing architecture was suggested in the work of [19] for software-defined load balancing using switch migration techniques. To support load shifting during switch migration, this framework selects the closest controller from the group of unloaded controllers. Unlike mice flows, which typically involve data packets loading at rates ranging from 1 packet per second (p/s) to 499 packets per second, our dynamic and adaptive load balancing system was specifically designed to address elephant flows. Elephant flows, on the other hand, entail data packets loading at rates between 501 packets per second and 5000 packets per second (p/s). The dynamic and adaptive load balancing framework's greatest strength is its ability to select the closest controller to accommodate

load shifts. This indicates that the closest neighbor controller will always be taken into consideration for migration during each phase of the migration process. This strategy, on the other hand, has the potential to reduce migration expenses as well as the amount of time required for execution. However, this procedure would lead to an increase in the amount of traffic congestion due to load shifting.

The CAMD which is also referred to as Controller Adaption and Migration Decision is a framework that was suggested by [12]. Within the confines of this architecture, the controller with the lightest load is chosen from the pool of unloaded controllers in order to participate in load shifting during switch migration. This approach is efficient, and it can manage a greater volume of traffic, provided that the incoming traffic is a mice flow (data packets load at a rate within 1p/s and 499p/s inclusively). The controller adaption and migration decision algorithm under the controller with the lightest load improved response time, throughput, and migration cost in comparison to the dynamic and adaptive load balancing framework [18] and Elasticon [19] approaches. However, the Load Balancing Rate (LBR) would be quite low if the controller's adaption and migration decision mechanism chose the switch that had the lightest load. In a similar manner, the controller adaption and migration decision technique results in a high packet loss, high response time, and a decrease in the efficiency of the network, along with poor throughput in the network. When the target controller is considered, the dynamic and adaptive load balancing [19] approach to switch selection is linked to the issue of increased congestion, whereas the controller adaption and migration decision [12] approach is linked to the issue of greater packet loss, increased response time, switch selection inefficiency, and low throughput. The majority of previous research on the switch migration problem simply takes into account any unloaded controllers without considering the migration efficiency. Meanwhile, a good demonstration would be the controller migrating a switch to a new master controller that has a higher level of efficiency (leaving maximum clustered resources free) for the purpose of preventing overload. In this particular piece of research, the focus is on the last switch migration scenario mentioned above in the second paragraph.

As a direct result of the problems that were described earlier with the frameworks for dynamic and adaptive load balancing, as well as the frameworks for controller adaptation and migration decision making. The aforementioned problems with the dynamic and adaptive load balancing as well as the controller adaption and migration decision frameworks were addressed by the research design that led to the development of an improved switch migration decision algorithm for software-defined network load balancing. This algorithm was given the name ISMDA[99]. The developed mechanism (ISMDA) will choose a switch with a heavy load for migration from the controller with an excessive load just as the choice of DALB, it will do so while migrating to the most suitable controller in order to leave the most clustered resources free. The developed framework's balancing module, which functions on each controller, is started during the migration step. In order to discover which group of controllers in the system were not being utilised to their full potential, the developed mechanism took into account both the variance and the average load state of the controller. This research went on to develop a migration model that detects the migration cost as well as the load-balancing variance for the choice of the optimal controllers from a group of unloaded controllers.

3.2 Developed ISMDA for SDN Controller Load Balancing Overview

The aim of this thesis is to provide a collaborative and adaptive optimization learning-based framework for solving controller placement in SD-WAN. The method developed in this chapter is consistent with the collaborative approach stated in the thesis' aim. (Please see 1.6). The software-defined network is built in the form of an undirected graph represented by the equation G = (V, E), in which V is the set of vertices and E is described as a set of connections. This research takes into consideration the premise that the controller has been suitably positioned in the network so that each controller governs a collection of switches.

3.2.1 ISMDA Load Balancing Strategy

Switch migration for load balancing is an efficient method of managing the controller's overload. This research presents a framework that uses a dynamic switch migration system to equitably distribute workloads among controllers. This method expedites the load-balancing process and maximises migration efficiency[99].

3.2.1.1 Associated Assumptions

This research presents the underlying presumptions that must be made in order to put the constructed framework to work.

- All of the controllers can communicate with one another, allowing for easy load distribution among them. This indicates that the load balancing technique is activated, and all of the switches in the network have easy access to all of the other controllers in the network, allowing them to migrate to the appropriate controller.

  .

- There is already a distributed data store (Hazelcast) present in the network. This data store connects different sets of controllers to one another to provide a logically centralized controller. Similarly, there is also a messaging library protocol called Zookeeper that allows for easy interaction.

- In the distributed environment, switches are concurrently connected to multiple controllers, with each controller serving one switch at a time. This design allows for redundancy and efficient load distribution. However, it's crucial to note that once a switch has been selected for migration, it cannot revert to its original controller until the entire migration process has been successfully completed. In the event that the migration process encounters difficulties or experiences extended delays, a fail-safe mechanism is in place. Specifically, switches are programmed to prioritize connecting to underloaded controllers within the network. Even if the migration process is temporarily stalled or fails, switches will not switch back to an overloaded controller, ensuring that network performance and load balancing are maintained.

3.2.1.2 ISMDA Framework Flowchart

A flowchart representation of the established framework, known as ISMDA, can be seen in figure 3.1.

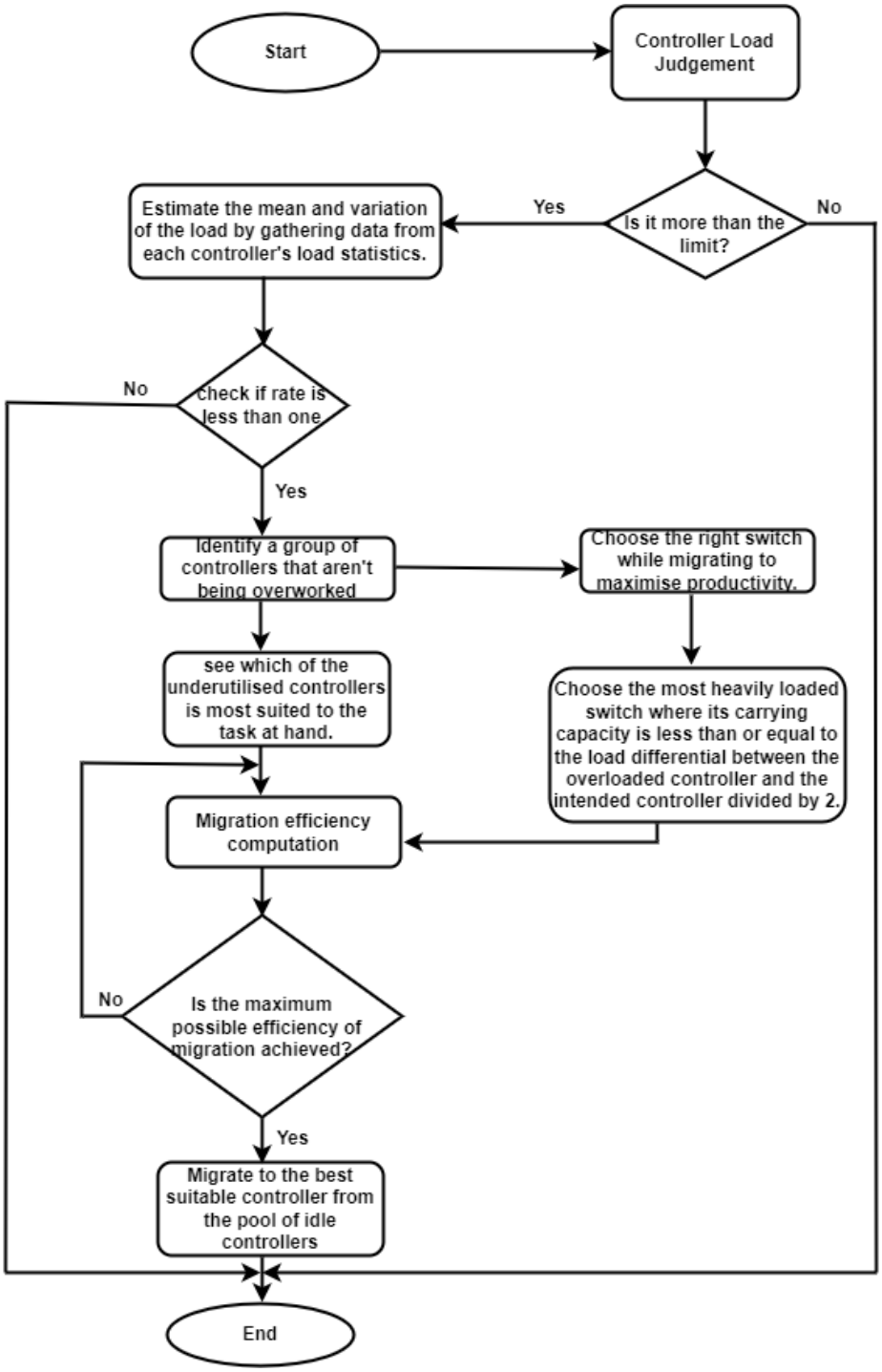

Figure 3.1: ISMDA framework flowchart.

The ISMDA that was established here is implemented on each controller, and they all work together to guarantee that the load is distributed evenly throughout the controllers. The ISMDA[99] functions on the controller itself, as a plug-and-play module. The controller performs periodic checks to determine

if the load has surpassed a certain limit (see Algorithm 2). To enhance total throughput and foster the establishment of stable, well-balanced networks, the ISMDA design intentionally prioritizes the set of under-loaded controllers for handling a higher volume of elephant data packets (flows) during switch migration operations. Here, it's important to note that "mice flows" typically encompass data packets loading at rates ranging from 1 packet per second (p/s) to 499 packets per second, while "Elephant flows" entail data packets loading at rates between 501 packets per second and 5000 packets per second (p/s). As soon as the load increases past a certain point, the load judgment module (module 1) is activated. The estimated mean and variance of the load are calculated by combining the controller's load data with that of other controllers. The controller simultaneously activates Module 1, which is in charge of load evaluation, Module 2, which chooses the switch that will be used, and Module 3, which is the target controller. All of these modules are responsible for controlling different aspects of the system. By using the calculated variance, a group of under-loaded controllers may be found from which the optimal one can be chosen. The ISMDA's switch module receives the flow operation from the load judgment module (see module 1). So that the overloaded controller's load is reduced by the same amount that the target controller's load is increased, the controller with the greatest load request would be migrated. The controller makes sure that the load on the target switch is less than or equal to half the difference between the load on the heavily loaded controllers and the load on the targeted controllers in order to ensure a smooth transition. To explore the effectiveness of selecting switches and controllers, respectively, the ISMDA created two different efficiency models that can be used interchangeably. In order to free up as many cluster resources as possible, the overload controller may migrate the newly-selected switch (maximum loaded) to the best-suited controller in the cluster with the help of a zookeeper (message library protocol).

3.2.1.3 Load Balancing Mechanism for ISMDA Strategy

Figure 3.2 depicts the conceptual model of the ISMDA, and it is composed of three separate modules that can be found running locally on each controller that contributes to the topology of the network. The load decision-making module (in module 1) is immediately activated by Algorithm 1 in the event that the controller's load capacity goes beyond a predetermined threshold.

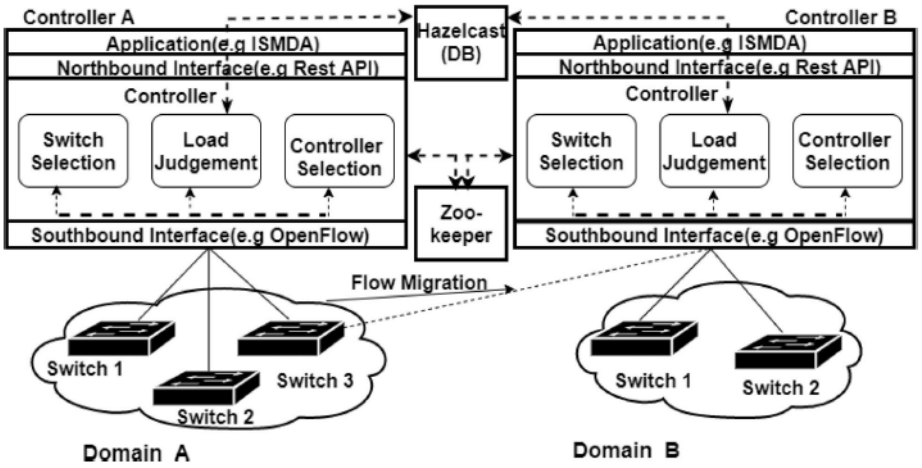

Figure 3.2: Load balancing framework.

The network's mean and variance for controller loads are determined by adding each controller's load to those of the other controllers in the network. The set of unloaded controllers in the network structure is determined by these conditions, where an unloaded controller is one whose load is lower than or equal to the average load of all the controllers. The load decision-making module (in module 1) prompts the switch selection module (in module 2) to migrate the highly loaded switch from the heavily loaded controller in order to increase selection efficiency due to the high load balancing rate. However, Modules 1 and 2 are followed by Module 3, which is the controller selection.
With migration efficiency in mind, it selects the best controller from the group of underutilized controllers, freeing up as many clustered resources as possible. The interaction between the controllers is connected through a cooperating service called Zookeeper. The data packets of Switch 3 in figure 3.2 of controller A in zone A will move its data packets to controller B in zone B whenever there is an imbalance in network demand.

The research goes on to detail the many components of ISMDA as well as its primary purposes. Algorithm 1 demonstrates the primary operation of ISMDA, and Algorithm 2 demonstrates the controller's dynamic threshold. The algorithm makes a request to the Load Decision-Making module (in module 1) whenever the controller's load status rises above a certain threshold. Module 2's Switch Selection is called concurrently with module 3's Target Controller Selection module to choose between available switches and controllers. As a direct consequence of this, the algorithm concludes, after which the switch moves to the targeted controller. The presentation of the ISMDA Overview can be found in Algorithm 1. This research also introduced an adaptive controller load rate in Algorithm 2. This rate improves how the network's controller threshold is updated.

3.2.1.4 ISMDA Algorithm 1

Modules 1, 2, and 3 are part of Algorithm 1, together with the adaptive controller rate algorithm (Algorithm 2). To determine if the controller load is too high, ISMDA Algorithm employs Algorithm 2. As a consequence of this, if the condition is met, then the three modules will each be set into motion to do their designated tasks.

Algorithm 1 Illustration of the ISMDA overview

1: **Input**: N, β', ρ, CL_{cur}
2: **Output**: Balanced Distributed Controllers
3: **if** $CL_{cur} > \beta'$ **then**
4: *Activate Load_Judgement_module*
5: *Activate Switch_Selection_module*
6: *Activate Target_Controller_Selection_module*
7: $Switch\_Migration(cl_j \leftarrow sw_k)$
8: Update N (current and target)
9: **end if**
10: **return** *Balanced Distributed controllers*

3.2.1.5 Dynamic Controller Threshold (Algorithm2)

Dynamic updating of the controller threshold is carried out using Algorithm 2. The threshold value is kept in the central repository. Each network controller verifies whether or not its current load is above a certain limit. Assuming that this is the case, it calculates the controller's mean load in the networks. It returns the original threshold if the calculated mean is less than or equal to that value. If that number is above or below the current threshold, the predicted average load is used instead. So that it may make

an informed judgement about how to distribute system loads, ISMDA must receive data from all of the system's controllers. Nonetheless, using a static controller threshold method, system performance can be degraded due to the regular gathering of load statistics data in the network for making load-balancing decisions. An adaptive load collection threshold strategy was proposed in this research to address the issue of degraded system performance (presented in Algorithm 2). This is kept in a centralised database that can be accessed by any controller.

Algorithm 2 ISMDA's Threshold for the Dynamic Controller

1: **Input**: Load list $\{CL_1, ..., CL_n\}$
2: **Output**: Dynamic (γ')
3: *Compute average load* $\overline{C}L(c_i) = \frac{1}{n}\sum_{i=1}^{n} CL(c_i)$
4: Assume $(\gamma') =$ **1600**
5: **if** $\overline{C}L(c_i) \leq Initial(\gamma')$ **then**
6: $(\gamma') =$ **1600**
7: **else if** $\overline{C}L(c_i) > Initial(\gamma')$ **then**
8: (γ')=$\overline{C}L(c_i)$
9: **end if**
10: **return** (γ')

3.2.1.6 Execute Judgment Module Load (Module 1)

This component keeps track of each controller's load data and relays that information in real time. The data reveals how the controller's central processing unit, network, and memory are being utilised. When a controller's load exceeds a predetermined threshold in Algorithm 2, it uses the system variance which is also referred to as system performance error and an estimate of the mean load of all other controllers to determine which controller is unloaded. The higher the quality of a network, the less room there is for error in its output. So these works classify, as unloaded, controllers whose weight is less or equal to the entire mean load of the networks. According to the author's description in [19], there are essentially two ways to obtain the SDN controller's load conditions. Both a centralised and decentralised load collection algorithm are examples of such techniques.

The load balancer is used in a centralised load collection technique to monitor and record the controller's current load. This method is inefficient and would only work on a restricted (small-sized) network. The distributed load collection mechanism, however, allows for any controller node to function as a load balancer. Regardless of the scale of the network, it can simply monitor, estimate, and report on its load data, making it more suitable and acceptable in large networks. There are two methods available for gauging the controller's workload. There are two approaches to this problem; one is the input metrics from the switch, while the other one is the performance metrics.

Input metrics from switches are used to characterise the controller's load by measuring things like the number of entries in the flow table, average message arrival rate, and response time. Since the switch input metric directly impacts controller resources, it was not used in this investigation. In this research, we focus on the performance measures that may be used to characterise the load status of a controller and how they relate to the controller's resources. The controller's metrics are central processing unit utilisation, memory usage, and bandwidth utilisation. Due to the multidimensional nature of the controller's resources and the wide variety of switch controller demands, it is difficult to make meaningful comparisons between them. Accordingly, a weighted total is assigned to integrate various resource kinds, with the weight being

adjusted to modify the controller resources' utilisation proportion. A rough estimate of the domain's overall CL load is made by each controller LC(ci). This is possible in real-time using the aforementioned measurements and their weighted values, which is expressed in Eq. 3.1

$$LC(c_i) = \begin{bmatrix} wgt_1 & wgt_2 & wgt_3 \end{bmatrix} \begin{bmatrix} LC_{bandwidth}(c_i) \\ LC_{memory}(c_i) \\ LC_{CPU}(c_i) \end{bmatrix}. \tag{3.1}$$

in which wgt_1, wgt_2, and wgt_3, represent the weights of network control productive capacity (bandwidth, CPU, and memory), and where $LC_{bandwidth}(c_i)$, $LC_{CPU}(c_i)$, and $LC_{memory}(c_i)$ denotes the actual bandwidth, CPU, and memory used on the control layer by flow rule instructions. For simplicity, let's assume λ_k represents the *packet_in* message at was transmitted from switch $s_k$to controller c_i at instant t. Due to the fact that such resources reside on the same control plane (uniformly distributed systems), a weighting (wgt_1, wgt_2, and wgt_3) system is required. Here, we employ a weight system to standardise the dissimilarities and similarities among the control plane's available assets. That way, you can learn how much of a resource one is actually utilising through the use of the available resources. The reduction of bias should be the primary concern when assigning weight to controller resources. In the absence of the weight, there will be a sharp increase in inconsistency as well as an increase in general error. As a result, the total burden that has been placed on the controller as a direct consequence of the incoming data packets messages is stated in Eq. 3.2

$$LC(c_i) = \sum_{s_k \in c_i} \lambda_k(t)\phi_{ki}. \tag{3.2}$$

$LC(c_i)$ stands for the controller's (c_i), current load,
ϕ_{ki} is the utilisation value of the control plane's resources, and $\lambda_k(t)$ is the total number of OpenFlow messages transmitted from switch s_k to controller c_i during the time interval t. In this investigation, we use the notation where ϕ_i stands for the controller's c_i resource utilisation value and the definition is defined in Eq. 3.3

$$\phi_i = \sum_{s_k \in c_i} \phi_{ki}. \tag{3.3}$$

As shown in the Eq. 3.4, the symbol ϕ_{ki} is the sum of the network utilisation caused by every switch s_k to controller c_i

$$\phi_{ki} = \left(\eta_i^a \frac{\lambda_k * a_i}{\kappa_i} + \eta_i^b \frac{\lambda_k * b_i}{\rho_i} + \eta_i^c \frac{\lambda_k * c_i}{\psi_i} \right). \tag{3.4}$$

Where λ_k stands for the Packet_in message that the associated switches send whenever they detect a change in the control load. Control event c_i for switch s_k is expressed by a_i, b_i, and c_i, which each denote the CPU time, memory, and bandwidth used by c_i. Values η_i^a, η_i^b, and η_i^c stand for the approximated weighted value of controller resources. For the memory, central processing unit, and network to be used efficiently, the formula $\eta_i^a + \eta_i^b + \eta_i^c = 1$ must hold.

Additionally, κ_i, ρ_i, and ψ_i stand in for the central processing unit, memory, and bandwidth of controller (c_i).
In order to make an accurate estimate of the control plane equilibrium level, the concept of the population

variance is offered. This is because all of the controllers are taken into consideration. This difference is represented in the Eq. 3.5 as

$$\gamma = \frac{1}{k} \sum_{i=1}^{k} \left(LC(c_i) - \overline{L}C(c_i) \right)^2. \tag{3.5}$$

in which γ is the variance, k is the number of controllers, $LC(c_i)$ symbolises the controller current load and $\overline{L}C(c_i)$ symbolises the controller mean load. In light of this, the load-balancing component would then initiate a load-balancing method with regard to the variance. Moreover, when there is an unbalance in controller load, the migration of the switches will begin.

Using data from other loads on the network, the module will calculate an average load factor which can be expressed in the Eq. 3.6 as

$$\overline{L}C = \frac{1}{k} \sum_{i=1}^{k} LC(c_i). \tag{3.6}$$

Algorithm 3 Module 1: Description of ISMDA's Load Judgement module

1: **Input**: n(load controller number), ρ_i, β'
2: **Output**: C_{UL}
3: max = NULL
4: **for** $i = 1$ *to* n **do**
5: *check controller load* $LC(c_i)$
6: max = max $+ LC(c_i)$
7: **end for**
8: *Calculate the mean load using* 3.6
9: *Estimate the load variance using equation* 3.5
10: **for** i=1 *to* k **do**
11: *Compute load balancing rate using equation* 3.7
12: **if** $\delta_i < 1$ **then**
13: $C_{Uloaded} \leftarrow c_i$
14: **else**
15: $C_{Oloaded} \leftarrow c_i$
16: **end if**
17: **end for**
18: **return** $C_{Uloaded}$

The pace at which a controller is able to bring its load closer to its average is known as its load balancing rate, and it is symbolized by the symbol δ_i. Specifically, this is shown in the Eq. 3.7

$$\delta_i = \frac{LC(c_i)}{\frac{1}{k} \sum_{i=1}^{k} LC(c_i)}. \tag{3.7}$$

After that, the module will compute the following three ratios, which will serve as the foundation for its migration decision-making.

- $\delta_i < 1$: it may be deduced that the controller's current load status is less than the typical load that is currently being experienced. This suggests that the controller is not operating at full capacity and has room for additional loads or a switch migration.
- $\delta_i > 1$: it may be deduced that the load that is now being placed on the controller is more than the load that is considered to be the present average. This indicates that the controller is overfull, which could cause the network to become imbalanced and cause additional costs.

- $\delta_i = 1$: This indicates that the load that is now being carried by the controller is equivalent to the current average load that is being carried by the active controllers as a whole. As a result, the controller is not operating in an overloaded or underloaded mode at this time. As a direct consequence of this, the controller is incapable of handling any more demand. This indicates that the load that is now being carried by the controller is equivalent to the load that is being carried by the active controllers as a whole. As a result, the controller is not operating in an overloaded or underloaded mode at this time. As a direct consequence of this, the controller is incapable of handling any more demand.

Following the techniques described above, controllers are classified into two distinct groups: the overfull controller group as well as the unloaded controller group. This is expressed as a mathematical expression as

$$\begin{cases} \delta_i > 1 & \text{controllerin overfull mode} \\ \delta_i = 1 & \text{controller in imbalanced mode} \\ \delta_i < 1 & \text{controllerin unloaded mode.} \end{cases}$$

3.2.1.7 Module for switch selection (Module 2)

In order to facilitate migration, this component or module chooses the high load switch provided by the overfull controller. The number of entries in the switch flow table, the arrival rate of packets, and the round-trip time were some of the helpful statistics that were provided by the load judgement module. The number of entries in the switch flow table and the round trip time are often utilised for the selection of the controller, whereas the mean arrival rate of the message is employed for both threshold computation and controller selection.

In accordance with the OpenFlow protocol, the switch selection process can benefit from taking into consideration the aforementioned three statistics. The number of messages sent from the associated switches during the specified time interval (t) directly relates to the amount of work performed by the controller. The switch is able to relay a plethora of messages to the controller, including Echo messages, Packet in messages, Hello messages, and others. Nevertheless, the Packet in messages is responsible for the majority of the weight placed on the controller. In this particular investigation, the typical Packet in messages that were transmitted from the switch to the controller was used to represent the controller's workload. The objective of the developed mechanism is to share the workload across a network of distributed controllers. This study assumed that in order to have a well-sustainable system, the load decrease of the overfull controller would not be greater than the increase of the target controller load in an ideal scenario. Following that, this study came to the conclusion that when choosing a switch for the purpose of migration, the load on the switch that is going to be transferred ought to be equal to or less than the difference between half of the overfull controller and the destination controller. Consequently, the decision to choose the switch is determined according to the constraints in Eq 3.8

$$SML_{Migrate} \leq \frac{CL_{Overfull} - CL_{destination}}{2}. \tag{3.8}$$

Algorithm 4 Module 2: ISMDA Switch_Selection module overview

Input: Overfull controller C_{OF}
Output: A list of migrated switches s_i from C_{OF}
Load switch set $\Delta = \{\}$
for $\forall$ s_i $\in C_{OF}$ **do**
  for $\forall$ $Flows$ $\in C_{OF}$ **do**
    $Identify\ migrated\ switch$
  end for
end for
$\left(\frac{1}{k}\sum_{i=1}^{k}(C_i)\right) / \max_{i=1}^{k}(C_i)$ *to find selection efficiency*
$s_n = \underset{s_n \in C_{OF}}{\operatorname{argmax}}\{\mathrm{P_{sn}}\}$
$Migrate\ LS_{Migrate} \leq \frac{LC_{Overloaded} - LC_{Inbound}}{2}$
Return *switch set* $\Delta\{\}$

$SML_{Migrate}$ reflects the weight of the transferred switch from the overfull controller $LC_{Overfull}$ reflects the weight of the overfull controller, and the $LC_{Inbound}$ reflects the weight on the controller that is best suited for the task at hand. This idea originates from [169], the NSGS technique for Indifference Zone strategy. This process notifies the switch migration component of its decision. The NSGS Method is an abbreviation of the entire sequential procedure and the Ranking and Selection (R&S) approach. NSGS is an abbreviation for **Nelson**, **Swann**, **Goldsman**, and **Song**, the first names of the authors of [169].

In the context of this study, it is anticipated that a switch will be able to establish connections with several controllers all at once. Nonetheless, only one controller, the master, will be actively serving the switch's needs at any given moment; the rest will be in either slave or equal mode. A messaging library protocol such as Zookeeper and ZeroMQ (ZMQ) can be used to make the transition in roles possible, and HA-TCP is what's utilised to make sure messages are sent between the switch and the controller.

3.2.1.8 Controller Selection Module (Module 3)

To ensure that the full amount of grouped resources are reserved, the objective of this module is to choose the controller that is the best fit to take on the load that will be transferred from the controller that is currently overloaded. This research developed a migration efficiency model that took into account both load balancing variance and the cost of migration. This model was then used to determine the best controller to use for the optimal selection. The migration cost and load balancing variation have both been established so that their application to the efficiency model that has been created can be improved.

When switch migration takes place, there is an improvement in the network's load balance. Nevertheless, this does result in additional migration costs for the network. As a result, one requires the introduction of the idea of migration cost. The majority of the costs associated with migration are made up of two primary aspects: the rise in the amount of work done by the controller, as well as the quantity of data that is passed between the controllers. In the process of moving s_k from the c_i computer to the c_j computer. The cost of migration can be stated in the Eq 3.9

$$cmig_{s_i c_j} = cmig_{ex} + cmig_{lc}. \tag{3.9}$$

where $cmig_{ex}$ represents the cost of exchanging messages and mc_{l_c} indicates the cost of dealing with an increased load.

The increase in load that takes place as a result of the intended controller accepting the migrating switch from the overload controller is what is meant by the term "load increased cost." This is represented in Eq. 3.10

$$cmig_{lc} = \begin{cases} \lambda_i cmig_{s_i c_j} - \lambda_i cmig_{s_i c_i}, & cmig_{s_i c_j} > cmig_{s_i c_i} \\ 0, & cmig_{s_i c_j} \leq cmig_{s_i c_i}. \end{cases} \tag{3.10}$$

where λ_i reflects the control messages generated by the switch s_k.

Algorithm 5 Module 3: ISMDA's Target_Controller_module illustration

Input: s_i, C_{underL}, $cmig_{lc}$
Output: Target Controller c_j
Obtain the current load on C_{underL}
for $\forall$ *controller* $c_i \in C_{underL}$ **do**
  for $j = 1$ *to* $n \in C_{underL}$ **do**
    Evaluate variance Ω *and* Ω^**using* (3.11)*and*(3.12)
    Compute migration efficiency using (3.13)
  end for
end for
Check that $LC(c_i) + LS(s_i) \leq LC(c_j)$
$C_{ST} = \underset{c_j \in C_{underL}}{\text{argmax}}\{C_{underL}\} : LC(c_i) + SL(s_i) \leq LC(c_j)$
update $\{C_{underL}\}$ *with current load state*
return *target controller*, c_j

Therefore, the load variance method needs to be implemented as a criterion for selection. In this analysis, the mean load of the N controllers was taken to be $\overline{L}C(c_i)$ and the controllers' load variance was employed as the balancing factor. This network selection factor prior to switch migration is re-defined from equation (5) and mathematically expressed in the Eq 3.11

$$\Omega = \frac{1}{k} \sum_{i=1}^{k} \left(LC(c_i) - \overline{L}C(c_i) \right)^2. \tag{3.11}$$

In the above equation, Ω represents the inherent network load variance that occurs before any switch relocation takes place. $LC(c_i)$ is what defines the load for each controller and $\overline{L}C(c_i)$ represents average controller load n. After the switch migration, the network's load selection factor makes certain that the amount of system bias is decreased to an acceptable level. This is defined in Eq. 3.12

$$\Omega^* = \frac{1}{k} \sum_{i=1, i \neq j}^{k} [LC^*(c_i) - \overline{L}C^*)^2 + LC^*(c_j) - \overline{L}C^*)^2], \tag{3.12}$$

with $LC^*(c_i) = \big(LC(c_i)$ - $\lambda_{si} c_{s_i c_i}\big)$ and $LC^*(c_j) = LC(c_j) + \lambda_{si} cmig_{s_i c_j}$.

After switch migration has taken place, the Ω^* in (3.12) merely represents the network's load variance. To further improve destination controller selection, migration efficiency is created to show a balance between the cost of the migration and the load balancing rate. In this analysis, the rate of load balancing is divided by the cost of migration to get the migration efficiency. This is written in Eq. 3.13

$$\Lambda = \frac{\mid \Omega^* - \Omega \mid}{cmig_{s_i c_j}}, \tag{3.13}$$

$$\forall s_i \in S, c_j \in C, J(s_i, c_j) = \{0, 1\} \tag{3.14}$$

$$\forall s_i \in S, \sum_{c_i \in C} J(s_i, c_j) = 1, \tag{3.15}$$

$$\exists c_i \in C, LC(c_i) \leq \delta' \tag{3.16}$$

Equation (3.13) restricts the number of connections between nodes, Equation (3.14) guarantees that the master controller is in charge of all switches at a time (t), and Equation (3.15) prevents all network controllers from becoming overfull at once.

Table 2. The arrival rate of the switch to controller flow

Controller set	CC1			CC2				CC3	
Switch set	s1	s2	s3	s4	s5	s6	s7	s8	s9
Switch Flow rate (KB/s)	800	600	600	600	1000	800	1400	400	600
Controller Threshold	3200								

3.2.2 Example for Demonstration

The purpose of this study is to demonstrate the importance of migration efficiency and load variance in switch migration situations with the same controller capacity using a simple example. Assuming that the controller CC = $\{CC1, CC2, CC3, CC4\}$ and their corresponding load CCL = **{800, 300, 500, 400}**p/s. The mean controller load of $\overline{LS} = 500p/s$. **600**p/s is anticipated to be the threshold of the controller (TC). As a result of the preceding calculation, $CC1$ will be the overfull controller. In a given distribution, variance evaluates how far a set of data is from its average value. Controller individual variances are $v(CC1) = 250000$, $v(CC2) = 40000$,$v(CC3) = 0$, and $v(CC4) = 10000$, **variance** of the controller's load before migration is **75000**. The overfull controller $CC1$ could either move excess load of **200** to $CC2$ or $CC4$. If controller $CC1$ selects $CC2$, the variance is therefore **{**$10000, 0, 0, 10000\}$, and the network's global load **variance** is **5000**$(p/s)^2$. If CC4 is picked, the variance of individual controller is now **{10000, 40000, 0, 10000}**p/s, and the load variance equals **15000**p/s^2. As a result, the migration efficiency for $CC2$ is **350**, whereas the migration efficiency for $CC4$ is 300. This analysis led to the conclusion that the optimal option would be a controller with better migration efficiency. Since the entire load balance will be in a stable mode, the controller $CC2$ will be judged to be the one that is best suitable for the situation.

Lemma 1: During the migration stage of this research project, a switch with the most loaded flow requests on an overfull controller is taken into consideration since this would increase the network's Load Balancing rate (LBR).

Proof 1: A load balancing rate can be used to evaluate how closely the model matches the observed load distribution. As may be seen in Table II, the study reveals the rate at which various switches reach the controller. The combined load on Controller C2 is $3800p/s$ at the time (t), making it overfull.

For load shifting, Controller C3, which has a higher migration efficiency, will be the preferred controller. This research explains how selecting a switch that has fewer flow requests has an effect on the LBR. For the purpose of determining LBR, equation (3.17) was taken into consideration in this study. LBR is defined as

$$\mathbf{LBR} = \left(\frac{1}{k}\sum_{i=1}^{k}(CC_i)\right) / \max_{i=1}^{k}(CC_i). \tag{3.17}$$

with $\{CC_i, ..., CC_n\}$ denotes the load list of the network components which includes the overfull controller's load within the 0–1 co-domain. The LBR is confirmed to be 0.59 before migration takes place. For example, if controller C2 chooses switch S4 for migration because it is the least loaded switch in its configuration, the LBR in equation (17) will be 0.71. Meanwhile, the LBR will become 0.94 if C2 makes the decision to select its switch with the current highest load for migration. According to the findings of the study, which can be derived from the analysis presented above, choosing a switch that has the largest flow request would result in a noticeable improvement in the effectiveness of the network.

3.3 Experimentation

This section explains the experiments that were conducted for this investigation. For the purpose of conducting a performance evaluation, this study employed a simulation-based methodology and measured the number of migrations, the packet loss, the throughput, and the response time. The Python code used in these experiments was developed specifically for this study by the research team and is available in the GitHub repository at [`https://github.com/dipokoya2003/ISMDA.git`]

This research was conducted using a personal computer that featured a Windows 10 Professional operating system, a processor from Intel Core i7-6700HQ running at 2.60 gigahertz, and 16 gigabytes of DDR3 memory operating at 1600 megahertz in order to build up the simulation environment. The experiment was carried out by utilising the Jupyter notebook compiler, which is a free and open-source software application that generates code that is capable of being performed in real-time. It was assumed throughout the operation that the threshold of the controller's load capacity was set to $\mathbf{2000}p/s$. Before any migration phase, the current load on controllers A-D was assumed to be 500p/s, the rate of system processing was set to 70%, and the total received load was initiated between (10000-26000)p/s by a size of 20. Each simulated controller had its Controller Threshold (CT) set at $\mathbf{2000}p/s$ before each iteration began, which is the value used by the proposed ISMDA when deciding on its migration method. For the purpose of simulating a realistic control environment, the processing rate was set to 70% before any migration phase began, and the current load being handled by controllers A, B, C, and D was configured to be $500p/s$. Each experiment was repeated 1,500 times, and the total received load was varied from (10000-26000)p/s in each run. When the load that is being computed is greater than the controller threshold that has been established, the developed ISMDA will activate Module 1. After that, the ISMDA will call the module for selecting the switches (module 2) as well as the module for selecting the controllers (module 3). The aforementioned modules will choose the high-load switch for migration, as well as the controller that is most suited to handle the inbound load. As part of this research, an improved switch migration decision mechanism has been developed to make sure that the maximum number of grouped resources would be accessible at all times during the migrating process. Therefore, when elephant flows make up the incoming traffic, this significantly improves the performance of the baseline frameworks. In

this research, the range of "elephant flow" is specified as $501p/s$ to $5000p/s$, while the range of "mice flow" is specified as $1p/s$ to $499p/s$

3.4 Result AND Discussion

This research introduces the ISMDA[99] method and evaluates it in relation to two other similar works that have been published previously by [19] and [136]. This is done so that the findings of the study can be affirmed. The DALB [19] mechanism, which is also known as the nearest controller selection method, can quickly lead to an increase in traffic congestion when accepting load shifting. This is because the communication overheads between switches and controllers increase when multiple switches can migrate into the same controller simultaneously. CAMD [136], method, which is also known as the Least-loaded controller selection strategy, is only useful for accepting load shifting in situations in which the inbound traffic load is minimal flow. In addition, the controller adaption and migration decision mechanism selection of a switch with the lowest flow request rate for migration is inappropriate and not optimal in comparison to a maximum utilised switch. As demonstrated in the final paragraph of Section 3, choosing a switch with the lowest flow request could not be more beneficial to the network than selecting a maximum-loaded switch. For this reason, ISMDA chose a maximum loaded switch when performing a migration, as well as uses the migration indexing factor to determine which controller is most suited to take on the load. Following is a discussion of the simulation results, with short explanations of the various performance metrics used in the analysis.

- Controller throughput: The term "controller throughput" refers to the maximum number of packets that a controller is capable of successfully processing at one time. The amount of traffic that was created while the simulation was being run was used to make an estimate of the average controller throughput. The total incoming load that was used for this research ranged between (10000 - 26000)p/s. The controller throughput of each of the studies that were examined as well as the proposed ISMDA can be seen in Figures 3.3, 3.4, and 3.5. As the graph shows, the proposed ISMDA is able to process more flow requests than the studies that were considered in the evaluation. Figure 3.6 presents a comparison of the average throughput achieved by each of the various algorithms.

  Figure 3.3 shows that the DALB method, in contrast to the CAMD approach, reliably processes more flow requests even under heavy demand, with a lower drop rate. Figure 3.5 shows that, on the same scale, DALB method was not consistent with the developed ISMDA.

  In contrast to the ISMDA and DALB algorithms, the CAMD algorithm in Figure 3.4 was unable to keep its consistency when the load was at its highest point. The decline rate went as low as 250. As a direct result of this, the overall throughput was decreased.

  ISMDA was able to effectively process a greater number of flow requests, as seen in Figure 3.5, compared to both DALB and CAMD. In comparison to DALB and CAMD, it maintains a more constant level of performance even at high loads, and its level of load reduction is also more manageable. The comparative results for the three different algorithms' average throughput are presented in Figure 6.

  As shown in Figure 3.6, the developed ISMDA algorithm accepted a greater volume of traffic than both the CAMD and DALB algorithms. The developed architecture shown in Figure 3.6 achieves an increase in controller throughput of around 7.4% over the CAMD and roughly 1.1% over the

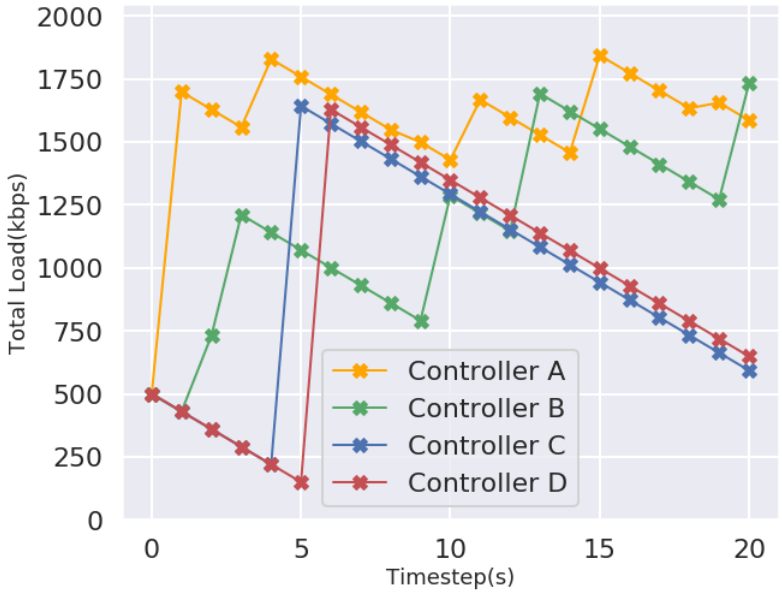

Figure 3.3: DALB Controller throughput

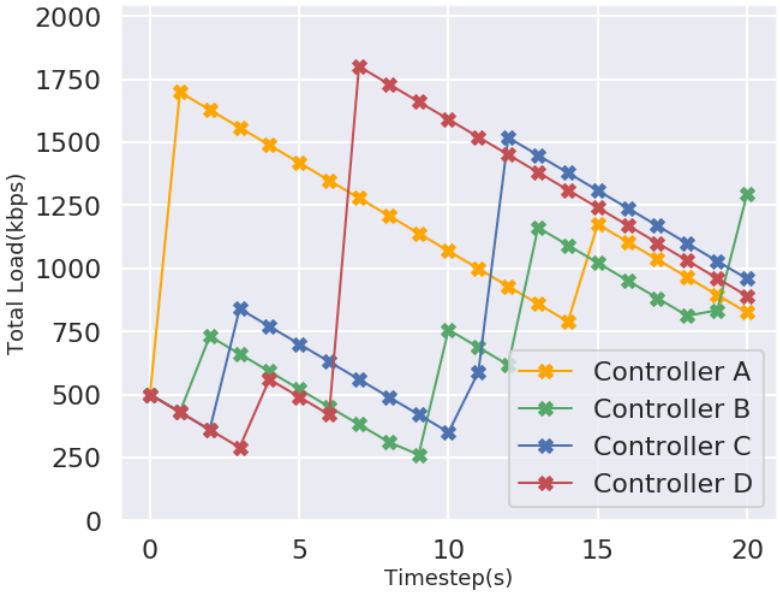

Figure 3.4: CAMD Controller throughput

DALB. This results in an overall throughput of approximately 444p/s for the typical controller. Despite the fact that DALB and ISMDA both have comparable throughputs, it has been demonstrated that the suggested ISMDA algorithm is superior to both DALB and CAMD in terms of its efficiency.

- Response time: It is anticipated that there will be an increase in response time in a particular network when there is an imbalance in the controller load. This study employed equation (3.18) provided by Netforecast [170] for response time computation. All parameters in this equation were held constant during evaluation, as the main goal of this research was to determine the impacts of packet loss on response time. Packet-loss values were derived from the number of rejected packets. The adopted expression is given as

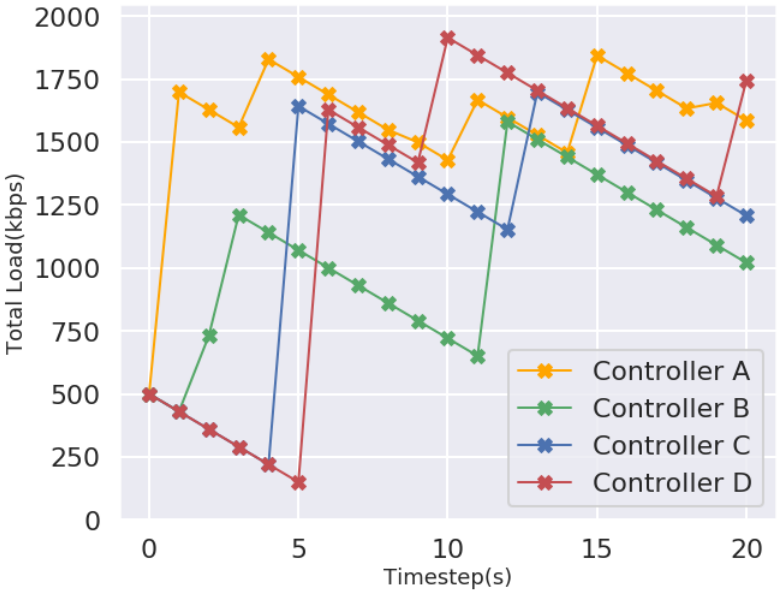

Figure 3.5: Proposed ISMDA Controller throughput

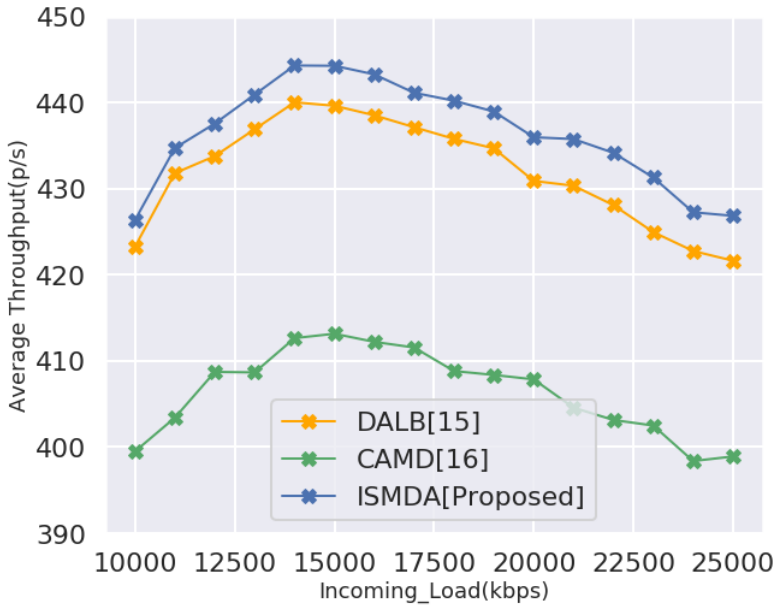

Figure 3.6: Throughput comparison of different Algorithm

$$RT = P_{dt} + T_{dt}, \tag{3.18}$$

where

$$P_{dt} = 2\Big[Rtd + C_{pt} + S_{tcp}\Big] + \Big[Rtd + \frac{[C_{cpt} + S_{pt}]}{2}\Big]\frac{AT-2}{mf} + P_{lr}\Big[\frac{AT-2}{mf} + 1\Big] + YT(\frac{L}{1-L}), \tag{3.19}$$

and

$$T_{dt} = \frac{MAX\left[8pyl\frac{1+OHR}{MPB} * Rtd\frac{pyl}{ws}\right]}{1-\sqrt{L}}. \tag{3.20}$$

The term "RT" stands for the response time, P_{dt} represents the propagation delay time, and T_{dt} stands for the transmission delay time. In equation 3.18, Rtd represents the round-trip delay, C_{pt} shows the current processing time, S_{tcp} represents the server TCP processing, and S_{pt} shows the server processing time. AT represents the application turns, mf shows the multiplexing factor, P_{lr} shows the packet-loss ratio, and Y represents the TCP timeout. In a similar manner, in equation 3.19, Pyl represents the length of the payload, OHR represents the overhead ratio, MPB represents the minimum path bandwidth, and WS represents the window size.

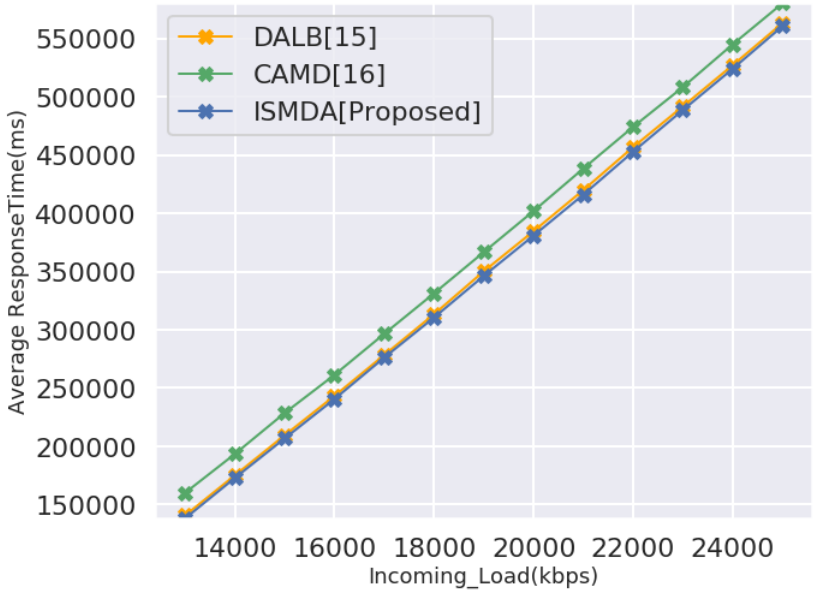

Figure 3.7: Comparison of Response Time different Algorithm

In this study, the authors assessed the three algorithms that were utilised in the simulation analysis by using the average response time. When there is an increase in the incoming load, there is likewise an increase in the mean response time of the three algorithms. Figure 7 shows that the reaction time of the developed algorithm (ISMDA) outperforms that of the controller adaption and migration decision algorithm and the dynamic adaptive load balancing algorithm, with around 5.7% and approximately 1% less, respectively. The study's effective procedure for choosing a controller to absorb inbound load from the overfull controller was crucial. At any point throughout the migration process, ISMDA will choose the most suitable controller to ensure that as many resources as possible are freed up in their clustered form. Because of the greater efficiency in resource utilisation ensured by ISDMA, fewer packets were rejected, and controller response times were drastically improved.

The number of Migration spaces: This is the frequency with which each algorithm must carry out a migration during an unbalanced or overload on the controller. Analysing and estimating the performance of the ISMDA (the developed mechanism), and the two compared algorithms, the dynamic adaptive load balancing and the controller adaption and migration decision algorithm were all accomplished by using the average count of the rejected packets. The average frequency with

which each algorithm was required to execute the migration is depicted in Figure 3.8. It is clear from this, that a higher total incoming load would naturally cause all three algorithms to perform more migrations than they currently do. The performance of the developed framework outperforms that of DALB and CAMD, with a decrease of 1.7% and 5.6%, respectively. This suggests that the established framework will result in fewer switch migrations occurring during controller load imbalance as compared to DALB and CAMD.

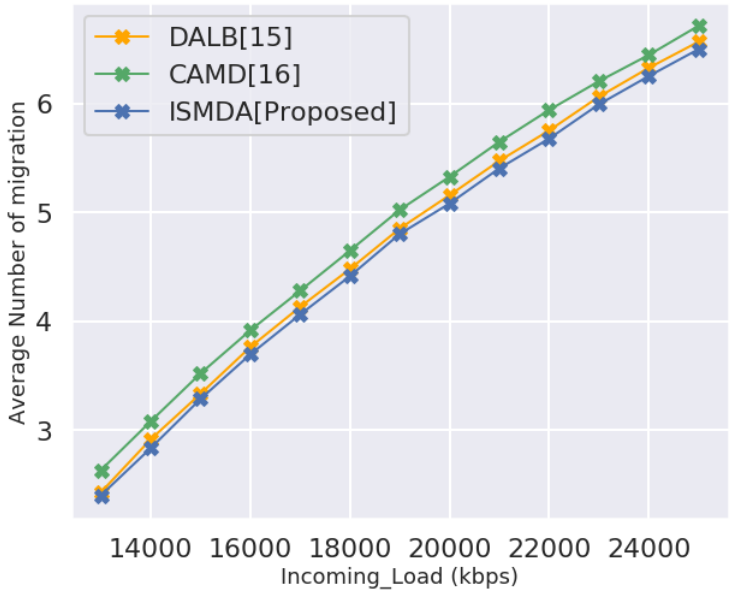

Figure 3.8: Comparison of several Migration Time Algorithms

At each stage of the migration process, the efficient method that was implemented and the selection of the controller that was the best fit are both deemed to be accountable. Because of this, there was a considerable drop in the total migration time occurrence.

- Packet Loss: The term "packet loss" refers to the number of data packets that are lost or discarded while a transmission is taking place. In this investigation, estimates were made on the typical amount of rejected packets that occurred during the performance evaluations of the three methods.

Figure 3.9 demonstrates that for all three approaches, the average number of packet-loss rises in proportion to the amount of incoming traffic. On the other hand, the controller adaption and migration decision algorithm had the highest average packet loss, whilst the ISMDA - the developed mechanism and the dynamic and adaptive load balancing had similar levels of mean packet loss. According to the findings of the analysis, ISMDA was more efficient than DALB and CAMD by an estimated 1% and 6.4%, respectively, in terms of the average packet loss. This is the case as a direct result of the efficient process that was proposed in this work. ISMDA can handle a higher load than either CAMD or DALB. As a direct result of this, a lower percentage of packets were rejected.

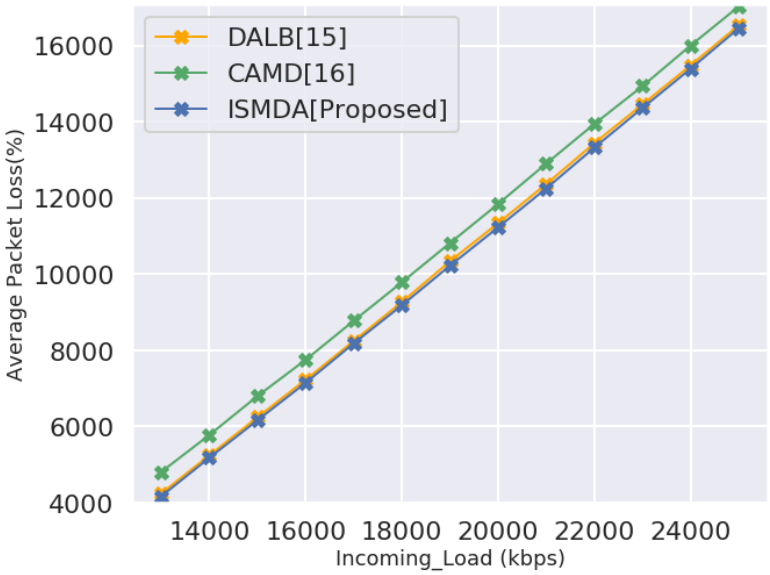

Figure 3.9: Comparison of Packet-Loss Rates of different Algorithm

3.5 Verification and Validation of the Developed Improved Switch Migration Decision Algorithm for SDN Load Balancing (ISMDA)

In this part of the thesis, the author shows that the developed ISMDA model has met its main goal in terms of quality and credibility by showing how it has done what it was meant to do. The verification process includes everything that goes into making a high-quality solution, like testing, analyzing the design, analyzing the specifications, and so on. The technique can be thought of as being fairly objective. In contrast, the process of validation is extremely subjective in nature. It involves making subjective decisions about how well a solution that has been proposed or made meets a real-world need.

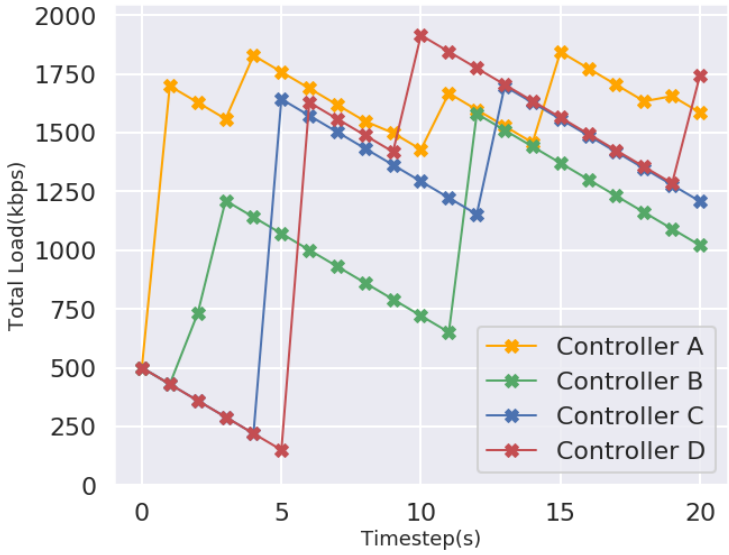

Figure 3.10: ISMDA Controller total received load

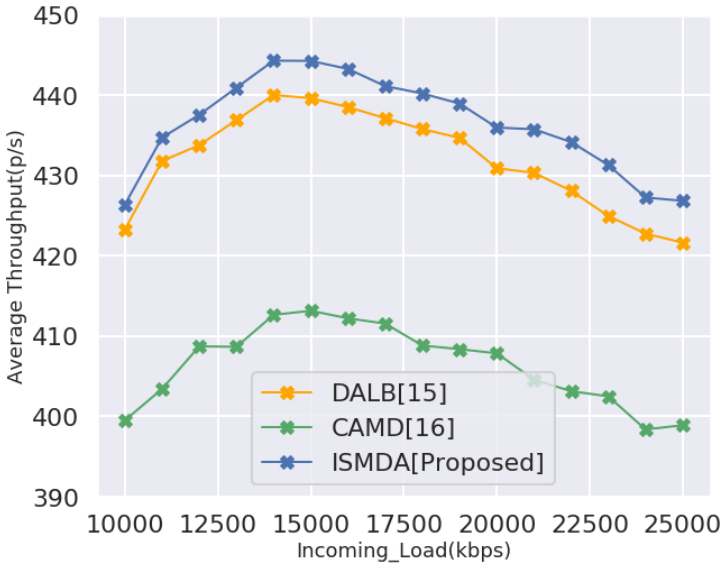

Figure 3.11: Throughput comparison of different Algorithm

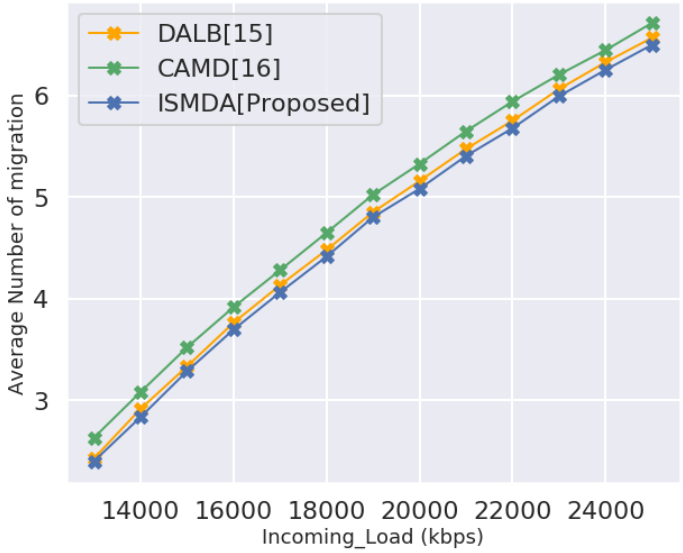

Figure 3.12: Comparison of several Migration Time Algorithms

The validation process includes many different steps, such as modeling the requirements, making a prototype, and testing with users. The specifications were followed closely during the planning and construction of the suggested solution. The developed solution aimed to address one of the gaps identified in this thesis's Chapter 2, namely the inability of existing switch migration algorithms for SDN load balancing to increase average network throughput and reduce the average number of migrations when an elephant-sized traffic flow is present. Looking at the figures in 3.10, 3.11, and 3.12 it is clearly seen that the developed ISMDA for SDN load balancing using switch migration techniques actually fulfills its intended design aim and meets the expected outcomes. In a similar way, the validation process involved comparing the results of the ISMDA model with those of the existing switch migration techniques for SDN load balancing when elephant traffic flows are present. The figures in 3.11, and 3.12 were used in the

validation process of the developed ISMDA model. As revealed by the figure in 3.11, the ISMDA average throughput outperforms the compared algorithms. In the same way, the developed model migrates less often than the algorithms that are already out there. As a result, this thesis concludes that the outcomes revealed by the 3.11, and 3.12 confirm the validity of the developed ISMDA model and have met its expectations. The reader is advised to see figures 3.3 and 3.4 for more information.

3.6 Conclusion Remarks

This section summarizes related works by presenting the key issues associated with existing switch migration techniques for SDN load balancing, assuming that the controller is optimally placed in SD-WAN deployment. According to the literature, the existing solution's inability (the switch migration technique for SDN load balancing) to perform efficiently when the incoming traffic flows are large is a major concern that needs urgent attention.

Several switch migration techniques have been developed in the literature to solve the aforementioned challenge. [19] developed a dynamic and adaptive load balancing architecture for software-defined load balancing using switch migration techniques. To support load shifting during switch migration, this framework selects the **closest controller** from the group of unloaded controllers. The CAMD, which is also referred to as Controller Adaption and Migration Decision, is a framework that was suggested by [12]. Within the confines of this architecture, the controller with the **"lightest load"** is chosen from the pool of unloaded controllers in order to participate in load shifting during switch migration. This method works well, and it can handle more traffic if the incoming traffic is a mice flow, which means that data packets load at a rate between 1 p/s and 499 p/s. This new method either had long migration times, long response times, or low throughput. When the target controller is considered, the dynamic and adaptive load balancing [19] approach to switch selection is linked to the issue of increased congestion, whereas the controller adaption and migration decision [12] approach is linked to the issue of greater packet loss, increased response time, switch selection inefficiency, and low throughput. The majority of previous research on the switch migration problem simply takes into account any unloaded controllers without considering the **migration efficiency**. In the meantime, a good example would be the controller moving a switch to a new master controller that is more efficient and frees up as many clustered resources as possible to prevent overload.

Consequent to the problems that were described earlier with the frameworks for dynamic and adaptive load balancing, as well as the frameworks for controller adaptation and migration decision making, this research addressed these challenges by *developing an improved switch migration decision algorithm for software-defined network load balancing(ISMDA)*. Just like DALB, the new mechanism (ISMDA) will choose a switch with a high load for migration from a controller with too much load. It will do this while migrating to the best controller to free up the most clustered resources. The developed framework's balancing module, which functions on each controller, is started during the migration step. In order to find out which group of controllers in the system were not being used to their full potential, a mechanism was made that took into account both the variance and the average load state of the controller. This research went on to develop a migration model that detects the migration cost as well as the load-balancing variance for the choice of the optimal controllers from a group of unloaded controllers.

Chapter 4

A Scalable Solution for solving Controller Placement problem in Software-Defined Networks

4.1 Introduction

Software-defined networking (SDN) is quickly becoming known in the networking world as a possible new networking paradigm. The control layer's detachment from the data-plane layer is what sets SDN apart from traditional networking [7]. The data-plane layer is made up of numerous streamlined packet forwarding switches, whereas the control layer consists of a group of devoted controllers that function as the SDN's intellectual "brains." This division makes the network directly programmable, which provides several advantages such as enhanced network utilisation effectiveness, simplified network administration, and the facilitation of network innovations. The most widely used communication interface for software-defined networks (SDN) is OpenFlow [1], which at the outset simplifies things by assuming there is just one controller. However, the performance and scalability of such a system with just a single controller may be compromised as the network scale grows over time. As a result, several different multi-controller strategies are subsequently put forth, which succeed in creating a similar fundamental architecture [2]. The Controller Placement Problem (CPP) poses a significant issue in multi-controller systems. While discussing software-defined networks, the CPP typically raises the question of where to deploy controllers and which switches to use in order to accomplish a specific objective. These objectives may include lowering latency, increasing reliability, improving energy efficiency, and so forth. Because it affects practically every aspect of SDN, including fault tolerance, network performance, and state distribution options, the CPP has generated a great deal of research attention [2]. While SDN has seen widespread adoption in data centre networks, traditional routing and traffic engineering practises persist in wide area networks. Although the placement of controllers is usually not a problem in LANs, it can have a big effect on the performance of an SD-WAN when many goals need to be optimised at the same time. In the context of SD-WAN, there has been a great deal of focus on determining the best location for controllers when there are multiple conflicting goals. Researchers have made use of metaheuristic algorithms. These algorithms include NSGA-II [8] also known as Non-dominated Sorting Genetic Algorithm II, and the MOPSO [161], also known as Multi-objective Particle Swamp Optimization to solve controller location issues in SDN. However, when more than three objectives needed to be considered, the scalability of these SD-WAN optimization algorithms became a problem [7]. In the same way, these algorithms are known to be com-

putationally expensive.

Based on the discussion in the concluding part of Chapter 3, see section 3.6 regarding the assumption that controller placement has already been placed in the topology (randomly positioned) for SDN load balancing. This is not a realistic assumption, and it cannot be implemented in a real network environment because randomly placing controllers in a large-scale environment is inefficient and can lead to poor network performance, which can also raise a service provider's capital expenditure (CAPEX) and operating expenses (OPEX). As a direct result, there is an urgent need to propose and develop a solution that can effectively optimise controllers in the SDN environment, especially in SD-WAN.

The controller placement can be positioned with reference to several performance metrics, which include resilience, inter-controller latency, switch-to-controller latency, and load balancing, to mention but a few. These performance metrics are known to be conflicted in nature, and as such, service providers need to carefully find a solution that will simultaneously optimise the controller regarding these performance metrics. This is a multi-or many-objective problem, also known as an optimization problem.

This chapter presents the first attempt to overcome the related difficulties in the literature by addressing controller location in large-scale deployments like SD-WAN, taking into account the established research goals and the observed gaps in the current body of knowledge. The developed strategy is a modified version of the original Mechanical Engineering-Based Non-Dominated Sorting Genetic Algorithm III (henceforth made reference to as ANSGA-III or adapted NSGA-III) [7].
Due to the scalability issues faced by the existing algorithms in the literature (MOPSO and NSGA-II) when dealing with objectives that are three and above. This research proposed and developed a modified version of the third edition of the Mechanical Engineering-based non-dominated sorting genetic algorithm in order to offer the ANSGA-III for the best location of controllers in the software-defined wide area network [7]. The developed strategy includes a special operator (repair operator) that helps in avoiding duplicate solutions in the Pareto-optimal sets and also makes it possible for the original NSGA-III [17] to be used in the SD-WAN domain. This is done by changing the original NSGA-III from its continuous optimization characteristics to the discrete optimization characteristics used for controller placement.

This chapter's initial portion will give an introduction to the subject matter. The problem definition will be examined in this chapter's second section after the introduction. The numerous objective functions that are utilised in the optimisation of controllers in SD-WAN will be presented in the following section. In the third section, the developed strategy for simultaneously optimising several conflicting objectives in the placement of controllers will be discussed. In addition, the description of several algorithms that constitute the developed algorithms will be investigated here. Such algorithms include reference-based, normalisation, niche, association, and repair-based operators. The experimentation that was carried out regarding the developed strategy will be discussed in section four, while the results and discussion of the experimentation will be examined in section five. The sixth and final section will bring the chapter to a close.

4.2 CPP Mathematical Design and Objective Functions

Within the scope of this study, the controller location issues in SD-WAN and the metrics used in optimising the controller were analysed. This method offered the ideal placement of controllers in respect to

the topology of the network infrastructure in order to satisfy a number of network criteria all at once. Although the latency that exists between the switch and the controllers to which it is connected is the most critical CPP condition, there are other competing goals that need to be taken into consideration. Among these objectives are the deployment cost, node-to-controller latency, resilience, load balancing and inter-controller latency.

Based on the findings of this study, an unconstrained, many-objective CPP was designed and implemented. Equations (4.1) through (4.6) introduce the objective functions of this research. These objective functions were used along with the decision variables in optimising CP. The SD-WAN is built in the form of an undirected graph represented by the equation $G = (V, E)$, in which V denotes the collection of vertices and E denotes the connections that exist between the vertices. Furthermore, to calculate placement, a distance matrix denoted by D holds information on the minimum path latency across each set of vertices in the topology. The latency between vertices n and vertices k is represented as d_{nk}. In this study, the authors normalised the latency values by dividing them by the diameter of the corresponding graph, which they express in D. The search area is constrained to a predefined list of $\binom{\alpha}{\beta}$ possibilities through which the required outcome can be obtained.

Meanwhile, the shortest path delay between any two vertices in the topology is stored in a distance matrix represented by D, which is used in the placement calculation.

In this research, we define a location as an α-element set, where α is a smaller set than V. The CPP uses the β-subset of V as its search space because it contains all the solutions. For demonstration purposes,the set $\Upsilon = 2, 7, 15, 18, 19$ represents the locations of the 5 controllers in a network with 21 vertices and a predetermined number of controllers such that $\Omega = 5$. It is assumed that the 5 controllers should be located at the vertices 2,7, 15, *and* 19. Note that switching the members of each subgroup does not yield a new permutation. Thus, in this particular case, there are $\binom{21}{5}$ possible placements in this topology, and since there are a set of criteria to be minimised, the author has denoted these placements with the symbols $\{k_1, k_2, \ldots, k_m\}$. Note that when there is no other feasible option Y for the search area, then Υ is the Pareto optimal choice. To be more precise, at least one index n must satisfy the condition, $\forall_n \; k_i(\Phi) \leq k_n(\Upsilon)$ and $k_n(\Phi) < k_n(\Upsilon)$. The purpose of solving the CPP is to determine the Pareto optimum set of the whole search area and the list of criteria for all Pareto optimal placements, that together constitute the Pareto Frontier, a collection of solutions.

4.2.1 Objective functions

A summary of the objective functions used in optimising CP was provided in this subsection. The audience is referred to [112] for further information on the examined objectives. When choosing where to position controllers, Ω of controllers, various competing objectives must be taken into account while determining controller placement. The maximum switch-to-controller latency and average switch-to-controller latency are indicated by the first two objective functions. This refers to the maximum and minimum latency between the controller that sits on the control plane and its associated switches. The maximum and mean switch-to-controller latency for every potential placement $\Omega \in 2^V$ and the provided distance matrix D are determined by equations (4.1) and (4.2).

$$\delta^{Lat\_max\_S2C}(\Omega) = \max_{v \in V} \min_{\omega \in \Omega} d_{v,\omega}, \tag{4.1}$$

$$\delta^{Lat\_avg\_S2C}(\Omega) = \frac{1}{|V|} \sum_{v \in V} \min_{\omega \in \Omega} d_{v,\omega}. \tag{4.2}$$

When several controllers are installed in large networks, they must communicate with one another and share data. As a result, when evaluating controller placement, inter-controller latency should be considered and minimised. Equations (4.3) and (4.4) perform the same functions as equations (4.1) and (4.2), but they compute inter-controller latency rather than the maximum and average forms of latency. This goal should be addressed in the CPP because it has a significant impact on controller coordination.

$$\delta^{Lat\_max\_S2C}(\Omega) = \max_{\omega_1, \omega_2 \in \Omega} d_{\omega_1, \omega_2}, \tag{4.3}$$

$$\delta^{Lat\_avg\_S2C}(\Omega) = \frac{1}{\binom{|\Omega|}{2}} \sum_{\omega_1, \omega_2 \in \Omega} d_{\omega_1, \omega_2}. \tag{4.4}$$

The latency-based objectives try to minimise the number of possible communication pathways inside the network, but the controller load balance is equally important for ensuring the steady operation of the network. To guarantee adherence and equitable load distribution across controllers, this research offers an imbalance metric in place of a balancing metric, as all other objective functions are minimised in this work [112]. To keep track of how many devices are assigned to each controller, the author uses the notation δ_ω, where *Omega* is the number of possible placements and *omega* is the number of controllers. In the equation, (4.5) ([112]), δ_ω represents the difference between the allocations of the two controllers with the fewest and the greatest amount of nodes, accordingly.

$$\delta^{imbalance}(\Omega) = \max_{\omega \in \Omega} \alpha_\omega - \min_{\omega \in \Omega} \alpha_\omega. \tag{4.5}$$

This research also took into account controller failure resilience as a cost function during optimal CP. This research assumes that $\mathrm{C} = 2^{\Omega} \setminus \{\emptyset\}$ contains all of the remaining placements after the failure of up to $(\beta - 1)$ controllers, then the mean node to controller latency for any failings condition is theoretically depicted by the equation (4.6).

$$\delta^{Lat\_avg\_S2C}(\Omega) = \frac{1}{|C|} \sum_{\Omega \in C} \left(\frac{1}{|V|} \sum_{v \in V} \left(\min_{\omega \in \Omega} d_{v,\omega} \right) \right). \tag{4.6}$$

4.3 The Adapted NSGA-III (ANSGA-III) for SD-WAN Controller Placement

This section of the thesis discusses the changed version of the non-dominated sorting genetic algorithm III, which will be called the ANSGA-III in the next few paragraphs. This approach is consistent with the optimization approach stated in the thesis' aim. (please see 1.6.) The NSGA-III [17] was developed in the field of industrial engineering to handle optimization problems with multiple competing objectives (greater than three). The controller location problem is a discrete optimization problem, whereas the present NSGA-III can only solve continuous optimization problems. As a result, it is not possible to use

it in a direct manner to solve the controller location problem in SD-WAN. Because of this limitation, the application of the existing NSGA-III to controller location problem did not result in the generation of a singular solution that was free of duplicates. NSGA-III used the strategy that NSGA-II had set up, but it also used the reference point-based technique and other techniques (like association, normalization and niching approaches) to enhance diversity preservation and convergence, and to make sure that the network could handle growth.

Because the majority of solutions were non-dominant and remained in the first layer even in the first generation, it was difficult to maintain a strong evolutionary pressure from top solutions toward ideal solutions. The investigated MOPSO and NSGA-II optimisation methods are ineffective when applied to solve the problem of scalability (more than three objectives) due to their reliance on crowding distance operators for conserving diversity and Pareto dominance strategies in ranking solutions, respectively [17] and [161]. Because the majority of solutions became non-dominant and remained in the initial level layer even in the early generation, it was difficult to sustain a strong genetic change among elite solutions toward an optimum solution. As a result, it became harder to find the best solutions, which made it harder to search for the best candidate solution in the most efficient way. Nonetheless, it has been demonstrated that a guiding mechanism in NSGA-III, such as a clustering operator with an equally dispersed reference point, aids in keeping the pool of solutions sufficiently broad and diverse. The algorithm has been independently verified as more accurate, making it similar to [140], and [171] which both agree that the NSGA-III self-adaptivity of the reference point list, obtained from the representative state of each point of reference over the course of multiple evaluations, makes it more efficient computationally. As a consequence, it is believed that the association, normalization, and niching techniques, as well as the reference point processes included in the NSGA-III procedure, account for the substantial convergence and diversity enhancement achieved by ANSGA-III [7] in comparison to NSGA-II.

It has been shown in the literature that the MOPSO and NSGA-II do not exhibit diversification characteristics throughout the Pareto set of solutions when there are more than three goals to simultaneously optimise (scalability challenge) [17] and the population size is greater than one hundred [172]. As a direct result of the shortcomings of existing algorithms, there is a need to build an algorithm that can choose diverse solutions. Hence, this substantiates the validity of the preliminary implementation of the modified ANSGA-III in SD-WAN.

4.3.1 The Description of the ANSGA-III as Developed

The suggested ANSGA-III is reviewed in this subsection after being introduced in Algorithm 6. The ANSGA-III model is given a set of well-defined reference point data Ref and the parent population represented as PP_t. The defined reference point data can be calculated following the systematic method of [173] represented as $Ref{=}\binom{N_{obj}+k-1}{k}$ where Ref = the total reference point in an Nobj objectives problem, Nobj represents the number of objectives while the k represent the number of division which is also a user-defined parameter.

The parent population at the next generation is represented by $PP_{(t+1)}$. The reference points are selected and created on a unit hyperplane in order to guarantee they are spread consistently over the entire scaled hyperplane (see Figure 1). Reference points produced in this way are often scattered throughout the normalized hyperplane, thus it stands to reason that the solutions generated from them will be widely

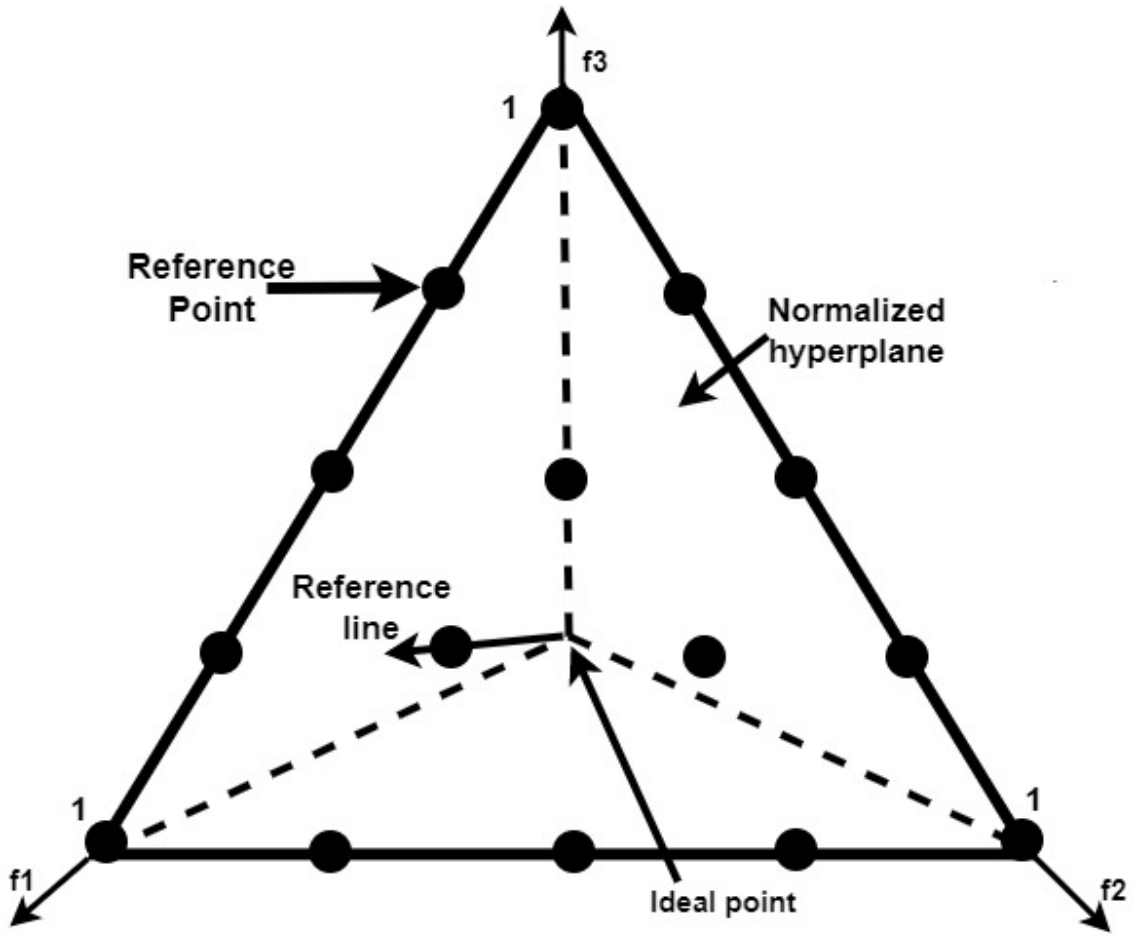

Figure 4.1: Reference point on a Unit hyperplane

distributed toward the Pareto optimal solution sets. This criterion is grounded in the diversity that is described in terms of the reference lines or reference points ([17]).

Line 3 of Algorithm 6, Φ_t holds the decision variables (solutions that are non-dominant) in the Front layer $\{FL_1, FL_2, \ldots, FL_M\}$ and i is the iteration number that is initialised to 1. In Line 4, the parent species (PP_t) was optimised through the use of the crossover (simulated binary) and mutation operators (polynomial mutation) to create the child species. At line 5, the introduced RBO employs a check mechanism on the offspring species set CR_t to eliminate non-viable offspring solutions and ensure that no duplicate solutions are produced after the final generation.

Algorithm 6 Proposed Adapted NSGA-III for SD-WAN Controller Placement Problem

1: **Input**:Ref =*structured reference points* δ^s
or γ^s *as aspiration points* PP_t *for the parent population*
2: **Output**: PP_{t+1}
3: Φ_t=$\emptyset$, i=1, Φ_t = *save non* − *dominated solutions*
4: CR_t=$Crossover + Mutation$
5: *Use repair operator technique on* CR_t
6: ρ_t=$PP_t \cup CR_t$
7: $(FL_1, FL_2, ...)$=*solutions that are not dominated*(ρ_t)
8: **repeat**
9: Φ_t=$\Phi_t \cup FL_i$ *and* $i = i + 1$
10: **until** $|\Phi_t| \geq N$
11: *Final front to be included* : $FL_l = FL_i$
12: **if** $|\Phi_t|$=N **then**
13: PP_{t+1}=Φ_t, *interrupt*
14: **else**
15: PP_{t+1}=$\bigcup_{j=1}^{l-1} FL_j$
16: *Points to be chosen from* $FL_l : K$=$N\ - |PP_{t+1}|$
17: *Normalise all objectives and generate reference set* γ^s : $Normalize\ (f^n, \Phi_t, \gamma^r, \gamma^s, \gamma^a)$
18: *Associate every candidate* ϕ *of* Φ_t *to its ref point* : $[\tau(s), d(s)]$ = $Associate\ (\Phi_t,\ \gamma^r,)\ \tau(s)$: *near ref point* d : *distance between* s *and* $\tau(s)$
19: *Compute niching of reference point* $j \in \gamma^r : \lambda_j = \sum_{\Phi \in \Phi t/FL_t} ((\tau(s) = j)\ ?1 : 0)$
20: *Select* K *individuals each at a time from* FL_l *to construct* PP_{t+1} : $Niching(K, \lambda_j, d, \gamma^r, FL_l, PP_{t+1})$
21: **Output** *Required Placements*
22: **end if**

In line 6, the parent species and the child species are merged together and kept in ρ_t. The non-dominant rank was performed on ρ_t and the decision variables were sorted in the order of their importance which is represented as $\{FL_1, FL_2, \ldots, FL_M\}$ in line 7. The audience are directed to [17] for further details on the NSGA-III and the methods it employs. The procedure of non-dominating sorting is carried out once again in line 8. In parallel, lines 9 and 10, added the solution from the initial fronts to the species set, and this process is repeated until the population set's size exceeds the size of population N.In line 11, the details of the final solution, which are required to be provided before condition 10 may be satisfied, are outlined. In lines 12 and 13, the algorithm interrupts and the number is incremented by 1 if the final front is included and the length of Φ_t = N. The fronts from the generation before are combined to the fronts of the generation after it on Lines 14 and 15, with the exception of the front that comes last. Line 16 outlines the selected locations from the previous front, while the normalization of objective functions and candidate solutions, as well as the generation of reference point sets denoted by γ^s, take place in Line 17. In line 18, individual candidate solution ϕ, which is a member of Φ_t, is connected with a reference point. A comparable clustering approach is utilized in this instance, but solely for reference points. It is worth noting that all solutions of the decision variables $\{FR_1, FR_2, \ldots, FR_M\}$ in X_t are interrelated. The niche count is computed on Line 19, and this specifies the number of different solutions that are associated with the Reference Line. Last but not least, beginning on line 20, K solutions are extracted one at a time from the last front. The normalization, RBO, niching, as well as association algorithms in the ANSGA-III are explained further in sections (4.3.2), (4.3.3), (4.3.4), and (4.3.5), respectively.

4.3.2 Repair-Based Operator (RBO) Algorithm Description in the ANSGA-III

The Repair-Based Operator (RBO) technique, innovatively developed as part of this research, represents a substantial enhancement to the NSGA-III algorithm. It was meticulously tailored to address the unique demands of the controller placement problem in SD-WAN and seamlessly integrated into the core of the original NSGA-III framework [17, 129]. The primary purpose of the RBO algorithm is to confine the search process exclusively to the feasible solution space. This strategic restriction serves as a guiding principle for the adapted NSGA-III, steering it toward the creation of unique and duplicate-free solutions. Moreover, the operator-based mechanism was designed with the specific intent of facilitating the replacement of continuous optimization characteristics with discrete optimization attributes within the existing NSGA-III framework. This transformation is pivotal since the controller placement problem inherently aligns with the realm of discrete optimization. This innovation constitutes a significant advancement for the NSGA-III algorithm, rendering it better equipped to tackle the distinct challenges posed by the research problem. Algorithm 12 provides a comprehensive breakdown of the RBO algorithm, elucidating its critical role in achieving the research objectives. The study seamlessly incorporates the developed repair operator-based mechanism into the adapted NSGA-III for the optimal placement of controllers in SD-WAN. This integration ensures both convergence and the generation of diverse solutions among the non-dominated alternatives.

In the case of Algorithm 12, the algorithm's input and output are read from lines 1 and 2, respectively. The algorithm takes as input the controller position values ($\alpha\beta$) and outputs the candidate solution value with no duplicate ($\alpha\beta$). In line 3 of the algorithm, the ($\alpha\beta$) is iterated through by counting rows (m) in ($\alpha\beta$). The starting controller position is ($\alpha\beta$), which is equal to the size of the entire dataset ((20). Starting at line 3, the method checks the number of rows (n) and iteratively moves through ($\alpha\beta$). Like in line 2, an empty list named γ is created in line 4, and in line 6, the algorithm turns the values $\alpha\beta$ to an integer by rounding down to zero. The ROB also performs a similar check in line 7 to ensure that γ in line 4 does not match the number in line 3. In addition, if the requirement in line 7 is met, the algorithm designates γ to be the ($\alpha\beta$) in line 8; otherwise, if the number exists inside γ, 0 is assigned to j (line 9). Because the value in $\alpha\beta[m, n]$ exists in γ up until γ reaches 20, the algorithm assigns the first number that is not in γ to z on line 10. The algorithm determines if z is outside of γ at lines 14 and 15. If this is the case, the first non-empty value not contained in γ is used. As a direct consequence of this, the number z is ascribed to γ in line 16.

Algorithm 7 *Repair − based Operator Algorithm*

1: **Input**: $\alpha\beta$: *Location variables for controllers*
2: **Output**: $\alpha\beta$: *Candidate solutions without duplication*
3: **for** *m in range* (*len* ($\alpha\beta[i]$)) : *check rows in* $\alpha\beta$ **do**
4: *Create a blank list* γ : *a blank list*
5: **for** *n in range* (*len* ($\alpha\beta[i]$)) : *check column number in a row* **do**
6: $\alpha\beta$ [*m*] [*n*] = *int* (*round* ($\alpha\beta$ [*m*] [*n*])) : *round variables within* $\alpha\beta$ *to* 0 *and transform to integer*
7: *verify if* γ *is missing the variable that is found in* $\alpha\beta$ [*m*] [*n*]
8: *Assign* γ = $\alpha\beta$ *once the requirement is met*
9: $j = 0$: *set* j = 0
10: **for** (*z in range* ($\alpha\beta$ [*m*] [*n*] 20)) : *let z equal the variable spanning* $\alpha\beta$ [*m*] [*n*]*up till* 20 **do**
11: **end for**
12: **end for**
13: **end for**
14: **if** *z not in* γ : *verify if the variable does not exist in* γ (*the first one* **then**
15: $\alpha\beta$ [*m*] [*n*] = *z* : *Assign the first value found that does not exist in* γ
16: *Attribute the variable of z to* γ
17: j = 1 : *assign* j = 1, *if variable is found for* γ *already*
18: **interrupt**
19: **if** $j == 0$ **then**
20: **interrupt**
21: **end if**
22: **end if**
23: **for** (*z in reversed* (*range* (0, $\alpha\beta$ [*m*] [*n*])) : *let z be the variable spanning* 0 *up till* $\alpha\beta$ [*m*] [*n*]/ **do**
24: **if** *z not in* γ : *if the variable does not found in* γ (*the first one* **then**
25: $\alpha\beta$ [*m*] [*n*] = *z* : *Assign the first variable found that does not exist in* γ
26: *Attribute the variable of z to* γ
27: $j = 1$ *if variable is found for* γ *already*
28: **interrupt**
29: **end if**
30: **end for**

In the same manner, if a number is found for γ, line 17 will assign 1 to the variable j. The algorithm will break either in line 18 or line 20 depending on the value of j in line 19. From line 23 to line 27, the procedure that should be completed in reverse is carried out. If the condition is true in line 24, then lines 25 and 26 will assign the first value discovered that is not in γ to the variable z. If the condition is false, then line 24 will return false. Last but not least, using the same procedure in reverse order, set j to 1 if a value has already been obtained for γ in line 27. If this requirement is fulfilled, the algorithm will terminate and stop at line 28.

4.3.3 Normalization algorithm description for the planned ANSGA-III

In this part, the normalisation procedure of the reference points as well as the complete population list regardless of the rankings is described, as well as the algorithm that goes along with it in the ANSGA-III. The normalisation procedure can be found in Algorithm 13. All objectives, reference points, and the population sets are normalised between 0 and 1 using this procedure. This phase is crucial because the reference points are constructed using the first quadrant of a single hyperplane. Furthermore, this approach assures that the scales of all objective variables are constant. The normalisation technique is described in depth in Algorithm 8, whereas the normalisation in the ANSGA-III is depicted in Figure 2 as discussed in [17]. Following this sub-section is a brief explanation of the algorithm 8.

Algorithm 8 Normalization $(\alpha^n, \beta_t, \gamma^r, \gamma^s/\gamma^a)$ *procedure*

1: **Input**:β_t, γ^s $(reference\ points)$ $or\gamma^a$ $(aspiration\ points)$
2: **Output**: $\alpha^n, \gamma^r(points\ of\ reference\ on\ scaled\ hyperplane)$
3: **for** $n = 1$ *to* N **do**
4: *Find ideal point* : $\delta_{i=n}^{min} = \min_{\beta \in \beta_t} \alpha_n(s)$
5: *Translate objectives* $\alpha'_n(s) = \alpha_n(s) - \delta_n^{min}\ \forall_s \in \beta_t$
6: *Find extreme points* $(\delta^{n^{max}}\ n = 1, ..., N) of \beta_t$
7: **end for**
8: *Calculate intercepts* ν_n *for* $n = 1, ..., N$
9: *Standardized objectives* (α^k) *using* $\alpha'_m(x) = \frac{\alpha_m(x) - \delta_m^{min}}{a_i}$
10: **if** γ_a *is given* **then**
11: *Map each* γ_a *point on scaled hyperplane* $\alpha'_m(x)$ *and keep the points in the list* γ^r
12: **else**
13: $\gamma^r = \gamma^s$
14: **end if**

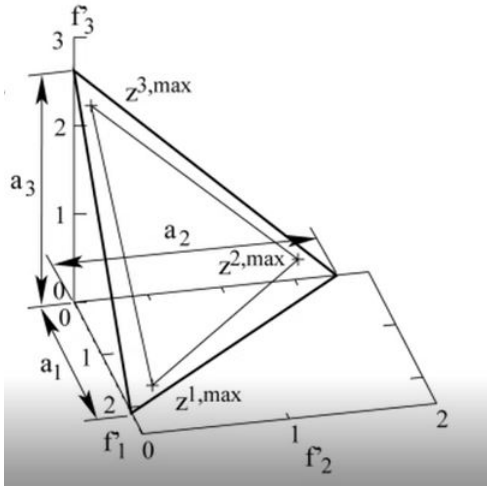

Figure 4.2: Graphical representation of the Normalization algorithm

The "ideal point" is obtained in Algorithm 8 using lines 3 through 6, which explains the minimum for every objective function vector generated per each objective. The objective in line 3 is translated into line 4 by deducting the minimal value $\delta_{m=n}^{min}$ from each objective. Translating objectives into the first quadrant of the hyper-plane will ensure that all objectives read positive values. This translation is required due to the fact that the reference points are generated based on the first quadrant and diversification is preserved with the help of reference point. The extreme solutions are computed on lines 6 through 8 since the extreme solutions do not intersect the $\{\alpha_1, \alpha_2, \alpha_3\}$ axis. These intercepts are calculated by stretching the plane and finding the point where it meets the objectives 5.1, 5.2, and 5.3. The transformed objective $\alpha_m(x)$ is divided by the intercept in line 9 to provide a normalised objective function. To put it another way, this restores the function to its primitive state. Last but not least, lines 10−14 verify that the population size, objectives and reference points are on the same axis.

4.3.4 Association algorithm description for the planned ANSGA-III

The association across each reference point and the nearest reference line is described in detail by the algorithm (9). The angle at which the points are perpendicular to the line is determined. The order of

the solutions has no bearing on this association.

Algorithm 9 Association (B_t, Γ^r) *procedure*

1: **Input**:γ^r, B_t
2: **Output**: $\rho(\beta \in B_t), d(\beta \in B_t)$
3: **for** *every reference point* $\gamma \in \Gamma^r$ **do**
4: *Compute reference line* $z = \gamma$
5: **end for**
6: **for** *every* $\beta \in B_t$ **do**
7: **for** *every* $z \in \gamma_r$ **do**
8: *Determine* $d^{\perp}(\beta, z) = ||(\beta - z^T/||z||^2)||$
9: **end for**
10: *Set* $\rho(s) = z : \mathrm{argmax_{z \in P^r} d^{\perp}(\beta, z)}$
11: *Set* $d(\beta) = d^{\perp}(\beta, \rho(\beta))$
12: **end for**

The inputs consist of the reference points and all the solutions in the front that correspond to those points, and the outputs consist of the reference lines along with the associated minimal solutions and minimum distance. Lines 3 and 4 contain the calculation that determines the associated reference line for each reference point that is provided. For each of the solutions in B_t and each reference line, lines 6*through*8 computed value for the perpendicular distance between the point and the line. In addition to this, the reference line that illustrates the lowest possible value of the solution is calculated. Last but not least, in lines 10 and 11, the reference line that has the solution not far from its distance are kept as $\rho(s)$ and $d(x)$, respectively, for future computation during niching process.

4.3.5 Niching technique description for the planned ANSGA-III

In this subsection, the niching strategy that was employed in the ANSGA-III will be discussed. The matching algorithm may be found in Algorithm (10). The Niching method is used to choose the solutions from the most recent front that are connected to the reference line.

Algorithm 10 Niching $(T, \alpha_{nc}, \delta, dst, \gamma^r, FL_l, PP_{t+1})$

1: **Input**:$T, \alpha_{nc}, \delta(\beta \in B_t), d(\beta \in B_t), \gamma^r, FL_l$
2: **Output**: PP_{t+1}
3: $\tau = 1$
4: **while** $\tau \leq T$ **do**
5: $NC_{min} = nc : \mathrm{argmin_{nc \in \gamma^r} \alpha_{nc}}$
6: $NC' = random(NC_{min})$
7: $I_{NC'} = \beta : \delta(\beta) = NC', \beta \in FL_l$
8: **if** $I_{NC'} \neq \emptyset$ **then**
9: **if** $\alpha_{nc'} = 0$ **then**
10: $PP_{t+1} = PP_{t+1} \bigcup \left(\beta : \mathrm{argmin_{B \in I_{NC'}}} \, d(\beta) \right)$
11: **else**
12: $PP_{t+1} = PP_{t+1} \bigcup arbitrary(I_{NC'})$
13: **end if**
14: $\alpha_{nc'} = \alpha_{nc'} + 1, FL_l = FL_l/s$
15: $\tau = \tau + 1$
16: **else**
17: $\gamma^r = \gamma^r/j'$
18: **end if**
19: **end while**

It is possible that the Last Fronts (LF) solutions will give rise to three niching scenarios ([17]). To make this clear, the **Scenario 1** depicts the circumstance in which there is single solution is associated with the reference line, **Scenario 2** describes the situation where no such solution exists, and **Scenario 3** depicts the circumstance where multiple solutions are associated with the reference line. In the Procedure (10), lines 3 and 4 show the solution from the LF, which is replicated one at a time until the population is full. The reference line with the lowest niche count value is discovered, and one reference line from lines 5 and 6 is picked at arbitrary. In line 7, a search is performed to identify the solution in the LF connected to the selected reference line in line 6. Line 15 eliminates the reference line as a conditional removal if there is no solution connected to the LF. As a result, if the solution from the LF (linked to the reference line) is null, then lines 8 through 19 are true. In the meanwhile, if a solution from the LF is linked to the reference line, the algorithm verifies in line 9 whether the niche count of the reference line is not zero (0). If requirement is met, the procedure will go to line 12, where it will choose a solution at random from the LF and include it in the generation that comes after it. In line 14, the niche count of the reference line is rise by one (1), and the previously chosen solution is removed from the sequence that is now being processed (the LF). Consequently, the counter is incremented by one in line 15 to undertake the next niching procedure. For diversity's sake, the solution relatively close to the reference line in line 10 will be chosen when numerous solutions of the LF are associated with the reference line and the niche count is zero. By contrast, if the number of niches is non-zero and several solutions from the LF are connected to the reference line. Any random solution might be chosen at random from the line that served as the desired reference (12 line). This would imply that one of the solutions from either the first or second front has already been associated with the reference line and diversification has been preserved. It is essential to keep in mind that the search performance would not be improved by any randomly picked solution that was located in close proximity to the reference line.

4.3.6 Coefficient of Variation in Percentage (PCV)

In this work, the Coefficient of Variation in Percentage, also known as PCV is used as a statistical measure of the variation in solutions throughout the non-dominant front regarding the mean of the objective function, regardless of the measuring method used. [174][175]. The PCV tool was used for a dominance and diversity study in [176] and for a software comparison in [177]. The PCV is calculated by dividing the standard deviation of the objective function by its mean, as proposed by [177]. The Standard Deviation and the Average of the Objective Function are calculated using the Distributed Evolutionary Algorithms in Python (DEAP) library. This research makes use of the DEAP library to gather statistics on the activities that are taking place in the optimization [139]. The PCV directly affects the features of diversification, which are proportional to it. This suggests that the diversification features are better when the PCV is greater, as indicated by the [178]. This result will be used to figure out how to use the diversity value that was found during this research.

4.3.7 Difference in Percentage (% Diff.)

This is a method that is used in statistics to express the variation (in percentage and as a fraction of the whole) between the characteristics of two different things at the same time.In this investigation, the percentage difference is employed to indicate the difference in diversity between the ANSGA-III alongside the NSGA-II and MOPSO algorithms. The percentage difference is utilised to describe the difference in

the level of diversity between the three algorithms. The [179] stated Percentage Difference (% Difference.) as

$$\%\ Difference. = \frac{Variations\ between\ two\ variables}{average\ of\ the\ two\ variables} * 100. \tag{4.7}$$

4.3.8 Parallel Coordinate Plot (PCP)

Traditionally, the evolutionary algorithm's solution vectors have been seen using a scatter plot in either two or three dimensions. The form, quality, and dispersion of a non-dominant collection of solutions, as well as the connection between different objectives, may be better grasped with this knowledge. Nevertheless, when there are more than three goals, it's possible that scatter plots in either two or three dimensions may be more difficult to grasp. For this scenario, a Parallel Coordinate Plot is preferable for investigating the corresponding solution sets (PCP). The PCP presents multifaceted data in the form of a two-dimensional chart, with each component of the main data being plotted along a vertical axis. With metaheuristic optimization approach ([180]), a visualising tool called a PCP has received very little attention. In order to further this evolution, this research employed PCP to visualise and evaluate the quality of the solution set supplied by the ANSGA-III as well as the other two examined algorithms. The parallel coordinates plot is a useful tool that makes the comprehension of high-dimensional data easier and more effective. In order to comprehend the behaviour of the six objectives along with the decision variables used in the placement of controllers, the PCP is used. It is generally known that the parallel coordinate plot scales well with the increasing complexity of the dataset, and it is also simple to construct. Based on the available literature, four criteria—convergence, divergence, coverage, and uniformity—have been established for assessing the quality of non-dominated set solutions in metaheuristic optimization techniques. This has been established in the body of academic work. Both convergence and divergence are considered in this investigation. The convergence of a solution is defined as the degree to which several solutions approach one another, or more specifically, the degree to which various solutions approach the actual Pareto front. In the meanwhile, the concept of divergence relates to the process of distinguishing one solution from another. This indicates that the solutions are maintained in isolation from one another. Obtaining a convergence solution while maintaining solution diversity is essential for any evolutionary method.

4.3.9 Hypervolume Performance Indicator (HPI)

In the literature, hypervolume performance metrics have received a lot of attention ([181]). This performance indicator calculates the feasible region that the non-dominant solution dominates. The HPI demonstrates the ability to capture both the convergence and diversity metrics in a single scalar as demonstrated by the [182]. The optimal solution was chosen with the assistance of the reference point by the hypervolume indicator. Because of this property, the HPI is more desirable than other indicators (Generational Distance, Spread, and Entropy), which call for the availability of the true Pareto Front. The true Pareto efficient solutions are unknown in environments such as SD-WAN controller placement and as a direct result, other indicators may not perform well. In continuation of the previously described advantages of the HPI, this research used a HPI as a measure to evaluate the quality of the ANSGA-III in comparison to the existing aforementioned optimization algorithms. The HPI had a scale that went

from zero to one inclusive. Better algorithm performance is associated with hypervolume indicators that are closer to one, according to the [182]. In contrast, when the HPI gets closer to zero, the quality of the performance starts to suffer. In order to evaluate the effectiveness of the developed ANSGA-III algorithm, this research compared the ANSGA-III algorithm to the two most popular equivalent SD-WAN controller placement algorithms, NSGA-II and MOPSO, employing HPI to gauge its efficacy. The performance of these algorithms was evaluated according to convergence and diversification metrics. The results of the experiment, together with a discussion of the findings, are described in the following paragraphs.

4.4 Experimentation

This experiment was carried out on a personal computer with a 2.70 GHz Intel Core i7-6820HQ CPU, 64GB of 1600 MHz DDR3 RAM, and Microsoft Windows 10 Professional Edition installed. The experimental setup was programmed in the Python programming language. Code was compiled using Jupyter Notebook version 6.3.0.. The Hypervolume performance indicator [183] was used in the computation of convergence using the Multi-objective optimization library in Python (pymoo version 0.5.0.). The open-source code can be viewed for free on the GitHub code repository at the following links: https://tinyurl.com/2p95ad26. This section of the research presented a real-world topological test case to illustrate the use of ANSGA-III. The primary goal of this research is to determine where on the Internet2 OS3E architecture [184] a set of $k = 5$ controllers should be placed in order to maximize the achievement of several, potentially competing objectives. This research employs the following six objective functions to optimize controller placements in the presence of many competing objectives. These objectives are resilience (maximum controller failure), load imbalance, maximum inter-controller latency, average inter-controller latency, as well as average switch-to-controller latency and maximum switch-to-controller latency. This network architecture (BtEurope) has 21 nodes, as documented by [184]. The adapted NSGA-III had its settings determined by the following factors. A 495 population size, 6 objective functions, 5 number of dimensions, 8 as a number of divisions, a 495 which represents the user-defined reference point, a 1.0 used as a crossover probability, and a mutation probability which is 1.0 divided by the number of dataset sizes were all used to arrive at these results. The dataset specified a $0 - 20$ range for the bounds, with 0 being the minimum and 20 the maximum. Following the foundational work of [129] and the studies of [185], this research started the experiment's iteration number at 100 and steadily raised it. It was discovered that 75% of the hardware computing capabilities were consumed during the experiment, and no more improvement was made at the five hundred iteration number. Meanwhile, an evaluation network scenario based on the internet zoo topology datasets was employed for this research. [184].

4.5 Results and Discussion

This research employed a set of six objective functions to guide the optimization of controller placements, addressing a multitude of competing objectives. These objectives included resilience (minimizing controller failure), load balancing, maximum inter-controller latency, average inter-controller latency, as well as average and maximum switch-to-controller latencies. To achieve these results, specific parameter settings were utilized: a population size of 495 individuals, six objective functions, five dimensions, eight

divisions, a user-defined reference point set to 495, a crossover probability of 1.0, and a mutation probability calculated as 1.0 divided by the number of dataset sizes. The dataset was bounded within the range of 0 to 20, with 0 representing the minimum and 20 the maximum values. In line with established practices in the field, NSGA-II and MOPSO were selected as the benchmark algorithms for evolutionary and combinatorial optimization problems. This aligns with the prevailing trends in the literature, facilitating a meaningful comparison with our proposed evolutionary algorithm. The experiments were executed on a personal computer equipped with a 2.70 GHz Intel Core i7-6820HQ CPU, 64GB of 1600 MHz DDR3 RAM, and Microsoft Windows 10 Professional Edition. The entire experimental setup was implemented using the Python programming language within Jupyter Notebook version 6.3.0. The Hypervolume performance indicator, sourced from the multi-objective optimization library in Python (pymoo version 0.5.0), was employed to assess convergence. For transparency and reproducibility, the open-source code is freely accessible on our GitHub code repository: https://tinyurl.com/2p95ad26. Additionally, the benchmark algorithm's source code was obtained from ' Yarpiz - Academic Source Codes and Tutorials https://yarpiz.com/ and adapted to incorporate the necessary parameters for optimal controller placement. In evaluating the algorithms' performance, the Hypervolume performance indicator played a pivotal role, offering a comprehensive assessment of their effectiveness.

4.5.1 Hypervolume Analysis Results

The Figure 4.3, shows a composite of Figures 4.4 through 4.6, for a comprehensive examination of the hypervolume indicators. The convergence graph of the three algorithms is depicted in Figure 4.7. The descriptive chart of the hypervolume indicator is displayed in Figures 4.4 through 4.6 for the ANSGA-III, NSGA-II, and MOPSO techniques, accordingly. Figures 4.4 through 4.6 shows hypervolume indicator descriptive charts for the ANSGA-III, NSGA-II, and MOPSO algorithms. The adapted NSGA-III, with a measurable HPI of 0.94876, was the most performed of the three techniques, as seen in the graph. The NSGA-II's with a measurable HPI was 0.94314, while the MOPSO algorithm's HPIr was 0.91168, the lowest of the three and the least close to 1. The modified version of NSGA-III's hypervolume improvement (0.9488), which is more than NSGA-II's (0.9431) and MOPSO's (0.9117) (all of which are between 0 and 1), is considered to be significant. This adheres to the core idea developed by [186] Pareto and put into practice by [187],[188], and [182]. The same meaning applies to Figures 4.4 through 4.6 as it does to Figure 4.3. In a similar manner, when it comes to the convergence, the ANSGA-III achieved the maximum (most near to 1) measurable HPI of 0.94876. This was the case among all three algorithms. The NSGA-II algorithm had a verifiable hypervolume indicator value of 0.93646 whereas the MOPSO algorithm achieved an identifiable HPI of 0.89348. The MOPSO algorithm achieved the mimimum value (the least closer to 1 among the three indicators). In the meanwhile, the number of generations in the convergence method, which was initially set at 500, was multiplied by 100 as a result of the computations performed by the internal library of DEAP. According to these results, the ANSGA-III method possesses the maximum level of convergence and diversity, whereas the MOPSO algorithm possesses the lowest level of convergence and diversity. Consequently, out of the three algorithms, only the ANSGA-III has the maximum convergence and diversity properties.

4.5.2 Convergence Analysis Results

The convergence chart of the three methods is depicted in Figure 4.7. In a follow-up to the research of [129], the algorithm's point of convergence is established when no more improvement leads to an optimal

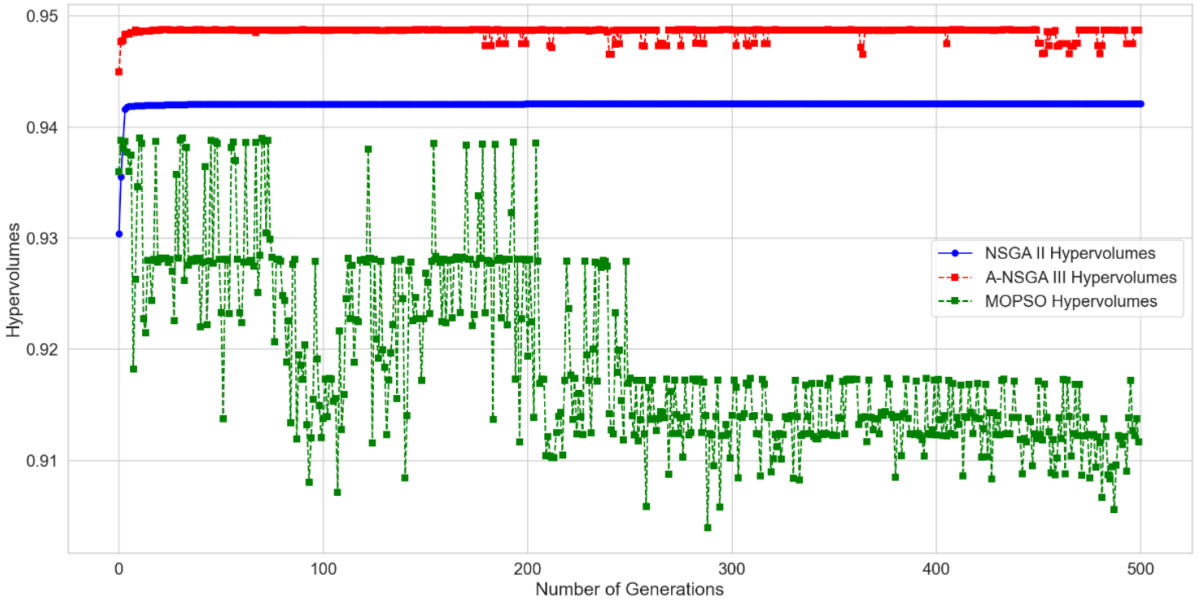

Figure 4.3: Measure of Hypervolume for the three Algorithms

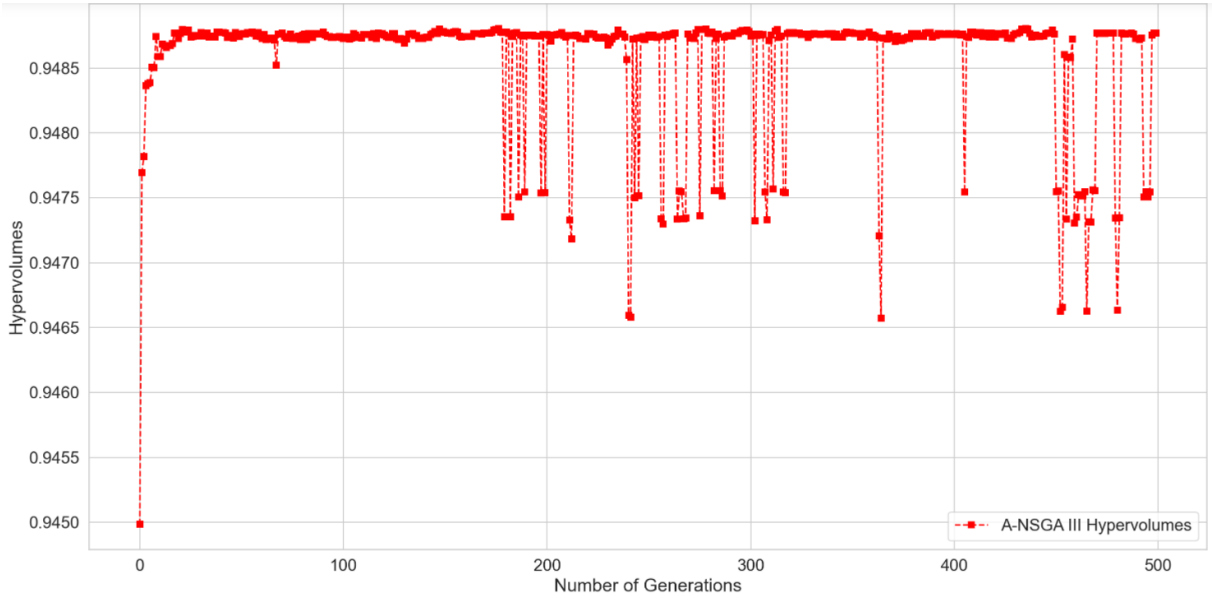

Figure 4.4: Measure of Hypervolume for the ANSGA-III Algorithms

solution. In this experiment, the number of iterations started at 100 and was slowly increased until it reached 500. However, during these 500 iterations, a new optimal solution did not appear. This research utilises the usage of the HPI to shed additional light on the nature of this convergence, which is displayed in Figure 4.7, in order to evaluate the convergence's quality further and provide a more in-depth analysis of it. The value of the measurable hypervolume indicator was 0.94876 for the ANSGA-III, which was the maximum among the three algorithms and the one that was closest to 1. The NSGA-II algorithm had a verifiable HPI of 0.93646 whereas the MOPSO algorithm achieved an identifiable HPI of 0.89348.The MOPSO algorithm achieved the lowest value (the least closer to 1 among the three indicators). In the

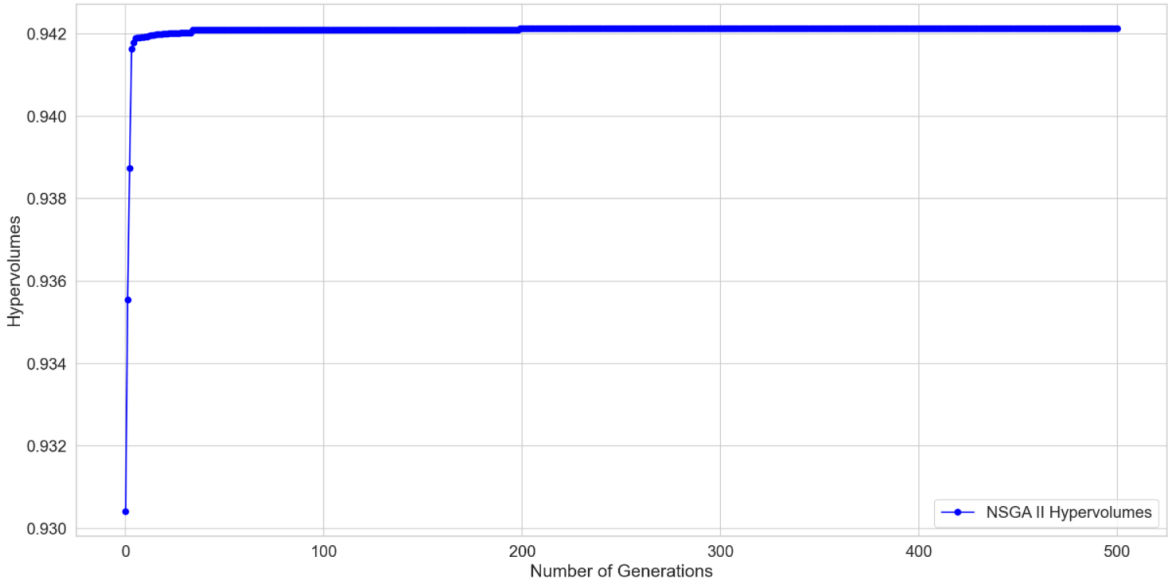

Figure 4.5: Measure of Hypervolume for the NSGA-II Algorithms

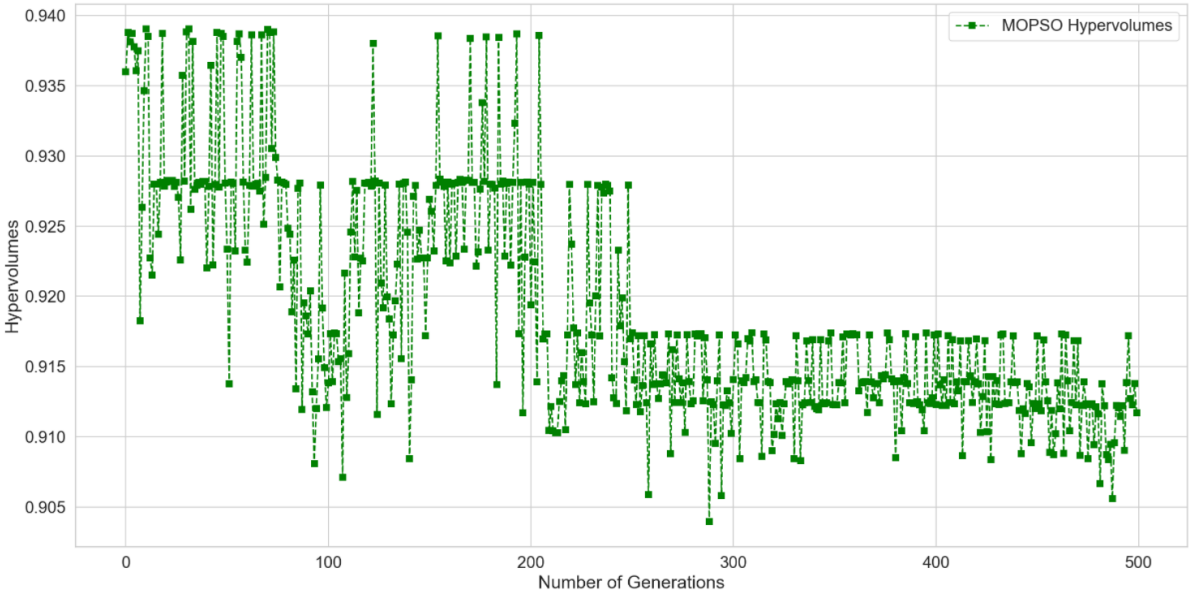

Figure 4.6: Measure of Hypervolume for the MOPSO Algorithms

meanwhile, the number of generations in the convergence method, which was initially set at 500, was multiplied by 100 as a result of the computations performed by the internal library of DEAP.

4.5.3 Percentage of Coefficient Analysis Results

Further, the deductive approach of the PCV and the standard deviation related to the six objectives across the three methods under consideration are displayed in Table 4.1. It was found in the Table 4.1, that the PCV [176] revealed substantially more inner genetic variation than the standard deviation utility did. As a consequence of this, the outcomes of the PCV were utilised to give further insight into the diversification features of the six objectives that were included in each one of the three algorithms

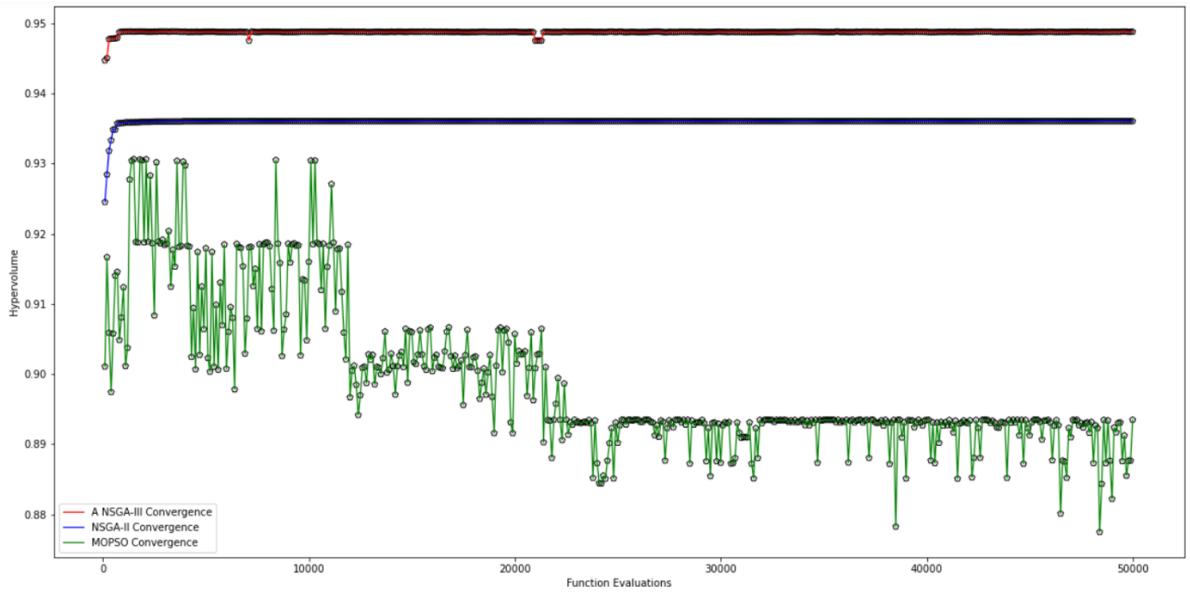

Figure 4.7: The three optimization algorithms' convergence graph

that were taken into consideration. The PCV directly affects the features of diversification, which are proportional to it. This suggests that the diversification features are better when the PCV is greater, as indicated by the [178]. The findings of the comparison may be seen in Table 4.1, which shows that the ANSGA-III has a higher PCV than the NSGA-II does for all six objectives. In a similar way, the ANSGA-III achieves a higher PCV than the MOPSO algorithm for five of the objectives, with the exception of the 2 objective. Because of this, it can be deduced that the ANSGA-III has the greatest overall gain in diversity compared to NSGA-II and MOPSO algorithms.

Table 4.1: Evaluation of Diversity Based on the Standard Deviation and the Variance Coefficient

		A-NSGA-III			NSGA-II			MOPSO		
SN	OF	AOF	SD	PCV (%)	AOF	SD	PCV (%)	AOF	SD	PCV (%)
1	0	0.2928	0.1079	36.8670	0.3651	0.0818	22.4033	0.4856	0.0312	6.4200
2	1	0.4067	0.0769	18.9024	0.4704	0.0633	13.4573	0.5078	0.0727	14.3162
3	2	0.6823	0.1613	23.6440	0.5607	0.1161	20.7105	0.3870	0.1128	29.1490
4	3	0.6271	0.2402	38.3098	0.6738	0.1554	23.0668	0.9022	0.0454	5.0297
5	4	0.0267	0.0066	24.8801	0.0318	0.0066	20.7946	0.1718	0.0201	11.7147
6	5	0.1068	0.0266	24.8808	0.1271	0.0264	20.7965	0.0430	0.0050	11.7140
		Percentage Total		167.4841			121.229			78.3436
		Percentage Difference					32.04%			72.52%

OF: Objective Function AOF: Average of Objective Function SD: Standard Deviation
PCV: Percentage Coefficient of Variation

Table 4.1 demonstrates that the adapted NSGA-III has the equivalent maximum PCV of (36.867, 18.9024, 23.644, 38.3098, 24.8801, 24.8808) respectively than the NSGA-II (22.4033, 13.4573, 20.7105,

23.0668, 20.7946, 20.7965) and the MOPSO approach (6.42, 14.3162, 29.149, 5.0297, 11.7147, 11.714) for the objective functions 0 through 5. In a similar vein, the percentage coefficient variation investigations reveal that the ANSGA-III possesses a percentage coefficient variation total of 167.4841%, which is a significant improvement over NSGA-II, which achieved a percentage coefficient of variation total of 121.229%, and the MOPSO approach, which achieved a PCV total of 78.3436% respectively.

4.5.4 Percentage Difference Analysis Results

The developed ANSGA-III performs better than both the MOPSO and NSGA-II, with a difference of 32.04% and 72.52%, respectively, when comparing the percentage differences (% Diff.) between the algorithms. This result supports the previous finding that, when the number of objectives is more than three, adapted NSGA-III outperforms MOPSO and NSGA-II in terms of diversification. This shows that the adapted NSGA-III method is better at scaling than both the MOPSO and NSGA-II models.

4.5.5 Experiment Execution Time Results

In a similar manner, Figure 4.8 displays the combined qualitative feature of the processing time (in secs) for the three models, whereas figures 4.9 through 4.11 display the qualitative feature of the processing time (in seconds) for each of the three models. The highest number of iteration for each model was limited to 500.

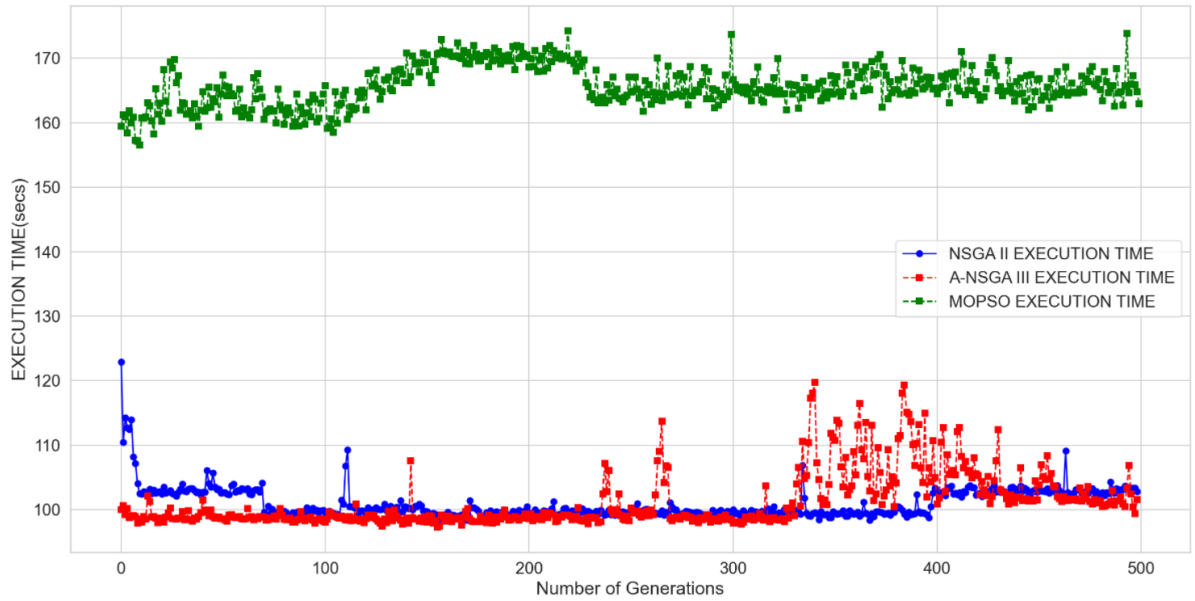

Figure 4.8: The three optimization algorithms' execution times

The modified NSGA-III achieved an average processing time of 100.961 seconds, which was quite close to the NSGA-II (100.766 secs). On the other hand, the MOPSO model required an average of 165.652 seconds to complete.

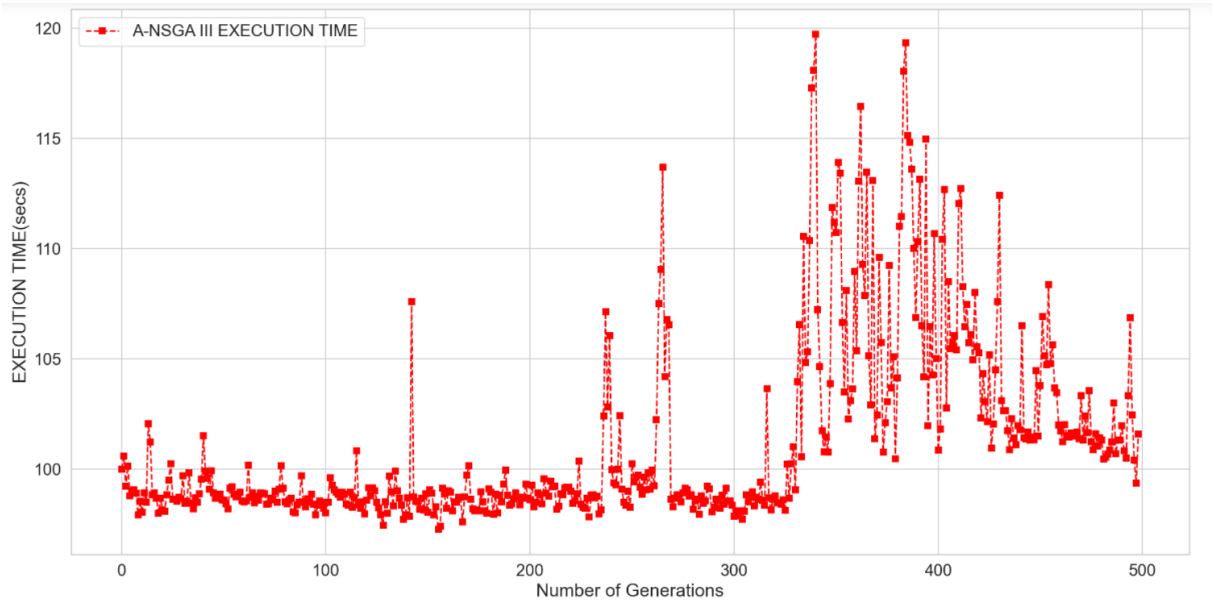

Figure 4.9: The ANSGA-III algorithm execution time

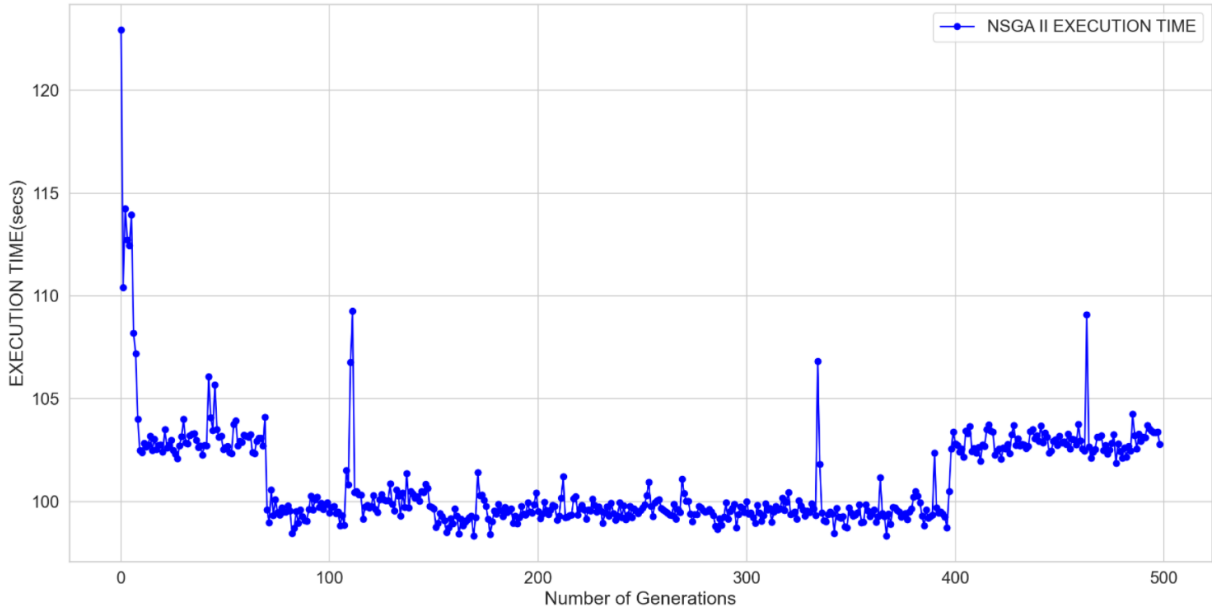

Figure 4.10: The NSGA-II algorithm execution time

4.5.6 Parallel Coordinate Plot (PCP) Result

Figures 4.12 through 4.14 show the parallel coordinate charts for all three methods. All three techniques were shown to converge on the true Pareto front. The diversity of solutions that stretched into the border was maintained by the proposed adapted NSGA-III, which did not cluster its solutions in a single location. On the other hand, it was discovered that solutions from the NSGA-II beginning with objective 1 and continuing through objective 6, were clustered together, and there was no indication of diversification anywhere in the objective solution space. Based on this conclusion, it appears that NSGA-II does not successfully preserve a sufficient degree of variation among the solutions. Similar to the NSGA-II method,

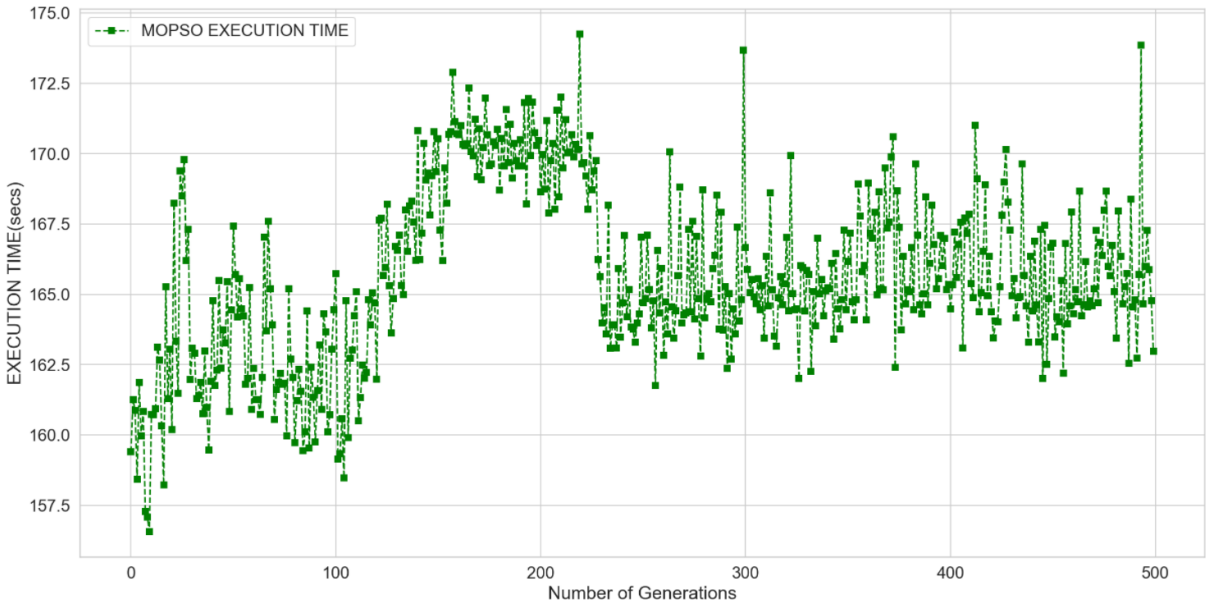

Figure 4.11: The MOPSO algorithm execution time

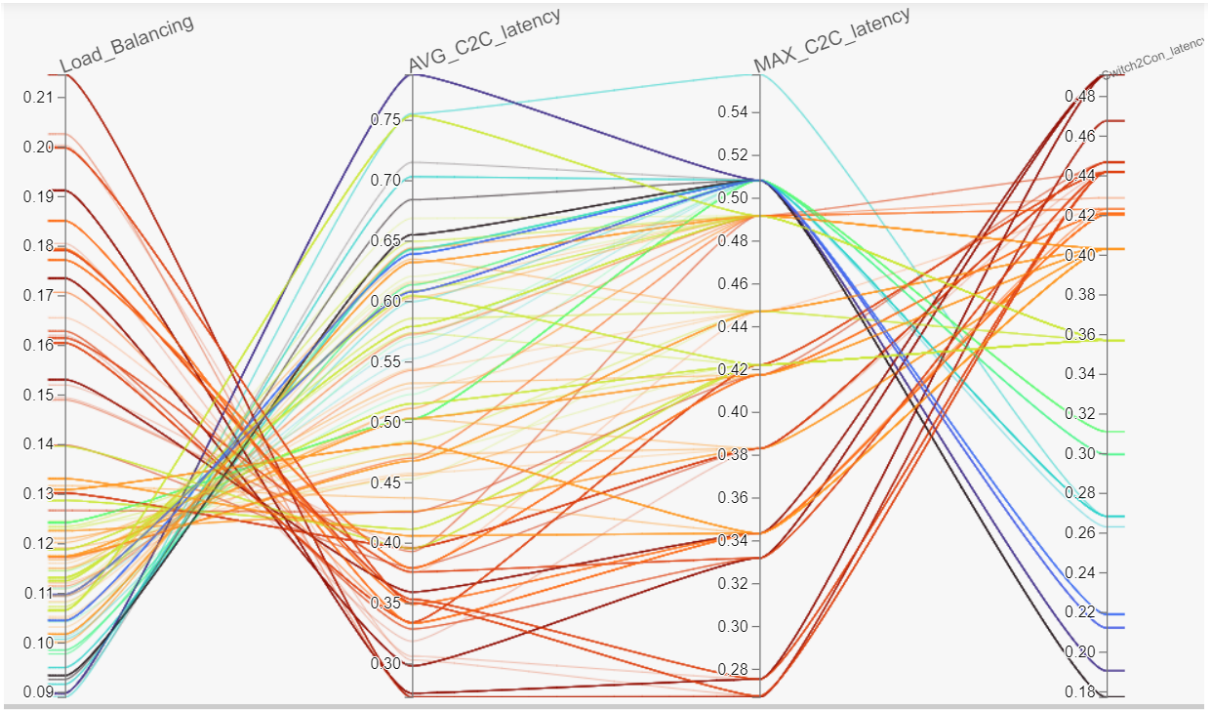

Figure 4.12: An ANSGA-III parallel coordinate visualisation of the solution

the MOPSO algorithm does not sustain a satisfactory convergence toward the genuine Pareto front. This is seen by the parallel coordinate plot, which ranges between 0.28 and 0.42. On the other hand, this contradicted what the ANSGA-III and NSGA-II demonstrated. Additionally, it was found that the MOPSO algorithm has difficulty covering the solution frontier for a number of objectives. Meanwhile, it appeared that the results obtained by the adapted NSGA-III and NSGA-II models achieved a satisfactory level of convergence throughout the whole Pareto front. These outcomes not only supported the evidence that

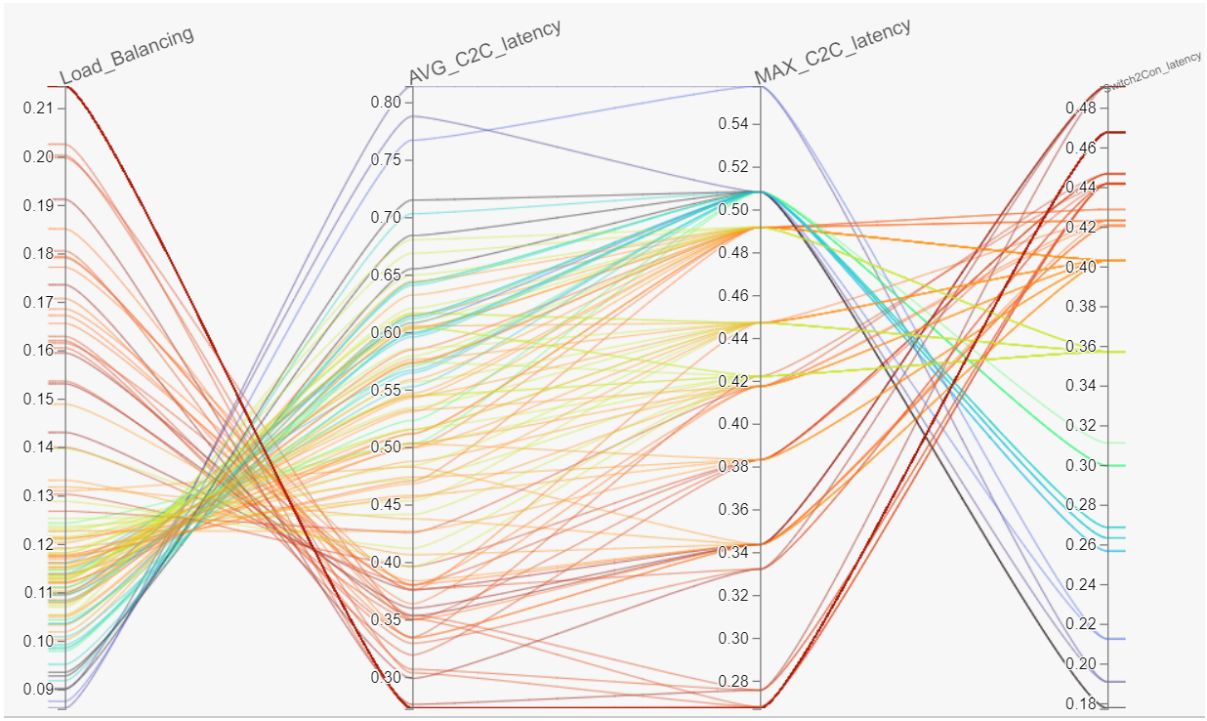

Figure 4.13: A NSGA-II parallel coordinate visualisation of the solution

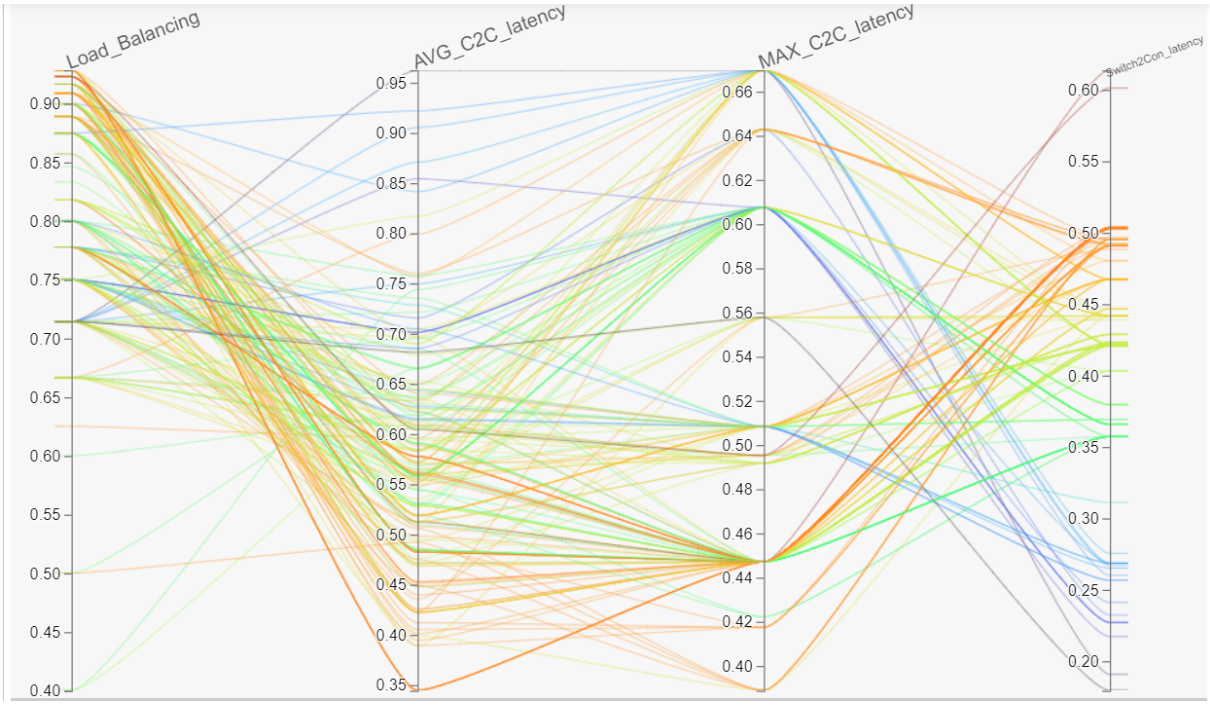

Figure 4.14: A MOPSO parallel coordinate visualisation of the solution

was provided with the hypervolume performance metrics, but they also complemented it.

Non-dominated solutions in three-dimensional space achieved by the three methods are shown in Figures 4.15 through 4.17. According to the results, the adapted NSGA-III has a more diverse set of

non-dominated solutions than the NSGA-II and MOPSO algorithms because its solution is more evenly distributed over the objective space. The solutions found by NSGA-II and MOPSO tend to cluster in a small number of locations rather than being evenly dispersed over the objective space. These clustering behaviors displayed by the NSGA-II and MOPSO algorithms demonstrate that these algorithms are not uniformly dispersed over the Pareto Fronts. Based on these results, it can be concluded that adapted NSGA-III solutions were dispersed throughout a broad region of the Pareto Front. The outcome was a positive addition to the outcomes that were acquired in the earlier results.

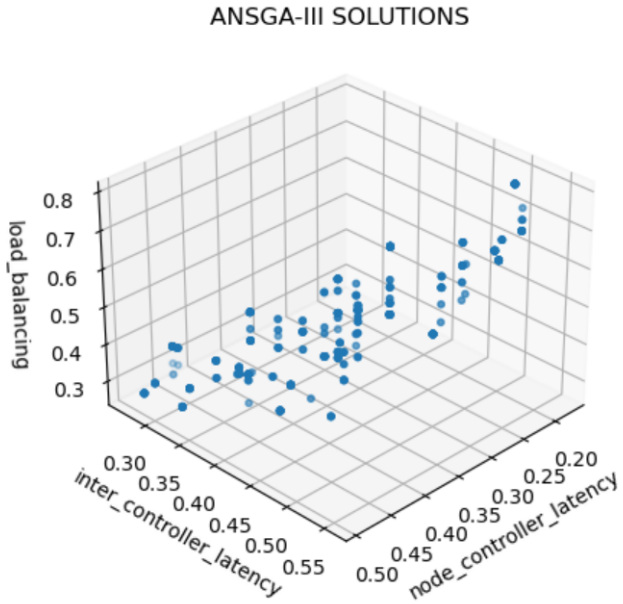

Figure 4.15: Scatter plots in three dimensions for ANSGA-III Pareto sets

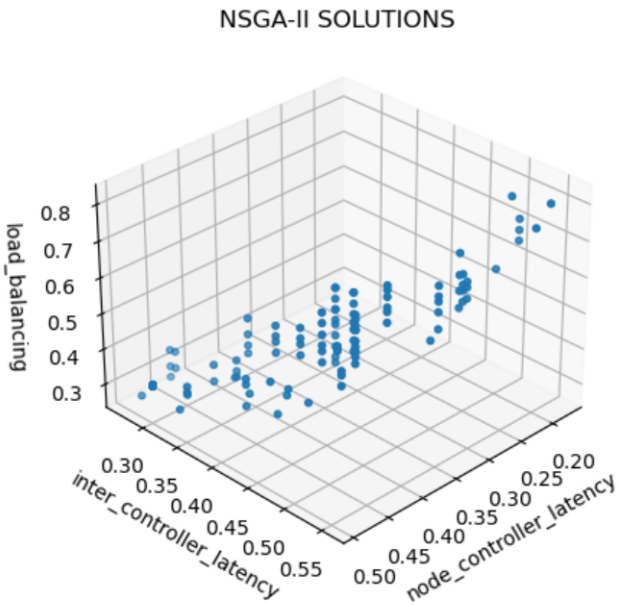

Figure 4.16: Scatter plots in three dimensions for NSGA-II Pareto sets

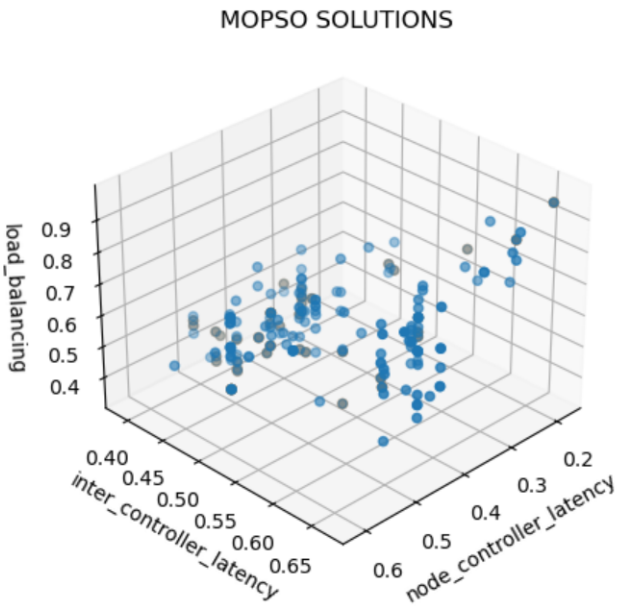

Figure 4.17: Scatter plots in three dimensions for MOPSO Pareto sets

4.6 Verification and Validation of the developed ANSGA-III

In this part of the thesis, the author shows that the developed model has met the main goal in terms of its quality and credibility by showing that it has done what it was meant to do. The verification process includes everything that goes into making a high-quality solution, like testing, analyzing the design, analyzing the specifications, and so on. The technique can be thought of as being fairly objective. In contrast, the process of validation is extremely subjective in nature. It involves making subjective decisions about how well a solution that has been proposed or made meets a real-world need. The validation process includes many different steps, such as modeling the requirements, making a prototype, and testing with users. The specifications were followed closely during the planning and construction of the suggested solution. The aim of the developed solution was to address one of the gaps (scalability) identified in Chapter 2 of this thesis, namely, the inability of the existing metaheuristic algorithms to work efficiently when the number of objectives to simultaneously optimize exceeds three. Looking at the Table in 4.2 it is clearly seen that the developed ANSGA-III for the optimization of controller placement in SD-WAN actually fulfills its intended design aim and meets the expected outcomes. The scalability is measured with the standard deviation and coefficient of variation which is directly proportional to the model diversification. The outcomes of the PCV in table 4.2 were utilized to give insight into the diversification features of the six objectives in each of the three algorithms that were taken into consideration. The PCV directly affects the features of diversification, which are proportional to them.

Table 4.2: Diversity Evaluation Using Standard Deviation and Variance Coefficient

		A-NSGA-III			NSGA-II			MOPSO		
SN	OF	AOF	SD	PCV (%)	AOF	SD	PCV (%)	AOF	SD	PCV (%)
1	0	0.2928	0.1079	36.8670	0.3651	0.0818	22.4033	0.4856	0.0312	6.4200
2	1	0.4067	0.0769	18.9024	0.4704	0.0633	13.4573	0.5078	0.0727	14.3162
3	2	0.6823	0.1613	23.6440	0.5607	0.1161	20.7105	0.3870	0.1128	29.1490
4	3	0.6271	0.2402	38.3098	0.6738	0.1554	23.0668	0.9022	0.0454	5.0297
5	4	0.0267	0.0066	24.8801	0.0318	0.0066	20.7946	0.1718	0.0201	11.7147
6	5	0.1068	0.0266	24.8808	0.1271	0.0264	20.7965	0.0430	0.0050	11.7140
		Percentage Total		167.4841			121.229			78.3436
		Percentage Difference					32.04%			72.52%

OF: Objective Function AOF: Average of Objective Function SD: Standard Deviation
PCV: Percentage Coefficient of Variation

This implies that the diversification features improve as the PCV increases, as indicated by the [178]. The results of the comparison may be seen in Table 4.2, which shows that the ANSGA-III has a higher PCV than the NSGA-II does for all six objectives. This shows that the ANSGA-III method is better at scaling than both the NSGA-II and MOPSO algorithms, which confirms the verification process of the developed solution and how it actually meets its intended purpose. In a similar way, this thesis uses the comparison of outputs as a validation process to make sure that the model that was built is credible and meets user satisfaction. The hypervolume performance indicator, which measures how much of the objective space the Pareto set solution takes up, was used to judge the quality of the ANSGA-III model and other optimization models like NSGA-II and MOPSO.

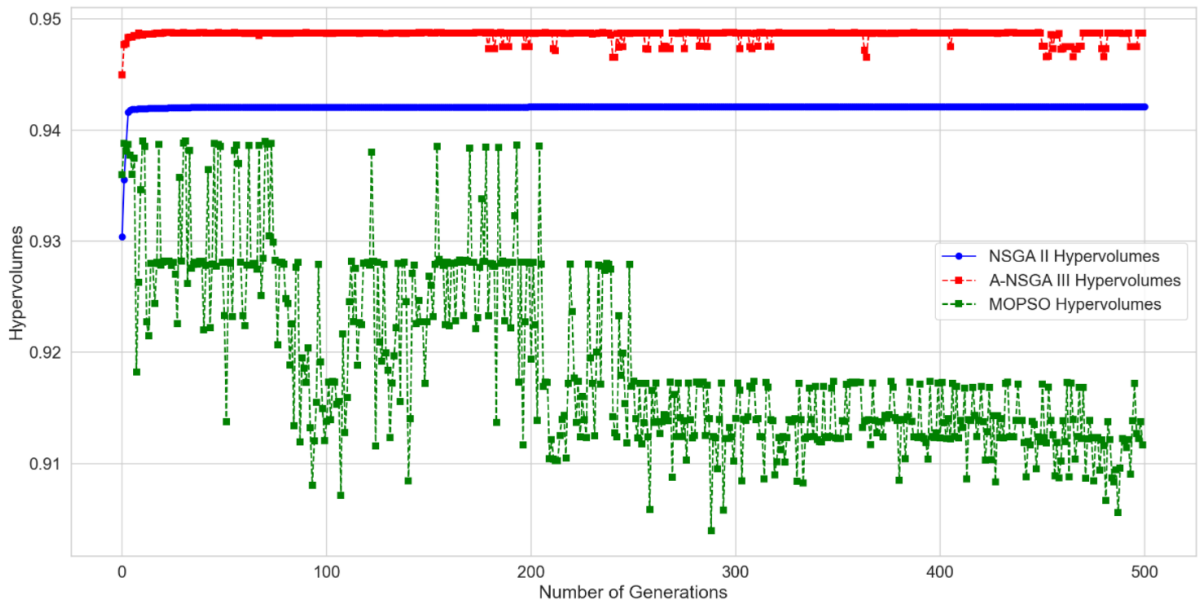

Figure 4.18: Hypervolume Indicator for the three Algorithms

As exhibited by the reference figure 4.18, the hypervolume indicator for the ANSGA-III shows the highest hypervolume (convergence and diversity), which confirms the validity of the developed model. This hypervolume indicator had a scale that went from zero to one, inclusive. Better algorithm performance is associated with hypervolume indicators closer to one; similarly, the indicator measures both convergence and diversity metrics and is recorded as a single scalar, as shown in 4.18.

4.7 Conclusion Remarks

There has been a lot of focus on finding the best location (in terms of both number and location) for SD-WAN controllers among a number of conflicting objectives. NSGA-II and MOPSO are two metaheuristic optimization strategies that have been used to find good solutions to the CPP in SD-WAN. However, these techniques were associated with a scalability challenge when there were more than three competing objectives that needed to be optimised simultaneously for SD-WAN controller placement. Therefore, this research introduced a modified version of the NSGA-III algorithm referred to as ANSGA-III to deal with the scaling problems encountered by the NSGA-II and MOPSO algorithms in the presence of more than three objectives. This research built and incorporated a RBO into the previously established industrial engineering-based NSGA-III in order to propose the ANSGA-III for optimal controller placement in the SD-WAN. The recommended ANSGA-III, NSGA-II, and MOPSO methods were put to the test using Internet zoo topology datasets that included six different objective functions. High convergence and diversification were shown by the suggested ANSGA-III over the NSGA-II and MOPSO methods in the presence of the scalability challenge. This was demonstrated by the HPI, the PCV, and the percentage difference, as well as the PCP. In addition, the suggested ANSGA-III was able to scale better than the other two methods. The results of the experiment revealed that the recommended ANSGA-III was effective in resolving the scalability challenges related to the optimal controller placement in the SD-WAN. As a direct consequence of this, the ANSGA-III approach was recommended in preference to both the NSGA-II method and the MOPSO method.

It was established that the NSGA-II and MOPSO are characterised by their incapacity to acquire a wide range of solutions among the non-dominant solutions. The ANSGA-III algorithm performs better than the NSGA-II and MOPSO algorithms when there are more than three objective functions. It is well known that both of the optimization strategies that were studied, MOPSO and NSGA-II, as well as the modified NSGA-III that was proposed, require a significant amount of computing effort. This has also been verified by the studies presented in this research. Even though the adapted NSGA-III supports scalability regarding the number of objectives that can be optimised simultaneously in the placement of controllers in software defined wide area networking. Neither the currently available solutions (MOPSO and the NSGA-II) nor the suggested adapted NSGA-III are able to automatically learn heuristics for combinatorial optimization tasks like SD-WAN controller placement. This is an issue that requires immediate attention. This research will present a less computationally expensive and intelligent framework as a direct result of this in the next chapter. The framework will be able to calculate controller placement in the face of several competing objectives in less time. Additionally, the solution will be able to predict controller placement when there is an expansion in the network topology rather than subjecting the datasets to optimization techniques each time.

Chapter 5

An Intelligent-based solution to address Controller Placement problem in SDN

5.1 Introduction

Combinatorial optimization problems come up in many different areas, such as decision-making, planning, telecommunications, transportation, routing, and scheduling [105]. This research is looking into the SD-WAN controller placement problem, which is an example of a combinatorial optimization problem. There are several different goals that need to be optimized at the same time to get the best controller placement. Adapted Non-Dominated Sorting Genetic Algorithm III (ANSGA-III) [7], Non-Dominated Genetic Algorithm II (NSGA-II) [107], and Multi-Objective Particle Swarm Optimization (MOPSO) [108] are examples of metaheuristic algorithms that have been suggested as ways to find near-optimal solutions. However, these approaches are associated with some drawbacks that need urgent attention. Existing methods have problems like not being efficient computationally, not being able to learn the heuristics of combinatorial optimization, and not being able to optimize controller placement well when there are more than three goals (except for ANSGA-III).

It has been confirmed in this study that the optimization algorithms explored, MOPSO and NSGA-II, as well as the ANSGA-III require a substantial amount of computational effort. (Please see Chapter 5, subsection 5.6.3, figure 5.19). Despite the fact that the adapted NSGA-III supports scalability in terms of the number of objectives that can be simultaneously optimized in SD-WAN controller placement, neither the currently available solutions (MOPSO and the NSGA-II) nor the suggested adapted NSGA-III can automatically learn heuristics (ability to predict the number and placement of controllers) for combinatorial optimization tasks such as SD-WAN controller placement. Due to the complex nature of the state-of-the-art algorithms (ANSGA-III, NSGA-II, and MOPSO), which are known to rely on heuristics that have been built by hand in order to make decisions that would otherwise be either too expensive to calculate or not adequately defined mathematically, an intelligent-based solution that is quick enough is therefore needed to compute both the optimal number of controllers and their placement [109][106]][110].

Due to the limitations of existing methods, this study suggests using a stochastic computational graph model with an ensemble learning approach to figure out where the best controller should be placed and how many controllers are needed for an SD-WAN deployment when different performance metrics are at odds with each other. The solution proposed in this study is a combination of a stochastic computational graph model and an Extreme Gradient Boosting Machine Learning Regression Model (SCGMEL). This

model solves the problem with the existing techniques used to place SD-WAN controllers. This study further suggests using an artificial neural network called learning vector quantization to predict where to put controllers in an SD-WAN topology, especially as the network grows. This approach is subject to the administrator and the service providers on account of its usage.

The proposed strategy is in line with the identified gaps in the second paragraph of 4.7. The first part of this chapter is going to be an introduction to the subject matter. After the introduction, the problem definition and the objective functions that were used to evaluate this method will be discussed. The proposed stochastic computational graph model with an ensemble learning model is explained in Section III of this study. Section IV discusses how this study was done through experiments, while the results and discussion of the proposed study, along with the comparison with the existing approach in the literature, are discussed in Section V. Finally, Section VI concludes the work.

5.2 Optimization Design for Controller Placement Problem

The research undertaken in this study delved into the critical realm of CPP within SD-WAN networks. In doing so, a novel approach for strategically situating controllers within the SD-WAN architecture was introduced. This method was specifically designed to cater to the simultaneous fulfillment of multiple, and often conflicting, network requirements. While the latency between switches and controllers stands out as the paramount consideration in CPP scenarios, it's imperative not to overlook other equally vital factors. These objectives encompass a spectrum of concerns, including but not limited to redundancy, equitable distribution of controller workloads (load balancing), and latency considerations in both switch-to-controller and inter-controller communication, spanning both average and worst-case scenarios [21]. This research not only addresses the core challenge of minimizing switch-to-controller latency but also recognizes the broader landscape of requirements that must harmoniously coexist within the SD-WAN network.

The optimization design for controller placement in SD-WAN is a critical research area given its pivotal role in enhancing network performance and efficiency [70]. This problem is inherently combinatorial, involving the intricate task of determining the optimal locations for SDN controllers in a way that simultaneously optimizes multiple, often conflicting, objectives [12]. In the realm of SD-WAN, where the network's performance is vital, optimization becomes paramount. The primary objectives typically include minimizing latency, ensuring load balancing, maximizing network dependability, and reducing operational costs. Achieving these goals is complex, as they often compete with one another [8]. For instance, reducing latency might involve placing controllers closer to end-user devices, but this could lead to uneven load distribution and decreased network dependability. To tackle this multifaceted challenge, researchers have turned to metaheuristic optimization algorithms [128]. ANSGA-III, NSGA-II, and MOPSO have been among the algorithms of choice. However, while these algorithms have shown promise in addressing controller placement issues, they are not without limitations. One of the most pressing issues is computational efficiency. As SD-WAN deployments scale up to handle large networks, the computational demands placed on these algorithms increase exponentially. This often leads to prohibitively long optimization times and renders these algorithms impractical for real-world, large-scale

deployments. Another critical limitation is the algorithms' inability to learn the heuristics of combinatorial optimization specific to SD-WAN controller placement [7]. In dynamic network environments, where traffic patterns, user demands, and network conditions are constantly changing, the ability to adapt and optimize in real-time becomes crucial [21]. Existing algorithms struggle to capture these nuanced heuristics and adapt their solutions accordingly. Furthermore, when faced with more than three conflicting objectives, most algorithms exhibit scalability challenges [7]. The inability to efficiently handle these scenarios limits their applicability in complex SD-WAN deployments where optimizing multiple objectives is essential. In response to these challenges, this research aims to pioneer innovative strategies that overcome computational complexity, harness optimization heuristics, and adapt to the dynamic nature of SD-WANs. Employing advanced machine learning techniques such as the Stochastic Computational Graph approach in conjunction with ensemble learning methods, notably XGBoost, and a supervised classification algorithm known as Learning Vector Quantization, this research aims to craft optimization and predictive models. These models are primed to assume a pivotal role in discerning the optimal number of controllers and their precise placements within the SD-WAN network. The envisaged result is a substantial enhancement in SD-WAN network performance and efficiency, effectively addressing the constraints inherent in current optimization methodologies. In conclusion, the optimization of controller placement in SD-WAN networks represents a multifaceted yet indispensable research domain. The pressing task at hand is to surmount challenges related to computational efficiency, heuristics learning, and scalability, all of which are instrumental in unlocking the full potential of SD-WAN technology. By pioneering innovative approaches and leveraging the capabilities of machine learning, this research aspires to blaze a trail toward more efficient, adaptable, and effective SD-WAN deployments in the immediate future.

Based on the outcomes of this study, an unconstrained, many-objective CPP for controller placement was designed. The objective functions of this study are described in Equations (5.1) through (5.6) to the reader. Specifically, the SD-WAN is constructed in the form of an undirected graph, denoted by the equation $G{=}(\Phi, E)$, where Φ is the set of nodes and E is the set of links between them. Meanwhile, the shortest path delay between any two nodes may be found in a distance matrix designated by the letter Dm, which can be used in the placement computation. $d\eta\theta$ described the delay that occurs between the node η and node θ. Take into consideration the fact that the authors of this study arrived at their basis for normalisation by dividing the delay recorded in Dm by the diameter of the graph that corresponded to it. In order to get the desired outcome, which is determined by the number of controllers that are sought, the search space is limited to a predetermined collection of $\binom{\sigma}{\rho}$ locations.

This study came to the conclusion that placement should be referred to as a σ-element, where ϕ is a subset of Φ. The search space for the CPP is the ϕ-subset of Φ, which is a space that includes all of the possibilities that might be chosen. In the event that there is a network design with 21 nodes and a predetermined number of controllers, such as $\phi{=}7$ for instance, the set $\rho{=}\{4, 5, 8, 10, 11, 13, 16\}$ denotes controller locations ($|CL|{=}7$. This is an example to explain the point. In order to fulfill the requirements of the assumption, the seven controllers have to be situated at nodes $4, 5, 8, 10, 11, 13, and 16$. It is essential to keep in mind that rearranging the members of any subset will not result in the formation of any new combination. As a consequence of this, in this particular case, the total number of possible locations in this network is $\binom{21}{7}$. It is anticipated that a set of objectives referred to as $\{\tau_1, \tau_2, \ldots, \tau_m\}$ will be reduced to the greatest extent possible. If there is no other potential placement δ in the search space then the solution ρ is the best possible alternative according to the Pareto principle. In other words,

all things considered, $\forall_\eta\ \theta_\eta(\delta) \leq \theta_\eta(\rho)$ and $\theta_\eta(\delta) < \theta_\eta(\rho)$ must have at least one index η. This study has to deal with the CPP to find the best set of solutions for the whole solution space as well as the set of objective values for all of the best placements, which together make up a set of solutions. In the next section, this study will discuss the objective functions that are used to find the best decision variables.

5.2.1 Objective functions

In this part of the study, the authors present a summary of the objective that was examined. The objective function in this study can also be referred to as the cost function or loss function. It is an important part of controller placement. The objective functions are used to optimize the decision variables (controller location). For further reading on the importance of the objective functions discussed in this section, the reader is referred to the study in [112]. When deciding where to put the controllers, there are several different priorities that need to be taken into account, each of which is contingent on the specific configuration of the controllers. This study has considered six different objectives that are conflicting in nature in optimizing controller placement. Although there are several objectives that can be used to optimize the placement, this research has carefully considered those that have a major effect on the service providers and that the user of SD-WAN cannot afford not to consider when optimizing controllers in the face of several conflicting objectives. The first two performance metrics (objective functions) display the maximum and average switch-to-controller latency. These two objectives ensure that average-case and worst-case latency between the controller and its associated switches are considered when optimizing placement, with the goal of minimizing both. The equations (5.1) and (5.2) calculate the maximum and average switch-to-controller latency for each controller placement $PL \in 2^{\Phi}$ and the defined distance matrix Dm, respectively).

$$\gamma^{Lat\_max\_N2C}(CL) = \max_{\phi \in \Phi} \min_{cl \in CL} d_{\phi,cl}, \tag{5.1}$$

$$\gamma^{Lat\_avg\_N2C}(CL) = \frac{1}{|\Phi|} \sum_{\phi \in \Phi} \min_{pl \in CL} d_{\phi,cl}. \tag{5.2}$$

In SD-WAN architecture, where several or more than one controller is deployed, there is a need for those controllers to communicate together to exchange information with each other. As a result, controller-to-controller latency should be taken into consideration when optimising the placement of controllers. This is generally referred to as "inter-controller latency," and it should be minimized. In a similar way to equations (5.1) and (5.2), the two equations in (5.3) and (5.4) do the same thing, but in this case, they determine the controller-to-controller latency. In this type of objective, both the average and worst-case scenarios (maximum inter-controller latency) are taken into account when deciding where to place controllers to improve the overall performance of the network. This performance metric (objective functions) should be looked into as part of the controller placement optimization because it significantly affects the way controllers work together overall.

$$\gamma^{Lat\_max\_N2C}(CL) = \max_{cl_1, cl_2 \in CL} d_{cl_1,cl_2}, \tag{5.3}$$

$$\gamma^{Lat\_avg\_N2C}(CL) = \frac{1}{\binom{|CL|}{2}} \sum_{cl_1, cl_2 \in CL} d_{cl_1,cl_2}. \tag{5.4}$$

While latency-based objectives focus on shortening network paths for data transmission, controller load balancing must also be taken into account while optimising controller placement in SD-WAN architecture to improve the reliability of network operation. In a capacitated controller placement environment where network reliability is the key performance indicator, one controller should not be overloaded while others are not underloaded. The load across the controller should be distributed equally. In this study, the switches associated with the controllers represent the load on them, and the number of switches assigned to the controller should be evenly distributed. As a result, the load-balancing objective characteristics should also be minimised. This study has considered an imbalance metric in order to minimise the loads across the controllers since the remaining objective functions were subjected to minimization [112]. The number of switches assigned to the controller is represented in this study as m_{pl}, where pl is the placement and m_{pl} is the number of switches that are connected to pl. The equation that describe this objective is mathematically denoted in (5.5) and is defined as the difference between m_{pl} two controllers with the lowest and highest assigned nodes.

$$\gamma^{imbalance}(CL) = \max_{cl \in CL} m_{cl} - \min_{cl \in CL} m_{cl}. \tag{5.5}$$

In this study, in addition to latencies and load balancing, controller failure as an objective function is also taken into account. This is called "resilience." This performance metric is important to the service providers because reliability is one of the most important things in any business. In case a controller fails, there should always be a way to ensure fault tolerance. This study has considered the minimum switch-to-controller latency failure that may occur during the optimization of controllers in SD-WAN. Assuming that CL $= 2^P \setminus \{\emptyset\}$ includes all potential positions left over from controller failures of up to $(n-1)$ controllers, the minimum switch-to-controller latency under any failure situation is represented in (5.6).

$$\gamma^{avg\_N2C}(CL) = \frac{1}{|CL|} \sum_{cl \in CL} \left(\frac{1}{|\Phi|} \sum_{\phi \in \Phi} \left(\min_{cl \in CL} d_{\phi,cl} \right) \right). \tag{5.6}$$

The objective functions, with the help of the distance matrix (containing the shortest distance between two locations), are what are used to optimize the decision variables (controller locations) for optimal placement of controllers in SD-WAN. This study deals with the problem of optimal controller placement in SD-WAN when there are several competing objectives to simultaneously optimize. Optimizing several competing objectives simultaneously is a combinatorial optimization problem that is NP-hard. An example of this combinatorial optimization problem is the CPP investigated in this study. In the literature, metaheuristic algorithms like MOPSO, NSGA-II, and ANSGA-III have been suggested as ways to solve controller placement problems. All of these algorithms, with the exception of the ANSGA-III, were connected to the problem of scalability (when the number of objectives exceeded three). Yet, these solutions that have been developed are known to be computationally inefficient and often lack intelligence (for example, the ability to predict how many controllers are needed to set up an SD-WAN). In the next part of the study, the proposed solution to the problem of where to put the controller and how many controllers are needed for SD-WAN will be discussed. This solution equally resolves the aforementioned challenges associated with the existing solutions for the placement of controllers.

5.3 Proposed Stochastic Computational Graph with Ensemble Learning Model for SD-WAN Controller Placement

The aim of this thesis is to provide a collaborative and adaptive optimization learning-based framework for solving controller placement in SD-WAN. The proposed method in this chapter is aligned with the aim of the adaptive optimization learning-based framework mentioned in this thesis in section 1.6. This study came up with a new optimization algorithm and a smart way to place controllers in SD-WAN when there were several goals that were at odds with each other. The proposed solution is presented in this section and is referred to as a stochastic computational graph model with ensemble learning (SCGMEL). The proposed SCGMEL algorithm will solve the problem of where to place controllers in SD-WAN, and it will also solve the problems with the current solution to the problem of where to place controllers in SD-WAN that has been proposed in the literature. The controller placement problem is about figuring out how many and where to put controllers when different goals are at stake. Large enterprise organizations, like SD-WAN, are examples of where controller placement problems exist. This study came up with a unique and novel solution for the placement of controllers in SD-WAN. These solutions are a mix of a stochastic computational graph model (stochastic gradient descent with momentum, learning rate, decay rate, and a dynamic computational graph) and an extreme gradient boosting decision tree model called XGBoost. After figuring out where to place the controllers with the SCGMEL algorithm, XGBoost was used to predict a good number of controllers for deploying SD-WAN. In the same way, this study used an artificial neural network called Learning Vector Quantization to predict where controllers should be placed. In the next part of this study, there will be a brief discussion about the flowchart and algorithm, as well as the different solutions that make up the proposed solution.

5.3.0.1 Stochastic Gradient Descent (SGD)

SGD is a popular optimization algorithm that is frequently used in machine learning applications [189]. SGD and Mini-Batch SGD are the transformed versions of Gradient Descent (GD). With stochastic gradient descent, instead of using all of the observations to figure out the gradient, only a random subset of them is used. Given a dataset with n observations ($i > 10^3$), GD considers all n points for reducing the cost function, making this algorithm extremely computationally intensive. SGD, on the other hand, adjusts the parameters with a single data point ($i = 1$) at each epoch. Mini-batch SGD involves adjusting model parameters using a subset of available data (j data points ($j < i$)). These ways of picking observations with a number of observations less than n make the procedure computationally efficient. In SGD, the value of the cost function drops off suddenly, making it less likely to become trapped at the local minimum and increasing the likelihood that it will eventually break free. To optimise a loss function, SGD, a popular gradient descent technique, is often utilised. It improves generalisation performance compared to batch gradient by updating model parameters using mini-batch samples at a time, which in turn requires less computation. The most basic type of gradient descent, known as batch gradient descent, updates parameters using all available samples at each iteration, which is both computationally intensive and has subpar generalisation performance.

This research uses SGD as an optimization technique to find the best way to place controllers in an SD-WAN, taking into account latency (both switch-to-controller and inter-controller latency), load balancing, and resilience (controller failure). When many goals are at odds with each other, the recommended solution is to use the stochastic gradient descent (mini-batch) principle to figure out where to put controllers. In the following sections, we'll go through the SGD and several additional optimization approaches and

parameters (learning decay rate, momentum, and a dynamic computational network) that work together to boost the SD-WAN's overall performance.

5.3.0.2 Learning Rate

In gradient descent, the learning rate [190] is used to scale the size of parameter updates as the algorithm moves downward. The number that is selected for the learning rate can have an effect on two different aspects of the algorithm: 1) how quickly the model learns, and 2) whether or not the cost function is minimized. In every method of machine learning, there are two types of parameters: those that can be learned by the machine and those that cannot. Machine-learnable parameters are those that algorithms can learn or estimate on their own while they are being trained on a specific dataset. However, data scientists will assign values to hyper-parameters to control the learning process and the final accuracy of the model. This is done in order to improve the accuracy of the predictions made by the model. In most cases, the learning rate is denoted by the symbol α. Finding the appropriate learning rate for a given challenge is not a simple task. Because of this, the optimization algorithm proposed in this study uses a decaying learning rate that slows down over time. The effect of a decaying learning rate is that the learning rate changes (becomes "dynamic") over time instead of staying the same (being "static").

5.3.0.3 Stochastic Gradient Descent with Momentum

SGD with momentum [191] is an extension of the stochastic gradient descent that ensures that a portion of the previous update of the vectors is put in memory while calculating the next one. The SGD with momentum is designed to speed up the optimization process. The fact, that the SGD progression of the search may bounce around the solution space and getting the right learning rate with SGD requires careful selection, which may result in the possibility of getting stuck in the local minimum if too high or too low, integrating momentum with the SGD will help to avoid local minimum and allow the algorithm to attain global optimum as well as speed up the optimization process (function evaluation reduction before attaining optimal results) with a more desirable end result. This study in the optimization part of the proposed solution has incorporated history into the parameter update (momentum) with the history being determined by the gradient that was seen in earlier iterations. As part of the suggested approach, the momentum is put to use in order to mitigate the impact of the learning rate and give strength to training stability (integrating first-order historical gradient). It is important to keep track of the most recent change to the vector so that you can factor it into the subsequent calculation.

5.3.0.4 Learning Rate Decay

Learning rate decay [192] is a mechanism that aids in the acquisition of complicated patterns and, as such, helps the model to converge to the global optimum and avoid the oscillation seen in conventional SGD without momentum and learning rate decay. The suggested SGD optimization model includes an autonomous learning rate decay technique. Using this method, one may determine the significance of the preceding update's (momentum's) contribution to the optimization process. This method gradually lowers the training pace, which lessens the need for human intervention and the number of adjustable settings. Decreases in the learning rate have the dual benefit of suppressing oscillation and speeding up convergence. To optimize a loss function, SGD, a popular gradient descent technique, is often utilised. It improves generalization performance compared to batch gradient by updating model parameters using mini-batch samples at a time, which in turn requires less computation. The most basic type of gradient descent, known as batch gradient descent, updates parameters using all available samples at each iteration,

which is both computationally intensive and has subpar generalization performance. As time goes on, the learning rate slows down, so the optimization techniques proposed in this research (a stochastic computational graph model) get closer and closer to an optimal solution where the controllers are placed in the best way possible (the minimum point). This hyper-parameter helps with the convergence of the proposed stochastic computational graph model and prevents overfitting.

5.3.0.5 Computational Graph

Computational graphs [193] are a way to represent a mathematical function in graph theory. Graph theory is predicated on the central premise that all components of a graph may be classified as nodes or edges. The "nodes" in a computational graph may be either input values or functions that combine those values. The edges get their final weights throughout the data transmission process. If an edge is leaving an input node, its weight will be set to the value of the input, but if it is leaving a function node, its weight will be calculated by multiplying the weights of the incoming edges by the parameter of the function. Popular machine learning frameworks like PyTorch and Tensor Flow [194] rely on the construction of these computational graphs to carry out the back-propagation process for a given network and to calculate gradients. Backpropagation is utilized to estimate the "gradient" of an input weight, which may be defined as the change in a loss that results from a relatively insignificant shift in that weight. After then, the weight is revised with the aid of the gradient and a learning rate with other optimization arguments like momentum and learning decay rate to cut down on the overall loss as much as possible and give strength to the training stability and speed up the convergence process. The objective functions used in this study for the optimization of controllers are represented mathematically. As a result, this study has adopted the use of the computational graph approach in writing the objective function as graph theory for ease of implementation and to help in the optimal performance of the proposed stochastic computational graph model. The dynamic computational network is made up of tensors (variables) that are enabled for gradients as well as functions (operations). Because the flow of data and the operations that will be done on the data are defined during runtime, the dynamic creation of the computational graph will take place in real time and in an iterative fashion. At each cycle, many gradients are created, and a computation graph is built to record the resulting gradient functions. PyTorch accomplishes this goal by building a computational graph that can be interacted with. It is possible to compute gradients in a variety of ways using this method due to the fact that each iteration of this procedure starts with a clean slate. Forward and backward computations are two distinct types of calculations that may be performed by utilizing the computational graphs in various ways.

The following terminologies provide definitions for a few of the most important terms in the field of computational graphs.

- A node in a network represents a variable. This variable might be a scalar, vector, matrix, tensor, or perhaps an entirely different kind of variable altogether.
- A function parameter and a data dependency are both represented as an edge in a graph. These are comparable to the node pointers that are used.
- A basic function that makes use of one or more variables is called an operation. In this context, only specific operations are permitted. Several operations can be combined to express more complicated functions than are included in this set.

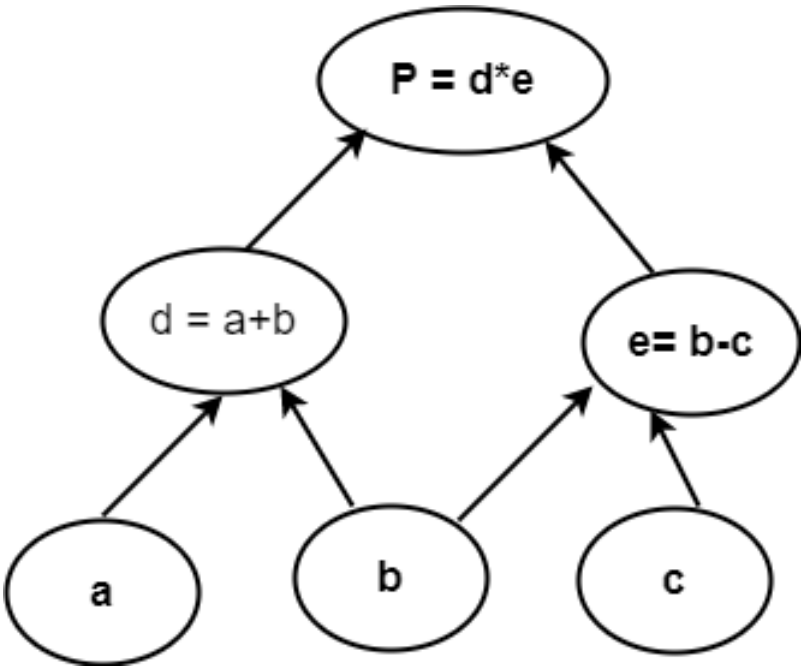

Figure 5.1: Build up of Computational Graph

Consider the following: **P**= $(a + b) * (b - c)$ as illustrated in 5.3.0.6
In order to facilitate a deeper level of comprehension, authors define two variables, d and e, as output variables for each operation. The equation now becomes: $d = a + b, e = b - c, \mathbf{P} = d * e$. It is possible to do arithmetic operations such as addition, subtraction, and multiplication. Nodes are the fundamental building pieces of a computational graph, each of which performs a unique operation and receives its own set of parameters[195]. The array's direction determines the flow of input data to other nodes in the network. The final output value can be calculated by first establishing the values of the input variables and then calculating the nodes of the graph in accordance with those values.

5.3.0.6 Computational Graphs Types

All deep learning mechanisms rely on the building of computation graphs in order to compute the gradient values necessary for gradient descent optimization [196]. The framework often handles backward differentiation so all you have to do is construct the forward propagation graph. This section will briefly discuss static and dynamic computational graphs.
Static graphs provide a number of benefits, one of which is that they make it possible to do sophisticated offline optimization and scheduling of graphs. This suggests that they would typically be quicker than dynamic graphs in general; nevertheless, the difference may not be noticeable in all use cases. The drawback is that managing structured data, including data with changeable sizes, is unsightly [197] [198]. The static graph is known to generally have the following two phases. In the first phase, architecture is defined (perhaps with fundamental flow control such as loops and conditionals) while in the second phase, one will either train the model or create predictions using a significant quantity of data.
Dynamic computational graphs: Computational graphs that are dynamic are those in which the graph is defined implicitly (by means such as operator overloading) as the forward computation is carried out [199]. The advantage of dynamic graphs is that they are more flexible. The library is less invasive and permits parallel graph construction and evaluation.

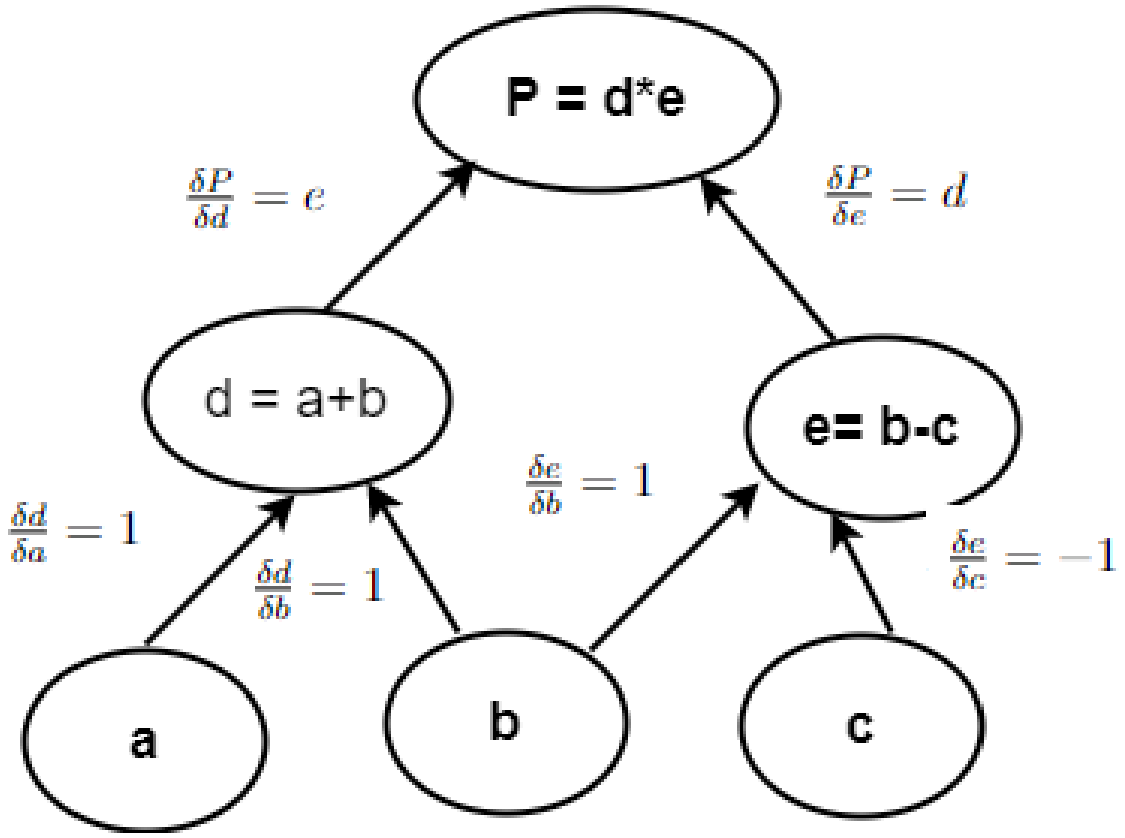

Figure 5.2: Illustration of Computational Graph

It is necessary to make use of the chain rule in order to do the evaluation of the partial derivatives of the final output variable with regard to the input variables $\alpha, \beta, and\ \gamma$. As a direct consequence of this, the derivatives may be written as follows:

$$\frac{\delta \mathbf{P}}{\delta \alpha} = \frac{\delta \mathbf{P}}{\delta d} * \frac{\delta d}{\delta a} = e * 1 = e$$

$$\frac{\delta \mathbf{P}}{\delta \beta} = \frac{\delta \mathbf{P}}{\delta d} * \frac{\delta d}{\delta b} = e * 1 = e$$

$$\frac{\delta \mathbf{P}}{\delta \gamma} = \frac{\delta \mathbf{P}}{\delta e} * \frac{\delta e}{\delta c} = d * -1 = -d$$

This provides us with an illustration of how computational graphs simplify the process of obtaining derivatives through the use of back-propagation.

5.3.0.7 Extreme Gradient Boosting (XGBoost)

XGBoost, or Extreme Gradient Boosting, is a renowned and potent machine learning algorithm extensively employed for constructing supervised models, whether for regression or classification tasks [200]. This algorithm is celebrated for its exceptional efficiency, adaptability, and versatility. It has been meticulously optimized to operate across various data formats, making it a robust choice for a wide array of machine-learning applications. One of the key strengths of XGBoost is its ability to perform parallel computation, which contributes significantly to its efficiency, accuracy, and practicality. At its core, XGBoost employs an ensemble technique known as "gradient boosting." This technique combines the predictions of

a set of simpler models, each with limited predictive power, to generate a highly accurate final prediction. XGBoost is renowned for its computational prowess and efficiency, surpassing many other algorithms, particularly in comparison to single decision trees. XGBoost is particularly resilient to outliers and leverages a collection of decision trees in its analysis. It incorporates both a tree learning algorithm and a linear model solver. Its selection as an ensemble learning method in this study is underpinned by its well-documented ability to elevate predictive accuracy and overall model performance. Unlike traditional ensemble methods that involve combining independently trained models, XGBoost operates through the collaborative synergy of multiple decision trees within the algorithm itself [201]. The result is an ensemble model that excels in a variety of tasks, including regression and classification. This study capitalizes on XGBoost's numerous advantages by proposing its use as a scalable tree-boosting model to predict the optimal number of controllers required for SD-WAN deployment. Data scientists commonly rely on XGBoost to achieve cutting-edge results in machine learning challenges. In essence, XGBoost is a versatile and robust machine learning tool suitable for a wide range of applications, including predictive modeling for SD-WAN controller placement.

5.3.0.8 Normal Distribution.

In statistics, the mean of a normal distribution is a measure of its central tendency [202]. The normal distribution is an incessant probability distribution in statistics that is used a lot in statistical analysis and modeling. The mean of a normal distribution is also known as its expected value or average value. It is denoted by the symbol μ and represents the point at which the distribution is symmetrically centered. The mean, or the most likely value, for a normal distribution is the same as the point where the distribution is most normally distributed. A normal distribution with an average of zero and a sigma of one has its peak near the mean. The mean value is the position on the horizontal axis where the curve is centered. The mean is a key parameter of the normal distribution and is used to calculate many other statistical measures, such as variance and standard deviation. It is also used in hypothesis testing when the sample mean differs from the null hypothesis in a statistically meaningful way [203].

5.3.0.9 Variance homogeneity or Levene's test

The Levene's or variance homogeneity test is a test in statistics that is used to detect whether or not the variances of two or more sets of data are equal to one another [204]. The assumption that the variances of the groups being compared have the same value is made by many statistical tests, including the t-test and the ANOVA; hence, this test is quite significant. Levene's test compares the absolute deviations of the data from their respective group means and tests whether these deviations are equal across all groups. The test can be conducted using different methods, such as the Brown-Forsythe test or Bartlett's test, depending on the underlying assumptions of the data. In the case of Levene's test, the assumption that there is equality in the variances across multiple groups of data is inferred to be the null hypothesis, while the assumption that at least one of the datasets among the group differs significantly can be regarded as the alternative hypothesis. In statistical analysis, the null hypothesis is ruled out and unequal variances are inferred if the probability value of the test's result is lower than the level of significance which is normally set at 0.05 threshold [205]. It is important to test for homogeneity of variances before conducting any statistical analysis that assumes equal variances, as violating this assumption can lead to biased results and incorrect conclusions.

5.3.0.10 Non-Parametric test of Kruskal-Wallis

The Kruskal-Wallis statistical analysis method is applied to the process of comparing the medians of relevant data sets in order to evaluate whether a statistically notable distinction exists between multiple groups of data [206]. It is a non-parametric test, which means it does not require the assumption of normality or equal variances in the data, making it suitable for analyzing non-normally distributed data. The Kruskal-Wallis test ranks the observations within each group and calculates a test statistic based on the sum of ranks [207]. The medians of all groups are assumed to be equal in the null hypothesis of the test and to be different from one another in at least one group in the alternative hypothesis. The null hypothesis is rejected and a statistically significant difference in medians between at least two groups is inferred if the probability (p-value) of the test's result is lower than the selected level of significance which is normally at 0.05 threshold. If the null hypothesis is found to be false, then post hoc tests, such as the Pairwise Wilcoxon rank-sum test or the Conover-Iman test, can be used to discover whether groups are substantially different from one another. The Kruskal-Wallis test is commonly used in fields such as biology, sociology, and economics when the assumptions of parametric tests, such as the one-way ANOVA, are not met.

5.3.0.11 Wilcoxon rank-sum pairwise method (Post Hoc-Test)

Wilcoxon rank-sum pairwise method is a non-parametric test in statistics that is used to compare the medians of two independent data sets to see if they differ statistically substantially [208]. The test entails ranking the observations from both groups and computing the total of each group's ranks. The test statistic is then computed as the smaller of the two sums of ranks, and a p-value is computed based on the test statistic's distribution under the null hypothesis of no difference between the groups [209]. If the probability value is lower than a predetermined level of significance, which is normally not more than 0.05, then the null hypothesis is ruled out, and it is shown that the medians of the two groups differ statistically significantly. The pairwise Wilcoxon rank-sum test is often used in situations where the assumptions of parametric tests, such as the t-test, are not met, such as when the data are non-normally distributed or have unequal variances. It is a robust test, meaning that it is not susceptible to the presence of outliers or non-normality in the data. When conducting multiple pairwise comparisons between groups, the significance level should be adjusted using a method such as the Bonferroni correction in order to keep the family-wise error rate under control.

5.3.0.12 Learning Vector Quantization

Learning Vector Quantisation is a production version of the supervised classification technique [210][211]. The terms "production version" and "prototype-based" are used interchangeably. A prototype is a test version of a product, service, or system that is developed to ensure the viability of an idea. Each class in the dataset is represented by one or more prototypes, and there may be more than one prototype per class. The class of the prototype that is geographically closest to new data points that are initially unknown is then allocated to those new data points. Defining a distance metric is necessary for the word "closest" to make any sense. There is no restriction on the number of prototypes that may be used per class; the only condition is that each class must have at least one prototype.

LVQ is a specific example of an artificial neural network that employs a winner-take-all Hebbian-based learning strategy [212]. It is quite similar to the self-Organising Maps (SOM) method, with a minor change. The LVQ system is represented by its prototypes, which are denoted by the notation

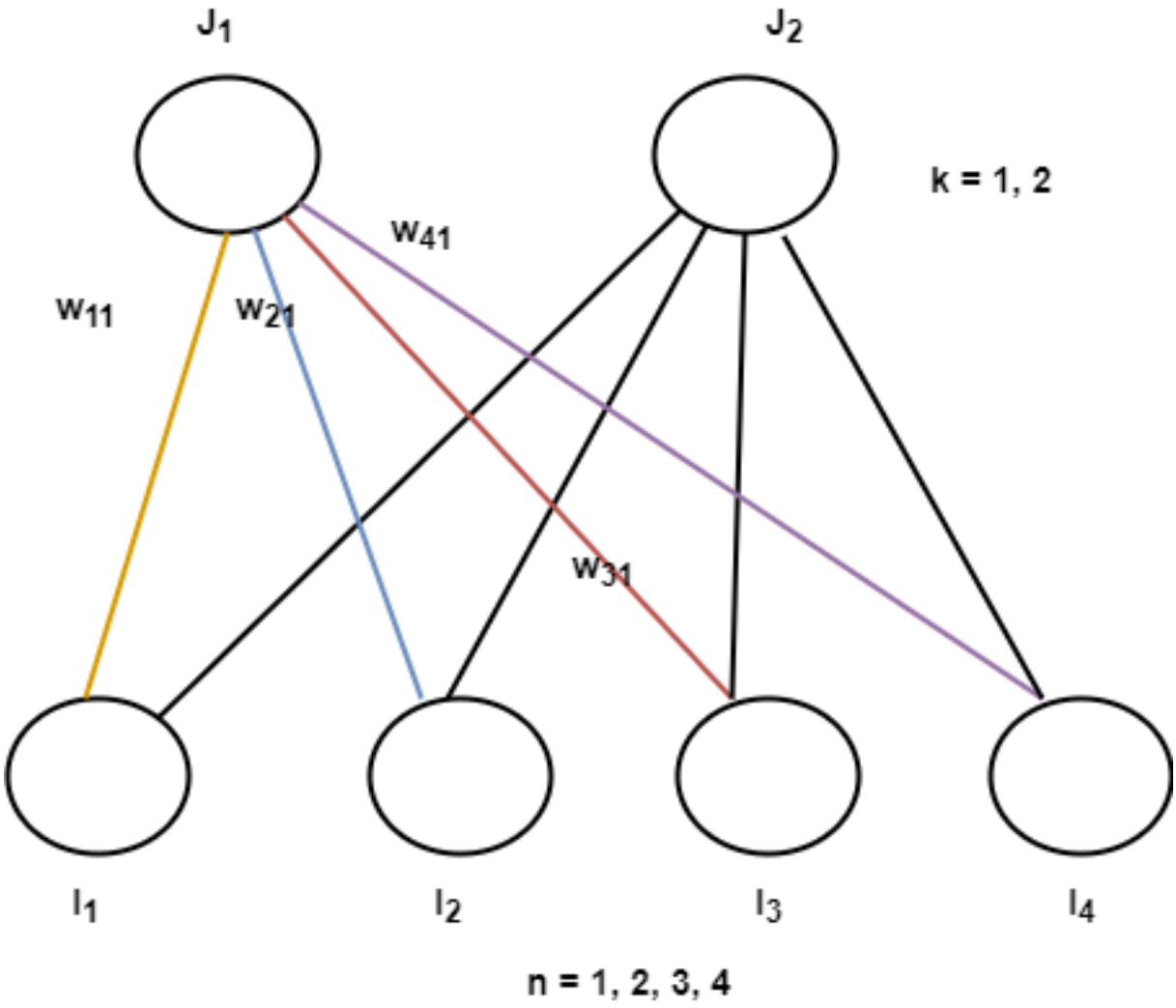

Figure 5.3: Conceptual Representation of LVQ networks

$W = (w_i, \ldots, w_n)$. In training algorithms that use a winner-take-all approach, the winner is either brought closer to the target if it properly classifies the data point or further away if it erroneously classifies the data point. One of the benefits of LVQ is that it generates prototypes that are simple and straightforward for specialists in the relevant application domain to understand. The LVQ networks are depicted in Figure 5.3, where the letters I and J stand for the input cluster and output cluster, respectively. The weights, denoted by W, are used to indicate how strong the connection between I and J. In the next part, we will examine an example of this method as well as the essential stages in further depth.

5.3.0.13 Example showing how LVQ works

The objective of this example is to illustrate the LVQ notion by providing a network of LVQs that contains five vectors that have each been assigned to one of two classes.

- Step 1: Based on the information presented in Figure 5.4 in the previous paragraph, each vector has

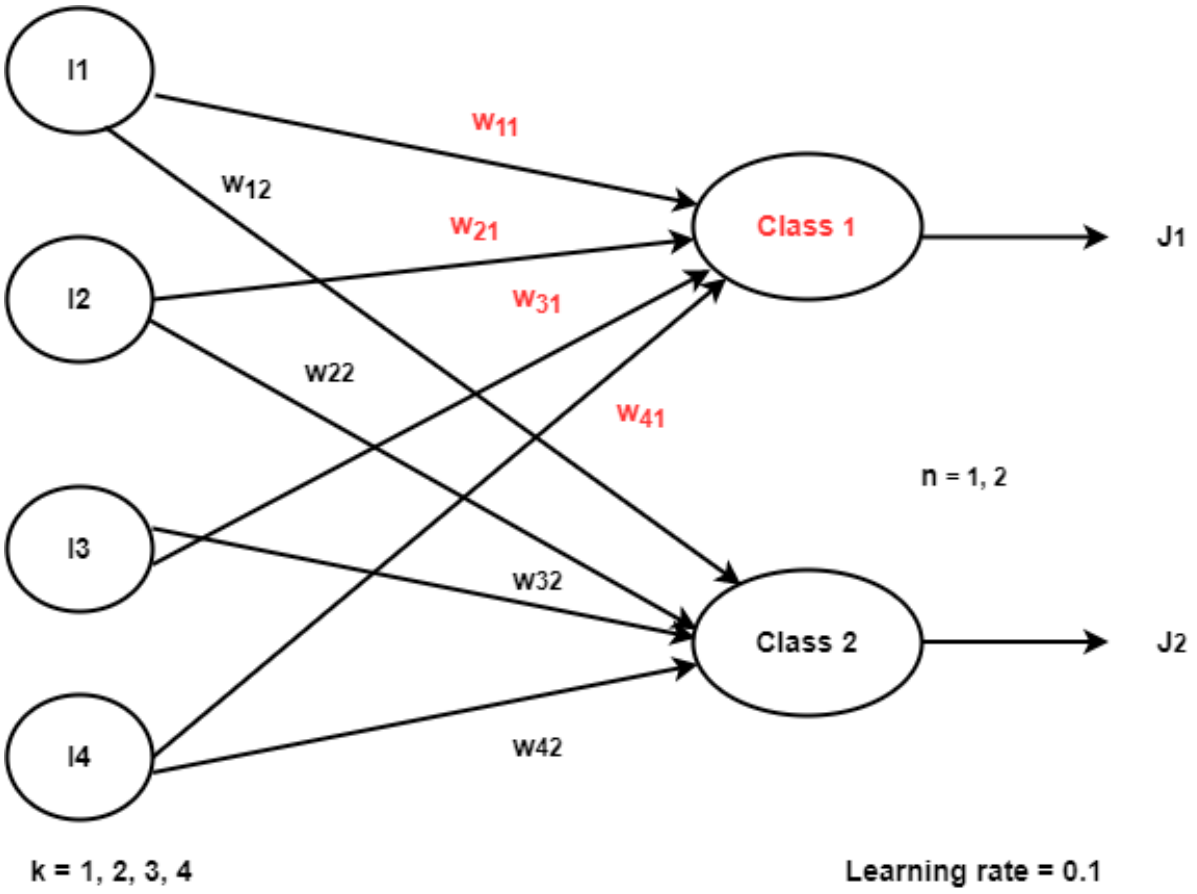

Figure 5.4: LVQ networks

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 2 \\ 1 \\ 1 \end{matrix}$$

four inputs designated as I1, I2, I3, and I4, and the number of output classes is two. Take the first two vectors into consideration as the starting weight vectors (codebook vectors, or the prototype), then use the remaining three vectors to create the input vector. The first two are taken due to the number of classes that exists in the network. Hence, $w_1 = [0011], w_2 = [1000]$. There are four inputs in each vector ($k = 1, 2, 3, 4$), and there are two classes (n=1,2) in total.

Hence, the initial weight matrix, W_{kn} = Therefore, the weight matrix, or the codebook vectors depending on the circumstance, consists of 4 rows and 2 columns. In the matrix W_{kn} =, the value denoted by column 1 is the w_1, and the value denoted by column 2 is the w_2.

- Step 2: The last three vectors out of a total of five vectors are now being taken into consideration at this level. During this part of the process, it is necessary to take into consideration the first input vector, which is the third vector in the starting vector $I_k = (i1, i2, i3, i4) = (0001)$, while the matching target cluster is represented by the number 2. In addition, using this formula, the

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Euclidean Distance between clusters n=1,2 and the first input vector will be determined.

$$dis_n = \sum_{k\in 1}^{n} (w_{kn} - i_k)^2. \tag{5.7}$$

Compute the distance between cluster n=1 (first column of the matrix) and first input vector, $dis_n = dis_1$ using

$$dis_1 = \sum_{k\in 1}^{n} (w_{k1} - i_k)^2. \tag{5.8}$$

$dis_1 = (0-0)^2 + (0-0)^2 + (1-0)^2 + (1-1)^2 = 1$, Also, repeat this procedure with the second codebook vector, which has the cluster number 2; the first input vector has the value $dis_n = dis_2$

$$dis_2 = \sum_{k\in 1}^{n} (w_{k2} - i_k)^2. \tag{5.9}$$

$dis_2 = (1-0)^2 + (0-0)^2 + (0-0)^2 + (0-1)^2 = 2$. After taking into account the minimal values, the winning cluster $n = 1$ is clear from the calculation of the Euclidean distance between the codebook vectors and the first input vectors in the remaining vectors given, here, $dis1dis2$. Therefore, only column $n = 1$ of the aforementioned matrix 5.4 will be updated to reflect the latest weights (codebook vectors or prototype). It is observed that $n = 1$, and the target cluster is 2. So the target cluster is not equal to the winning cluster. The below equation is used to update the weight when the winning cluster $\neq$ *target cluster*.$\text{Wei}_{kn}(new) = wei_{kn}(old) - \beta[i_k - wei_{kn}(old)]$. The β here represents the learning rate which is set to 0.1. i_k, represent the input vector, $n = 1$, the first column of the weighted matrix. $Wei_{11}(new) = 0 - 0.1(0-0) = 0$, $Wei_{21}(new) = 0 - 0.1(0-0) = 0$, $Wei_{31}(new) = 1 - 0.1(0-1) = 1.1$, and $Wei_{41}(new) = 1 - 0.1(1-1) = 1$. The new matrix after the update becomes

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

- Step 3: In this phase, the second input vector is considered such that $I_k = (i1, i2, i3, i4) = (1100)$ and the corresponding target cluster = 1.The same procedure in steps 1 and 2 are followed, but in this case with the new updated weight. Euclidean distance will be computed following the procedure in the previous step 1 and 2. When n =1 $dis_1 = (0-1)^2 + (0-1)^2 + (1.1-0)^2 + (1-0)^2 = 4.21$, and when $n = 2$, $dis_2 = (1-1)^2 + (0-1)^2 + (0-0)^2 + (0-0)^2 = 1$. In this case, $dis_2 << dis_1$, which implies that the winning cluster $n = 2$ considering the minimum value. Hence, the update will be performed on column $n = 2$ of the above 5.4. Since the winning cluster

$\neq target\ cluster, this equation \text{Wei}_{kn}(new) = wei_{kn}(old) - \beta[i_k - wei_{kn}(old)]$ will be used to update with the same learning rate $\beta = 0.1$. The weighted matrix will then be computed as $Wei_{12}(new) = 1 - 0.1(1-1) = 1$, $Wei_{22}(new) = 0 - 0.1(1-0) = -0.1$, $Wei_{32}(new) = 0 - 0.1(0-0) = 0$, as well as $Wei_{42}(new) = 0 - 0.1(0-0) = 0$. The new matrix after the update becomes

$$\begin{bmatrix} 0 & 1 \\ 0 & -0.1 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

- Step 4: In this phase, the third input vector is considered such that $I_k = (i1, i2, i3, i4) = (0110)$ and the corresponding target cluster = 1.The same procedure in step 1 and 2 are followed, but in this case with the new updated weight. Euclidean distance will be computed following the procedure in previous steps 1 and 2. After the computation,when n =1 $dis_1 = (0-0)^2+(0-1)^2+(1.1-1)^2+(1-0)^2 = 2.01$, and when n= 2, $dis_2 = (1-0)^2 + (-0.1-1)^2 + (0-1)^2 + (0-0)^2 = 3.21$. In this case, $dis_1 << dis_2$, which implies that the winning cluster n = 1 considering the minimum value.Since the winning cluster = target cluster, this equation $Wei_{kn}(new) = wei_{kn}(old) + \beta[i_k - wei_{kn}(old)]$ will be used to update with the same learning rate $\beta = 0.1$.The weighted matrix will then be computed as $Wei_{11}(new) = 0 + 0.1(0-0) = 0$, $Wei_{21}(new) = 0 + 0.1(1-0) = 0.1$, $Wei_{31}(new) = 1.1 + 0.1(1-1.1) = 1.09$, and $Wei_{41}(new) = 1 + 0.1(0-1) = 0.9$. The new matrix after the update becomes

$$\begin{bmatrix} 0 & 1 \\ 0.1 & -0.1 \\ 1.09 & 0 \\ 0.9 & 0 \end{bmatrix}$$

  By using a winner-take-all concept and input vector clustering, Learning Vector Quantization automatically updates the matrix weights.

This study has proposed the learning vector quantization classification algorithm to predict the placement of the controller in the SD-WAN deployment. The next section will now present the flowcharts and the algorithms of the proposed stochastic computational graph model with an ensemble learning approach, as well as the classification algorithm used in the prediction of controller placement.

5.4 The Proposed Stochastic Computational Graph Model with Ensemble Learning Approach, as well as the LVQ Flowcharts and Algorithms for SD-WAN Controller Placement

In this section, the flowcharts and the algorithms of the proposed stochastic computational graph model with an ensemble learning approach, as well as the classification algorithm used in the placement of controllers, will be discussed. The flowcharts will be presented first, followed by the algorithm, and then their discussion will follow.

In Figure 5.5, the following inputs are loaded: the datasets (coordinates-based), which comprise the latitude and longitude of different cities for possible controller placement locations; the learning rate α, which is used to control the movement of the model down the hill; the momentum, which is used to stabilize the training and help speed up the convergence process; and the learning rate decay, which helps to dynamically decrease the learning rate as well as help in suppressing oscillation and speeding up the convergence process. The number of epochs, which is the number of iterations the model takes in optimizing the parameters; the data loader, which helps to load the tensors batch by batch during the training; and early stopping (callback operation), which is a monitoring technique to avoid overfitting of the model and ensure the model terminates when there is no longer improvement in the loss. The second step in the flowchart ensures that the datasets are converted to a tensor format for easy readability and operation with the dynamic computational graph. Then, the haversine distance function is employed to calculate the distance between two geographical cities. The third box of the flowcharts defined various objective functions used with the decision variables to optimize controller placement. These objective functions include switch-to-controller latency on the average and worst-case scenarios, inter-controller latency on the average and worst-case scenarios, load balancing, which was converted to load imbalance for easy computation, and resilience (controller failure). The fourth box uses PyTorch tensors to generate the initial controller location randomly from the datasets. The optimal number of controllers here is seven. The fifth box is then used to clone the initial controller location to help in the comparison of the initial loss and the final loss when the model finishes its optimization. Meanwhile, little noise of 0.0001 is added to the tensors to enhance the gradient computations. Similarly, in the sixth box of the flowchart, the dynamic computational graph is used to track the operation of the tensors, and then containers to store all losses such as the train loss, validation loss, and overall loss are created. In the next box, the container for best loss was first used to store the initial losses, which were subjected to updates during the model iteration. Similarly, the final location container was used to store the final location losses. The iteration begins in the next box, and the dataloader is used to load the tensors batch by batch. This approach is known as "mini-batch stochastic gradient descent." In the next box, all losses were computed, and back-propagation with the help of PyTorch was used in the automatic differentiation and calculation of gradients. In the tenth box, the controller location was updated using various parameters, such as the learning rate, the decay rate, and the momentum (see Figure 5.5). In the next box, the "loss function" is defined, where all the losses are added into a single loss value with the appropriate weight to show the importance and strength of each objective function. In the twelfth box, losses were recorded between the tensors and the rest of the position during the completion of the train and the validation cycle. This is required because mini-batch SGD is considered and the dataloader was used to load the tensors batch by batch. The train losses contain losses stored by batch, while the validation losses contain losses at the complete epoch. The next box will sum up all the value losses and store them in the overall losses container. Meanwhile, the next box employs the use of a callback operation and tracks the losses of the model, such that when the current loss is less than the minimum of the previous losses, the value is recorded, which implies

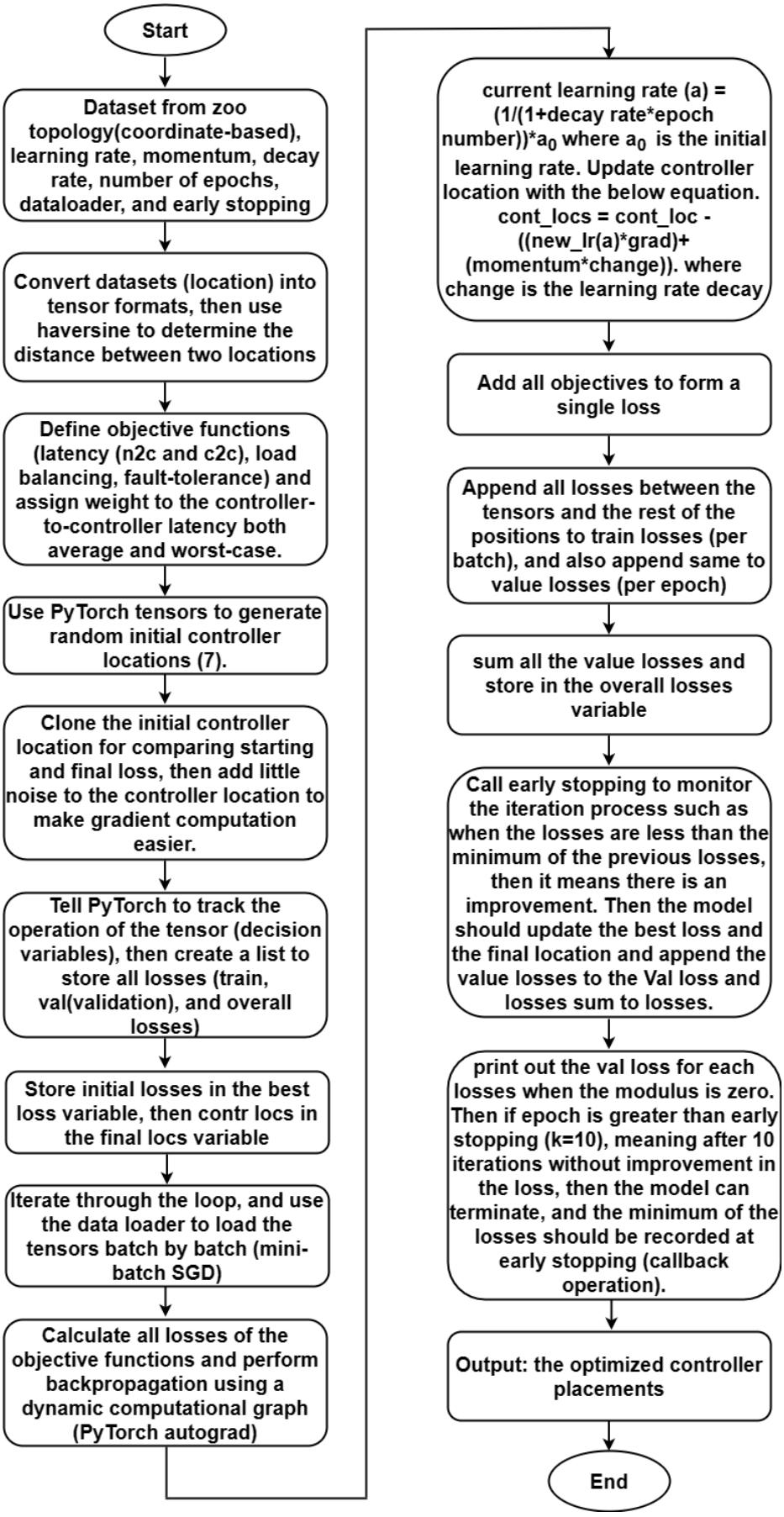

Figure 5.5: The flowchart of the proposed stochastic computational graph model

that there is an improvement after the iteration. The model then updates the best locs and the final locs defined in the seventh box, and also, updates the val losses and losses defined in the previous box. However, in the final box, if no improvement occurs after ten iterations, the model will terminate and record the losses recorded at this early stopping, which will now be the optimized controller placement. This means that the model has converged and reached the global minimum.

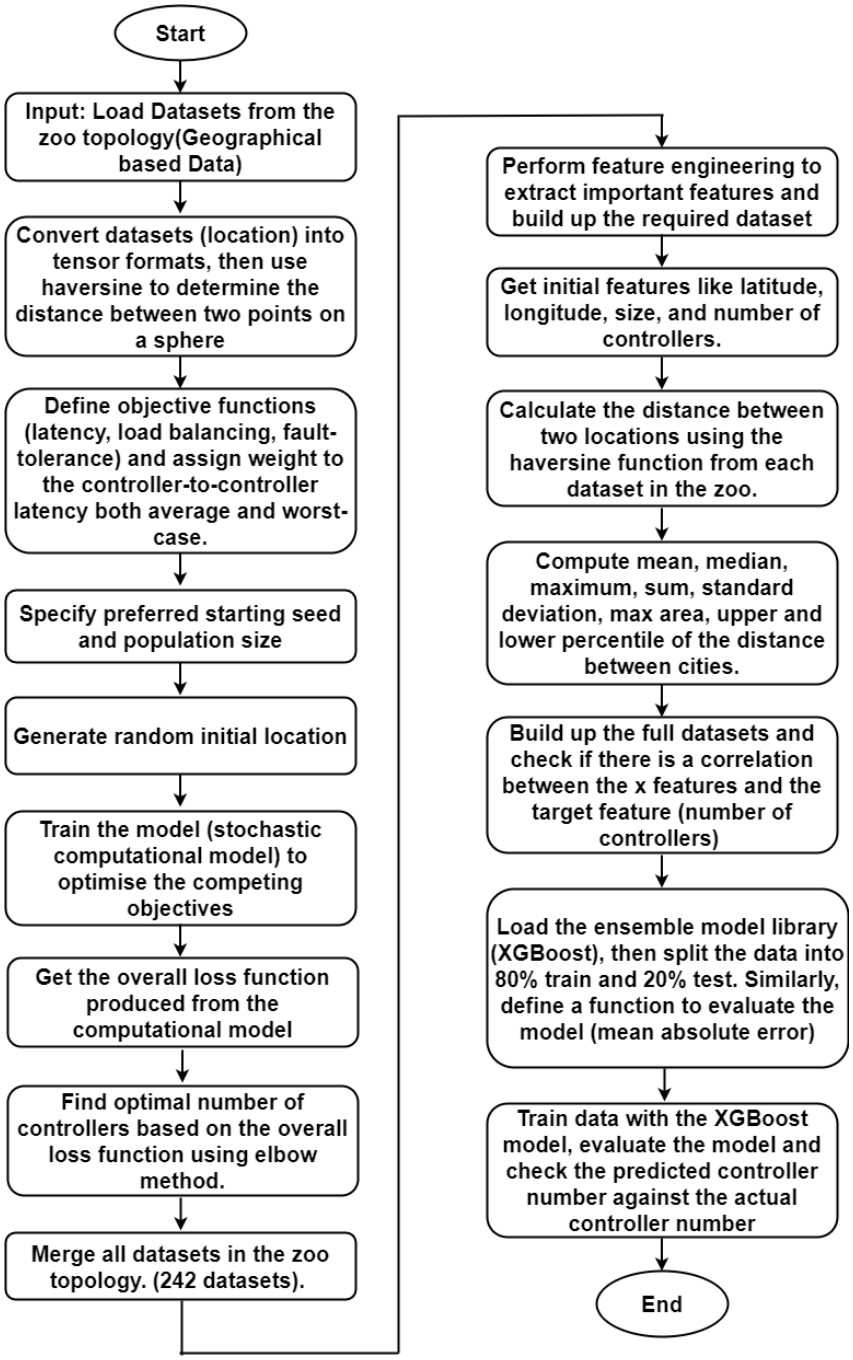

Figure 5.6: The flowchart of the proposed ensemble learning model (XGBoost)

This study now presents the next flowchart which describes the flowchart of the ensemble learning model employed in this research to determine the optimal number of controllers that is needed in the optimal placement of controllers in SD-WAN. The proposed ensemble learning model and its discussion

are presented in Figure 5.6. The optimal number of controllers has achieved with the use of the proposed stochastic computational model and elbow method techniques, which were used to identify the right number of controllers (using the overall loss estimated); thereafter, the initial controller location and the final controller location generated by the use of the model with feature engineering were used in the prediction of the suitable number of controllers, which was achieved with the help of XGBoost.

In Figure 5.6, the first approach that was carried out is the same as the one flowchart in 5.5. The proposed stochastic computational graph model was first used to obtain the placement of controllers. This is done because the overall loss obtained from this approach was used to calculate the optimal number of controllers needed in the SD-WAN deployment. Note that the entire dataset from the zoo topology [184] was subjected to evaluation in this study to optimise the number of controllers needed for each of the datasets represented in the zoo topology.

The first box in 5.6 describe the input which is geographical datasets obtained from zoo topology. This dataset comprises of 242 different service providers' WAN datasets. The zoo topology is a directory where live WAN datasets are kept for research purposes. The second box converts this datasets into tensors format then haversine distance formula is used to calculate distance between locations. In the third box, objective functions that were used to compute placement were defined, while the population size and a seed value were specified for reproducibility in box four. Similarly, box five generates a random initial controller location. The proposed stochastic computational graph was used in Box 6 to optimise the decision variables regarding the objective functions. The overall loss function, which was obtained from the overall loss calculated by the model in the previous box, was used in box seven to estimate the number of controllers. Further explanation on this will be given after the presentation of the results. In the box, the optimal number of controllers was achieved with the help of the elbow method. The next boxes begin the process of controller number prediction with the help of the XGBoost model. In the ninth box of the flowcharts, datasets from the zoo topology were merged together. Feature engineering was extracted from Box 10 to build up more necessary features for the prediction. In box eleven, input, the final location of controllers, the size of the number of datasets, and the number of controllers for each dataset in the topology are obtained. Box twelve calculates the distance between the locations in the zoo topology datasets. Box 13 describes those features that were extracted from the zoo topology datasets. Such features as the mean, median, sum, standard deviation, maximum area, and upper and lower percentiles of the distance between locations were computed in this box. The next box checks for correlation between the X features and the y features (number of controllers). This is done to see if the model will be able to perform well or not. Box fifteen loads the necessary libraries for the XGBoost model, the dataset was split-ted into training and testing with 80% and 20% respectively. Similarly, the mean absolute error, which is a performance metric, as defined in this box to help evaluate the performance of the model. The last box describes the training process with the XGBoost model and how the required number of controllers was obtained.

Following the given description of the proposed stochastic computational model with ensemble learning approach flowcharts, the algorithms of the proposed solution and the learning vector quantization used in the placement of controllers will now be described. The algorithm 11 below is the proposed optimization algorithm for the optimization of several conflicting objectives in the placement of SD-WAN controllers using the Stochastic Computational Graph model with an ensemble learning approach. This proposed optimization algorithm is briefly discussed below.

Line 1 consists of the input to the model which are the required dataset (coordinate-based), data-loader (which is used for loading tensors in batches), user control parameters learning rate, decay rate, momentum, number of epochs, and early stopping. The necessary libraries (PyTorch, Numpy, and Pandas, to mention but a few) that help in the computation of the algorithm were loaded in Line 2. The datasets in longitude and latitude were loaded in Line 3, while Line 4 converted the coordinates-based datasets to tensor format. In Line 5, the haversine distance function was used in the computation of the distance between two locations. Line 6 defines the objective functions using a computational graph approach. In Line 7, generate random initial controller locations. While in line 8, the initial controller location is being cloned so as to be able to compare the initial loss and the final loss after the completion of the training process. A little noise is added to the controller location to make the computation of gradients easier in Line 9. Line 10 tracks the operations that each tensor performs (the seven controller positions in this case). In line 11, lists were created as a placeholder that holds the train losses, val losses, and the losses sum values. Line 12 defines val losses as the losses after all epochs are done. Similarly, lines 13 and 14 serve as placeholders for the best loss and final location, respectively. Note that initial losses are the loss of starting losses, and as training progresses, the losses with the lowest value will be kept in the placeholder known as "best loss." Line 15 iterates through the number of epochs, and in Line 16, the algorithm runs through the batches with the help of the data loader, which is used to load the tensors. In Lines 18 and 19, the algorithm takes into account each tensor in the controller location (Line 7) and computes all of the objective functions with a computational graph, and performs automatic differentiation using PyTorch autograd. Lines 20 and 21 update the controller location by taking the difference in the new learning rate multiplied by the gradient plus the momentum multiplied by the decay rate from the controller location. In Line 23, all objectives are added together to form a single loss. In Line 24, append the losses of the position tensors (final controller location) and the controller location (initial controller location) to the train losses (losses at each batch). Similarly, all losses of the position sensor and controller location to val losses (after each epoch). Line 26 sums the validation losses together and stores them in losses. If the epoch is greater than zero, it checks if the losses are less than the sum of all previous losses, and then it updates the final location and the best loss in Lines 27–34. In line 35, a callback rate was employed to avoid over-fitting. Meanwhile, the printout messages were set to every 4 epochs, such that the outcome of the iteration will only be printed out when the remainder of the epoch and the predetermined number of epochs that were set are equal to zero. Line 36 returns val loss for each loss, whereas line 37 will only execute after 10 epochs have been executed to see if there is an improvement or not since the early stop was set to ten.

Algorithm 11 *Proposed Optimization Algorithm for SD−WAN Controller Placement using Stochastic Computational Graph model with ensemble learning approach*

1: **Require**: $Dataset,\ Learning\ rate(\emptyset),\ n_{epochs},\ dataloader(dl),\ momentum,\ decay-rate,\ early\ stopping$
2: *Load necessary libraries*
3: *Load datasets* $(coordinate-based)$
4: *Convert datasets into tensor formats*
5: $Define\ haversine\ distance\ function = 2(r)arcsin\left(\sqrt{sin^2\left(\frac{\beta_1-\beta_2}{2}\right)+cos(\beta_1)cos(\beta_2)sin^2\left(\frac{\alpha_1-\alpha_2}{2}\right)}\right)$
6: $Define\ objective\ functions\ using\ the$ *computational graph approach*
7: $Generate\ random\ initial$ *controller location*
8: *Clone the initial controller location for later use*
9: *Add little noise to controller location to enhance gradients computation*
10: *Track the operation of the decision variables*
11: *Create list to store* $train_{losses},\ val_{losses},\ losses_{sum}$
12: $val_{losses},$ = *calculate loss after all epoch is done*
13: $best_{loss} = initial_{losses}$
14: $final_{locs} = cont_{locs}$
15: **for** *epochs in* $range(n_{epochs})$ **do**
16: *Run through batches using dl*
17: **for** pos_{tensor} *in dl* **do**
18: *Calculate* $loss2,\ loss,\ loss.backward$ *with the help of dynamic computational graph*
19: $loss2 = first\ + second\ objective,\ loss = all\ objective,\ and\ loss.backward$
20: *new learning rate* $(\alpha)=\frac{1}{1+decay\ rate*epoch*initial\ learning\ rate(\alpha_0)}$
21: $cont_{locs}.data-=cont_{locs}-(\alpha*grad)+(momentum*decay\ rate)$
22: **end for**
23: *Add all objective functions to form a single loss*
24: $append\left(all_{losses}(pos_{tensors}, cont_{locs})\right) to\ train_{losses}$
25: $set\ all_{losses}\big(pos_{tensors}, cont_{locs})\big) to\ val_{losses}$
26: *sum* val_{losses} *and store in losses*
27: **if** $epoch > 0$ **then**
28: **if** $losses\ < min(losses_{sum}$ **then**
29: $final_{locs} = cont_{locs}.clone()$
30: $best_{loss} = val_{loss}$
31: *append* val_{loss} *to* val_{losses}
32: *append losses to* $losses_{sum}$
33: **end if**
34: **end if**
35: **if** $epoch\ \bmod verbose==0$ **then**
36: *return each* $val_lossfor$ *each losses*
37: **if** $epoch > early-stopping$ **then**
38: **if** $min(losses_{sum}[-early-stopping:]) > min(losses_{sum}$ **then**
39: **end if**
40: **end if**
41: **end if**
42: *return best score recorded at* $early-stopping$
43: **end for**
44: **Expected** : *optimized controller placement positions*

In lines 38–41, after waiting for 10 epochs to run, the algorithm will check if the minimum in the last ten losses is greater than the current loss. If this happens, it means that there is no improvement in the

loss, so it should break the process. Then the best score should be returned at an early stop in line 42, and finally, line 43 returns the optimized controller placement positions.

Following the discussion above, this study will now present the proposed ensemble learning algorithm (see 12). The same explanation given in 5.6 also explains Algorithm 12. This study will now present the learning vector quantization, an artificial neural network used for the prediction of controller placement (see 13).

Algorithm 12 *Proposed Ensemble Learning Algorithm to Predict Optimal Controller Number in SD − WAN Controller Placement*

1: **Require**: *Required coordinates based Datasets*
2: *Convert datasets into tensor formats*
3: *Define haversine distance function* $= 2(r)arcsin\left(\sqrt{sin^2\left(\frac{\beta_1-\beta_2}{2}\right)+cos(\beta_1)cos(\beta_2)sin^2\left(\frac{\alpha_1-\alpha_2}{2}\right)}\right)$
4: *Define objective functions using the computational graph approach*
5: *Specify seed and population size*
6: *Generate random initial controller location*
7: *Use Stochastic computational graph model to optimize decision variables with the given objectives*
8: *Obtain overall loss values from the model in previous step*
9: *Find optimal number of controller based on the overall loss using elbow method*
10: *Merge all datasets in the zoo topology*
11: *Perform feature engineering to extract important features*
12: *Get initial features like latitude longitude number of controller size of the datasets*
13: *Compute mean median maximum sum standard deviation max area upper and lower percentiles of the distance between locations*
14: *Build up the datasets and check if there is correlation between the features and the target*
15: *Load the ensemble model library initiate the XGBoost model and split the data into X and y*
16: *Train the model and evaluate it with Mean absolute error*
17: **Expected** : *the predicted number of controllers*

5.4.1 Proposed Learning Vector Quantization for the Controller Placement predictions.

The Algorithm 13 provides a description of the suggested improved learning vector quantisation for SD-WAN controller placement. This technique is used in order to better predict the future locations of controllers. The proposed algorithm will be described in further detail below.

Line 1 describes the datasets, which include the data obtained from the optimization method; the learning rate; a tuning parameter that characterises the step size and is denoted here as θ; the number of epochs; and the number of codebook vectors (prototypes or the weight). In line 2, the output of the algorithm, which is the predicted controller placement, is described in its entirety. The dataset is described in its original values (multi-label/multi-class dataset) in line 3 of the algorithm. During this time, line 4 will convert the dataset into multi-label binary data by populating a two-dimensional array with zeros based on the row size. In line 5, the X variable stands for the encoding training examples, and in line 6, the y variable stands for the encoding target examples. In line 7, the enumeration of the dataset was done by mapping the corresponding values to the index of the dataset with one. In line 8, the X and y were broken up into X_{train}, X_{test}, y_{train}, y_{test}. Line 9 computes the distance between the codebook vectors and the training datasets and finds the minimum distance using euclidean distance. Line 10 denotes the training process of the model, which was carried from lines 11-28. Line 11 iterates the labels through

the transpose of the training data represented by y^T_{train}. Lines 12 and 13 combine to label all of the entire training features X_{train}, yielding a one-dimensional array of features and the target. Line 13 describes the scenario when the random codebook vectors are generated from the entire train. Line 14 loops through the epoch's range before running line 15, which computes the rate, which is represented by $rate{=}\emptyset * (1 - (epoch/n_{epochs}))$. In the same way, lines 16 and 17 go through each row in the train and find the closest match between each row and the $codebook_{label}$. In line 18, the algorithm considers the rows in the train without the last row, which is the label, and updates the weights with the following in lines 19-23: It computes error as the difference between each of the rows and each of the bmu (best matching unit). The bmu is the closest codebook vector. Line 20 updates the codebook vectors by using the $bmu[i] + {=} rate * error$ if the bmu without the label is equal to each row of the train without the label. Otherwise, the model employs this equation $bmu[i] - {=} rate * error$ to update the codebooks. Lines 24-28 end the conditional statement and the control sequence iteration. Meanwhile, lines 29–41 describe the prediction part of the algorithm. Lines 29 and 30 iterate through $codebook_{label}$ in all the codebooks and select codebook vectors that do not have the target features in all positions where the codebooks are one. This is so because the ones represent the top five values, which are our controller positions. Note that the multi-label binary consists of only zeros and ones alone. Lines 31-34 show the model iterating through each row in X_{test} and the codebook iterating through each $codebook_{label}$, after which the Euclidean distance between the codebook and each row is computed in lines 33 and 34. In line 35, predict probability was employed due to the multi-class label approach, and the inverse of the minimum of the codebooks was computed in line 35. Line 36 ends the for-loop, while lines 37 and 38 return the different numbers of labels. On line 39, the predicted values are decoded back to their original values. Similarly, line 40 argsorts the predicted values such that the values are in ascending order. Lastly, line 41 returns the predicted controller positions. The original dataset and the encoded dataset that were discussed above are represented in Figures 5.7 and Figure 5.8 respectively. In the discussion of the algorithm "13," it is revealed that the encoded dataset is what will be fitted to the model, and when the model returns the outcome, it will return the original values by employing argsort, which returns the sorted indices of an array. In other words, the dataset will be decoded before it is fitted to the model.

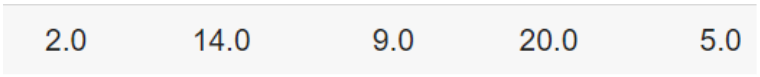

Figure 5.7: Demonstration of datasets used in their original values

Algorithm 13 *Proposed Adapted Learning Vector Quantisation (LVQ) for SD − WAN Controller Placement Prediction*

1: **Input**: *Datasets*, $\emptyset(Learning\ rate)$, n_{epochs}, $n_{codebooks}$
2: **Output**: *predict controller placement positions*
3: *Given dataset in its original values consisting of X training instances and y targets*
4: *Decompose dataset into multi − label instances by creating a* $2 - D$ *array full of zeroes with rows size*
5: *Let X represents the encoded training instances*
6: *Let y represents the encoded target instances*
7: *Enumerate dataset by mapping the corresponding values to the index of dataset with ones*
8: *Split X, and y into* X_{train}, X_{test}, y_{train}, y_{test}
9: *Compute* $get_{closest}$ *using squared Euclidean distance* $= \sum(cv_{ij} - xi)^2 : where\ get_{closest}\ is\ minimum$
10: Fit_{train} *model from lines* $11 - 28$
11: **for** $label\ \in \mathbf{y}_{train}^{T}$ **do**
12: $train$=$np.concatenate([X_{train},\ label])$
13: $codebook_{label}$=$random_{subset}\ of\ train$
14: **for** $epoch\ \in range(n_{epochs})$ **do**
15: $rate$=$\emptyset * (1 - (epoch/n_{epochs}))$
16: **for** $row\ \in train$ **do**
17: bmu=$get_{closest}(codebook_{label},\ row)$
18: **for** $i\ \in range(len(row) - 1)$ **do**
19: $error$=$row_i - bmu[i])$
20: **if** $bmu[-1]$==$row[-1]$ **then**
21: $bmu[i] + = rate * error$
22: **else**
23: $bmu[i] - = rate * error$
24: **end for**
25: **end for**
26: **end for**
27: **end for**
28: **end if**
29: **for** $codebook_{label}$ *in codebooks* **do**
30: $codebook_label_1$=$codebook_{label}$ *where* $codebook_{label}[-1]$ *is* 1
31: **for** *row in* X_{test} **do**
32: **for** $codebook \in codebook_label_1$ **do**
33: $dist_{code}$=$euclidean_{distance}(codebook, row)$
34: **end for**
35: $proba_{label}row$=$1/(\min of\ dist_{code})$
36: **end for**
37: *Returns the n different numbers of labels*
38: **end for**
39: *Re − Decompose prediction back to the original controller values*
40: $sorted_{pred} = argsort\ prediction$
41: **Output** : *Predict controller placement positions*$(last\ five\ columns\ of\ sorted_{pred})$

```
array([0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
      dtype=int16)
```

Figure 5.8: Demonstration of dataset in their encoding format

5.5 Experimentation

This experiment was carried out on a personal computer outfitted with a 2.70 GHz Intel Core i7-6820HQ processor, 64GB of 1600 MHz DDR3 memory, and Microsoft Windows 10 Professional Edition. The experimental setup was programmed in the Python programming language. The code was compiled using Jupyter Notebook version 6.3.0.. The machine learning framework such as PyTorch and TensorFlow PyTorch were employed within the proposed stochastic computational model with ensemble learning approach. The TensorFlow PyTorch was used to create the decision variables. The computational graph was used to express the objective function in the form of graph theory, and the gradient was calculated with the automatic back-propagation approach using PyTorch's auto-grad engine. A dataloader was used to load tensors batch by batch. The code can be assessed at no cost as open-source code on the GitHub code repository at the following link: https://tinyurl.com/2p95ad26. A real-world dataset from the Zoo topology [184] was used in this experiment for the computation of controller placement in SD-WAN. The primary goal of this research is to determine where in the WAN topology is suitable to place controllers such that several conflicting objectives are simultaneously optimized. Based on the experiment carried out in this study, 7 is revealed as the optimal number and is considered in this experiment as the number of controllers needed in the deployment of SD-WAN. The chosen number of controllers is in line with the overall loss and cost. The following six objective functions were used along with the decision variables in optimising the controller. These objectives are average switch-to-controller latency, average inter-controller latency, load balancing, maximum switch-to-controller latency, maximum inter-controller latency, and resilience. The following parameters were used in the proposed stochastic computational model with an ensemble learning approach: These parameters include: learning rate = 0.1, the number of decision variables = 7, the number of objective functions = 6, the early stopping (callback operation) = 10, the number of epochs = 200, the seed value = 12, momentum = 8, decay rate = 0.6, and the training and test data = 80% and 20% respectively. Similarly, this research employs the same zoo topology (Internet 2 OS3E) as discussed above, but it automates the entire datasets in the zoo topology of approximately 242 datasets (each dataset contains a different number of nodes) for the prediction of the number of controllers suitable for each of the datasets based on the conflicting objectives employed in this research. The Haversine function was used to calculate the distance between two points in this case. An extreme gradient-boosting decision tree algorithm was proposed for the number of controller predictions in this study. This was subjected to performance evaluation with other existing regression models. These include logistic regression, K-nearest neighbors, and random forest.

Similarly, this research proposed a classification algorithm referred to as Learning Vector Quantisation for the prediction of controller placement in SD-WAN and this was compared with the existing classification algorithms in order to evaluate and validate the proposed algorithm. K-nearest neighbors algorithm, Random forests, Linear regression, XGBoost, CatBoost, Graph neural network, Feed Forward Network, and Ensemble learning were among the existing algorithms that were compared. The dataset used for the purpose of this classification was obtained from the outcome of the proposed optimization algorithm used in this research. The dataset was encoded before being fitted into the model to improve its performance of the model. The number of codebooks used was set to 40, the learning rate was set to 0.2, and the number of epochs was set to 70. All these were the best outcomes from Optuna, which was employed in this research to automate the tuning of parameters. The dataset was split into training and testing phases. 70% of the data was used in the training phase, while the remaining 30% was used in the testing phase. The Grid Search and Optuna were used in the hyper-parameter tuning of the adapted classification

algorithm (LVQ) to enhance the model and use the best parameters in the training phase of the model, although Optuna had better performance. Simultaneously, an open-source just-in-time compiler referred to as Numba was used in this research to compile the Python code to C to make computation and the speed of the proposed algorithm faster.

5.6 Results and Discussion

This section provides a summary of the experiment's findings. The results of the A-NSGA-III, NSGA-II, and MOPSO algorithms are compared to the outcomes of the stochastic computational graph model with an ensemble learning technique. The findings of the experiment are depicted in Figures 3 through 19. The primary objective of this research is to determine where in the WAN topology SD-WAN controllers should be placed to achieve various network requirements and how many controllers are required to set up the topology. The experiment leveraged six (6) aspects of the controller placement problem in SD-WAN to accomplish the results provided in this study using performance metrics such as execution time, average CPU usage, average core load, and total CPU usage. These were the average switch-to-controller latency, the maximum switch-to-controller latency, the average controller-to-controller latency, the maximum controller-to-controller latency, resilience regarding controller failure, and load balancing. Similarly, the suggested ensemble learning (XGBoost) model was compared to three more regression models (logistic regression, random forest, and K-nearest neighbor). Meanwhile, the proposed learning vector quantization was subjected to performance evaluation with a mean number of accurate controller locations and accuracy metrics. The proposed learning vector quantization classification algorithms were compared with state-of-the-art classification algorithms. These include k-nearest neighbor, random forest, logistic regression, xgboost, catboost, graph neural network, and feed-forward neural network.

In order to ensure a fair comparison, the suggested method and the compared algorithm were subjected to identical data observations in all experiments undertaken for this study. These methods provide the initial (randomly generated) and final (results from the stochastic computational graph model) controller locations, execution time, average CPU use, average core load, and total CPU utilization. The computational resources measure was fully accounted for in the experiment in order to obtain the desired outcome. Note that the total CPU usage is the product of execution time and the average CPU utilization. All the algorithms were combined to enhance comparison and validation. The initial and final controller locations were mapped back to the corresponding location data frames (coordinates-based form), and the haversine distance function was used to calculate the distance between the locations. Input and final losses based on the starting controller location and the final controller location were calculated with the help of the objective functions. The difference in total loss between the initial and final losses was calculated. This was done for all exploited objective functions. Then the average of all these losses for all objectives in each algorithm was extracted for evaluation. All this will help in judging the performance of the models. In the parts that follow, the experimental outcomes are analyzed and discussed.

This study will now show the image depicting one of the datasets utilized in the experiment, as well as the initial and final controller positions, in the next section. This was solely mentioned in the research to convey visual assistance.

5.6.1 The BtEurope Dataset and initial and final controller location images

The dataset used in this research was obtained from the Internet2 0S3E network zoo topology [184]. The internet zoo topology comprises of 260 different wide area network service providers' datasets. It is a repository where service providers store their WAN data for research purposes. The BtEurope is one of the datasets used in this experiment. BtEurope, for example, contains 21 geographically separated cities in Europe. The image of this dataset is shown in Figure 5.9 below. Meanwhile, the entire datasets in the zoo topology were used during the learning of heuristics in this research work in order to find the suitable number of controllers that should be deployed in a network. The usable datasets in the zoo topology were about 242, for a total of about 260

Figure 5.9: Demonstration of the BtEurope Image

Next to the BtEurope datasets is the scatter plots showing the figure of the initial and final controller location. This is captured in both Figure 5.10 and Figure 5.11 respectively. This starting and final location was solely included in the research to convey the visual assistance. The initial controller location was randomly generated with the dynamic computational graph approach during the experiment of the stochastic computational model proposed in this study. The final controller location is the final optimized controller location obtained from the model.

Following the starting and final controller location scatter plots displayed in Figures 5.10 and Figure 5.11 respectively, this study will now present the proposed stochastic computational graph model results in the next section. This result shows the outcome when momentum and decay rate are not added to the experiment, and the second graph shows the overall loss outcome when momentum and decay rate are included in the experiment.

5.6.2 Outcome of the proposed stochastic computational graph models

In this section, the outcome of the proposed stochastic computational graph model results are shown. This result shows the outcome when momentum and decay rate are not added to the experiment, and the second graph shows the overall loss outcome when momentum and decay rate are included in the experiment.

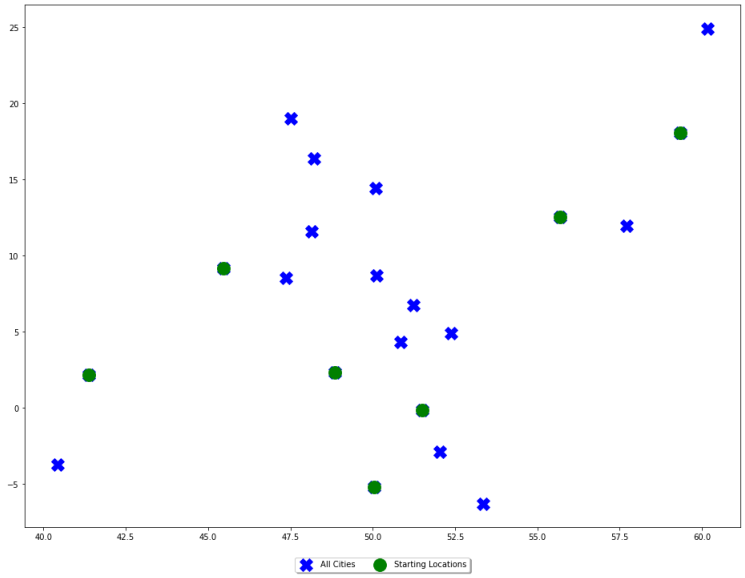

Figure 5.10: A scatter plot showing the Starting location of controllers

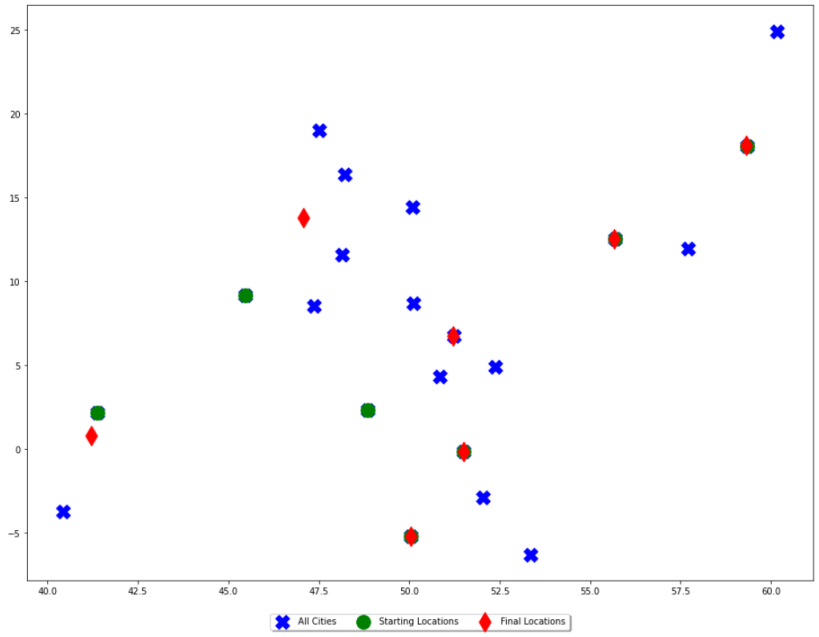

Figure 5.11: A scatter plot showing the Final location of controllers

```
Training ended by early stopping, best score at epoch 209
Best scores at epoch 209 maxN2c: 247.27 meanN2C: 365.30 maxICL: 14.56 meanICL: 14.89 Overall: 1580.04
```

Figure 5.12: Outcome of the proposed stochastic computational graph model without decay rate and momentum

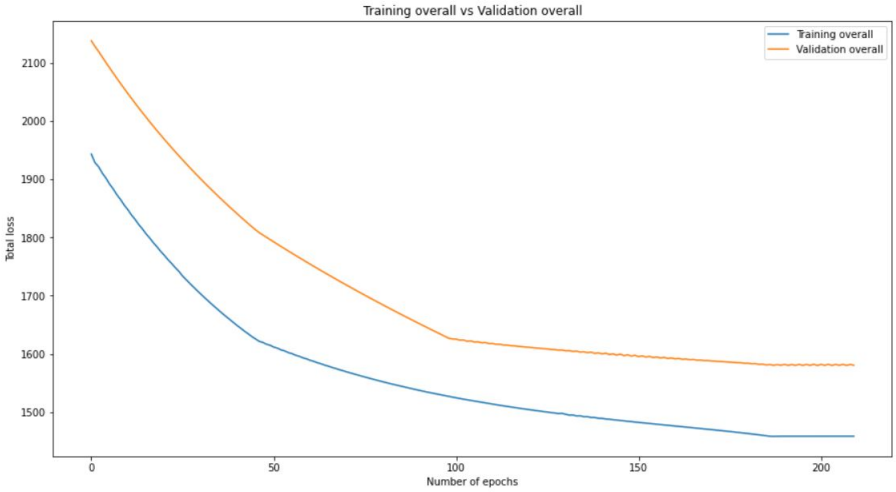

Figure 5.13: Outcome of the proposed stochastic computational graph model without decay rate and momentum

The proposed stochastic computational graph model is depicted in Figure 5.13. This is the graph without the decay rate and momentum. The model works by using the traditional learning rate as its parameter. Although several trials and errors of this learning rate were investigated before finally choosing the most suitable one, the overall loss obtained from this approach is 1580.04. This can be seen in Figure 5.12.

The next four graphs depicted in Figures 5.14, 5.15, 5.16, 5.17 represent the outcome of the proposed stochastic computational graph model with the user control parameters such as decay rate and momentum. Figures 5.14, show the overall initial and final loss obtained by the model. Similarly, the graph in Figure 5.16 shows the training and validation of the switch-to-controller latency with respect to their losses. Meanwhile, the graph in the 5.17 represents the training overall and validation overall loss. The total loss is plotted against the number of epochs in both Figures 5.16 and 5.17.

```
Our starting Cities: [ 7  9 10 20 13  8 14]
Our starting Locations: [[48.85342     2.34881   ]
 [41.3888      2.159     ]
 [50.05001    -5.19999   ]
 [59.332592   18.064909  ]
 [55.675953   12.565539  ]
 [45.464283    9.18952   ]
 [51.50854    -0.12573001]]
Initial maxN2c: 489.01 meanN2C: 441.27 maxICL: 1416.41 meanICL: 1452.65
Overall for initial locs: 4988.71
```

Figure 5.14: Outcome of the proposed stochastic computational graph model with momentum and learning decay rate

```
Training ended by early stopping, best score at epoch 66
Best scores at epoch 66 maxN2c: 193.08 meanN2C: 408.87 maxICL: 1630.59 meanICL: 1606.86 Max tolerance:1048.33 Load Balancing:3.00 Overall: 883.41
```

Figure 5.15: Outcome of the proposed stochastic computational graph model with momentum and learning decay rate

In contrast to the overall loss obtained with the proposed stochastic computational graph model without decay rate and momentum, the outcome (overall loss) of the solution with decay rate and momentum was able to minimize loss with less use of computational resources.

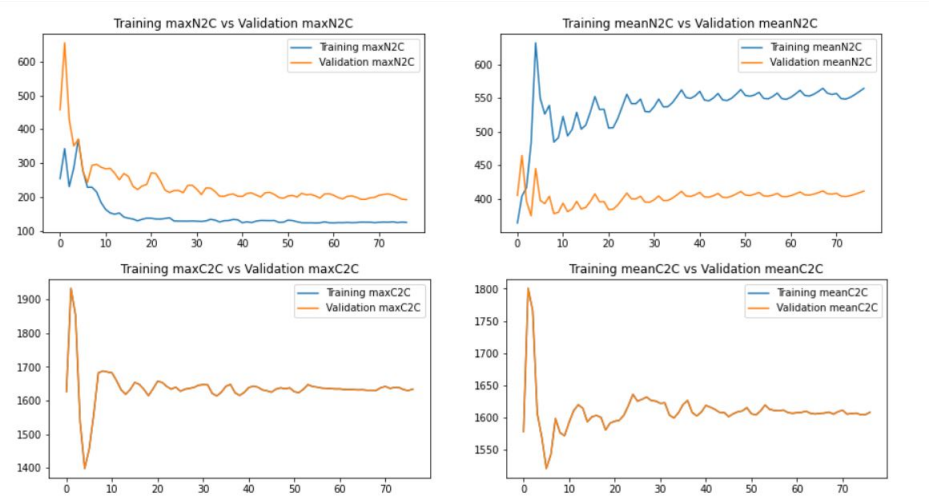

Figure 5.16: Outcome of the proposed stochastic computational graph model with momentum and learning decay rate

The convergence process was faster as compared with the solution without the decay rate and momentum. This further proves the theoretical concept and how strong the effect of decay rate and momentum may be on the model. It is revealed from the graph in Figure 5.16 how fast the convergence process is compared to the figure in Figure 5.13. The overall loss obtained from the graph in Figure 5.16 is 883.41 as against 1580.04 obtained in the solution of Figure 5.13 which experiment was carried out without the parameter of decay rate and momentum.

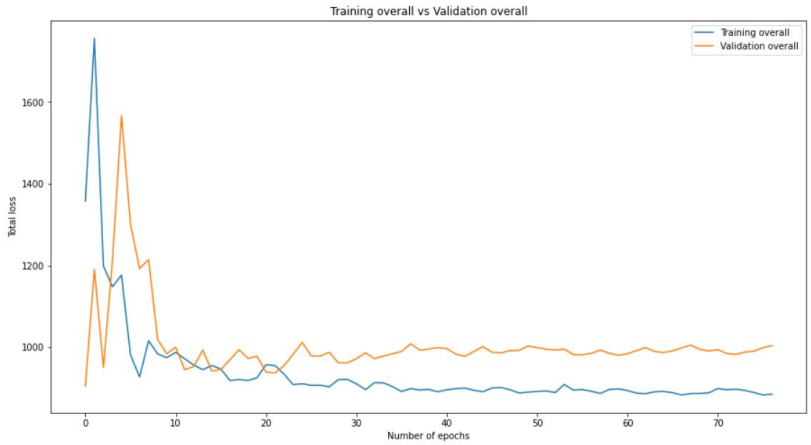

Figure 5.17: Outcome of the proposed stochastic computational graph model with momentum and learning decay rate

In Figure 5.16, the convergence happens at around 70 epochs, as against the convergence in Figure 5.13 which happens at around 209 epochs. In summary, the solution with momentum and a slower decay rate is recommended over the solution without a slower decay rate. This recommended solution will ensure the stability of the training process and speed up the convergence of the model. In addition, it will help in suppressing oscillation movements associated with the traditional approach. This study will now present the number of controllers used and its discussion in the next subsection.

5.6.3 The proposed number of controller using elbow method

In this part of the study, the proposed number of controllers using the elbow method is presented in Figure 5.18. Controller placement in SD-WAN addresses two challenges: the optimal location and the optimal number of controllers needed for the deployment. The researchers in the literature choose the number of controllers to deploy in SD-WAN in advance. This is not a good approach because choosing the number of controllers to be installed in advance may lead to poor overall network performance (high network overhead) if the number selected is too small. However, it may also increase the service providers' capital expenditure if the number selected is too high. Contrary to the approach in the literature, this study has carefully selected the number of controllers needed in the SD-WAN deployment with respect to the overall losses obtained by the optimization model proposed in this study. This approach considered all the objective functions used in optimizing the decision variables.

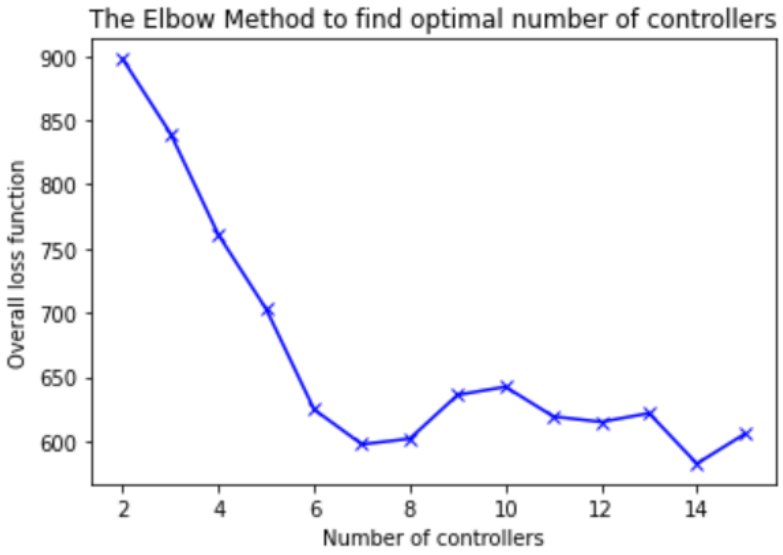

Figure 5.18: Graph showing the Optimal Controller Number

Figure 5.18 revealed the graph of the optimal number of controllers considered for the placement of controllers. This overall loss (y-axis) is plotted against the number of controllers (x-axis). Looking at the graph, it is revealed that 7 controller is optimal. Although, 14 is the number of controllers that has the minimum value of the overall loss, however, the point of inflection is considered in this study. This is where the trade-off comes into play. The cost of installation for 14 controllers is more expensive compared to the cost of installing 7 with a small difference in an overall loss. It will be a wise decision and more cost-effective to choose the optimal number of controllers to install at this point of inflection than to go for a high number of controllers when the difference in overall loss can be negligible.
In the next section, the discussion on the proposed stochastic computational graph model compared with

the existing optimization algorithms used in the optimization of controller placement in SD-WAN will now be presented. The compared solutions are Adapted Non-Dominated Sorting Genetic Algorithm-III (ANSGA-III), Multi-Objective Particle Swarm Optimization (MOPSO), and Non-Dominated Sorting Genetic Algorithm III (NSGA-III). The suggested method and the compared algorithm were subjected to identical data observations in all experiments undertaken for this study. Performance metrics such as execution time, average CPU usage, and total CPU utilization were used to assess the performance of the models.

Consequently, it is recommended for the operator of a wide area network who is using the software-defined network architecture or who plans to use it in the future to employ this approach in choosing the number of controllers for their deployment.

5.6.4 The performance comparison between the proposed solution and the existing optimization algorithms

In this section, the performance comparison between the proposed solution and the existing optimization algorithms will be discussed. The proposed stochastic computational graph model is compared with the three other existing optimization algorithms used in the optimization of controller placement in SD-WAN. The compared solutions are Adapted Non-Dominated Sorting Genetic Algorithm-III (ANSGA-III), Multi-Objective Particle Swarm Optimization (MOPSO), and Non-Dominated Sorting Genetic Algorithm III (NSGA-III). Performance metrics such as execution time, average CPU usage, and total CPU utilization were used to assess the performance of the models (please see: https://tinyurl.com/2p95ad26).

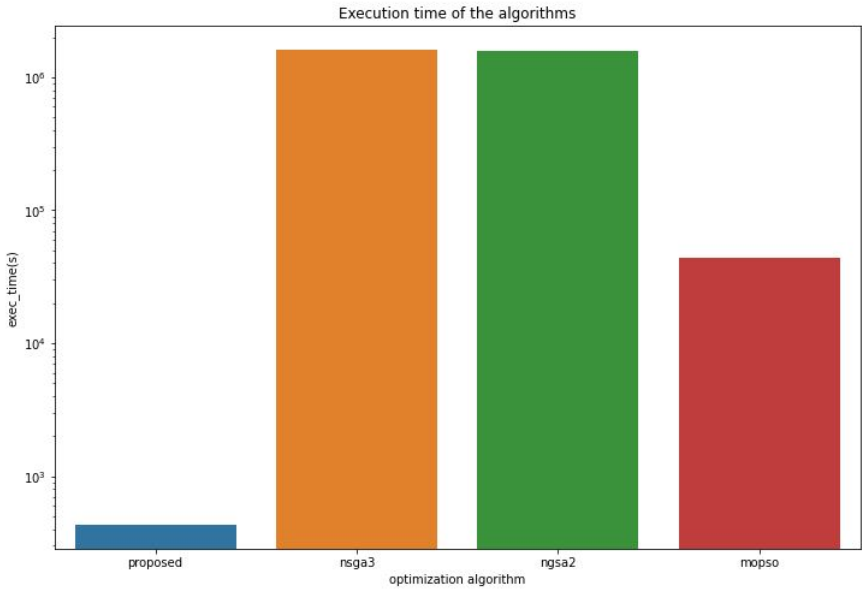

Figure 5.19: Graph showing the Execution of the four Algorithms

The graphs of the proposed optimization algorithm and the existing optimization techniques for the placement of controllers in SD-WAN are represented in Figures 5.19, 5.20, and 5.21 respectively. The graph in Figures 5.19 under the same experimental condition proves that the proposed stochastic computational graph model outperforms others with respect to the execution time. In this experiment, the total CPU was achieved through the product of the execution time and the average CPU usage. The

time module (DateTime) was used in this experiment to calculate the algorithm's running time. The datetime.now() function was used before the first line of the script, and the end time was saved before using the same function before the last line of the script. This time module can be used to achieve different purposes in the running of the experiment. Examples of these include saving the timestamp at the beginning and end of the algorithm as well as finding the difference between the start and end times, which gives the outcome referred to as the "execution time." The proposed solution as depicted in Figure 5.19 shows that the stochastic computational graph model is computationally inexpensive. Consequent to the result exhibited by the graph in Figure 5.19, this study concludes that the proposed solution efficiently performs better than the NSGA2, ANSGA-III, and MOPSO algorithms, with a percentage decrease of 99.972%, 99.974%, and 99.012% respectively. Hence, it is recommended over the state-of-the-art optimization algorithms that are compared in this research.

Similarly, Figure 5.20 shows the average CPU usage of the proposed and state-of-the-art optimization algorithms. CPU utilisation refers to the utilisation of processing resources by the computer. Python System and Process Utilities, or psutil, is a package that can be used to get information about running processes and how the system is being used (CPU, memory, and load). This library was used in all of the optimization algorithms that were considered during this research. It was mostly used to keep an eye on the system and control processes that were already running. As revealed by the graph in 5.20 , the proposed stochastic computational graph model exhibits the lowest CPU usage. Consequent to the result exhibited by the graph in Figure 5.20, this study concludes that the proposed solution efficiently performs better than the NSGA2, ANSGA-III, and MOPSO algorithms, with a percentage decrease of 41.2%, 41.7%, and 43.81% respectively. Based on these results, the proposed optimization algorithm uses the least CPU resources. Hence, it is recommended over the state-of-the-art optimization algorithms that are compared in this research.

In addition, the graph in 5.21 shows the total CPU usage of the proposed optimization algorithm and state-of-the-art optimization algorithms. In this experiment, the total CPU was achieved through the product of the execution time and the average CPU usage. The time module (DateTime) was used in this experiment to calculate the algorithm's running time. The datetime.now() function was used before the first line of the script, and the end time was saved before using the same function before the last line of the script. This time module can be used to achieve different purposes in the running of the experiment. Examples of these include saving the timestamp at the beginning and end of the algorithm as well as finding the difference between the start and end times, which gives the outcome referred to as the "execution time."

Similarly, CPU usage refers to the use of processing resources by the computer. Python System and Process Utilities, or psutil, is a package that can be used to get information about running processes and how the system is being used (CPU, memory, and load). This library was used in all of the optimization algorithms that were considered during this research. It was mostly used to keep an eye on the system and control processes that were already running. As revealed in the graph, it can be seen that the proposed optimization algorithm exhibits the least total CPU usage and, as such, efficiently performs over the ANSGA-III, MOPSO, and NSGA-II with a percentage decrease of 99.983%, 99.985%, and 99.446% respectively. Based on these results, the proposed optimization algorithm uses the least CPU resources. Hence, it is recommended over the state-of-the-art optimization algorithms that are compared in this research.

Following the analysis and discussion of the computational resources used by each of the optimization algorithms in this research, this study will now discuss the losses minimized by these algorithms. This was achieved with the help of the objective functions used in this experiment and the distance between the locations calculated using the haversine function (please see: https://tinyurl.com/2p95ad26).

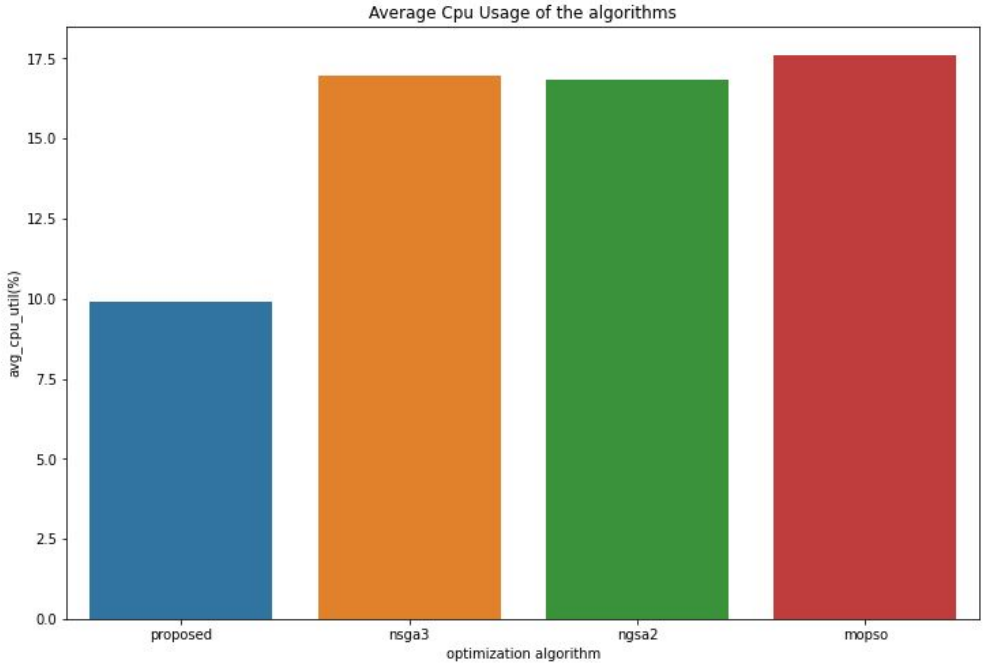

Figure 5.20: Graph showing the Average CPU usage of the four Algorithms

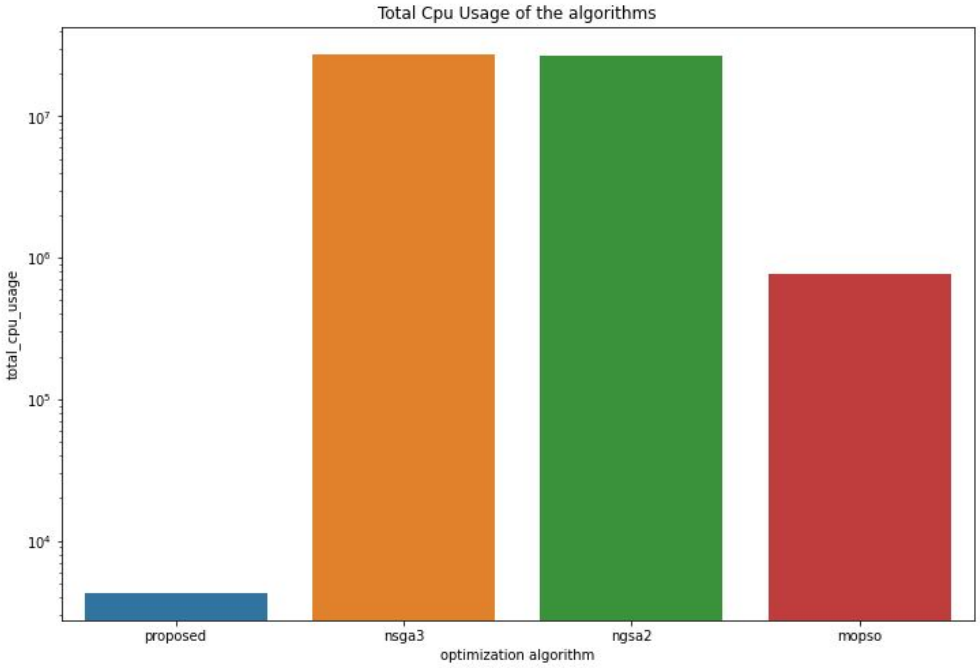

Figure 5.21: Graph showing the Total CPU usage of the four Algorithms

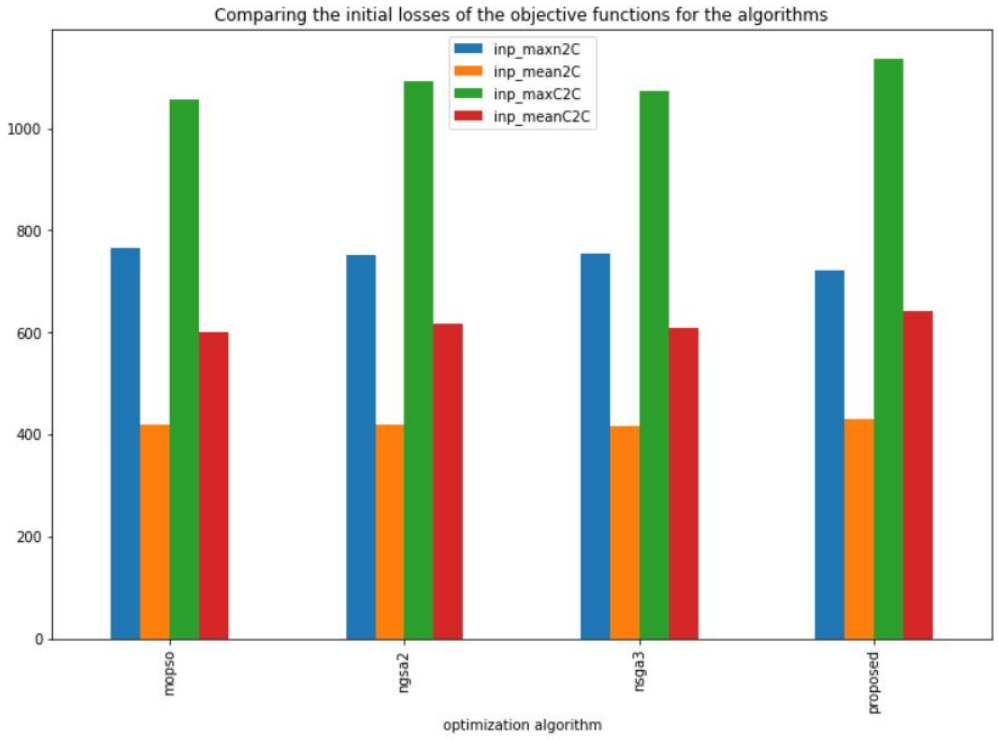

Figure 5.22: Graph showing the initial losses of the four Algorithms

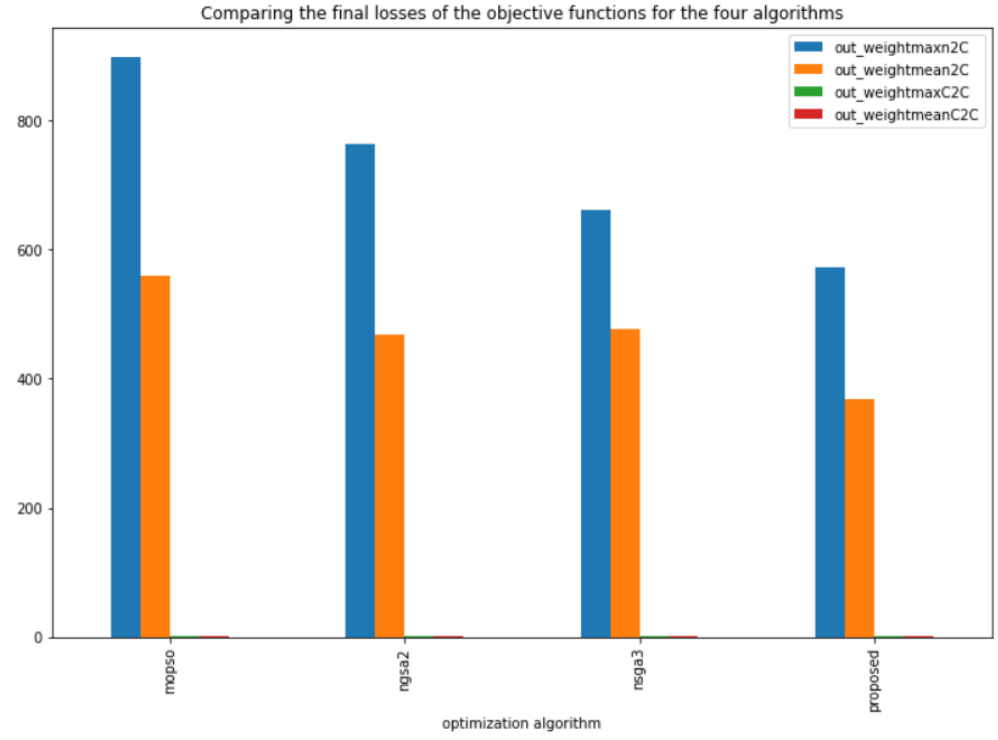

Figure 5.23: Graph showing the final losses of the four Algorithms

The losses minimized by each of the optimization algorithms used in this experiment are displayed in Figures 5.22 and 5.23. The initial losses were captured from the calculation of the losses between different algorithms at the initial controller location, while the final losses were captured from the distance between

the final controller locations. The final controller location was achieved with the help of each algorithm's optimized controller locations after subjecting the initial controller location to the respective optimization algorithms (Please see UPDATEDPLOT1 in : https://tinyurl.com/2p95ad26).) As exhibited by the graph in Figure 5.23, it is revealed that the proposed stochastic computational graph model was able to minimize the objective far better than the compared algorithms. This result further proves the result achieved in Figure 5.17. In the next subsection, this study will now analyze and discuss the proposed ensemble learning model and the other regression models used in the prediction of the optimal number of controllers while deploying SD-WAN.

5.6.5 The performance comparison between the proposed ensemble learning model and the other regression models

In this section, this study will now analyze and discuss the proposed ensemble learning model and the other regression models used in the prediction of the optimal number of controllers while deploying SD-WAN. To predict the optimal number of controllers, the following data was used across the proposed ensemble learning model (XGBoost) and the rest of the machine learning regression models: These include the number of controllers, achieved through the capture of the overall loss with the elbow method during the optimization of controller placement conducted with the help of the proposed stochastic computational graph model. The size of the datasets, the distance between locations in the datasets calculated with the haversine function (longitude and latitude), the mean distance, median distance, difference in the highest and lowest coordinate points, the maximum distance, the upper and lower percentile distance, and the standard deviation distance (please see: https://tinyurl.com/2p95ad26) were all calculated.

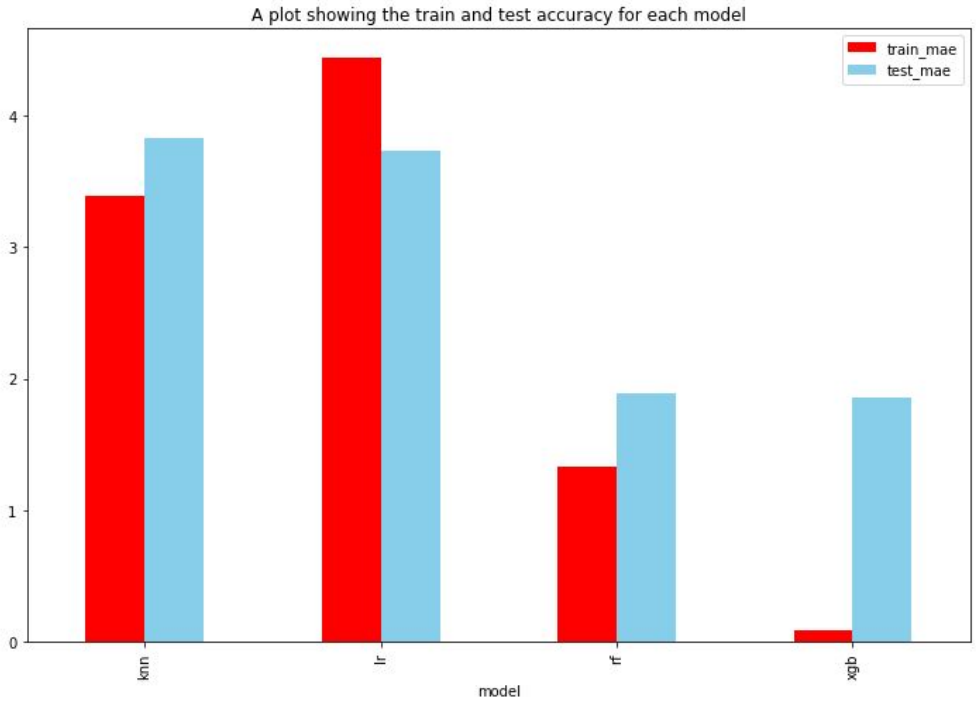

Figure 5.24: Graph showing the train and test accuracy for each model

	fit_time	pred_time	train_mae	test_mae	model
0	0.004425	0.007415	3.390244	3.829268	knn
1	0.016930	0.006021	4.436261	3.729883	lr
2	0.376290	0.044729	1.334114	1.883536	rf
3	0.135608	0.004229	0.089354	1.855751	xgb

Table 5.1: Table showing the performance of each model

The following outcomes of the prediction computed by the proposed ensemble learning model will now be presented in Figures 5.24, 5.1, 5.25, and 5.2. Figure 5.24 display the graph of the train and test accuracy of all the regression models used in this study. The mean absolute error was used to assess the performance of these models. As can be seen from this graph, the proposed ensemble learning model (XGBoost) came up with the least mean absolute error. The lower the mean absolute error, the better the models. This implies that the proposed ensemble learning model was able to reduce more error between the actual and predicted errors. As a direct result of this outcome, this study concludes that the proposed XGBoost outperforms other comparable regression models. Similarly, table 5.1 shows the inferential statistics achieved from these models. This result complements the graph in Figure 5.24.

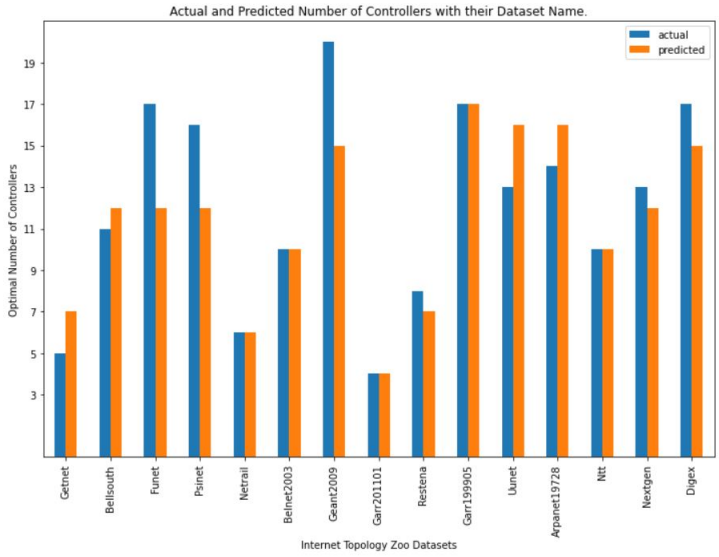

Figure 5.25: Graph showing the proposed XGBoost controller number prediction

Table 5.2: Table showing the proposed XGBoost controller number prediction

98	5	7	Getnet
30	15	17	BtLatinAmerica
25	20	13	Bics
16	10	10	Belnet2003
168	8	7	Restena
195	13	16	Uunet
97	15	15	Geant2012
194	11	12	UsCarrier
67	12	13	Garr199901
120	15	14	Intellifiber
154	10	10	Ntt
202	19	15	York
79	19	19	Garr201004
69	17	17	Garr199905
145	6	6	Netrail
55	17	15	Digex
45	15	14	Cogentco
84	9	9	Garr201104
146	19	15	NetworkUsa

In addition, Figure 5.25 and Table 5.2 show the controller number prediction achieved by the proposed ensemble learning model and the inferential statistics information of the controller number prediction of randomly selected datasets for visual representation. As exhibited in Figure 5.25, it can be seen that the proposed ensemble learning model actually performs better in its prediction because the difference between the actual and predicted error is negligible and it is able to predict accurately most of the time. In the next section of this study, the proposed classification algorithm (learning vector quantization) and the existing classification algorithms will be presented. This research has further used this classification algorithm to predict future controller placement for SD-WAN deployment. This controller placement complements the proposed stochastic computational graph for future controller placement. This proposed classification algorithm is at the discretion of the user or SD-WAN service providers when expanding the network topology.

5.6.6 The performance comparison between the proposed classification algorithm (Learning Vector Quantization and the existing classification algorithms)

This section of the study will now discuss the proposed classification algorithm (learning vector quantization), and the existing classification algorithms. This research has further used this classification algorithm to predict future controller placement for SD-WAN deployment. This controller placement complements the proposed stochastic computational graph for future controller placement. This proposed classification algorithm is at the discretion of the user or SD-WAN service providers when expanding the network topology. A method called Multi-Class Learning Vector Quantization (MCLVQ) was proposed to predict the placement of controllers in SD-WAN. The suggested classification model was validated using some of the existing algorithms in the same class. These algorithms are KNN, Random Forest, Logistic Regression, XGBoost, CatBoost, the Ensemble Model (XGBoost, Random Forest, and CatBoost), Graph Neural Network (GNN), and Feed Forward Network (FFN). The dataset (the final controller location)

achieved from the output of the proposed stochastic computational graph model was used in the optimization of several conflicting objectives in the placement of controllers. This dataset was originally in a multi-class, multi-label format and was converted into a multi-class, binary classification format and then used to train the model. This brought down the number of dimensions and made it easier for the suggested model to figure out how the features related to the target. Because of the nature of this research, a new metric called "the mean number of accurate controller locations" was developed to measure how well the classification models worked. With the help of Optuna, a software framework for automatic hyper-parameter optimization, the process of optimizing the hyper-parameters that were used to train the model was done automatically. These parameters include the number of folds, learning rate, number of epochs, and number of codebooks. To make the computation faster, Numba, an open-source just-in-time compiler, was also used to compile the Python code to C (please see: https://tinyurl.com/2p95ad26).

Table 5.3: Table showing the datasets used in the classification algorithms in their original format

	output_0	output_1	output_2	output_3	output_4	output_5	output_6
0	2.0	6.0	16.0	11.0	20.0	17.0	19.0
1	0.0	5.0	2.0	9.0	12.0	13.0	16.0
2	2.0	6.0	9.0	10.0	20.0	14.0	18.0
3	0.0	5.0	8.0	9.0	16.0	17.0	20.0
4	18.0	1.0	4.0	10.0	11.0	16.0	20.0

```
1 y
```

```
array([[0, 0, 1, ..., 0, 1, 1],
       [1, 0, 1, ..., 0, 0, 0],
       [0, 0, 1, ..., 1, 0, 1],
       ...,
       [0, 1, 1, ..., 0, 1, 1],
       [0, 1, 0, ..., 1, 0, 1],
       [0, 0, 0, ..., 0, 1, 0]], dtype=int16)
```

Figure 5.26: Figure showing the converted datasets to binary classification formats

```
1 y[0]
```

```
array([0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1],
      dtype=int16)
```

Figure 5.27: Figure showing the converted datasets to binary classification formats

The outcome of the proposed classification algorithms and the other compared classification algorithms are displayed in Figures 5.26, 5.27, 5.28, 5.4, 5.33, 5.30, 5.34, and 5.35. The table in 5.3 displays the dataset used in the training of the proposed learning vector quantization and the compared classification

algorithms. Similarly, the Figure in 5.26 and 5.27 shows the conversion of these original datasets in 5.3 into binary classification format to enhance the performance of the training model.

Table 5.4: Table showing the inferential statistics information of the classification algorithms

	fit_time(secs)	predict_time(secs)	train_acc	test_acc	train_score	test_score	model
0	81.709807	1.843094	0.908272	0.844508	5.665143	5.126667	lvqmc
1	55.186260	0.426579	0.908272	0.844508	5.665143	5.126667	mcLVQ
2	0.015092	3.166598	0.818857	0.815810	5.287429	5.244000	knn
3	18.830309	5.239218	1.000000	0.888254	7.000000	5.834667	rf
4	0.308197	0.009693	0.860653	0.862095	5.524571	5.534667	lr
5	14.529471	1.561821	0.976435	0.886921	6.809143	5.808000	xgb
6	23.220212	0.384406	0.999374	0.896381	6.997714	5.918667	cat
7	56.245652	7.269588	0.999592	0.895429	6.997714	5.906667	vote
8	145.457107	4.600025	0.794222	0.793605	4.002286	3.982667	gnn
9	45.347118	0.996642	0.893524	0.879365	5.724571	5.642667	FFN

Similarly, the inferential statistics information of all the classification algorithms is revealed in table 5.4. The parameters used are the fit time (secs), the predicted time (secs), the training accuracy, the test accuracy, the training score, and the test score. The train and test scores were achieved from the developed mean accurate location. This mean of "accurate location" is the performance metric developed in this study to measure the performance of the classification algorithms with respect to the number of controllers. It is revealed from this table that the proposed learning vector quantization performs well along with the catboost and logistic classification algorithms with respect to prediction time. The prediction time is the time taken for the model to predict the controller placement outcome. In addition, the accuracy of the proposed learning vector is similar to that of the other compared classification algorithms. All accuracy of the classification algorithms except graph neural network lies in the range of $80s$. The table next to the 5.27 further complements this discussion that the proposed learning vector quantization performs well in the prediction of controller placement along with the other classification algorithms.

```
1 actual_locs[:10]
```

```
[[9, 19, 13, 2, 16, 17, 20],
 [6, 11, 13, 2, 16, 18, 20],
 [9, 11, 14, 16, 1, 7, 20],
 [8, 9, 5, 11, 13, 15, 17],
 [5, 9, 11, 12, 15, 17, 18],
 [17, 16, 10, 9, 1, 15, 20],
 [16, 0, 7, 5, 3, 11, 20],
 [9, 5, 14, 2, 16, 17, 20],
 [11, 14, 1, 16, 17, 18, 20],
 [18, 17, 0, 6, 5, 9, 20]]
```

Figure 5.28: Figure showing the actual controller placement position

```
[[9, 17, 16, 18, 13, 20, 19],
 [2, 17, 18, 20, 16, 13, 11],
 [20, 7, 17, 1, 14, 11, 9],
 [6, 15, 8, 16, 7, 9, 11],
 [11, 20, 16, 19, 15, 1, 17],
 [1, 16, 20, 18, 10, 17, 9],
 [5, 2, 15, 9, 16, 3, 20],
 [3, 18, 2, 16, 17, 6, 20],
 [17, 16, 14, 20, 11, 7, 1],
 [10, 17, 6, 9, 2, 20, 19]]
```

Figure 5.29: Figure showing the predicted controller placements of the proposed learning vector quantization

```
[[11, 2, 18, 13, 16, 19, 20],
 [18, 15, 2, 13, 16, 11, 20],
 [15, 1, 20, 11, 16, 9, 14],
 [3, 6, 17, 13, 11, 9, 15],
 [1, 2, 20, 17, 9, 11, 15],
 [1, 20, 10, 18, 17, 15, 16],
 [15, 5, 2, 9, 3, 16, 20],
 [6, 2, 17, 14, 16, 9, 20],
 [1, 7, 17, 11, 20, 14, 16],
 [14, 5, 0, 17, 6, 20, 9]]
```

Figure 5.30: Figure showing the predicted controller placement of the catboost model

Figures 5.28, 5.34, and 5.30 show the actual and predicted controller placement by the proposed learning vector quantization, as well as the best-performing classification algorithms in terms of accuracy and the mean number of accurate locations. This figure shows that both the proposed classification algorithm and the catboost did a good job of predicting where the controllers should be placed. In Figure 5.34, the numbers $9, 17, 16, 18, 13, 20, and$ 19 refer to the positions of the cities in the datasets. For examples, 9 represents Barcelona; 17 represents Dublin; 16 represents Madley; and 18 represents Brussels; 13 represents Copenhagen; 20 represents Gothenburg; and 19 represents Amsterdam. All these geographical cities are from the BtEurope datasets. In most cases, the two of them were able to correctly predict six of the seven controllers. So, this showed that the proposed algorithm could work well in all situations, so it was recommended for predicting where controllers should be placed in SD-WAN.

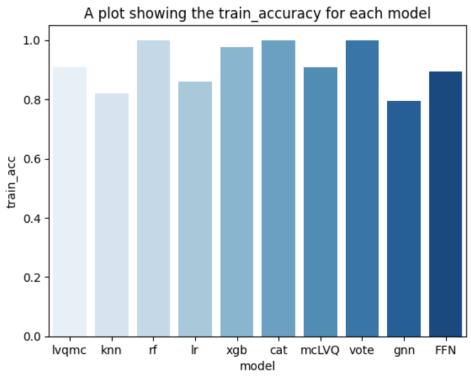

Figure 5.31: The bar plot of the classification algorithms train accuracy

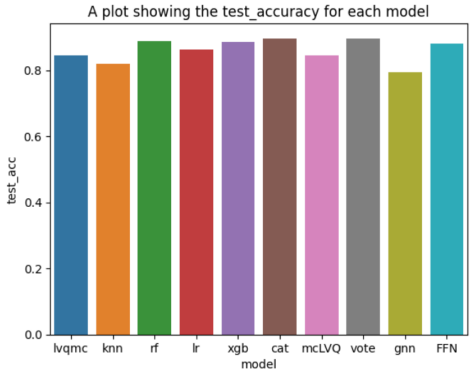

Figure 5.32: The bar plot of the classification algorithms train accuracy

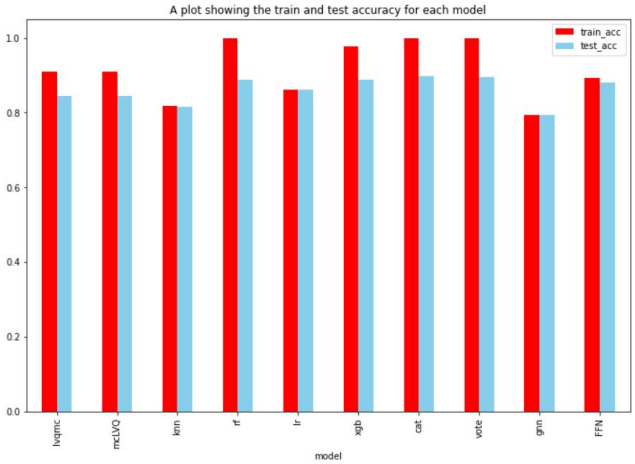

Figure 5.33: The bar plot of the classification algorithms train and test accuracy

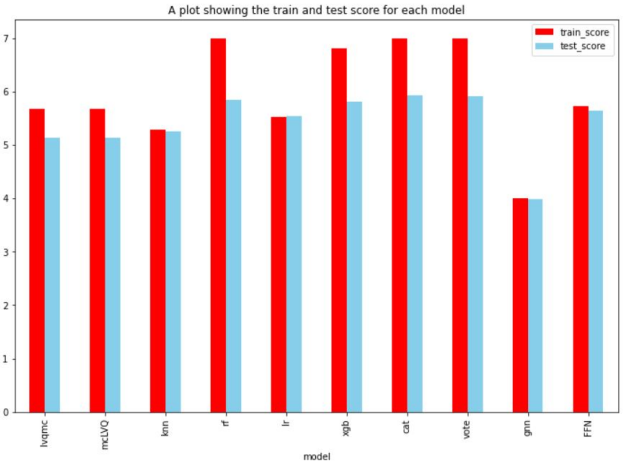

Figure 5.34: The bar plot of the classification algorithms train and test score

In conclusion, the graph in Figures 5.33, 5.34, 5.35 shows the bar plot of the classification algorithms' train and test accuracy, train and test score, and the merged bar plot of the classification algorithms' fit and predict time. Figure 5.33 shows the bar plot of the train and test accuracy for all the classifications. This bar plot shows that all of the algorithms worked well, except for the k-nearest neighbor, which

did not work as well as the others. Similarly, Figure 5.34 shows the training and test scores of all the classification algorithms. All classification algorithms except the graph neural network perform well and are able to score above 80% in the number of accurate controller placement predictions. Lastly, the merged bar plot revealed in Figure 5.35 shows the time taken for the classification algorithm model to train and predict the placement of controllers. From the outcome displayed in this part of the study, it is concluded that the proposed learning vector quantization efficiently performs well along with similar classification algorithms. Hence, it is recommended for use in the prediction of controller placement in SD-WAN.

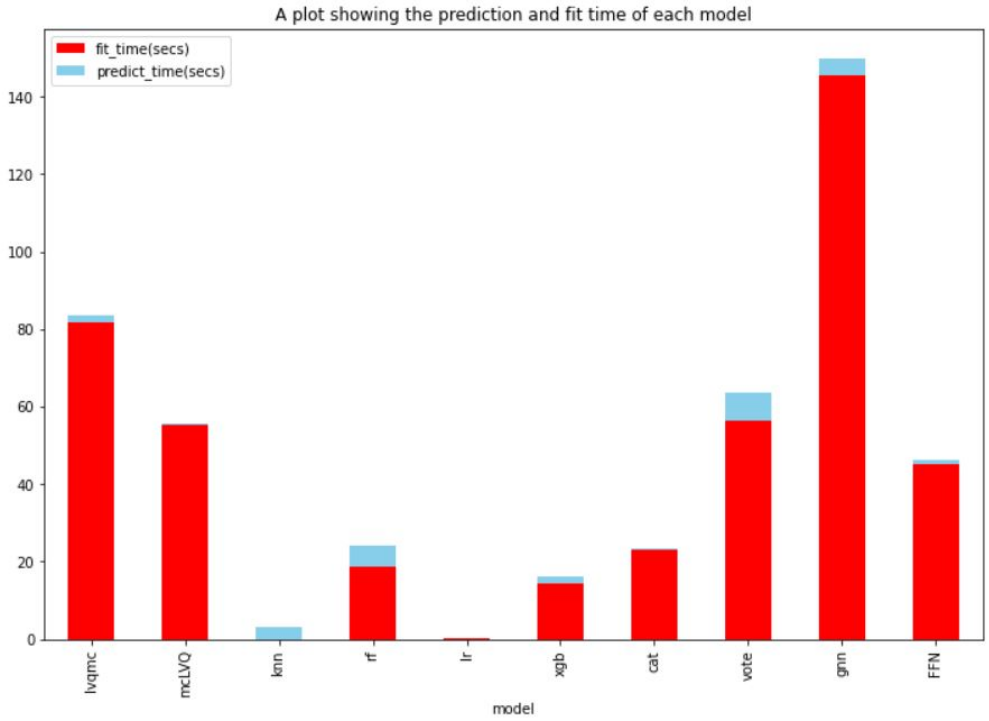

Figure 5.35: The merged bar plot of the classification algorithms fit and prediction time

5.7 Verification and Validation of the proposed Stochastic Computational model

In this part of the thesis, the author shows that the proposed model has met the main goal in terms of its quality and credibility by showing that it has done what it was meant to do. The verification process includes all of the steps involved in creating a high-quality solution, such as testing, design analysis, specification analysis, and so on. The technique can be thought of as being fairly objective. In contrast, the process of validation is extremely subjective in nature. It requires making subjective judgments about how well a solution that has been presented or produced addresses a demand that exists in the real world. The validation process encompasses a variety of activities, such as modeling the requirements, prototyping, and user testing. The specifications were followed closely during the planning and construction of the suggested solution. The aim of the proposed solution is to address the challenges identified in 2 of this thesis, which include scalability, high computational complexity, and the lack of an existing method to learn the heuristics of combinatorial optimization. Looking at the figure in 5.6 and the algorithm 11, it is clearly seen that the proposed stochastic computational graph model for the

optimization of controller placement in SD-WAN actually fulfills its intended design aim and meets the expected outcomes. The loss/objective function progression is shown in Figure 5.36 which further reveals how the proposed model reduces by finding the optimal location. In a similar fashion figure 5.37 shows the optimal location of controllers. Both figures confirm the verification process of the proposed model and how it actually meets its intended purpose.

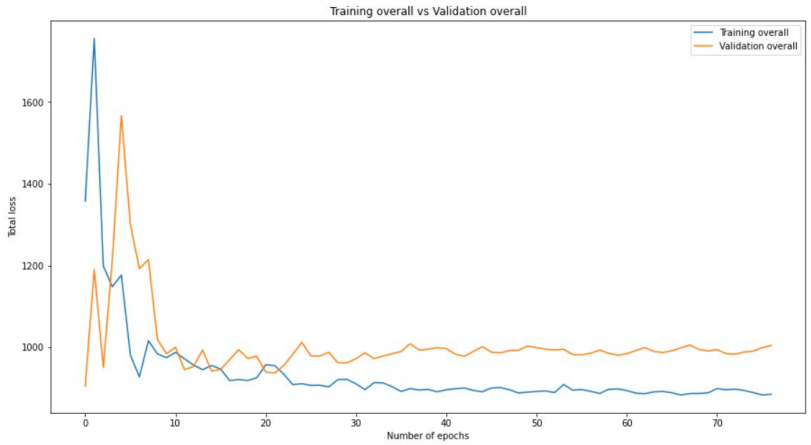

Figure 5.36: Outcome of the proposed stochastic computational graph model with momentum and a learning decay rate

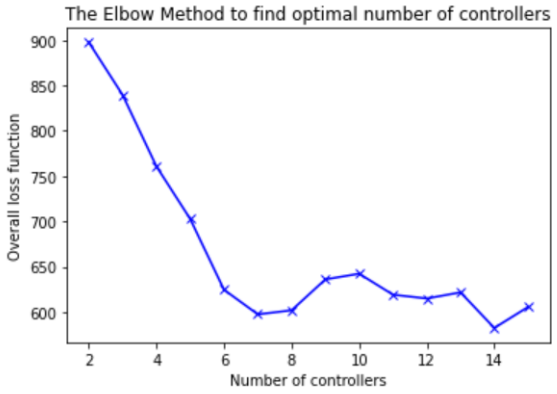

Figure 5.37: Graph showing the Optimal Controller Number

The overall loss function and its progression towards the optimal solution in both figure 5.36 and 5.37 clearly confirm the intention of designing this model. The intention of the author is to build a solution that would be able to reduce the overall loss of the objective functions used along with the decision variables (controller locations) in optimizing controller placement. Meanwhile, in a supervised machine learning approach, model performance can be validated with unseen data called a "validation or test set." On the other hand, the proposed stochastic computational graph model is a reinforcement learning method, not a supervised learning method. In this case, the outputs are compared and the model is tested with different data sets. Such datasets include BtLatinAmerica, BtNorthAmerica, NetworkUsa, and TataNld which are all extracted from the datasets [184] deposited to the online repository by the service providers. The figure in 5.38 shows that the proposed stochastic computational graph model

performs efficiently over the existing models (ANSGA-III, NSGA-II, and MOPSO). The total CPU usage (the product of average CPU usage and execution time) exhibited by the figure in 5.38 confirms the validity of the proposed model.

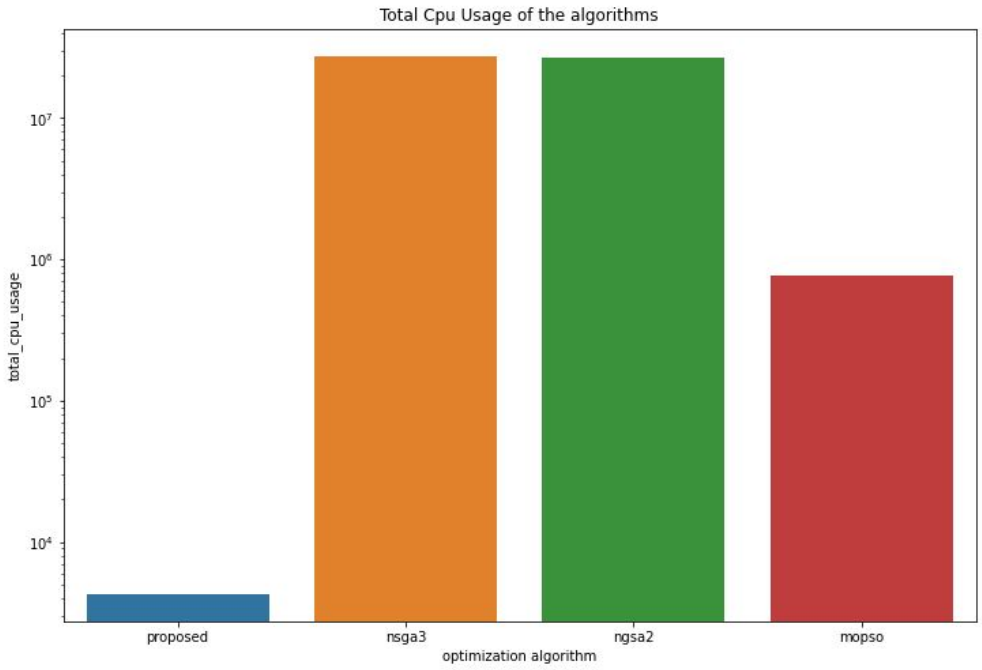

Figure 5.38: Graph showing the Total CPU usage of the four Algorithms

5.7.1 Inferential Statistical Analysis for Controller Placement Algorithms

The aim of this analysis is to use statistics to find out if the proposed controller placement algorithm (SCGMEL) really has a low and significant output loss compared to the MOPSO, NSGA-II, and ANSGA-III algorithms. This analysis is important because it will help find the best way to place controllers in the SDN so that output loss is kept to a minimum. The descriptive graph chart is shown below to give an understanding of the considered and reviewed optimization algorithms before conducting the inferential statistics.

Following the descriptive chart in 5.40 is the sample data used in conducting the inferential statistical analysis. The sample data represent the losses obtained by each of the optimization algorithms. The network final output loss means of the algorithms are displayed in table 5.5. Meanwhile, its variances are 25139.335528, 3067.111557, 32561.492155, and 33732.179519 for Proposed SCGMEL, MOPSO, NSGA-II, and ANSGA-III respectively.

```
The network final output loss means of the algorithms are:
 proposed     936.018623
mopso       1459.347972
nsga2       1221.599093
nsga3       1142.605371
```

Figure 5.39: Final output loss Average

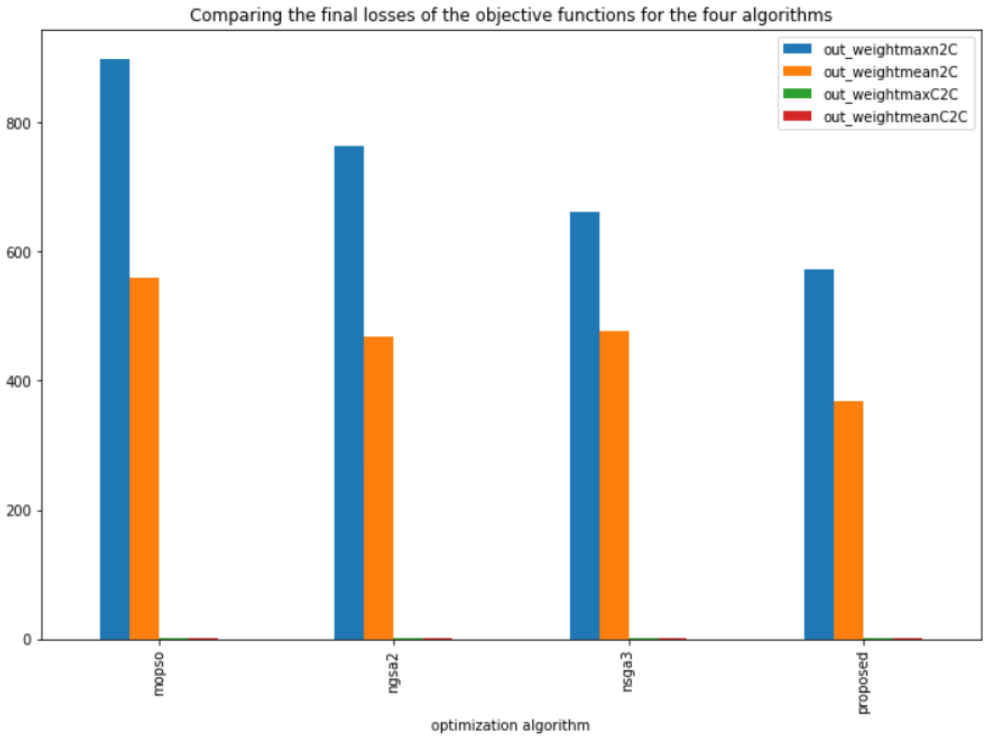

Figure 5.40: Graph showing the final losses of the four Algorithms

Table 5.5: Sample data representing losses obtained from the four Algorithms

	proposed	mopso	nsga2	nsga3
0	716.016170	1493.835447	1318.060786	1205.792011
1	714.692806	1493.928038	1443.265407	1079.906553
2	1073.149020	1389.168394	844.720857	1024.933980
3	690.896103	1492.154173	773.148163	1135.833079
4	1013.050896	1389.585855	1157.688084	1220.144165
5	1013.075670	1540.748289	1396.240934	939.967940
6	1013.146655	1365.462906	1021.786994	1039.683992
7	999.735857	1438.179971	1220.144165	1275.039106
8	737.644693	1387.394529	1364.921533	1028.227812
9	865.387042	1389.075804	797.485422	1200.618064
10	1331.641651	1438.347339	1117.091896	773.188143

Interpretation: Looking at Figure 5.39, the proposed algorithm has the network final output loss means of 936.018623 while the MOPSO algorithm has the highest network final output loss means of 1459.347972. This study is interested to know if there is a significant difference among these fours controller based on the network final output loss data.

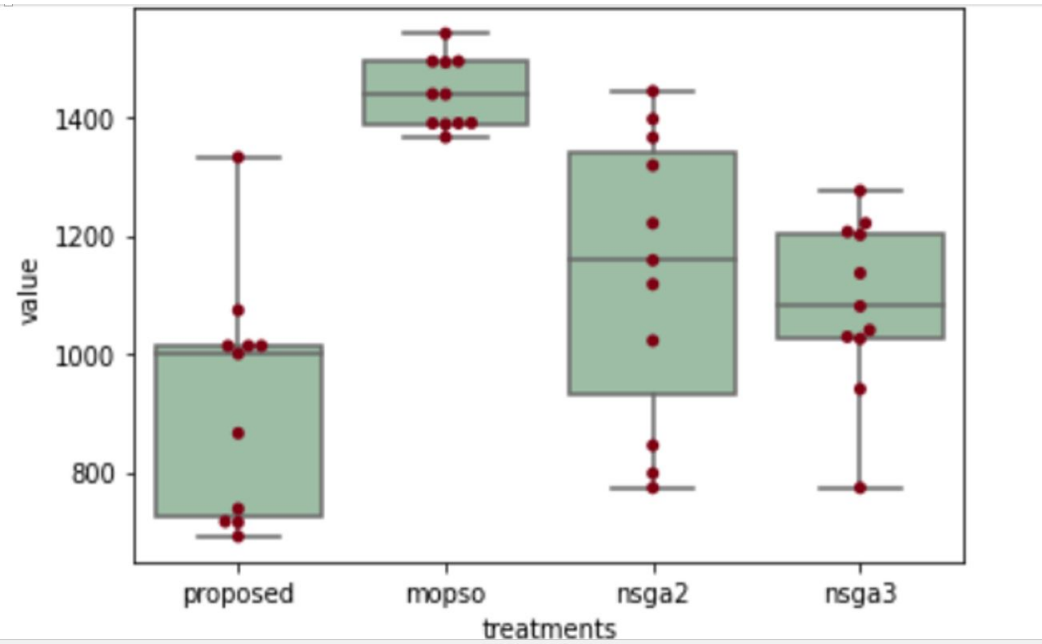

Figure 5.41: Box Plot for the Optimization Algorithm

Each box represents each controller optimization algorithm, there is no presence of outliers based on the revealed box plot. However, there is a need for further investigation to actually know if the data obey the normality assumption or not.

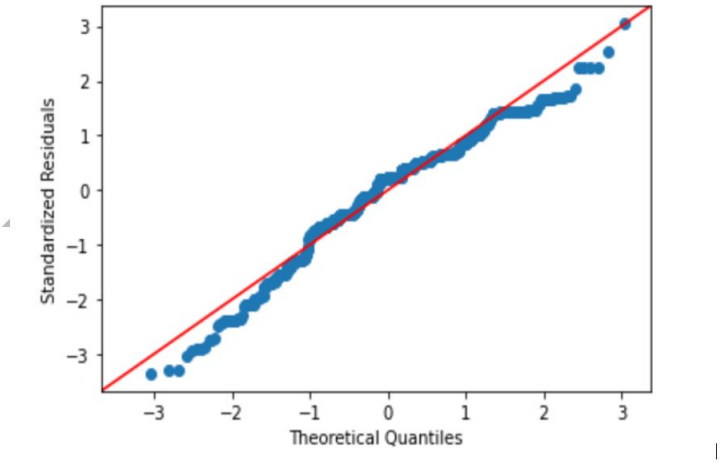

Figure 5.42: Standardized Residual Plot

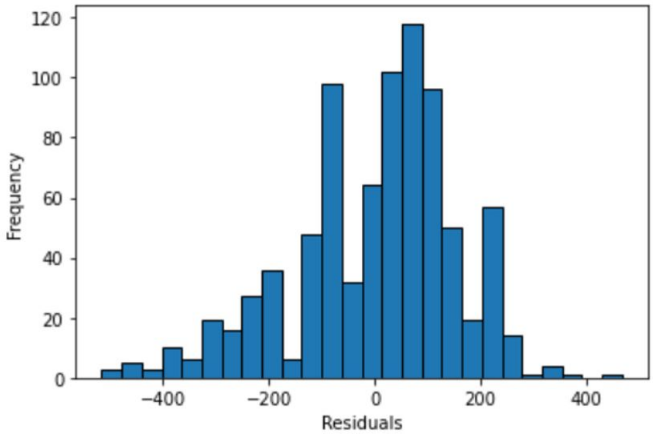

Figure 5.43: Histogram Plot

Interpretation of standardized residuals plot: As the standardized residuals lie closely around the 45-degree line, it suggests that the residuals are likely to be approximately normally distributed. **Interpretation of histogram plot:** In the histogram, the distribution looks approximately left-tailed and suggests that residuals are not like to be approximately normally distributed. The normal distribution and homogeneity test will be conducted for confirmation of what is suspected in Figure 5.42 and 5.43. In order to conduct the normality and homogeneity test, this research has employed statistical tools known as Shapiro-Wilk and Levene's tests respectively.

5.7.1.1 Shapiro-Wilk test for Normality assumption test

H0: The dataset is derived from a normal distribution using a 5% significance level.
H1: The dataset is not derived from a normal distribution using a 5% significance level.

```
0.9675437211990356 1.1072497085873323e-12
==============================
 Decision: Reject the Null hypothesis
```

Figure 5.44: Shapiro-Wilk Normality Test result

Interpretation: The dataset is NOT derived from normal distribution at a 5% significant level. The p-value is less than the significant level.

5.7.1.2 Homogeneity of variances test

H0: The variances of the four samples taken from the populations are equal to one another.
H1: The variances of the four samples taken from the populations are not equal to one another.
Levene's and Barlett's test
The below shows the result obtained from Levene's and Barlett's equality of variance test respectively.

	Parameter	Value
0	Test statistics (W)	37.4571
1	Degrees of freedom (Df)	3.0000
2	p value	0.0000

Figure 5.45: Levene's method equality of variance result

```
270.3730332464351 2.5630732141706028e-58
```

	Parameter	Value
0	Test statistics (T)	270.373
1	Degrees of freedom (Df)	3.000
2	p value	0.000

Figure 5.46: Barlett's method equality of variance result

Interpretation: As the p-value, for both Bartlett's test ($0.0000 < 0.05$) and Levene's test ($0.0000 < 0.05$) is significant, we reject the null hypothesis and conclude that four algorithms do not have equal variances. On this note, the parametric approach would not be used since the main two assumptions of normality and homogeneity are violated. Hence, this work resulted in the use of the non-parametric statistical approach for further analysis.

5.7.1.3 Kruskal-Wallis test

This test replaces the parametric test dues to the violation of the homoscedasticity of normality and variance assumption.
HO: There are no significant differences among network final output loss medians for the four algorithms at a 5% significant level.
H1: At least one of the algorithms produces the network final output loss median that is significantly different from other algorithms at a 5% significant level.

```
Kruskal-Wallis Test
-------------------
H statistic = 547.4398221375352
p value = 2.4937473240269875e-118
```

Figure 5.47: Kruskal-Wallis Test result

Decision: There is significant evidence to reject the null hypothesis at a 5% significant level. Conclusion: At least one of the algorithms produces the network final output loss median that is significantly different from other algorithms at a 5% significant level. Since there is evidence that at least one of the algorithms produces the network final output loss median that is significantly different from other algorithms at a 5% significant level, hence the test will compare the algorithms pair wisely using the

Wilcoxon rank-sum pairwise method.

5.7.1.4 Post HOC Test

Post HOC Test

Wilcoxon rank-sum Pairwise method: The median will be compared for the difference. The median for the final output loss is:

```
The median will be compared for difference. The median for the final output loss for:
Proposed Algorithm   : 998.4189305
Mopso Algorithm      : 1492.154173
NSGA-II Algorithm    : 1318.060786
NSGA-III Algorithm   : 1135.833079
```

Figure 5.48: Final Output loss Median for the Optimization Algorithm

The analysis of each of the optimization algorithms is now performed with the Post-HOC test.

```
Proposed Algorithm and Mopso Algorithm Comparism
================================================
RanksumsResult(statistic=-17.636206316605595, pvalue=1.2989416968608584e-69)
```

Figure 5.49: Comparison of the Proposed and MOPSO Algorithm

The null hypothesis is ruled out. It is concluded that there is a significant distinction between the two median values of the two algorithms that were examined at the 5% significance level. This implies that the proposed algorithm has a significantly low final output loss than the MOPSO algorithm at a 5% significant level.

```
Proposed Algorithm and NSGA-II Algorithm Comparism
==================================================
RanksumsResult(statistic=-13.538190193544098, pvalue=9.305331701826858e-42)
```

Figure 5.50: Comparison of the Proposed and NSGA-II Algorithm

The null hypothesis is ruled out. It is concluded that there is a significant distinction between the two median values of the two algorithms that were examined at the 5% significance level. This implies that the proposed algorithm has a significantly low final output loss than the NSGA-II algorithm at a 5% significant level.

```
Proposed Algorithm and NSGA-III Algorithm Comparism
================================================
RanksumsResult(statistic=-11.100109469074988, pvalue=1.252903356368298e-28)
```

Figure 5.51: Comparison of the Proposed and NSGA-III Algorithm

The null hypothesis is ruled out. It is concluded that there is a significant distinction between the two median values of the two algorithms that were examined at the 5% significance level. This implies that the proposed algorithm has a significantly low final output loss than the NSGA-III algorithm at a 5% significant level.

```
NSGA-II Algorithm and NSGA-III Algorithm Comparism
================================================
RanksumsResult(statistic=4.727172125357244, pvalue=2.2766826600821443e-06)
```

Figure 5.52: Comparison of the NSGA-II and NSGA-III Algorithm

The null hypothesis is ruled out. It is concluded that there is a significant distinction between the two median values of the two algorithms that were examined at the 5% significance level. This implies that the NSGA-III algorithm has a significantly low final output loss than the NSGA-II algorithm at a 5% significant level.

```
Mopso Algorithm and NSGA-III Algorithm Comparism
================================================
RanksumsResult(statistic=16.71312029007129, pvalue=1.0518462090458277e-62)
```

Figure 5.53: Comparison of the MOPSO and NSGA-III Algorithm

The null hypothesis is ruled out. It is concluded that there is a significant distinction between the two median values of the two algorithms that were examined at the 5% significance level. This implies that the NSGA-III algorithm has a significantly low final output loss than the MOPSO algorithm at a 5% significant level.

```
Mopso Algorithm and NSGA-II Algorithm Comparism
================================================
RanksumsResult(statistic=15.348329502594478, pvalue=3.6342465335086066e-53)
```

Figure 5.54: Comparison of the MOPSO and NSGA-II Algorithm

The null hypothesis is ruled out. It is concluded that there is a significant distinction between the two median values of the two algorithms that were examined at the 5% significance level. This implies that the NSGA-II algorithm has a significantly low final output loss than the MOPSO algorithm at a 5% significant level. Further to the investigation carried out in this research using inferential statistics, the outcome

revealed that the proposed SCGMEL actually performed efficiently over the compared algorithms as it ensured that the objective functions were adequately minimized, which confirmed the outcome revealed by the descriptive analysis displayed in 5.40.

5.8 Conclusion Remarks

This section gives a summary of the proposed intelligent-based solution to address the controller placement problem in SD-WAN. The controller placement problem is a problem associated with large-scale enterprises. Such enterprises include wide-area networks (service providers). The controller placement problem deals with the location of a suitable position to place controllers and the number of controllers needed for such a deployment. There is a need to address this problem when there are several conflicting objectives to be simultaneously optimized. Metaheuristic algorithms have been suggested in the literature for the placement of controllers in SD-WAN. It has been confirmed based on the investigation and experiment conducted in this study that these metaheuristic algorithms (ANSGA-III, NSGA-II, and the MOPSO) developed in the literature, with the exception of ANSGA-III, were associated with a scalability challenge (where there were more than three objectives to be optimized). Yet, these solutions are known to be computationally expensive and lack the intelligence mechanism to predict the number of controllers required for SD-WAN deployment. The developed solutions in the literature randomly select the number of controllers to deploy in advance, except for some solutions that used the K-means algorithm to determine the optimal number of controllers, which is limited to just one performance metric. Choosing any number of controllers to be deployed in a network is not an effective approach because this has a direct impact on the cost of installation, which is likely to be too low or too high for the topology. The impact of this approach is that the overall performance of the networks may be degraded when the number of controllers deployed is too low. On the other hand, when the number of controllers deployed is too high, this may result in an increase in the service provider's capital expenditure. Hence this study proposes a stochastic computational graph model with an ensemble learning approach for the placement of SD-WAN controllers in the presence of several competing objectives. The proposed solution is a combination of an optimization algorithm (stochastic computational graph model) and a scalable extreme gradient decision tree algorithm (XGBoost). The solution proposed can be used to address the controller placement problem and alleviate the associated challenges with the existing metaheuristic algorithms used in the placement of controllers. The stochastic computational graph model is used as an optimization technique to optimise controller placement, while XGBoost is used to predict the optimal number of controllers needed for deployment. This study further proposed an artificial neural network (learning vector quantization) that further predicted the placement of controllers, especially when the network topology expanded. The proposed solution was carried out using the network topology obtained from the Internet Zoo topology. The proposed stochastic computational graph model was evaluated and validated with the existing optimization algorithms. These include the ANSGA-III, NSGA-II, and MOPSO, respectively. Execution time, average CPU utilization, and total CPU usage were used as performance metrics to assess the performance of these models. The outcomes revealed that the proposed solution efficiently perform better over the ANSGA-III, NSGA-II, and MOPSO, in terms of the computational resources with percentage decrease of 99.983%, 99.985%, and 99.446% respectively. Similarly, the ensemble learning model (XGBoost) was subjected to performance evaluation with other regression models (random forest, knn, and linear regression) using the mean absolute error performance metric tool. The outcomes

revealed that the proposed ensemble learning model efficiently performs better over the compared regression models with the mean absolute error of 1.855751. Finally, the proposed classification algorithms were compared with several existing classification algorithms. The existing algorithms include knn, random forest, logistic regression, xgboost, catboost, graph neural networks, and feed-forward neural networks. The outcome reveals that the proposed artificial neural network classification model efficiently performs along with the other existing algorithms. The accuracy score of the proposed learning vector quantization is 84% as opposed to the highest accuracy score (the catboost model), which is 89%, however, in terms of controller placement prediction, the proposed solution was able to predict similarly to the catboost model. They both predicted six controller placements accurately out of the seven actual placements. The prediction process time also demonstrates that the recommended classification technique outperforms other currently employed classification algorithms. In Conclusion, the performance of the optimization techniques (stochastic computational graph) proposed in this research, along with the existing techniques (ANSGA-III, NSGA-II, and MOPSO), was further subjected to statistical analysis (inferential statistics) to complement the result exhibited by the descriptive analysis in subsection 5.7.1. This helps this research to further check if there is truly a statistical difference in the performance of the proposed and existing optimization techniques for controller placement. The outcome of the inferential statistics, as exhibited in 5.7.1 further confirmed that the stochastic computational graph outperforms other existing optimization techniques. As a result of the outcomes revealed by the experiment in this study, the proposed stochastic computational graph model is therefore recommended over other existing techniques for the placement of the controller in SD-WAN. This research has proposed a strategy that enables SDN operators to choose the right number and placement of SDN controllers in order to get the best network performance. The suggested framework is intelligent, scalable, and computationally efficient when compared to the most recent optimization technique.

Chapter 6

Conclusion and Future Work

6.1 Introduction

In conclusion, this research effectively tackles the fundamental challenge of determining the optimal number and placement of controllers within extensive SDN architectures, wherein the optimization must account for a range of potentially conflicting objectives. Throughout this study, a comprehensive examination of established optimization algorithms, including NSGA-II, MOPSO, and ANSGA-III (known for scalability), has been conducted. This review has critically assessed their limitations, encompassing factors like scalability, computational efficiency, and notably, their lack of inherent intelligence when it comes to determining and predicting the optimal number of controllers required for deployment.

This study's paramount contribution lies in the introduction of two innovative optimization strategies: the Adapted Non-Dominated Sorting Genetic Algorithm III (ANSGA-III) and the Stochastic Computational Graph with Ensemble Learning method (SCGMEL). ANSGA-III's capabilities have been augmented with a repair operator-based mechanism, leading to improvements in convergence, diversity, and scalability compared to conventional methodologies. Simultaneously, SCGMEL capitalizes on stochastic gradient descent, a weighted sum approach, a computational graph model, and the XGBoost algorithm, resulting in a remarkable enhancement of computational efficiency and the ability to intelligently predict the optimal number and strategic placement of controllers. Furthermore, this research encapsulates the development of the Improved Switch Migration Decision Algorithm (ISMDA), a vital element of the holistic solution. ISMDA excels in facilitating balanced load distribution across controllers, showcasing its prowess in effectively managing the distribution of workloads among underutilized controllers through its migration efficiency strategy.

The research's efficacy is substantiated through rigorous experimentation across diverse datasets, including BtEurope and other datasets sourced from the Zoo topology. The results undeniably underscore the superiority of the Collaborative and Adaptive Optimization Learning-Based Framework over prevailing optimization algorithms, as evidenced by enhanced computational complexity management and improved prediction accuracy.

In essence, this study emerges as a pivotal contributor to the SDN and SD-WAN domain by furnishing a solution that is both scalable and intelligent, all while efficiently optimizing controller placement. The proposed framework empowers SDN operators to make well-informed decisions pertaining to the optimal number and strategic placement of controllers, ultimately translating to heightened network performance

and optimal resource utilization. The research's commitment to open-source code and comprehensive experimental methodologies further bolsters its credibility and applicability.

To encapsulate, this research transcends the boundaries of SD-WAN controller placement optimization, laying a sturdy foundation for future advancements within the dynamic landscape of software-defined networking. The novel strategies outlined herein hold the promise of revolutionizing network performance management, effectively addressing the evolving demands of modern network environments.

6.2 Summary of Contributions

This thesis' primary contributions are:

- In Chapter 3, ISDMA was offered as a solution to the problem of SDN load imbalance. The problem arises if there is an excessive flow in the inbound traffic load. Module 1—the load decision-making module; Module 2—switch selection; and Module 3—target controller selection all utilized by ISDMA. As soon as the load on a controller rises over a particular limit, the balancing module of the controller is triggered to ensure that there is an appropriate distribution of load across controllers. The balancing module searches through the controller set for the unloaded controller, taking into account the average load status and variation among the controllers. The presented approach strategically moves maximum-loaded switches from an overloaded controller to the most suitable controller in a group of unloaded controllers, freeing up as many clustered resources as possible. Additionally, this study developed a model to assess the migration's efficiency, revealing a connection between the extent of changes in migration cost and the effectiveness of load balancing. The developed ISMDA mechanism surpassed the existing controller adaptation and migration decision algorithms, as well as, DALB algorithms in terms of the frequency of migration spaces, response time, throughput, and packet loss, as determined by the results. According to the results of the simulation that was run under the situation of bulk data flow, the developed ISMDA is more effective than the two compared algorithms (CAMD and DALB). With the ISMDA method introduced, there is now a more balanced utilization of resources among the distributed controllers. Additionally, it increases controller throughput, shortens the time needed for migration in the network, and makes the control plane more resistant to packet loss and slow response times. In conclusion, ISMDA demonstrates higher efficiency compared to DALB and CAMD, resulting in approximately 1% and 6.4% lower average packet loss, respectively. Moreover, it enhances controller throughput by around 7.4% compared to CAMD and approximately 1.1% over DALB. ISMDA also outperforms DALB and CAMD with a decrease of 5.7% and 1%, respectively, in terms of controller response time. The improvement in the performance of the ISMDA solution is a result of the migration efficiency strategy developed in this research.

- In Chapter 4, an ANSGA-III algorithm was suggested as a solution to address the scalability concerns linked with the NSGA-II and MOPSO evolutionary techniques when dealing with more than three objectives. In order to attain the best controller placement in SD-WAN, this investigation integrated a repair-based operator into the established mechanical engineering-based NSGA-III method, leading to the formulation of the proposed ANSGA-III. To compare the performance of ANSGA-III, NSGA-II, and MOPSO evolutionary techniques, internet zoo topology datasets with

six objective functions were utilized. The assessment encompassed diverse metrics, such as the percentage coefficient of variation (PCV), parallel coordinate plots (PCP), percentage difference, and hypervolume indicator. According to the PCV analysis, ANSGA-III achieved a PCV aggregate of 167.4841%, while NSGA-II and the MOPSO algorithm attained a PCV aggregate of 121.229% and 78.3436%, respectively. The traceable hypervolume indicator value of ANSGA-III was the highest among the three algorithms, at 0.94876%. NSGA-II had a detectable hypervolume indicator value of 0.93646%, and the MOPSO algorithm had the lowest value at 0.89348%. The results showed that ANSGA-III outperformed NSGA-II and MOPSO algorithms regarding convergence and diversification when faced with scalability challenges. The experiment demonstrated that the adapted NSGA-III successfully addressed the scalability issues related to the CPP in SD-WAN. Consequently, the ANSGA-III method was preferred over the NSGA-II and MOPSO evolutionary techniques, depending on specific use cases.

- In Chapter 5, a stochastic computational graph with an ensemble learning model as well as a learning vector quantization classification algorithm were proposed to address the CPP in SD-WAN when there are multiple goals that are at odds with each other. The same stochastic computational graph model suggested in this study was used as a backbone along with the XGBoost model to obtain a suitable number of controllers with respect to several conflicting objectives and enhance the ability to predict the number of controllers suitable for SD-WAN deployment. The entire dataset (zoo topology) in this scenario was automated in order to learn the heuristics of the combinatorial tasks and help in the prediction of controllers. This has a positive consequence on the network's overall performance. The controller will not only be computed for the location, but the number of controllers to be deployed will also be taken into consideration. This will improve the network's overall performance and make sure that service providers do not spend too much money on installing controllers. Similarly, this thesis came up with a classification algorithm whose backbone runs on the LVQ for the prediction of controller placement. This prediction of placement improves the network's overall performance due to the reduction in time it takes to calculate controller locations as the network topology increases. SCGMEL exhibited exceptional computational efficiency, surpassing ANSGA-III, NSGA-II, and MOPSO by 99.983%, 99.985%, and 99.446% respectively. The XGBoost regression model performed significantly better in predicting the number of controllers with a mean absolute error of 1.855751 compared to 3.829268, 3.729883, and 1.883536 for KNN, linear regression, and random forest, respectively. The proposed LVQ-based classification method achieved a test accuracy of 84% and accurately predicted six of the seven controller locations.

6.3 Reflection on the Research Questions and Achievement of Objectives

- **How can an enhanced migration decision algorithm for controller placement be formulated to effectively tackle the complexities of SDN load balancing?**

  Ans: This study successfully formulated and implemented an improved switch migration decision algorithm within the SDN architecture. The algorithm was designed to optimize the distribution of workloads among controllers, contributing to efficient controller placement The addressed research question led to the development of an enhanced switch migration decision algorithm targeting load

balancing within the SDN context. Existing literature often focuses on scenarios where incoming traffic consists of mice flows with a low flow rate, leading to improved migration efficiency in the data plane. However, such approaches fall short when dealing with high incoming traffic loads. This thesis introduces novel techniques for improved switch migration, specifically designed to tackle network issues arising from substantial inbound data traffic. In this approach, when the load on the current controller exceeds a predefined threshold, the algorithm's balancing unit is invoked on each controller. This balancing unit ensures even distribution of the load across all controllers. By evaluating the variation in load relative to the average load of all controllers, the underloaded controller is identified. The introduced method, known as the Improved Switch Migration Decision Algorithm (ISMDA), efficiently reallocates heavily loaded switches from an overloaded controller to an underloaded one, optimizing the distribution of critical resources. A migration efficiency model was also established, demonstrating a trade-off between migration cost variability and load balancing. This innovative approach provides a practical solution to address the challenges of load balancing in SDN environments, particularly in scenarios with high inbound data traffic loads.

- **How can the optimization of SD-WAN controllers be achieved in the context of multiple conflicting objectives, surpassing the count of three?**

  Ans: This research has pioneered the development of a groundbreaking repair operator-based mechanism, seamlessly integrated into the ANSGA-III framework. The primary objective was to achieve optimal controller placement within SD-WAN environments

  The research question focusing on optimizing SD-WAN controllers in the presence of multiple conflicting objectives (more than three) is addressed through the development and integration of a repair operator-based mechanism into the Adaptive Non-Dominated Sorting Genetic Algorithm III (ANSGA-III). Traditional optimization techniques struggle to effectively manage scenarios where more than three objectives need simultaneous optimization. This study proposes a novel approach by introducing a repair operator-based mechanism into the engineering-based ANSGA-III, specifically designed to determine optimal controller placement. The research ensures that unfeasible solutions are discarded during the crossover and mutation processes, and prevents the creation of duplicate solutions that might appear as optimal outcomes. The repair operator mechanism replaces continuous optimization features in the existing NSGA-III with discrete optimization attributes. Furthermore, the proposed strategy enhances convergence and diversity across the Pareto Front by employing techniques such as normalization, association, reference points, and niching approaches. This holistic approach not only enhances the effectiveness of optimization but also enables the concurrent optimization of multiple competitive objectives during SD-WAN controller placement. The result is a robust, adaptable, and efficient optimization strategy that successfully addresses the challenge of optimizing SD-WAN controllers in the presence of several conflicting objectives.

- **How can machine learning methodologies be leveraged to facilitate the acquisition of heuristics for solving intricate combinatorial optimization problems, such as the placement of SD-WAN controllers?**

  Ans: This research has put forward a pioneering automated learning-based decision-making model to achieve optimal controller placement. This model draws upon three key components: a stochastic computational graph, an ensemble learning model, and learning vector quantization.

The research question concerning the application of machine learning to facilitate the learning of heuristics for combinatorial optimization problems, like SD-WAN controller placement, is addressed by proposing an automated learning-based decision-making model. This model leverages a combination of techniques, including a stochastic computational graph, an ensemble learning model, and learning vector quantization. A comprehensive analysis of the research landscape reveals that both exhaustive and meta-heuristic approaches come with significant computational overhead and lack the ability to learn the intricate heuristics required for solving combinatorial optimization tasks such as SD-WAN controller placement. In response to this challenge, this study introduces a novel solution that capitalizes on a stochastic computational graph, an ensemble learning model, and learning vector quantization.

This proposed solution serves a dual purpose: predicting the number and optimal placement of controllers within an SD-WAN topology and optimizing controller placement even amidst competing objectives. It effectively addresses the limitations posed by exhaustive and meta-heuristic approaches by incorporating a stochastic and dynamic computational graph. Moreover, the ensemble learning model is harnessed for predicting the optimal number of controllers, ensuring a multifaceted optimization of SD-WAN controller placement while accounting for various conflicting objectives. Furthermore, the research introduces a classification algorithm, built on the foundation of Learning Vector Quantization (LVQ), to predict controller placement. This algorithm enhances the capacity to determine optimal controller locations in a comprehensive manner.

In conclusion, this research delivers a sophisticated solution that harnesses machine learning's capabilities to unravel the complexities of combinatorial optimization problems in SD-WAN controller placement. By combining the power of a stochastic computational graph, an ensemble learning model, and learning vector quantization, this approach sets new benchmarks for predictive accuracy and optimization efficiency in the context of SD-WAN controller placement.

6.4 Future work

While this research has made significant contributions to optimizing Software-Defined Networking (SDN) controller placement and load balancing, there are several avenues for future research to further enhance the field. Some potential future work includes:

- Utilizing Graph Neural Networks (GNN) and Graph Attention Networks (GAT): In future research, exploring the use of GNN and GAT as optimization algorithms and classification algorithms could be highly beneficial. GNNs and GATs have shown promise in handling graph-structured data, making them suitable for solving complex problems like controller placement and load balancing in SD-WAN. Integrating these neural network architectures into the proposed Stochastic Computational Graph Model with Ensemble Learning (SCGMEL) and Learning Vector Quantization (LVQ) classification algorithm can lead to more intelligent and efficient solutions.

- Extending Controller Placement to Cloud Services, 5G, and IoT: To make the controller placement solution more versatile, future work can extend the research to manage cloud services, 5G cellular networks, and IoT devices. This would involve adapting the proposed algorithms to address the unique requirements and challenges of these domains, considering factors like varying traffic patterns, mobility, and resource constraints.

- Integration of SDN and Network Virtualization: Investigating the integration of SDN and network virtualization to support multiple virtual networks running on a single SD-WAN architecture can result in greater flexible and scalable network management. This approach allows network operators to efficiently utilize resources and adapt the network to different service requirements dynamically.
- Full Integration of SD-WAN with Artificial Intelligence (AI) and Machine Learning (ML): In the future, fully integrating SD-WAN technology with AI and ML can bring significant benefits to network automation, traffic routing optimization, and proactive network management. ML models can be employed to predict and avoid network outages before they occur, enhancing network reliability and performance.
- Enhancing Controller Placement Objectives: Expanding the controller placement objectives to include metrics like packet arrival and departure wait times, controller capacity, and power consumption can lead to more comprehensive optimization models. Considering queuing delay and power consumption in the optimization process can improve the overall network performance and resource efficiency.
- Learning Vector Quantization as Regression Technique: Investigating the use of Learning Vector Quantization (LVQ) as a regression technique instead of a classification algorithm can enhance prediction accuracy and flexibility. Adapting LVQ to perform regression tasks can provide more accurate predictions for the number of controllers needed, allowing for more fine-grained control over controller placement decisions.

By addressing these future research directions, the proposed controller placement and load-balancing solutions can be extended to tackle new challenges and adapt to the evolving landscape of SDN, cloud services, 5G, IoT, and artificial intelligence. The combination of advanced optimization techniques, flexible controller placement strategies, and integration with cutting-edge technologies can pave the way for intelligent, efficient, and resilient networks in the future.

Appendix A

Glossary

ANSGA-III	Adapted Non-dominated Sorting Genetic Algorithm III
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application Specific Integrated Circuit
BGP	Border Gateway Protocol
BGP LS	Border Gateway Protocol Link State
CAPEX	Capital Expenditure
CORD	Corporate office re-architected as a data center
CAMD	Controller Adaption and Migration Decision
CPP	Controller Placement Problem
CPU	Central Processing Unit
DALB	Distributed Algorithm and Load Balancing
DCN	Data Centre Networks
EA	Evolutionary Algorithm
GNN	Graph Neural Networks
GAT	Graph Attention Networks
ISMDA	Improved Switch Migration Decision Algorithm
I2RS	Interface to the Routing System
K-NN	K Nearest Neighbour
LAN	Local Area Networks
LISP	Locator ID Separation Protocol
LBR	Load Balancing Rate
LVQ	Learning Vector Quantisation
ML	Machine Learning
MCLVQ	Multi-Class Learning Vector Quantisation
MD-SAL	Model-Driven Software Engineering Layer
MDSE	Model-Driven Software Engineering Layer
MOPSO	Multi-Objective Particle Swarm Optimisation
NETCONF	Network Configuration
NSGA-II	Non-dominated Sorting Genetic Algorithm II
NSGA-III	Non-dominated Sorting Genetic Algorithm II
OF-CONFIG	OpenFlow Configuration

OVSDB	Open vSwitch Database Management
PCEP	Path Computational Element Protocol
POCO	Pareto Optimal resilient Controller
PSO	Particle Swarm Optimisation
RL	Reinforcement Learning
REST API	Representational State Transfer
SCGMEL	Stochastic Computational Graph Method and Ensemble Learning
SDN	Software Defined Networking
SD-WAN	Software-Defined Wide Area Network I
SN	Simulated Annealing
TLS	Transport Layer Security
XGBOOST	Extreme Gradient Boosting
WAN	Wide Area Networks

Appendix B

Code Repository for the Thesis

For more information about the code developed for this thesis click on the following link in Github :

- The shortened URL:
  https://tinyurl.com/2p95ad26

Bibliography

[1] J. Liu, S. Zhang, N. Kato, H. Ujikawa, and K. Suzuki, "Device-to-device communications for enhancing quality of experience in software defined multi-tier lte-a networks," *IEEE Network*, vol. 29, no. 4, pp. 46–52, 2015.

[2] Y.-D. Lin, P.-C. Lin, C.-H. Yeh, Y.-C. Wang, and Y.-C. Lai, "An extended sdn architecture for network function virtualization with a case study on intrusion prevention," *IEEE Network*, vol. 29, no. 3, pp. 48–53, 2015.

[3] J. Liu, Z. Jiang, N. Kato, O. Akashi, and A. Takahara, "Reliability evaluation for nfv deployment of future mobile broadband networks," *IEEE Wireless Communications*, vol. 23, no. 3, pp. 90–96, 2016.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for sdn," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, 2014.

[6] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, vol. 3, pp. 10–5555, 2010.

[7] O. Adekoya and A. Aneiba, "An adapted nondominated sorting genetic algorithm iii (nsga-iii) with repair-based operator for solving controller placement problem in software-defined wide area networks," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 888–901, 2022.

[8] V. Ahmadi and M. Khorramizadeh, "An adaptive heuristic for multi-objective controller placement in software-defined networks," *Computers & Electrical Engineering*, vol. 66, pp. 204–228, 2018.

[9] C. Gao, H. Wang, F. Zhu, L. Zhai, and S. Yi, "A particle swarm optimization algorithm for controller placement problem in software defined network," in *International Conference on Algorithms and Architectures for Parallel Processing*, pp. 44–54, Springer, 2015.

[10] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[11] V. Ahmadi, A. Jalili, S. M. Khorramizadeh, and M. Keshtgari, "A hybrid nsga-ii for solving multi-objective controller placement in sdn," in *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, pp. 663–669, IEEE, 2015.

[12] K. S. Sahoo and B. Sahoo, "Camd: a switch migration based load balancing framework for software defined networks," *IET Networks*, vol. 8, no. 4, pp. 264–271, 2019.

[13] J. Hollinghurst, A. Ganesh, and T. Baugé, "Controller placement methods analysis," in *2016 6th International Conference on Information Communication and Management (ICICM)*, pp. 239–244, IEEE, 2016.

[14] F. Bannour, S. Souihi, and A. Mellouk, "Scalability and reliability aware sdn controller placement strategies," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–4, IEEE, 2017.

[15] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "Poco-plc: Enabling dynamic pareto-optimal resilient controller placement in sdn networks," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 115–116, IEEE, 2014.

[16] L. Mamushiane, "Towards the development of an optimal sdn controller placement framework to expedite sdn deployment in emerging markets," Master's thesis, Faculty of Engineering and the Built Environment, 2019.

[17] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints," *IEEE transactions on evolutionary computation*, vol. 18, no. 4, pp. 577–601, 2013.

[18] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon; an elastic distributed sdn controller," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 17–27, IEEE, 2014.

[19] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, and M. Zhu, "A load balancing strategy of sdn controller based on distributed decision," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 851–856, IEEE, 2014.

[20] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

[21] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in sdn-based core networks," in *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*, pp. 1–9, IEEE, 2013.

[22] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications surveys & tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[23] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[24] M. Awais, M. Asif, M. B. Ahmad, T. Mahmood, and S. Munir, "Comparative analysis of traditional and software defined networks," in *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*, pp. 1–6, IEEE, 2021.

[25] P. Čisar, D. Erlenvajn, and S. Maravić Čisar, "Implementation of software-defined networks using open-source environment," *Tehnički vjesnik*, vol. 25, no. Supplement 1, pp. 222–230, 2018.

[26] N. Gupta, M. S. Maashi, S. Tanwar, S. Badotra, M. Aljebreen, and S. Bharany, "A comparative study of software defined networking controllers using mininet," *Electronics*, vol. 11, no. 17, p. 2715, 2022.

[27] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79–95, 2015.

[28] R. Wazirali, R. Ahmad, and S. Alhiyari, "Sdn-openflow topology discovery: An overview of performance issues," *Applied Sciences*, vol. 11, no. 15, p. 6999, 2021.

[29] R. Deb and S. Roy, "A comprehensive survey of vulnerability and information security in sdn," *Computer Networks*, vol. 206, p. 108802, 2022.

[30] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[31] S. Schaller and D. Hood, "Software defined networking architecture standardization," *Computer standards & interfaces*, vol. 54, pp. 197–202, 2017.

[32] G. Mine, J. Hai, L. Jin, and Z. Huiying, "A design of sd-wan-oriented wide area network access," in *2020 International Conference on Computer Communication and Network Security (CCNS)*, pp. 174–177, IEEE, 2020.

[33] P. Segeč, M. Moravčik, J. Uratmová, J. Papán, and O. Yeremenko, "Sd-wan-architecture, functions and benefits," in *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pp. 593–599, IEEE, 2020.

[34] S. Wallin and C. Wikström, "Automating network and service configuration using {NETCONF} and {YANG}," in *25th Large Installation System Administration Conference (LISA 11)*, 2011.

[35] B. Pfaff, B. Lantz, B. Heller, *et al.*, "Openflow switch specification, version 1.3. 0," *Open Networking Foundation*, pp. 39–46, 2012.

[36] T. Čejka and R. Krejčí, "Configuration of open vswitch using of-config," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 883–888, IEEE, 2016.

[37] A. Rodriguez-Natal, M. Portoles-Comeras, V. Ermagan, D. Lewis, D. Farinacci, F. Maino, and A. Cabellos-Aparicio, "Lisp: a southbound sdn protocol?," *IEEE Communications Magazine*, vol. 53, no. 7, pp. 201–207, 2015.

[38] D. Lopez, O. G. de Dios, Q. Wu, and D. Dhody, "Pceps: Usage of tls to provide a secure transport for the path computation element communication protocol (pcep)," tech. rep., 2017.

[39] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pp. 1–6, IEEE, 2014.

[40] A. Sgambelluri, F. Paolucci, F. Cugini, L. Valcarenghi, and P. Castoldi, "Generalized sdn control for access/metro/core integration in the framework of the interface to the routing system (i2rs)," in *2013 IEEE Globecom Workshops (GC Wkshps)*, pp. 1216–1220, IEEE, 2013.

[41] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit, "Sdn and openflow evolution: A standards perspective," *Computer*, vol. 47, no. 11, pp. 22–29, 2014.

[42] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020.

[43] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 918–953, 2016.

[44] L. Mamushiane and T. Shozi, "A qos-based evaluation of sdn controllers: Onos and opendaylight," in *2021 IST-Africa Conference (IST-Africa)*, pp. 1–10, IEEE, 2021.

[45] M. B. Dissanayake, A. Kumari, and U. Udunuwara, "Performance comparison of onosand odlcontrollers in," *J Res Technol Eng*, vol. 2, pp. 94–105, 2021.

[46] I. Šeremet and S. Čaušević, "An analysis of reconvergence delay when using bgp-ls/pcep as southbound protocols," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 415–420, IEEE, 2019.

[47] J. Kaur, "Sdn controller (open daylight) based implementation of pcep (path computation element protocol) for network path instantiation," 2021.

[48] G. Huang and H. Y. Youn, "Proactive eviction of flow entry for sdn based on hidden markov model," *Frontiers of Computer Science*, vol. 14, pp. 1–10, 2020.

[49] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *International conference on passive and active network measurement*, pp. 347–359, Springer, 2015.

[50] O. Blial, M. Ben Mamoun, R. Benaini, *et al.*, "An overview on sdn architectures with multiple controllers," *Journal of Computer Networks and Communications*, vol. 2016, 2016.

[51] M. He, A. M. Alba, E. Mansour, and W. Kellerer, "Evaluating the control and management traffic in openstack cloud with sdn," in *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR)*, pp. 1–6, IEEE, 2019.

[52] J.-S. Weng, J. Weng, Y. Zhang, W. Luo, and W. Lan, "Benbi: Scalable and dynamic access control on the northbound interface of sdn-based vanet," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 822–831, 2018.

[53] S. Askar and F. Keti, "Performance evaluation of different sdn controllers: A review," 2021.

[54] Y. Li, X. Guo, X. Pang, B. Peng, X. Li, and P. Zhang, "Performance analysis of floodlight and ryu sdn controllers under mininet simulator," in *2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pp. 85–90, IEEE, 2020.

[55] A. Azzouni, M. Nguyen, G. Pujolle, *et al.*, "Topology discovery performance evaluation of opendaylight and onos controllers," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pp. 285–291, IEEE, 2019.

[56] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "Sdn controllers: A comparative study," in *2016 18th mediterranean electrotechnical conference (MELECON)*, pp. 1–6, IEEE, 2016.

[57] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, 2016.

[58] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of opendaylight sdn controller," in *2014 20th IEEE international conference on parallel and distributed systems (ICPADS)*, pp. 671–676, IEEE, 2014.

[59] M. Zaher, "A comparative and analytical study for choosing the best suited sdn network operating system for cloud data center," *Annals of Emerging Technologies in Computing (AETiC)*, vol. 6, no. 1, 2022.

[60] S. Rowshanrad, V. Abdi, and M. Keshtgari, "Performance evaluation of sdn controllers: Floodlight and opendaylight," *IIUM Engineering Journal*, vol. 17, no. 2, pp. 47–57, 2016.

[61] M. Islam, N. Islam, M. Refat, *et al.*, "Node to node performance evaluation through ryu sdn controller," *Wireless Personal Communications*, vol. 112, no. 1, pp. 555–570, 2020.

[62] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in sdn," *International Journal of Network Management*, vol. 28, no. 3, p. e2018, 2018.

[63] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101–118, 2018.

[64] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 344–355, 2017.

[65] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pp. 18–25, IEEE, 2013.

[66] M. C. Penna, E. Jamhour, and M. L. Miguel, "A clustered sdn architecture for large scale wson," in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pp. 374–381, IEEE, 2014.

[67] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *IEEE transactions on network and service management*, vol. 12, no. 1, pp. 18–33, 2015.

[68] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A k-means-based network partition algorithm for controller placement in software defined network," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2016.

[69] M. Tanha, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE, 2016.

[70] L. Mamushiane, J. Mwangama, and A. A. Lysko, "Controller placement optimization for software defined wide area networks (sdwan)," 2021.

[71] K. S. Sahoo, S. Sahoo, A. Sarkar, B. Sahoo, and R. Dash, "On the placement of controllers for designing a wide area software defined networks," in *TENCON 2017-2017 IEEE Region 10 Conference*, pp. 3123–3128, IEEE, 2017.

[72] D. Zeng, C. Teng, L. Gu, H. Yao, and Q. Liang, "Flow setup time aware minimum cost switch-controller association in software-defined networks," in *2015 11th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QSHINE)*, pp. 259–264, IEEE, 2015.

[73] M. He, A. Basta, A. Blenk, and W. Kellerer, "Modeling flow setup time for controller placement in sdn: Evaluation for dynamic flows," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2017.

[74] K. Sood and Y. Xiang, "The controller placement problem or the controller selection problem?," *Journal of communications and information networks*, vol. 2, no. 3, pp. 1–9, 2017.

[75] J. Nagano and N. Shinomiya, "Efficient information sharing among distributed controllers of openflow network with bi-connectivity," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, pp. 320–324, IEEE, 2015.

[76] T. Zhang, A. Bianco, and P. Giaccone, "The role of inter-controller traffic in sdn controllers placement," in *2016 IEEE conference on network function virtualization and software defined networks (NFV-SDN)*, pp. 87–92, IEEE, 2016.

[77] L. Han, Z. Li, W. Liu, K. Dai, and W. Qu, "Minimum control latency of sdn controller placement," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 2175–2180, IEEE, 2016.

[78] L. Zhu, R. Chai, and Q. Chen, "Control plane delay minimization based sdn controller placement scheme," in *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, IEEE, 2017.

[79] T. Li, Z. Gu, X. Lin, S. Li, and Q. Tan, "Approximation algorithms for controller placement problems in software defined networks," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 250–257, IEEE, 2018.

[80] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*, pp. 1–6, IEEE, 2011.

[81] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The sdn controller placement problem for wan," in *2014 IEEE/CIC International Conference on Communications in China (ICCC)*, pp. 220–224, IEEE, 2014.

[82] P. Xiao, Z.-y. Li, S. Guo, H. Qi, W.-y. Qu, and H.-s. Yu, "Ak self-adaptive sdn controller placement for wide area networks," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 7, pp. 620–633, 2016.

[83] H. Aoki and N. Shinomiya, "Controller placement problem to enhance performance in multi-domain sdn networks," in *Proc. ICN*, p. 120, 2016.

[84] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed sdn control layer," in *2014 IFIP Networking Conference*, pp. 1–9, IEEE, 2014.

[85] Y. Jiménez, C. Cervelló-Pastor, and A. J. Garcia, "Defining a network management architecture," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–3, IEEE, 2013.

[86] M. Guo and P. Bhattacharya, "Controller placement for improving resilience of software-defined networks," in *2013 Fourth International Conference on Networking and Distributed Computing*, pp. 23–27, IEEE, 2013.

[87] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Communications*, vol. 11, no. 2, pp. 38–54, 2014.

[88] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: An enhanced controller placement strategy for improving sdn survivability," in *2014 IEEE Global Communications Conference*, pp. 1909–1915, IEEE, 2014.

[89] N. Perrot and T. Reynaud, "Optimal placement of controllers in a resilient sdn architecture," in *2016 12th International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 145–151, IEEE, 2016.

[90] B. P. R. Killi and S. V. Rao, "Controller placement with planning for failures in software defined networks," in *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pp. 1–6, IEEE, 2016.

[91] B. P. R. Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 514–527, 2017.

[92] E. Calle, D. Martínez, M. Mycek, and M. Pióro, "Resilient backup controller placement in distributed sdn under critical targeted attacks," *International Journal of Critical Infrastructure Protection*, vol. 33, p. 100422, 2021.

[93] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha, "Optimal controller placement in software defined networks (sdn) using a non-zero-sum game," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pp. 1–6, IEEE, 2014.

[94] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.

[95] H. Aoki, J. Nagano, and N. Shinomiya, "Network partitioning problem to reduce shared information in openflow networks with multiple controllers," in *Proc. ICN*, p. 262, 2015.

[96] J.-M. Sanner, Y. Hadjadj-Aoufi, M. Ouzzif, and G. Rubino, "Hierarchical clustering for an efficient controllers' placement in software defined networks," in *2016 Global Information Infrastructure and Networking Symposium (GIIS)*, pp. 1–7, IEEE, 2016.

[97] L. Yao, P. Hong, W. Zhang, J. Li, and D. Ni, "Controller placement and flow based dynamic management problem towards sdn," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, pp. 363–368, IEEE, 2015.

[98] S. Hegde, R. Ajayghosh, S. G. Koolagudi, and S. Bhattacharya, "Dynamic controller placement in edge-core software defined networks," in *TENCON 2017-2017 IEEE Region 10 Conference*, pp. 3153–3158, IEEE, 2017.

[99] O. Adekoya, A. Aneiba, and M. Patwary, "An improved switch migration decision algorithm for sdn load balancing," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 1602–1613, 2020.

[100] A. Ruiz-Rivera, K.-W. Chin, and S. Soh, "Greco: An energy aware controller association algorithm for software defined networks," *IEEE communications letters*, vol. 19, no. 4, pp. 541–544, 2015.

[101] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE communications letters*, vol. 19, no. 1, pp. 30–33, 2014.

[102] A. Sallahi and M. St-Hilaire, "Expansion model for the controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 21, no. 2, pp. 274–277, 2016.

[103] S. Auroux, M. Draxler, A. Morelli, *et al.*, "Crowd dynamic network reconfiguration in wireless densenets," in *Proc. European Conference on Networks and Communications (EuCNC 2014)*, 2014.

[104] S. Auroux, M. Dräxler, A. Morelli, and V. Mancuso, "Dynamic network reconfiguration in wireless densenets with the crowd sdn architecture," in *2015 European Conference on Networks and Communications (EuCNC)*, pp. 144–148, IEEE, 2015.

[105] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman, "Learning combinatorial optimization on graphs: A survey with applications to networking," *IEEE Access*, vol. 8, pp. 120388–120416, 2020.

[106] M. Lombardi and M. Milano, "Boosting combinatorial problem modeling with machine learning," *arXiv preprint arXiv:1807.05517*, 2018.

[107] A. Jalili, V. Ahmadi, M. Keshtgari, and M. Kazemi, "Controller placement in software-defined wan using multi objective genetic algorithm," in *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, pp. 656–662, IEEE, 2015.

[108] L. Liao, V. Leung, Z. Li, and H.-C. Chao, "Genetic algorithms with variant particle swarm optimization based mutation for generic controller placement in software-defined networks," *Symmetry*, vol. 13, no. 7, p. 1133, 2021.

[109] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[110] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[111] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[112] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.

[113] S. Lange, S. Gebert, J. Spoerhase, P. Rygielski, T. Zinner, S. Kounev, and P. Tran-Gia, "Specialized heuristics for the controller placement problem in large scale sdn networks," in *2015 27th International Teletraffic Congress*, pp. 210–218, IEEE, 2015.

[114] H. S. Naning, R. Munadi, and M. Z. Effendy, "Sdn controller placement design: For large scale production network," in *2016 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pp. 74–79, IEEE, 2016.

[115] A. Xu, S. Sun, Z. Wang, X. Wang, and L. Han, "Multi-controller load balancing mechanism based on improved genetic algorithm," in *2022 International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–8, IEEE, 2022.

[116] N. S. Radam, S. T. F. Al-Janabi, and K. S. Jasim, "Multi-controllers placement optimization in sdn by the hybrid hsa-pso algorithm," *Computers*, vol. 11, no. 7, p. 111, 2022.

[117] L. Alouache, S. Yassa, and A. Ahfir, "A multi-objective optimization approach for sdvn controllers placement problem," in *2022 13th International Conference on Network of the Future (NoF)*, pp. 1–9, IEEE, 2022.

[118] P. Aravind, G. S. Varma, and P. P. Reddy, "Simulated annealing based optimal controller placement in software defined networks with capacity constraint and failure awareness," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 8, pp. 5721–5733, 2022.

[119] J. Hemagowri and P. T. Selvan, "A hybrid evolutionary algorithm of optimized controller placement in sdn environment," *Computer Assisted Methods in Engineering and Science*, 2023.

[120] E. Borcoci, R. Badea, S. G. Obreja, and M. Vochin, "On multi-controller placement optimization in software defined networking-based wans," in *ICN*, vol. 2015, p. 273, 2015.

[121] H. Bo, W. Youke, W. Chuan'an, and W. Ying, "The controller placement problem for software-defined networks," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 2435–2439, IEEE, 2016.

[122] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Computer Networks*, vol. 112, pp. 24–35, 2017.

[123] B. Zhang, X. Wang, and M. Huang, "Multi-objective optimization controller placement problem in internet-oriented software defined network," *Computer Communications*, vol. 123, pp. 24–35, 2018.

[124] H. Kuang, Y. Qiu, R. Li, and X. Liu, "A hierarchical k-means algorithm for controller placement in sdn-based wan architecture," in *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pp. 263–267, IEEE, 2018.

[125] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Capacity-aware and delay-guaranteed resilient controller placement for software-defined wans," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 991–1005, 2018.

[126] S. Mohanty, P. Priyadarshini, S. Sahoo, B. Sahoo, and S. Sethi, "Metaheuristic techniques for controller placement in software-defined networks," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*, pp. 897–902, IEEE, 2019.

[127] B. Zhang, X. Wang, L. Ma, and M. Huang, "Optimal controller placement problem in internet-oriented software defined network," in *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pp. 481–488, IEEE, 2016.

[128] S. Liu, H. Wang, S. Yi, and F. Zhu, "Ncpso: a solution of the controller placement problem in software defined networks," in *International conference on algorithms and architectures for parallel processing*, pp. 213–225, Springer, 2015.

[129] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[130] A. Jalili, M. Keshtgari, and R. Akbari, "Optimal controller placement in large scale software defined networks based on modified nsga-ii," *Applied Intelligence*, vol. 48, no. 9, pp. 2809–2823, 2018.

[131] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in sdn," *IEEE communications surveys & tutorials*, vol. 22, no. 1, pp. 472–503, 2019.

[132] A. Chehouri, R. Younes, J. Perron, and A. Ilinca, "A constraint-handling technique for genetic algorithms using a violation factor," *arXiv preprint arXiv:1610.00976*, 2016.

[133] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on evolutionary computation*, vol. 7, no. 2, pp. 117–132, 2003.

[134] A. P. Wierzbicki, "The use of reference objectives in multiobjective optimization," in *Multiple criteria decision making theory and application*, pp. 468–486, Springer, 1980.

[135] A. Ksentini, M. Bagaa, and T. Taleb, "On using sdn in 5g: The controller placement problem," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2016.

[136] K. S. Sahoo, B. Sahoo, R. Dash, and N. Jena, "Optimal controller selection in software defined network using a greedy-sa algorithm," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 2342–2346, IEEE, 2016.

[137] K. S. Sahoo, B. Sahoo, R. Dash, and M. Tiwary, "Solving multi-controller placement problem in software defined network," in *2016 International Conference on Information Technology (ICIT)*, pp. 188–192, IEEE, 2016.

[138] K. S. Sahoo, A. Sarkar, S. K. Mishra, B. Sahoo, D. Puthal, M. S. Obaidat, and B. Sadun, "Metaheuristic solutions for solving controller placement problem in sdn-based wan architecture," in *ICETE 2017-Proceedings of the 14th International Joint Conference on e-Business and Telecommunications*, 2017.

[139] F.-M. De Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, and C. Gagné, "Deap: A python framework for evolutionary algorithms," in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pp. 85–92, 2012.

[140] H. Seada and K. Deb, "U-nsga-iii: A unified evolutionary algorithm for single, multiple, and many-objective optimization," *COIN report*, vol. 2014022, 2014.

[141] M. A. Bagha, K. Majidzadeh, M. Masdari, and Y. Farhang, "Improving delay in sdns by metaheuristic controller placement," *International Journal of Industrial Electronics Control & Optimization*, vol. 5, no. 3, 2022.

[142] A. Sapkota, B. B. R. Dawadi, C. S. R. Joshi, *et al.*, "Multi-controller placement optimization using naked mole-rat algorithm over software-defined networking environment," *Journal of Computer Networks and Communications*, vol. 2022, 2022.

[143] V. S. Thalapala, A. Mohan, and K. Guravaiah, "Woaccpp: Wisdom of artificial crowds for controller placement problem with latency and reliability in sdn-wan," 2022.

[144] M. M. Kazemian and M. Mirabi, "Controller placement in software defined networks using multi-objective antlion algorithm," *The Journal of Supercomputing*, pp. 1–24, 2022.

[145] A. A. Qaffas, S. Kamal, F. Sayeed, P. Dutta, S. Joshi, and I. Alhassan, "Adaptive population-based multi-objective optimization in sdn controllers for cost optimization," *Physical Communication*, vol. 58, p. 102006, 2023.

[146] M. Latah and L. Toker, "Artificial intelligence enabled software-defined networking: a comprehensive overview," *IET networks*, vol. 8, no. 2, pp. 79–99, 2019.

[147] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[148] M. Negnevitsky, *Artificial intelligence: a guide to intelligent systems*. Pearson education, 2005.

[149] S. Faezi and A. Shirmarz, "A comprehensive survey on machine learning using in software defined networks (sdn)," *Human-Centric Intelligent Systems*, pp. 1–32, 2023.

[150] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco, "A survey on machine learning techniques for routing optimization in sdn," *IEEE Access*, vol. 9, pp. 104582–104611, 2021.

[151] A. Lakhan, M. A. Mohammed, O. I. Obaid, C. Chakraborty, K. H. Abdulkareem, and S. Kadry, "Efficient deep-reinforcement learning aware resource allocation in sdn-enabled fog paradigm," *Automated Software Engineering*, vol. 29, pp. 1–25, 2022.

[152] Y. Tian, Y. Zhang, and H. Zhang, "Recent advances in stochastic gradient descent in deep learning," *Mathematics*, vol. 11, no. 3, p. 682, 2023.

[153] F. Huang and S. Gao, "Gradient descent ascent for minimax problems on riemannian manifolds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[154] N. Ketkar, J. Moolayil, N. Ketkar, and J. Moolayil, "Automatic differentiation in deep learning," *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, pp. 133–145, 2021.

[155] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for optimization*. Mit Press, 2019.

[156] Y. Wu, S. Zhou, Y. Wei, and S. Leng, "Deep reinforcement learning for controller placement in software defined network," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1254–1259, IEEE, 2020.

[157] C. Chen, F. Xue, Z. Lu, Z. Tang, C. Li, *et al.*, "Rlmr: Reinforcement learning based multipath routing for sdn," *Wireless Communications and Mobile Computing*, vol. 2022, 2022.

[158] M. Xiang, M. Chen, D. Wang, and Z. Luo, "Deep reinforcement learning-based load balancing strategy for multiple controllers in sdn," *e-Prime-Advances in Electrical Engineering, Electronics and Energy*, vol. 2, p. 100038, 2022.

[159] A. Yazdinejad, E. Rabieinejad, A. Dehghantanha, R. M. Parizi, and G. Srivastava, "A machine learning-based sdn controller framework for drone management," in *2021 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, IEEE, 2021.

[160] J. Chen, W. Xiao, X. Li, Y. Zheng, X. Huang, D. Huang, and M. Wang, "A routing optimization method for software-defined optical transport networks based on ensembles and reinforcement learning," *Sensors*, vol. 22, no. 21, p. 8139, 2022.

[161] S. Kukkonen and K. Deb, "Improved pruning of non-dominated solutions based on crowding distance for bi-objective optimization problems," in *2006 IEEE International Conference on Evolutionary Computation*, pp. 1179–1186, IEEE, 2006.

[162] C. Tipantuña and X. Hesselbach, "Nfv/sdn enabled architecture for efficient adaptive management of renewable and non-renewable energy," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 357–380, 2020.

[163] C. Qi, J. Wu, G. Cheng, J. Ai, and S. Zhao, "An aware-scheduling security architecture with priority-equal multi-controller for sdn," *China Communications*, vol. 14, no. 9, pp. 144–154, 2017.

[164] G. Cheng, H. Chen, Z. Wang, and S. Chen, "Dha: Distributed decisions on the switch migration toward a scalable sdn control plane," in *2015 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, IEEE, 2015.

[165] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, "Star: Preventing flow-table overflow in software-defined networks," *Computer Networks*, vol. 125, pp. 15–25, 2017.

[166] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 7–12, 2013.

[167] C. Liang, R. Kawashima, and H. Matsuo, "Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers," in *2014 Second International Symposium on Computing and Networking*, pp. 171–177, IEEE, 2014.

[168] W. H. F. Aly and A. M. A. Al-anazi, "Enhanced controller fault tolerant (ecft) model for software defined networking," in *2018 Fifth International Conference on Software Defined Systems (SDS)*, pp. 217–222, IEEE, 2018.

[169] B. L. Nelson, J. Swann, D. Goldsman, and W. Song, "Simple procedures for selecting the best simulated system when the number of alternatives is large," *Operations Research*, vol. 49, no. 6, pp. 950–963, 2001.

[170] M. Miyagi, K. Ohkubo, M. Kataoka, and S. Yoshizawa, "Performance prediction method for web-access response time distribution using formula," in *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No. 04CH37507)*, vol. 1, pp. 905–906, IEEE, 2004.

[171] Y. Yuan, H. Xu, and B. Wang, "An improved nsga-iii procedure for evolutionary many-objective optimization," in *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pp. 661–668, 2014.

[172] G. G. Yen and H. Lu, "Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 253–274, 2003.

[173] I. Das and J. E. Dennis, "Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems," *SIAM journal on optimization*, vol. 8, no. 3, pp. 631–657, 1998.

[174] M. Marcisz, "Practical application of coefficient of variation," in *Proceedings of the 13th international congress on energy and mineral resources (CIERM 2013). Santander: National Association of Spanish Mining Engineers*, pp. 202–208, 2013.

[175] H. Abdi, "Coefficient of variation," *Encyclopedia of research design*, vol. 1, pp. 169–171, 2010.

[176] A. K. Thukral, R. Bhardwaj, V. Kumar, and A. Sharma, "New indices regarding the dominance and diversity of communities, derived from sample variance and standard deviation," *Heliyon*, vol. 5, no. 10, p. e02606, 2019.

[177] D. A. Agunbiade, S. O. Folorunso, K.-K. A. Abdullah, and P. I. Ogunyinka, "Two-phase sampling for stratification: Application to software industry," *Annals. Computer Science Series*, vol. 15, no. 2, 2017.

[178] J. Mwaura, A. P. Engelbrecht, and F. V. Nepomuceno, "Diversity measures for niching algorithms," *Algorithms*, vol. 14, no. 2, p. 36, 2021.

[179] T. J. Cole and D. G. Altman, "Statistics notes: What is a percentage difference?," *Bmj*, vol. 358, 2017.

[180] M. Li, L. Zhen, and X. Yao, "How to read many-objective solution sets in parallel coordinates [educational forum]," *IEEE Computational Intelligence Magazine*, vol. 12, no. 4, pp. 88–100, 2017.

[181] H. Ji and C. Dai, "A simplified hypervolume-based evolutionary algorithm for many-objective optimization," *Complexity*, vol. 2020, 2020.

[182] K. Shang, H. Ishibuchi, L. He, and L. M. Pang, "A survey on the hypervolume indicator in evolutionary multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 1, pp. 1–20, 2020.

[183] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020.

[184] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[185] Y. Tian, H. Wang, X. Zhang, and Y. Jin, "Effectiveness and efficiency of non-dominated sorting for evolutionary multi-and many-objective optimization," *Complex & Intelligent Systems*, vol. 3, no. 4, pp. 247–263, 2017.

[186] T. Wagner, N. Beume, and B. Naujoks, "Pareto-, aggregation-, and indicator-based methods in many-objective optimization," in *International conference on evolutionary multi-criterion optimization*, pp. 742–756, Springer, 2007.

[187] D. Brockhoff, T. Friedrich, and F. Neumann, "Analyzing hypervolume indicator based algorithms," in *International Conference on Parallel Problem Solving from Nature*, pp. 651–660, Springer, 2008.

[188] Z. Li, X. Wang, S. Ruan, Z. Li, C. Shen, and Y. Zeng, "A modified hypervolume based expected improvement for multi-objective efficient global optimization method," *Structural and Multidisciplinary Optimization*, vol. 58, no. 5, pp. 1961–1979, 2018.

[189] E. Yazan and M. F. Talu, "Comparison of the stochastic gradient descent based optimization techniques," in *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pp. 1–5, IEEE, 2017.

[190] J. Chee and P. Toulis, "Convergence diagnostics for stochastic gradient descent with constant learning rate," in *International Conference on Artificial Intelligence and Statistics*, pp. 1476–1485, PMLR, 2018.

[191] Y. Liu, Y. Gao, and W. Yin, "An improved analysis of stochastic gradient descent with momentum," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18261–18271, 2020.

[192] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?," *arXiv preprint arXiv:1908.01878*, 2019.

[193] M. Looks, M. Herreshoff, D. Hutchins, and P. Norvig, "Deep learning with dynamic computation graphs," *arXiv preprint arXiv:1702.02181*, 2017.

[194] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[195] M. Schaarschmidt, S. Mika, K. Fricke, and E. Yoneki, "Rlgraph: Modular computation graphs for deep reinforcement learning," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 65–80, 2019.

[196] A. Brunel, D. Mazza, and M. Pagani, "Backpropagation in the simply typed lambda-calculus with linear negation," *Proceedings of the ACM on Programming Languages*, vol. 4, no. POPL, pp. 1–27, 2019.

[197] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[198] J. Schulman, N. Heess, T. Weber, and P. Abbeel, "Gradient estimation using stochastic computation graphs," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[199] A. Bou and G. De Fabritiis, "Pytorchrl: Modular and distributed reinforcement learning in pytorch," *arXiv preprint arXiv:2007.02622*, 2020.

[200] B. Pan, "Application of xgboost algorithm in hourly pm2. 5 concentration prediction," in *IOP conference series: earth and environmental science*, vol. 113, p. 012127, IOP publishing, 2018.

[201] R. Gayathri, S. U. Rani, L. Čepová, M. Rajesh, and K. Kalita, "A comparative analysis of machine learning models in prediction of mortar compressive strength," *Processes*, vol. 10, no. 7, p. 1387, 2022.

[202] M. C. Korkmaz, C. Chesneau, and Z. S. Korkmaz, "The unit folded normal distribution: A new unit probability distribution with the estimation procedures, quantile regression modeling and educational attainment applications," *Journal of Reliability and Statistical Studies*, pp. 261–298, 2022.

[203] C. M. Stein, "Estimation of the mean of a multivariate normal distribution," *The annals of Statistics*, pp. 1135–1151, 1981.

[204] Z. Yi, Y.-H. Chen, Y. Yin, K. Cheng, Y. Wang, D. Nguyen, T. Pham, and E. Kim, "Brief research report: A comparison of robust tests for homogeneity of variance in factorial anova," *The Journal of Experimental Education*, vol. 90, no. 2, pp. 505–520, 2022.

[205] M. Kostetckaia and M. Hametner, "How sustainable development goals interlinkages influence european union countries' progress towards the 2030 agenda," *Sustainable Development*, vol. 30, no. 5, pp. 916–926, 2022.

[206] J. Jamco and A. M. Balami, "Analisis kruskal-wallis untuk mengetahui konsentrasi belajar mahasiswa berdasarkan bidang minat program studi statistika fmipa unpatti," *PARAMETER: Jurnal Matematika, Statistika dan Terapannya*, vol. 1, no. 1, pp. 39–44, 2022.

[207] P. E. McKight and J. Najab, "Kruskal-wallis test," *The corsini encyclopedia of psychology*, pp. 1–1, 2010.

[208] J. Liu, S. Ma, W. Xu, and L. Zhu, "A generalized wilcoxon–mann–whitney type test for multivariate data through pairwise distance," *Journal of Multivariate Analysis*, vol. 190, p. 104946, 2022.

[209] D. D. Boos and S. Duan, "Pairwise comparisons using ranks in the one-way model," *The American Statistician*, vol. 75, no. 4, pp. 414–423, 2021.

[210] D. Nova and P. A. Estévez, "A review of learning vector quantization classifiers," *Neural Computing and Applications*, vol. 25, no. 3, pp. 511–524, 2014.

[211] D. Somasundaram, F. Zhang, S. Wang, H. Ye, Z. Zhang, and B. Zhang, "Learning vector quantization neural network for surface water extraction from landsat oli images," *Journal of Applied Remote Sensing*, vol. 14, no. 3, p. 032605, 2020.

[212] J. Ravichandran, M. Kaden, and T. Villmann, "Variants of recurrent learning vector quantization," *Neurocomputing*, vol. 502, pp. 27–36, 2022.