Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Dynamic routing optimization in software-defined networking based on a metaheuristic algorithm

Junyan Chen[1,2], Wei Xiao[1], Hongmei Zhang[2*], Jiacheng Zuo[3] and Xinmei Li[1]

Abstract

Optimizing resource allocation and routing to satisfy service needs is paramount in large-scale networks. Software-defined networking (SDN) is a new network paradigm that decouples forwarding and control, enabling dynamic management and configuration through programming, which provides the possibility for deploying intelligent control algorithms (such as deep reinforcement learning algorithms) to solve network routing optimization problems in the network. Although these intelligent-based network routing optimization schemes can capture network state characteristics, they are prone to falling into local optima, resulting in poor convergence performance. In order to address this issue, this paper proposes an African Vulture Routing Optimization (AVRO) algorithm for achieving SDN routing optimization. AVRO is based on the African Vulture Optimization Algorithm (AVOA), a population-based metaheuristic intelligent optimization algorithm with global optimization ability and fast convergence speed advantages. First, we improve the population initialization method of the AVOA algorithm according to the characteristics of the network routing problem to enhance the algorithm's perception capability towards network topology. Subsequently, we add an optimization phase to strengthen the development of the AVOA algorithm and achieve stable convergence effects. Finally, we model the network environment, define the network optimization objective, and perform comparative experiments with the baseline algorithms. The experimental results demonstrate that the routing algorithm has better network awareness, with a performance improvement of 16.9% compared to deep reinforcement learning algorithms and 71.8% compared to traditional routing schemes.

Introduction

With the growth of network scale, allocating network resources has become increasingly essential. Due to the tight coupling between the control plane and data plane in a traditional network architecture, routing algorithms cannot collect the state information of the network from a global perspective and plan forwarding paths accordingly. This may result in low network utilization and severe congestion in large-scale environments. Software-defined networking (SDN) [1] introduces a new approach that utilizes a programmable control plane to determine how different data flows are forwarded. The network's control logic is then transmitted from delivering devices such as routers and switches to software controllers, thereby decoupling the data plane from the control plane and simplifying network management [2]. As shown in Fig. 1, the control plane and data plane components connect through southbound application programming interfaces (APIs), such as OpenFlow [3]. In contrast, network policies or applications (such as routers, load balancers, and firewalls) can be implemented on the control

*Correspondence:
Hongmei Zhang
hmzhang@guet.edu.cn
[1] School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China
[2] School of Information and Communication, Guilin University of Electronic Technology, Guilin 541004, China
[3] School of Computer Science and Technology, Soochow University, Suzhou 215031, China

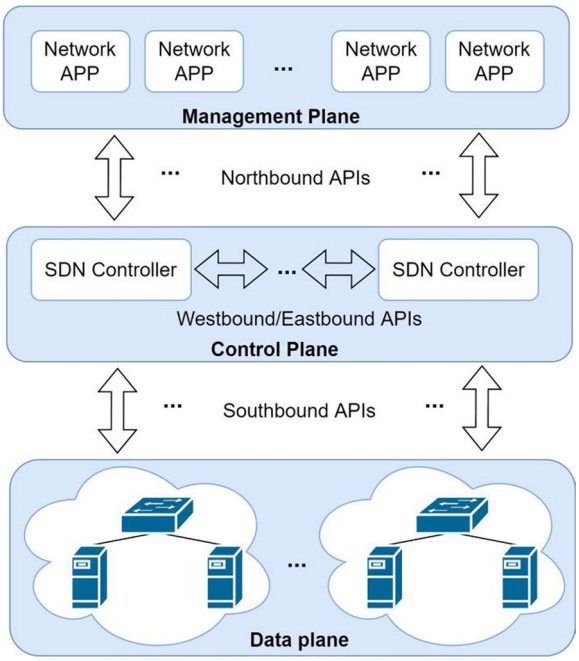

Fig. 1 SDN Architecture diagram

or management planes and interact with the controller through northbound APIs (such as RESTful API [4]).

In SDN, traditional routing algorithms, such as open shortest path first (OSPF) [5], cannot quickly adapt to dynamic traffic characteristics [6]. This limitation hinders the effective allocation of network resources and gradually becomes a bottleneck. To overcome this bottleneck, scholars have proposed using deep learning (DL) to solve routing problems [7, 8]. Existing DL methods such as graph neural network (GNN) [9] and long short-term memory (LSTM) [10] have achieved specific results in extracting network features and predicting network states. Training DL algorithm models does not require complex assumptions and modelling of network environments and can obtain optimized routing decisions through precise inference based on input data. This makes data-driven DL routing methods better adapted to different network application scenarios and routing optimization goals than traditional routing methods [11]. However, existing deep learning-based intelligent routing algorithms cannot guarantee their security and robustness in complex and changing network environments and require high deployment costs. In addition, due to the offline training nature of deep learning, there exists a reality gap that cannot display the same exceptional performance achieved during training in practice.

Using reinforcement learning [12] algorithms to make routing decisions is a recent research hotspot. Q-learning [13] is a classic form of reinforcement learning that stores state-action representations in Q-tables [6, 14]. However, as networks become more complex, the corresponding state-action space increases, and Q-tables may become very large. The algorithm requires a long learning time and can lead to high memory usage. Deep reinforcement learning (DRL) [15] uses deep neural networks to approximate state-action tables to overcome the scalability issues of Q-tables. Recent research has applied DRL methods to complex problems in the field of communication networks [16–19]. However, DRL-based intelligent routing algorithms lack vigorous scalability and sufficient robustness. They focus more on the convergence and accuracy of the algorithm, and training and deployment solutions in practical scenarios need to be improved.

In general, due to the current limitations of routing devices' computing power, deploying DL and DRL-based routing algorithms in large-scale networks necessitates further enhancement in terms of algorithm robustness and generalization. In contrast, optimizing SDN routing with heuristic algorithms may be more practical. Heuristic algorithms are problem-oriented algorithms that may not guarantee convergence to nearly optimal solutions but can obtain competitive solutions within a reasonable time frame. As an extension of heuristics, metaheuristics provide a problem-independent algorithm framework that can iteratively improve the solution quality of a given fitness function to find nearly optimal solutions [20]. The development of metaheuristic algorithms has made new progress in solving optimization problems. The African vulture optimization algorithm (AVOA) [21] is a bioinspired metaheuristic algorithm with global search ability and fast convergence speed. Inspired by this algorithm, this paper proposes an African vulture routing optimization (AVRO) algorithm for network routing optimization, which has stable convergence performance and strong global optimization capability. To achieve the network optimization goal of load balancing, we add link centrality measurement to population initialization to enhance AVRO's topological awareness. At the same time, we set additional optimization stages for the algorithm to approximate the leader vulture, thus strengthening the algorithm's development and improving convergence stability. The main contributions of this paper are as follows:

(1) A mathematical model is constructed for the network link load balancing problem, and optimization objectives are developed for the SDN routing algorithm.
(2) A routing optimization scheme, AVRO, is designed based on a metaheuristic algorithm. AVRO adds link betweenness measurement in population initialization to enhance the topological awareness of the algorithm. Additionally, it adds an optimization stage based on development and exploration to

strengthen the algorithm's performance and improve the model's convergence stability.

(3) The AVRO model is compared with traditional routing algorithms, deep learning-based routing algorithms, and classic reinforcement learning-based routing algorithms in three different network topologies and two different traffic intensities under simulation environments, demonstrating the superior load-balancing performance of the algorithm proposed in this paper.

The remainder of this paper is structured as follows: Sect. " Related work" presents related work on SDN routing problems; Sect. " Mathematical modelling for network optimization" describes the network model and optimization objectives; Sect. " Design of routing optimization algorithm based on AVRO" introduces the workflow of the routing algorithm based on AVRO; Sect. " Experimental evaluation" verifies the algorithm's performance through simulation experiments; and finally, Sect. " Conclusion" presents the conclusion and future directions of work.

Related work

The SDN paradigm has been widely applied to various communication network problems, such as distributed routing [8], mobile edge computing [16], data center networks [17], optical networks [19], named data networking [22], and vehicular ad hoc network routing [23]. Routing is a core activity in SDN, as it routes data from source nodes to destination nodes, significantly affecting network performance, such as energy consumption, latency, and packet transmission rate. To maximize network utility, researchers have proposed many routing optimization algorithms. Existing research on routing optimization methods includes traditional routing methods, deep learning-based routing algorithms, reinforcement learning-based routing algorithms, heuristic-based routing algorithms, and dedicated algorithms designed for dynamic routing scenarios.

The traditional routing algorithm is the OSPF protocol [5], which only selects the shortest path for forwarding routes. However, in real network environments, the available bandwidth of paths dynamically changes over time, making it difficult for traditional routing algorithms to accurately perceive the current network status and take appropriate actions accordingly. When the service demand bandwidth reaches the bottleneck of a link, it not only dramatically reduces user experience but also may cause severe network congestion, leading to a significant waste of network resources [11].

In recent years, the development of artificial intelligence (AI) technology based on deep learning has progressed rapidly, and many studies have used AI models to solve routing optimization problems. The general deep learning-based routing solution takes network topology and network status information as input, and the model makes appropriate routing decisions based on the input information [11]. Existing intelligent routing solutions based on deep learning models mainly generate routing paths in a hop-by-hop manner. Shin et al. [8] designed a distributed intelligent routing algorithm based on GNN to utilize topology information further. By deploying the GNN parameter update function on each router and using GNN to extract network structure and state information, the iterative process of GNN topology modelling can be completed in a distributed manner, with good scalability and distributed routing decision-making. Compared with traditional routing methods, it has lower information exchange costs and faster routing convergence speed when the network environment changes. However, generating routing paths hop-by-hop is prone to causing loops, and deploying the above solution requires routers to have powerful model computing capabilities. It may also require modifications to existing routing protocols. Therefore, deploying the above solution under an existing computer network architecture would incur high costs and affect network scalability.

Deep learning methods can also assist in making routing decisions and improve the efficiency of routing algorithms. The authors in [24] combined GNN and LSTM models using a deep learning model based on graph neural networks to establish relationships between network status, network topology, traffic matrices, and routing path models, and used the established model to assist heuristic algorithms in calculating routing strategies. Chen et al. [25] used the LSTM algorithm to predict network traffic on the SDN application plane. Using deep learning models to assist traditional routing algorithms effectively improves the performance of traditional routing optimization algorithms. However, the process of replacing traditional routing algorithms with deep learning-based route generation algorithms remains a protracted endeavor. Firstly, DL loss functions typically exhibit non-convexity, posing challenges in pursuing globally optimal solutions. Secondly, DL must grapple with gradient vanishing or explosion issues, often rendering gradient-based optimization methods ineffectual. Finally, the practical deployment of DL models encounters impediments due to scale and inference speed constraints [18]. These constraints underscore the challenges confronted during real-world implementation.

Reinforcement learning is another popular direction for intelligent routing algorithms. The authors in [6] proposed the RSIR algorithm, which uses Q-learning to make decisions on the next hop node, optimizing network load balancing, latency, and packet loss accordingly. The authors in [14] directly represented Q-Learning states and paths in the action space for implementing multipath routing with guaranteed flow. These Q-learning-based methods

use a Q-table for learning and decision-making, but their perception capabilities could be improved, leading to poor performance. Deep reinforcement learning is a combination of deep learning and reinforcement learning, which replaces the Q-table with a deep neural network, allowing the agent to observe the state of the environment and learn its characteristics through deep learning perception processing, and then train its decision-making skills through reinforcement learning [16]. Fu et al. [17] proposed a DQN-based routing policy for creating different routing methods for mouse and elephant flows in data center networks to optimize throughput, latency, and packet loss. The authors in [18] implemented a new architecture that combines multiagent reinforcement learning (MARL) and GNN to minimize network congestion. The authors in [19] proposed a DRL intelligent routing algorithm based on ensemble learning and information propagation neural networks to maximize the utilization of optical transport networks. These DRL-based intelligent routing algorithms can achieve good performance after some training iterations. However, due to the computational complexity of DRL algorithms, their deployment on hardware devices is limited, and more mature solutions are needed for these algorithms to be practically applicable.

In intelligent routing algorithms, metaheuristic algorithms can be an alternative to DRL, as their performance is comparable to machine learning-based routing algorithms. The authors in [26] proposed a knowledge-based enhanced ant colony system algorithm to utilize the knowledge obtained from SDN controllers for placing virtual network functions (VNFs) while simultaneously allocating primary and redundant paths for flows to manage services. In [20], the authors proposed a multicriteria heuristic (MCH) for solving online VNF placement and routing problems, using genetic algorithms to learn hyperparameters for the online MCH model to minimize total power consumption in NFV infrastructure. The authors in [27] proposed a fault tolerance metaheuristic-based scheme (FTMBS) for controller placement problems in wireless software-defined networks. The authors in [28] proposed a novel clustering algorithm, inspired by the natural flying soaring technique of the albatross bird, that efficiently selects network cluster heads to solve the controller placement problem, intending to achieve network reliability and enhance the network lifetime. However, these algorithms only consider globally optimal solutions provided by the algorithms without taking into account the network's volatility and complexity, neglecting the instability of algorithm training and convergence.

In the context of dynamic routing algorithms, the authors in [29] introduced the Ant Colony Optimization Algorithm for Dynamic Routing in Software Defined Networking (ACOSDN). This algorithm effectively handles dynamic network fluctuations, reduces congestion, enhances throughput, and mitigates latency and packet loss concurrently. However, the ant colony algorithm's convergence rate is sluggish and tends to converge to local optima, potentially leading to network performance degradation. In [30], the authors proposed a route path selection approach based on link quality estimation and critical switch awareness. This method elevates data throughput and packet delivery rates by introducing multiple constraint parameters, including link latency, link transmission rate, and critical switch switching frequency scores. Nevertheless, it augments the criteria for routing calculations without adopting a global optimization perspective, which may impose limitations on network performance. In contrast, our work focuses on globally optimizing network routing within a backbone network scenario characterized by minimal topology changes and excludes considerations for routing in dynamic topologies.

Table 1 summarizes pertinent literature on intelligent routing algorithms, focusing on their key points, approaches, objectives, algorithms, and limitations. It underscores the imperative for research in the following manner: there is a need for a model capable of rapid convergence, independent of hardware constraints, and capable of perceiving a network's global perspective to optimize routing in response to network fluctuations, thus enhancing network performance. In this regard, this paper proposes the African vulture routing optimization algorithm for SDN routing optimization. This paper improves the convergence speed and robustness of metaheuristic algorithms, making them better suited to adapt to changes in the SDN network state.

Mathematical modelling for network optimization

We describe a network using G, as shown in Eq. (1), where V represents the set of forwarding nodes in the data forwarding layer, and E represents the set of all links e, as shown in Eq. (2), where m is the number of links.

$$G = (V, E) \tag{1}$$

$$E = [e_1, e_2, \ldots, e_m] \tag{2}$$

We use c_i to represent the rated bandwidth of each link e_i and define C as the sequence of network link-rated bandwidths, as shown in Eq. (3), where $|C| = |E| = m$.

$$C = [c_1, c_2, \ldots, c_m] \tag{3}$$

We use t_{i}^{τ} to describe the flow passing through edge e_i at time τ, which is routed by the SDN controller, and define T^{τ} as the sequence of network traffic, as shown in Eq. (4).

$$T^{\tau} = \left[t_1^{\tau}, t_2^{\tau}, \ldots, t_m^{\tau}\right] \tag{4}$$

Table 1 Related work

Literature	Schemes	Aim	Algorithm	Limitations
Shin [8]	Distributed Intelligent Routing	Utilize topology information effectively to improve routing decisions	GNN-based algorithm	Prone to causing loops, requires powerful model computing capabilities, and may need modifications to routing protocols
Rischke [14]	Reinforcement Learning Routing	Optimize network load balancing, latency, and packet loss	RSIR algorithm using Q-learning	Limited perception capabilities and potential performance issues
Wang [16]	Deep Reinforcement Learning	Optimize throughput, latency, and packet loss for mouse and elephant flows in data center networks	DQN-based routing policy using deep neural networks and reinforcement learning	Computational complexity limits hardware deployment
Bernárdez [18]	Multiagent Reinforcement Learning	Minimize network congestion	Combination of MARL and GNN	Computational complexity may limit hardware deployment
Chen [19]	Ensemble Learning DRL	Maximize the utilization of optical transport networks	DRL intelligent routing algorithm based on ensemble learning and information propagation neural networks	Computational complexity may limit hardware deployment
Rusek [24]	Deep Learning-Assisted Routing	Establish relationships between network status, topology, traffic matrices, and routing path models	GNN and LSTM models combined with heuristic algorithms	Process of replacing traditional routing algorithms with deep learning-based ones is challenging due to non-convex loss functions, gradient issues, and practical deployment constraints
Farshin [26]	Knowledge-Based Metaheuristics	Utilize knowledge from SDN controllers for VNF placement and routing	Enhanced ant colony system algorithm with knowledge integration	Neglects network volatility and complexity, instability in algorithm training and convergence
Samarji [27]	Fault tolerance metaheuristic	Maximize the network connectivity, maximize the load balance among controllers, minimize the worst-case latency, and maximize the network lifetime	Genetic algorithm and greedy randomized adaptive search problem algorithm	The impact of load distribution of faulty controller on the network performanc is not analyzed
Samarji [28]	Energy soaring-based routing	Selects the network cluster heads for solving the controller placement problem	Energysoaring routing algorithm adopted from the albatross bird	Without factoring in the network's instability and intricacies
Raouf [29]	Ant Colony Optimization	Handle dynamic network fluctuations and reduce congestion, latency, and packet loss	ACOSDN algorithm using Ant Colony Optimization	Sluggish convergence and potential convergence to local optima
Isyaku [30]	Route Path Selection Optimization	Elevate data throughput and packet delivery rates with link quality estimation and constraint parameters	Route path selection optimization approach based on link quality estimation and switch awareness	Doesn't adopt a global optimization perspective, may limit network performance

We utilize u_i^τ to describe the usage of edge e_i at time τ, as calculated by Eq. (5). U^τ refers to the collection of utilization rates for links at time τ, as shown in Eq. (6).

$$u_i^\tau = \frac{t_i^\tau}{c_i} \tag{5}$$

$$U^\tau = \left[u_1^\tau, u_2^\tau, \ldots, u_m^\tau\right] \tag{6}$$

Our focus is primarily on optimizing network routing from a load-balancing perspective. Load-balancing is a vital optimization objective in intelligent network optimization. We achieve load-balancing by considering the full range of link utilization U^τ, defined as G^τ at time τ and calculated by Eq. (7). The optimization objective is defined as Eq. (8), with Eq. (9) representing the restraint condition for achieving the optimization objective, which aims to minimize the full range of link utilization. By reducing the difference of link utilization among each link, the network traffic can be evenly distributed among the links, avoiding the concentration of network traffic on a single link, and thus achieving load balancing.

$$G^\tau = max\left(U^\tau\right) - min\left(U^\tau\right) \tag{7}$$

$$min\left(G^\tau\right) \tag{8}$$

$$subject to G^\tau \in [0, 1] \tag{9}$$

We make routing decisions using the OSPF algorithm based on link weights, which calculates the shortest path for a node pair's traffic demand using link weights. The optimal path changes as its usage increases due to congestion. Therefore, the network's load status affects the optimal path selection. Assigning link weights based on the current network load condition is an essential issue in the OSPF routing algorithm.

This paper gives weights to each link based on the AVRO algorithm. For population-based metaheuristic optimization algorithms, the population represents the solutions to the problem, and fitness corresponds to the optimization objective of the problem. The AVRO algorithm maximizes the population fitness by moving the population positions in an acceptable computational time and space to obtain a better feasible solution. Expressly, multiple populations of vultures represent alternative solutions, and in each iteration, the vulture population moves towards better fitness. We compute the fitness of each vulture population and choose the one with the highest fitness as the solution for that iteration. Based on the objective optimization Eq. (8), this study describes the AVRO population using Eq. (10) and calculates population fitness using Eq. (11), where P^τ in Eq. (10) represents the collection of link weights, and p_i^τ describes the weight of link i at time τ.

$$P^\tau = \left[p_1^\tau, p_2^\tau, \ldots, p_m^\tau\right] \tag{10}$$

In Eq. (11), the variables α_1, α_2, and α_3 are adjustable parameters used to fine-tune the range of fitness and the weighting of optimization objectives. The part of the tuning experiment exhibits numerical values, which possess no intrinsic significance and may be adjusted based on the range of values pertinent to one's particular objectives. In the equation, the full range of link utilization G^τ is negatively correlated with population fitness, meaning that a smaller link utilization spread results in a higher network load balancing coefficient.

$$Fitness = \alpha_1 - log\left(\left(G^\tau + \alpha_2\right)^{\alpha_3}\right) \tag{11}$$

Design of routing optimization algorithm based on AVRO

The AVRO algorithm devised five steps to mimic the lifestyle of African vultures, as shown in Fig. 2. First, initialize the vulture population, then perform T-step iteration, calculate the fitness of the vulture population, and select the best vulture for the next stage. First, calculate whether the vulture is satiated for each population vulture. In the early training stage, if the vulture is stuffed, it will enter the exploration stage; otherwise, it will enter the development stage. In the later stage of training, it will directly enter the optimization stage.

AVRO algorithm

Table 2 defines the primary mathematical symbols associated with the AVRO algorithm.

(1) The first stage: Identifying the finest vulture among the population

In the AVOA algorithm, the initial population is formed by randomly generating solutions within the search range, as shown in Eq. (12), where r_1 is a random variable that takes values between 0 and 1.

$$P(i) = r_1 * (ub - lb) + lb \tag{12}$$

Instead of randomly generating the population, Eq. (13) is utilized to initialize the vulture population in this paper. Here, parameter δ_1 can take any value to adjust the range of b, representing the edge betweenness, i.e., the proportion of shortest paths passing through a given edge in the network.

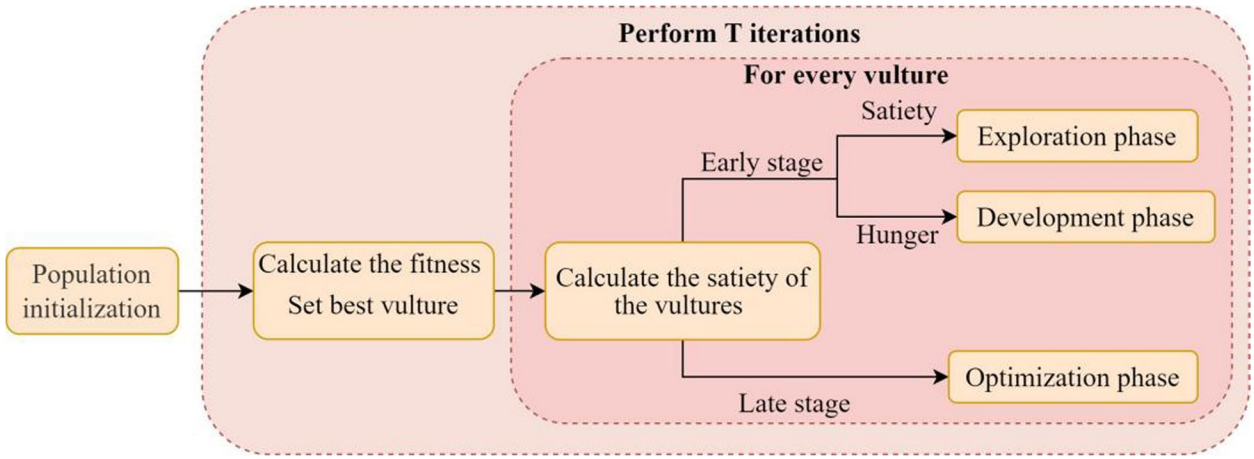

Fig. 2 The general workflow of AVRO algorithm

Table 2 The principal mathematical symbols associated with the AVRO algorithm

symbol	definition	formula
$\alpha_1, \alpha_2 \alpha_3$	Adjustable parameters with no intrinsic significance	(11)
r_n	Random real numbers that follows a uniform distribution from 0 to 1, n is a positive integer from 1 to 8	-
i	The i-th iteration	-
T	Maximum Number Of Iterations	-
b	The proportion of shortest paths passing through a given edge in the network	(13)
δ_1	Adjustable parameters to adjust the range of b	(13)
$R(i)$	Leader vulture (the two vultures with the highest fitness) in iteration i	(14)
$Best_1$	The best solution	(14)
$Best_2$	The second-best solution	(14)
F	The satiety rate of vultures	(17)
$P(i)$	The vector position of the vulture in the i-th iteration	-
$D(i)$	Exploration distance of vultures in the i-th iteration	(20)
$d(i)$	The distance between vultures and leader vultures	(24)
ub	The upper bound of algorithm search interval	-
lb	The lower bound of algorithm search interval	-
δ_2	Adjustable parameters to adjust the range of F in the optimization stage	(36)

$$P(i) = r_1 * (ub - lb) + lb + \delta_1 * b \tag{13}$$

The central idea of the OSPF algorithm in computer networks is to utilize the shortest paths within the network. The higher the edge betweenness centrality, the more shortest paths pass through that edge, leading to a higher utilization rate and potential congestion. Thus, increasing the initial weight of links with high betweenness centrality can reduce the number of shortest paths passing through such edges, thereby avoiding situations where traffic congregates on only a few edges. Similar to the concept of course learning [31], during population initialization, the AVRO algorithm is explicitly informed about the information it needs to explore to guide the algorithm toward better solution spaces and improve its performance.

After the formation of the initial population, the fitness of all solutions is calculated. The best solution $Best_1$ and second-best solution $Best_2$ in each iteration are candidates for leader vulture $R(i)$, as shown in Eq. (14).

$$R(i) = \begin{cases} Best_1, p_1 = L_1 \\ Best_2, p_2 = L_2 \end{cases} \tag{14}$$

In (14), L_1 and L_2 are custom parameters, which represent the probability of selecting the leader vulture. p_i is calculated using the roulette wheel selection method [32], as shown in Eq. (15).

$$p_i = \frac{L_i}{\sum_{i=1}^{n} L_i} \tag{15}$$

(2) The second stage: Identification of the satiety rate for the vultures

The vultures must feed to gain the energy necessary for survival. Satiated vultures are able to venture further in search of sustenance, while their famished counterparts lacking the requisite energy are forced to seek food in proximity to high-energy vultures and become more aggressive in the process. Equation (17) models this process by using F to represent the satiety of the vultures and balancing algorithmic development with exploration. In Eq. (17), t is defined by Eq. (16), and this kind of simulation behavior has been used before [21]. When $|F|$ is greater than 1, the vultures search for food in different areas, and AVRO enters the exploration stage. If $|F|$ is less than 1, AVRO enters the development stage, and the vultures search for food near their current location. During training, development and exploration alternate, with the early stages emphasizing exploration and the later stages emphasizing development to promote algorithm convergence.

$$t = h \times \left(sin^{\gamma}\left(\frac{\pi}{2} \times \frac{i}{T}\right) + cos\left(\frac{\pi}{2} \times \frac{i}{T}\right) - 1 \right) \tag{16}$$

$$F = (2 \times r_2 + 1) \times z \times \left(1 - \frac{i}{T}\right) + t \tag{17}$$

Within Eqs. (16) and (17), i denotes the current iteration, and T represents the total iterations. r_2 is a random number between 0 and 1. z is a random number between -1 and 1, where values below 0 indicate hunger in the vulture, while values above 0 signify satiation, with satiation declining over time. h represents a random number between -2 and 2.

(3) The third stage: Exploration

When $|F| \geq 1$, the vulture enters an exploration phase and randomly searches the environment. As shown in Eq. (18), $P(i+1)$ represents the location of the vulture in iteration $i+1$. If the generated random number r_{G_1} is greater than or equal to parameter G_1, then Eq. (19) is executed. Otherwise, Eq. (21) is used.

$$P(i+1) = \begin{cases} Eq(19), if G_1 \geq r_{G_1} \\ Eq(21), if G_1 < r_{G_1} \end{cases} \tag{18}$$

$$P(i+1) = R(i) - D(i) \times F \tag{19}$$

$$D(i) = |X \times R(i) - P(i)| \tag{20}$$

According to Eq. (19), the vulture forages around the leader vulture $R(i)$, with $D(i)$ defined by Eq. (20) representing the exploration distance of the vulture. X is the distance randomly moved by the vulture, adding randomness to the exploration phase. It is obtained using the formula $X = 2 \times r$, where r represents a random number between 0 and 1.

$$P(i+1) = R(i) - F + r_3 \times ((ub - lb) \times r_4 + lb) \tag{21}$$

In Eq. (21), ub and lb indicate the upper and lower bounds of the algorithm search interval, representing the positions of the population. r_3 and r_4 are random numbers between 0 and 1. The use of r_4 adds randomness to the distribution of solutions within the search interval, increasing the diversity of the algorithm's exploration.

(4) The fourth stage: Development

If $|F|$ is less than 1, AVRO enters the development phase, which is aimed at improving the convergence efficiency of AVRO. There are also two stages within the development phase, each utilizing two different strategies determined by parameters G_2 and G_3, both predefined in the [0,1] range.

Development (Stage One): When $|F|$ falls within the range [0.5,1), AVRO enters the first stage of its development phase. In this stage, a random number between 0 and 1, denoted as r_{G_2}, is generated. If r_{G_2} is greater than or equal to parameter G_2, then a food competition process is executed; otherwise, a rotating flight process is performed, as outlined in Eq. (22).

$$P(i+1) = \begin{cases} Eq(23), if G_2 \geq r_{G_2} \\ Eq(27), if G_2 < r_{G_2} \end{cases} \tag{22}$$

Food Competition: When $|F| \geq 0.5$, it indicates that the vultures have relatively abundant energy. Vultures with sufficient energy are reluctant to share food with others, while weaker vultures gather around healthier ones to search for food. When many vultures converge on a single food source, population conflict may arise. Equations (23) and (24) are utilized to model this process. Random variable r_5, which takes values between 0 and 1, is introduced to increase the randomness of the process. Equation (24) is used to obtain the distance $d(t)$ between a vulture and the leader vulture.

$$P(i+1) = D(i) \times (F + r_5) - d(t) \tag{23}$$

$$d(t) = R(i) - P(i) \tag{24}$$

Rotating Flight: The vulture's rotating flight behavior is simulated using a spiral model. The vulture's position is updated using Eq. (27), while S_1 and S_2 are obtained using Eqs. (25) and (26). Random variables r_6 and r_7 take values between 0 and 1.

$$S_1 = R(i) \times \left(\frac{r_6 \times P(i)}{2\pi}\right) \times cos(P(i)) \tag{25}$$

$$S_2 = R(i) \times \left(\frac{r_7 \times P(i)}{2\pi}\right) \times sin(P(i)) \tag{26}$$

$$P(i+1) = R(i) - (S_1 + S_2) \tag{27}$$

Development (Stage Two): If $|F|<0.5$, AVRO enters the second stage of its development phase, during which intense fighting breaks out among vultures due to their convergence. First, a random number between 0 and 1, denoted as r_{G_3}, is generated. If r_{G_3} is greater than or equal to parameter G_3, then the vulture converges towards the leader vulture. Otherwise, an aggressive food competition process is carried out, as outlined in Eq. (28).

$$P(i+1) = \begin{cases} Eq(31), if G_3 \geq r_{G_3} \\ Eq(32), if G_3 < r_{G_3} \end{cases} \tag{28}$$

Vulture Convergence: Vultures converge towards the leader vulture, using Eqs. (29) and (30) to calculate the convergence position. Here, $Best_1(i)$ represents the optimal solution in the i-th iteration, while $Best_2(i)$ represents the second-best solution. Subsequently, all vultures are gathered using Eq. (31), initiating competition for food.

$$A_1 = Best_1(i) - \frac{Best_1(i) \times P(i)}{Best_1(i) - P(i)^2} \times F \tag{29}$$

$$A_2 = Best_2(i) - \frac{Best_2(i) \times P(i)}{Best_2(i) - P(i)^2} \times F \tag{30}$$

$$P(i+1) = \frac{A_1 + A_2}{2} \tag{31}$$

Food Competition: When $|F|<0.5$, the leader vulture becomes hungry and weak, lacking sufficient energy to compete with other vultures for food. Other stronger vultures become aggressive in their search for food. Equation (32) is utilized to model this behavior. $LF(x)$ simulates the vulture's flight process, utilizing a Levy Flight *(LF)* model [33], as shown in Eq. (33), where σ is a part of the definition of the *LF* model, x represents the problem dimension and β is a fixed parameter that takes a value of 1.5. Random variables u and v are drawn from a normal distribution, as shown in Eq. (34).

$$P(i+1) = R(i) - \left|d(t)\right| \times F \times LF(x) \tag{32}$$

$$LF(x) = 0.01 \times \frac{u \times \sigma}{|y|^{\frac{1}{2}}}, \sigma = \left\{\frac{\Gamma(1+\beta) \times sin(\pi\beta/2)}{\Gamma[(1+\beta)/2] \times \beta \times 2^{(\beta-1)/2}}\right\}^{1/\beta} \tag{33}$$

$$u \sim N\left(0, \sigma^2\right), v \sim N(0,1) \tag{34}$$

(5) The Fifth stage: Optimization

In the original algorithm, when $|F|$ is greater than 1, vultures search for food in different areas, and AVRO enters an exploration phase. On the other hand, if $|F|$ is less than 1, AVRO enters a development stage where vultures search for food near the optimal solution. Eq. (17) is utilized to compute F. The convergence process and final convergence performance vary depending on the maximum number of iterations. As the maximum number of iterations changes, F also changes during the training process, and its reduction rate slows down, resulting in slower convergence. This approach aims to explore the maximum fitness during the training process but does not directly yield stable convergence during the training phase.

When deploying heuristic algorithms in computer networks, models with stable outputs are desirable. Hence, an optimization phase is added to the AVRO algorithm. During this phase, the AVRO algorithm still executes the second development phase, and an improved formula for computing F is obtained as shown in Eq. (35). r_8 is a random number between 0 and 1. The optimization phase is initiated after a certain number of iterations t_{train}.

$$F = (2 \times r_8 + 1) \times z \times \left(1 - \frac{i}{t_{train}}\right) + t, if t < t_{train} \tag{35}$$

Equation (36) determines the range of F in the optimization stage, where $\text{clip}(F, -\delta_2, \delta_2)) = \max(\min(F, \delta_2), -\delta_2)$ represents limiting F to $[-\delta_2, \delta_2]$. δ_2 is a parameter that can be adjusted. The optimization phase does not affect the exploration and development processes of the vultures. Instead, it reduces the activity range of the vultures to minimize fluctuations in algorithm performance, leading to stable convergence effects.

$$F = clip(F, -\delta_2, \delta_2), if t \geq t_{train} \tag{36}$$

The workflow of the routing optimization algorithm based on AVRO

Algorithm 1 outlines the workflow of the AVRO algorithm. The algorithm takes as input the population size *N*, the maximum number of iterations, and the number of iterations used for training, and outputs the positions of the population, which in this paper are link weights. First, the population is initialized (line 1), followed by training iterations. Throughout the training process, it is imperative to procure the initial state of the network (line 3). Subsequently, the fitness of each member within the vulture population is computed (line 4), and potential leader vultures are selected (line 5). For each vulture population, a leader vulture is randomly selected from the potential candidates (line 7), and the *F* parameter is updated using the formula (lines 8–9). If $|F| \geq 1$, the exploration phase is entered; otherwise, the development phase is entered. During the exploration phase, the population is updated based on Eq. (18) (lines 11–12). The development phase is divided into two stages, where Eqs. (22) and (28) are used to update the population (lines 14–19). Finally, discern the optimal resolution amidst the vulture community and implement it towards routing (line 20–22).

The AVRO algorithm needs to solve a routing problem in each iteration *T*, and each population needs to be evaluated separately. Therefore, the time complexity of AVRO is *O(POP)*, where *POP* represents the size of populations. The space required by the AVRO algorithm is mainly used to store the problem's solution. Thus, the spatial complexity of AVRO is *O(POP * dim)*, where *dim* represents the dimension of the solution and the number of links in this paper.

Algorithm 1 AVRO algorithm for routing optimization

Input：	The population size N, maximum number of iterations T, Training iteration t_{train}
Output：	The location of Vulture
01	Initialize the random population *P*:*(i = 1, 2, ..N)* using Eq.(13)
02	for *i* in *T* do
03	Get network status *S*
04	Calculate the fitness values of vulture using *S* and Eq.(11)
05	Set P_{Best_1}, P_{Best_2} as the location of vulture
06	for (each vulture (P_i)) do
07	Select *R(i)* using Eq. (14)
08	if $i < t_{train}$ Update the *F* using Eq. (35)
09	else Update the *F* using Eq. (36)
10	if($\|F\| \geq 1$) then
11	if ($G_1 \geq r_{G_1}$) then Update the location vulture using Eq. (19)
12	else Update the location vulture using Eq. (21)
13	if ($\|F\| < 1$) then
14	if ($\|F\| \geq 0.5$) then
15	if ($G_2 \geq r_{G_2}$) then Update the location vulture using Eq. (23)
16	else Update the location vulture using Eq. (27)
17	else
18	if ($G_3 \geq r_{G_3}$) then Update the location vulture using Eq. (31)
19	else Update the location vulture using Eq. (32)
20	Set *best vulture* as the P_i with the highest fitness
21	Calculate the shortest path for the traffic matrix using *best vulture*
22	Routing based on the shortest path

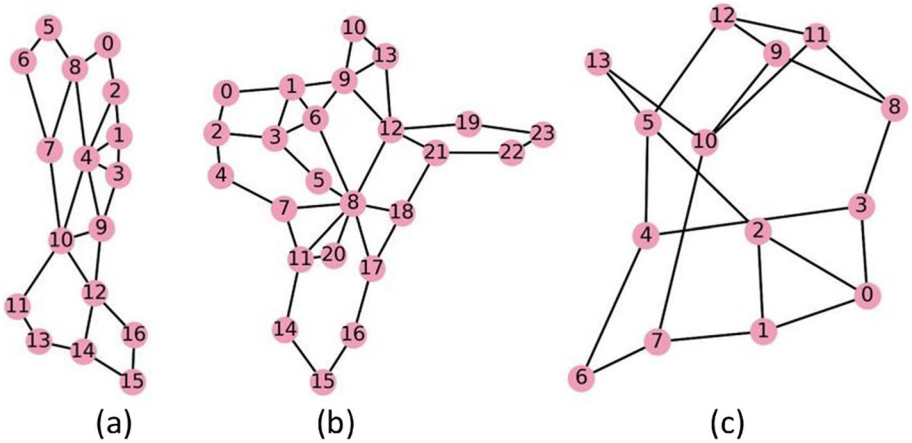

Fig. 3 Experimental network topologies: **a** GBN; **b** GEANT2; **c** NSFNET

Experimental evaluation

Experimental environment settings

The experimentation and evaluation in this paper were conducted using a simulation environment that emulates data plane network interactions in SDN. Our experiments were carried out on an AMD Ryzen 7 2700 CPU, Nvidia GTX1080 graphics card, and 64 GB RAM. Three real network topologies, GBN, GEANT2, and NSFNET [19], each possessing distinct graph characteristics and containing links with 10 GB of bandwidth, were considered for model training and evaluation. The network topology structure is depicted in Fig. 3. The connectivity between nodes in the GBN and GEANT topologies varies significantly, with some nodes connected to only two nodes and others related to several nodes. We can call the nodes with many connections as central nodes, which may have a higher link betweenness because they may have more shortest paths. In contrast, the GEANT topology has more nodes and a more complex network structure, making routing optimization more difficult. The NSFNET topology has a more evenly distributed network with node degrees ranging from 2–4.

In this paper, two dynamic traffic intensities (Traffic-1 and Traffic-2) were implemented in the three network topologies to verify the model's generalization performance in different network scenarios. Traffic-1 was generated using the gravity model, while Traffic-2 was generated using the uniform model. Each traffic intensity corresponds to 200 demand matrices. During the training process, a total of 7000 episodes were run, with one traffic demand matrix being switched for each episode. The traffic matrices are square matrices of rank equal to the number of network nodes, with each element of the matrix representing the traffic demand to be allocated by a particular node in a given time slot.

Parameter settings

This paper selects an appropriate parameter value for the AVRO agent through experimentation. In the experiment, we choose the GBN network topology and Traffic-1 traffic intensity as the experimental environment, and record the fitness during training to plot a curve. For the convenience of observation, the curve aggregated 100 fitness values at each iteration, displaying an estimate of the central tendency, as shown in Fig. 4.

Figure 4(a) displays the training results of the AVRO algorithm with different population sizes. The algorithm's performance exhibits significant differences under different population sizes. When the population size is 2, the AVRO algorithm cannot explore a better solution space, while at 5, the population's fitness fluctuates greatly, reaching optimal performance at 20. The population size represents the number of optional solutions, and as the population size increases, the algorithm's search ability increases but also contains more uncertainty, leading to more significant performance fluctuations. However, because increasing the population size affects the speed of algorithm decision-making, this paper chooses 10 for training.

Figure 4(b) shows the parameter δ_2 tuning results of the optimization stage, where this parameter's value is set to 0.0001. The smaller the value of δ_2, the more stable the convergence of the algorithm. The final convergence fitness value of the model has no relationship with this parameter. In theory, this value affects the convergence performance by limiting the value of F during the

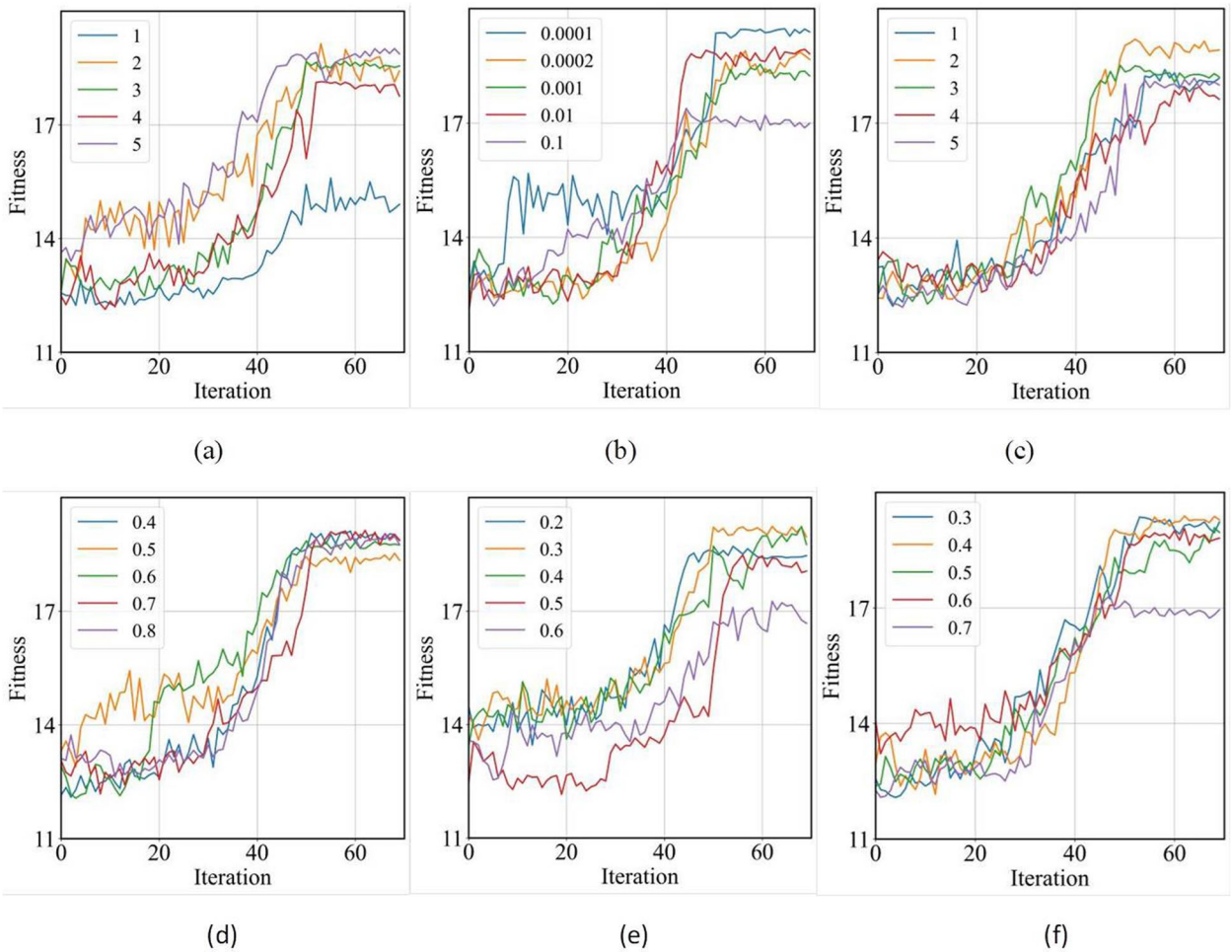

Fig. 4 Experimental results of adjusting parameters: **a** POP; **b** δ_2; **c** γ; **d** G_1; **e** G_2; **f** G_3

optimization stage. It only affects the convergence stability during the optimization stage but does not affect the training before the optimization stage. The difference in fitness value during the optimization stage is mainly caused by the randomness of the algorithm during the training process.

Figure 4(c) displays the training results when the γ parameter in Eq. (16) takes different values, with $\gamma=2$ achieving the best performance. The figure reveals the nonlinear relationship between parameter γ and the algorithm's fitness. Within a certain moderate range of parameter γ, the algorithm can achieve higher fitness. Setting γ too high may decrease algorithm performance. This is because γ affects the baseline of satiety F, and as γ increases, the value of t_3 decreases, which has less impact on F.

Figure 4(d) shows the parameter tuning results of G_1 (Eq. (18)), where the algorithm's convergence is almost identical under different parameters. This is because G_1 determines the exploration method of the population, which means that the two exploration methods perform similarly.

Figure 4(e) shows the parameter tuning results of G_2 (Eq. (22)). As G_2 increases, the fitness of the model's convergence exhibits a trend of first increasing and then decreasing, with $G_2= 0.3$ achieving relatively optimal performance. G_2 balances food competition and rotational flight in the first stage of development, as G_2 decreases, the probability of entering the hovering flight stage decreases, while the probability of entering food competition increases. This suggests that food competition causes the relatively hungry vultures to cluster around the satiated ones, which can enhance the development performance of the algorithm.

Figure 4(f) displays the parameter tuning results of G_3 (Eq. (28)).In the figure, the algorithms with parameter settings ranging from 0.3 to 0.6 exhibit similar performance, but the algorithm with a parameter setting of 0.7 has poor convergence performance. This suggests that a large value of G_3 can lead to the algorithm falling

Table 3 Related parameters and values of AVRO

Parameter	Value
POP	10
δ_1	2
δ_2	0.0001
γ	2
G_1	0.4
G_2	0.3
G_3	0.4
ub	1
lb	2
α_1	5
α_2	0.1
α_3	19

into an early convergence trap. G_3 balances food competition and vulture clustering in the second stage of development. Increasing G_3 leads to vultures always moving around the leader vulture, unable to obtain enough energy to explore the solution space, thus posing a risk of early maturity.

Table 3 presents the values obtained for several significant parameters and adjusted parameters of AVRO.

Performance evaluation

To evaluate the algorithmic performance of AVRO, this study compares it with AVOA [21], DDPG [34], GTO [35], RSIR [6], and OSPF [5] under the same optimization objective. We collect the fitness in the evaluation and plot it as a curve. The curve aggregates 100 fitness values for each value of iteration and displays the estimated central trend and the 95% confidence interval of the estimate.

- DDPG is a classical deep reinforcement learning algorithm whose state space includes link traffic, link utilization, link ingress/egress demands, and the agent's action in the previous time slot. The action space is set as the weight of all links in the network, and the reward is consistent with the fitness design of this study. The DDPG algorithm outputs link weights based on the state. Then it calculates the routing strategy using the link weights, consistent with the AVRO algorithm.
- GTO, also known as Gorilla Troops Optimizer, is a metaheuristic algorithm that mathematizes the collective social habits of gorillas. This algorithm shares the same optimization objective as the one in this paper, and it also outputs link weights.
- RSIR is designed based on the Q-learning algorithm of reinforcement learning. It is a hop-based routing algorithm whose state space is designed as nodes in the network, and the action space is designed as optional next hops. The reward setting in this paper is related to the remaining bandwidth of links, packet loss rate, and delay, and is adjusted by three hyperparameters accordingly. When compared with our model, the reward setting is focused on the remaining bandwidth of links. When traffic reaches a certain node, RSIR continues to determine the next node for routing the traffic, thereby allocating network traffic.
- OSPF is a classic routing algorithm that calculates the shortest path between node pairs using the Dijkstra algorithm. Traffic will be routed directly through the path with the least number of hops. It should be noted that among these algorithms, OSPF is not implemented based on SDN controllers, while other algorithms are implemented within the SDN architecture.

Fitness evaluation

For ease of observation, Table 4 shows the fitness values of each algorithm after converging, averaged over the last 1000 iterations. "-1" and "-2" in the table represent the traffic intensities for Traffic-1 and Traffic-2, respectively.

The fitness comparison of each algorithm under Traffic-1 traffic intensity is shown in Fig. 5. In the GBN network, although the initial performance of the improved AVRO algorithm is not as high as that of the AVOA algorithm, after ten iterations, the fitness of the AVRO

Table 4 The fitness of each algorithm in each network scenario

	GBN-1	GBN-2	GEANT-1	GEANT-2	NSFNet-1	NSFNet-2
AVRO(ours)	19.10	17.32	16.38	12.61	19.57	19.29
AVOA [21]	16.74	17.09	15.63	12.28	17.79	19.06
DDPG [34]	17.77	14.72	12.80	11.36	16.40	16.16
GTO [35]	17.58	14.45	9.91	10.18	17.59	15.72
RSIR [6]	16.26	14.61	10.45	12.19	16.50	10.86
OSPF [5]	11.93	7.57	6.46	9.56	16.27	14.11

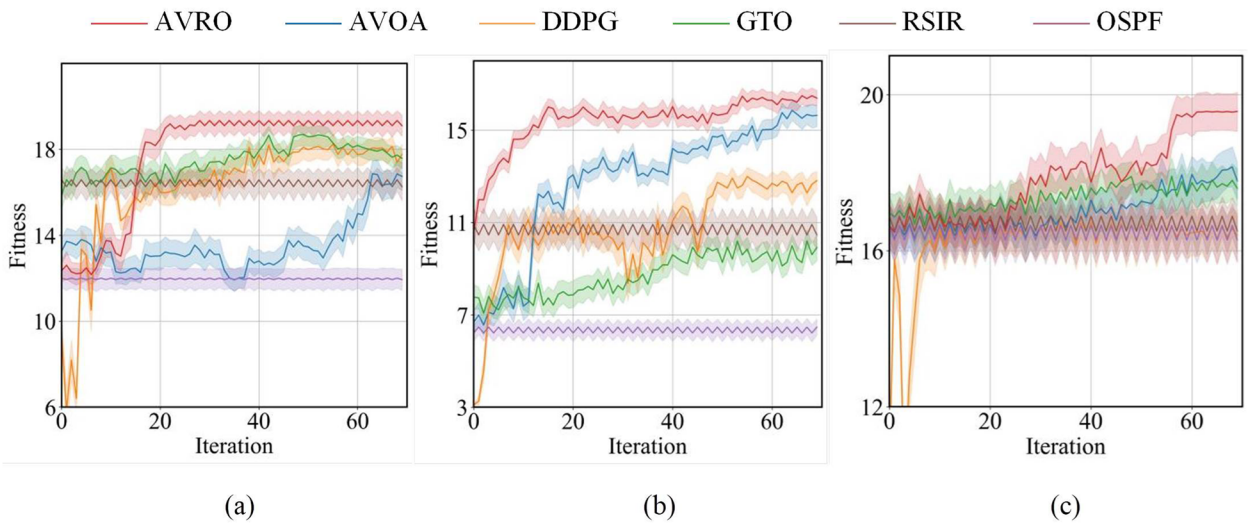

Fig. 5 Fitness evaluation of algorithms under Traffic-1 traffic intensity: **a** GBN; **b** GEANT2; **c** NSFNET

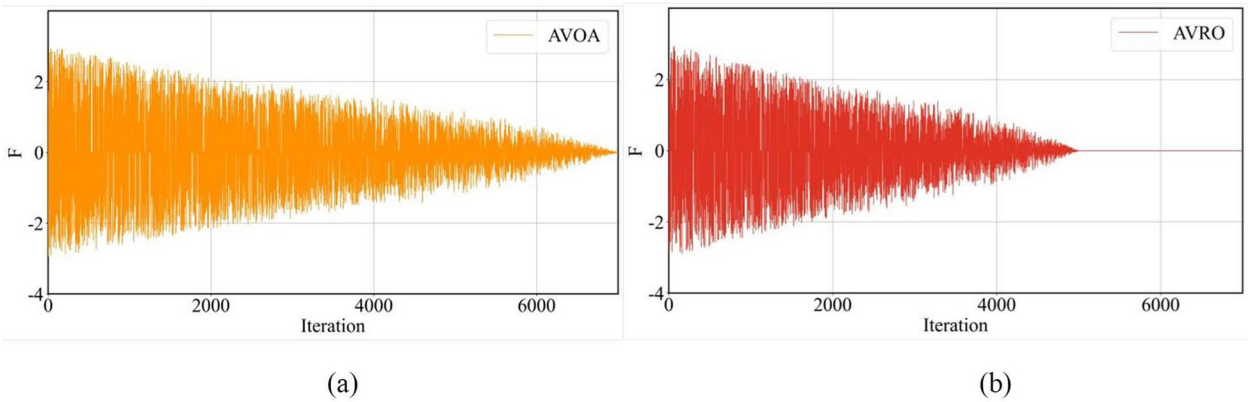

Fig. 6 Performance of F: **a** AVOA algorithm; **b** AVRO algorithm

algorithm surpasses that of the AVOA algorithm and can reach convergence before the optimization stage. This indicates that the improved population initialization algorithm does not directly enhance the performance of the AVRO algorithm but makes it easier for the algorithm to learn valuable knowledge and accelerate convergence. In the GEANT2 and NSFNET networks, the initialization performance of the AVRO algorithm is superior to that of the AVOA algorithm because population initialization explicitly affects the solution space. The AVOA algorithm almost does not converge under Traffic-1 traffic intensity because formula (17) calculates F, which is related to the current and maximum iteration numbers. As the number of iterations increases, the trend of F becomes closer to $F=0$, as shown in Fig. 6. However, due to the preliminary development stage, the algorithm prematurely converges. The AVRO algorithm is very close to $F=0$ after the optimization stage, and the algorithm's fitness slightly improves in a narrow range. This is because the AVRO algorithm stops exploring and reduces its development scope, moving towards the vulture in a more refined solution space.

During the training phase, the DDPG algorithm exhibits significant instability and relatively poor convergence performance. This is because continuous action greatly expands the exploration space of the task, making learning more difficult. The AVRO algorithm does not need to consider tasks' actions and space as the DRL algorithm, so it is more difficult to get trapped in local optima and has better global optimization ability. Additionally, the Q-network's learning is faced with the challenge of overestimation, which directly propagates its fitting error to the policy network through gradient descent, resulting in poor stability and performance of the DDPG algorithm. The GTO algorithm performs better than DDPG in fitness for GBN and NSFNET but worse than DDGP

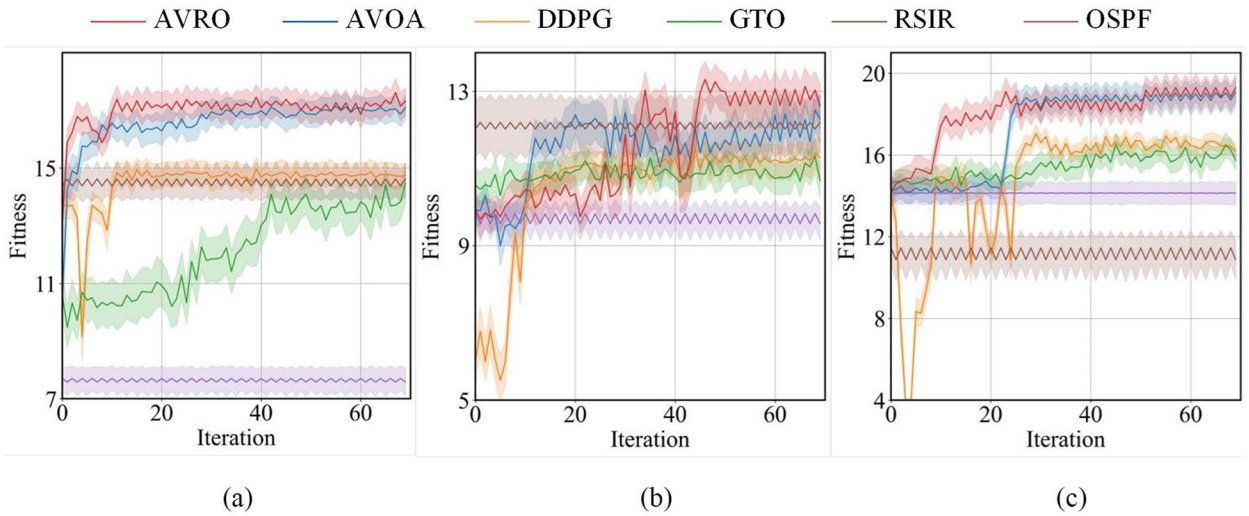

Fig. 7 Fitness evaluation of algorithms under Traffic-2 traffic intensity: **a** GBN; **b** GEANT2; **c** NSFNET

in GEANT2. Its initial performance is relatively good, but its search capability is poor. The global search capability of the GTO algorithm during the exploration stage is limited, which is inferior to that of the AVRO algorithm. The performance of the RSIR and OSPF algorithms remains relatively fixed due to the experimental environment setup, as they are successively trained on 200 traffic matrices. While RSIR can make better decisions based on network information, its learning capacity is limited, and it performs worse than DDPG trained to converge in most network conditions. Under the same fitness metric, the OSPF algorithm performs worse than the others. Unlike AVRO's dynamic routing strategy, its routing decision is fixed, and it cannot dynamically adjust routing strategies according to network conditions, resulting in potential congestion.

The fitness comparison of each algorithm under Traffic-2 traffic intensity is shown in Fig. 7. In the GBN and NSFNET networks, the AVRO algorithm converges faster than the AVOA algorithm. AVRO's optimization in the GEANT2 and NSFNET networks is relatively less pronounced than that under Traffic-1, with more significant performance fluctuations. However, AVRO's fitness experiences a slight range improvement during the optimization stage before stabilizing and outperforms the AVOA algorithm. The DDPG algorithm's convergence speed is consistent with that of the AVOA algorithm, but its fitness is lower. The GTO algorithm performs slightly worse than the DDPG algorithm, with some fluctuations in fitness trends. The performance of the RSIR algorithm in GEANT2 is particularly noteworthy due to the network's pronounced centrality, which surpasses that of the other two topologies. By leveraging the guidance of residual bandwidth, RSIR can effectively avoid routing through these highly central edges when making decisions. The OSPF algorithm outperforms the RSIR algorithm in the NSFNET topology. This can be attributed to the evenly distributed nodes within the range of 2–4 and the relatively uniform distribution of Traffic-2. In this scenario, the RSIR algorithm is challenging to learn helpful knowledge.

This paper uses minimizing the maximum link utilization as an optimization objective, as shown in Eq. (8). We hope the maximum link utilization is as tiny as possible

Table 5 The maximum link utilization of each algorithm in each network scenario

	GBN-1	GBN-2	GEANT-1	GEANT-2	NSFNet-1	NSFNet-2
AVRO(ours)	39.2%	43.6%	45.2%	58.1%	38.0%	39.3%
AVOA [21]	45.2%	43.5%	47.7%	59.4%	42.9%	39.2%
DDPG [34]	41.5%	50.5%	56.7%	62.4%	47.0%	45.8%
GTO [35]	42.6%	51.9%	68.2%	66.8%	43.3%	48.4%
RSIR [6]	48.0%	51.7%	66.8%	60.9%	46.5%	66.9%
OSPF [5]	60.6%	78.2%	83.1%	69.4%	47.3%	53.8%

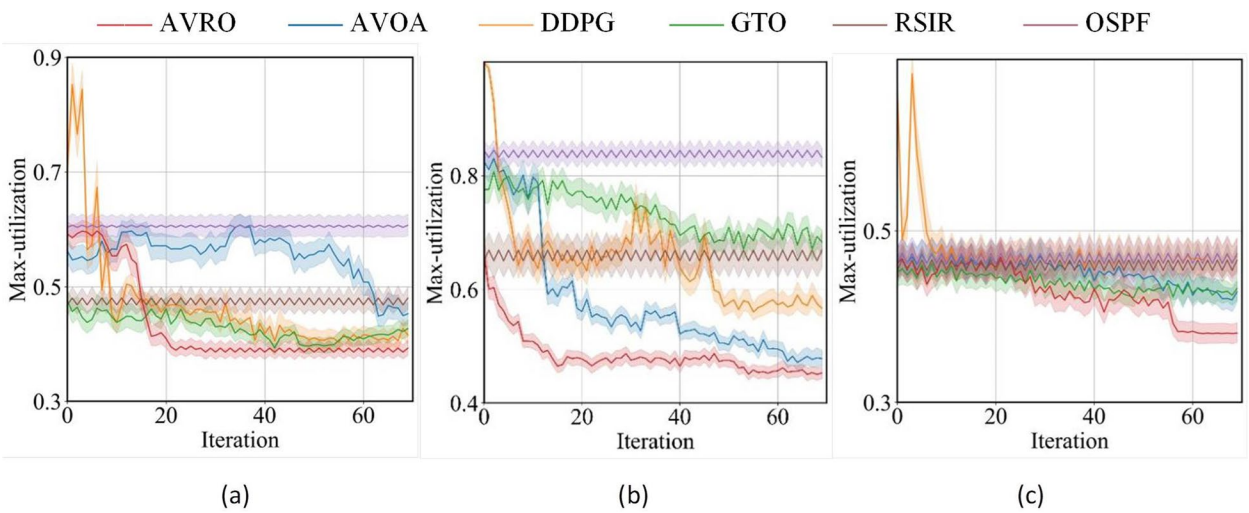

Fig. 8 Evaluation of maximum link utilization of algorithms under Traffic-1 traffic intensity: **a** GBN; **b** GEANT2; **c** NSFNET

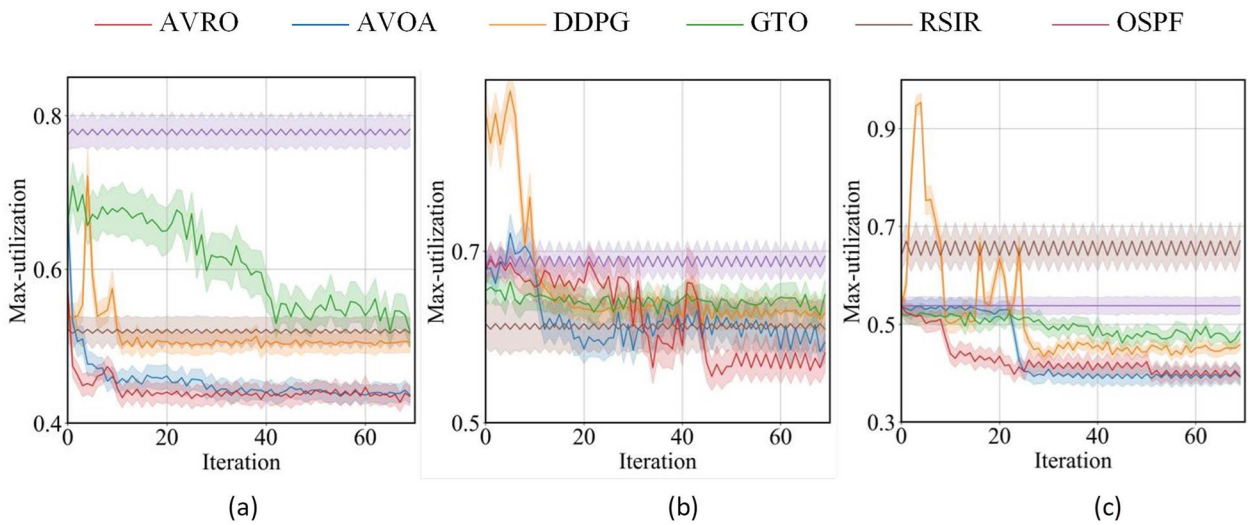

Fig. 9 Evaluation of maximum link utilization of algorithms under Traffic-2 traffic intensity: **a** GBN; **b** GEANT2; **c** NSFNET

to achieve the load balancing goal. This can also avoid network congestion. Similarly, Table 5 shows each algorithm's maximum link utilization values after convergence, averaged over the last 1000 iterations.

Figure 8 shows the maximum link utilization of each algorithm under Traffic-1. The AVRO algorithm exhibits a decreasing trend in maximum link utilization under Traffic-1, ultimately converging to a lower position relative to other algorithms. AVRO and AVOA have small confidence intervals, indicating more stable algorithmic performance. Compared to other algorithms, the AVRO algorithm can find relatively optimal solutions in less time. The AVRO algorithm can minimize the maximum link utilization under the optimization objective of this paper, achieving load balancing. Furthermore, using the full range of link utilization as an optimization objective avoids situations where some links in the network remain idle for long periods, reducing network resource waste.

Figure 9 shows the maximum link utilization of each algorithm under Traffic-2. Under the NSFNET topology,

the AVRO algorithm's maximum link utilization is slightly higher than that of the AVOA algorithm because of the objective optimization setting, which biases the AVRO algorithm towards balanced link utilization. Regarding other algorithms, the DDPG algorithm exhibits significant instability during the training stage but has a small range of confidence intervals after convergence. The GTO algorithm performs relatively poorly in training. Although the RSIR algorithm's fixed decision-making can perform well, it cannot surpass other models. The OSPF algorithm's maximum link utilization is relatively high, indicating poor load-balancing capability. The AVRO algorithm outperforms these baseline algorithms through global optimization and fast convergence.

Conclusion

SDN decouples the network's data and control planes, making network management more straightforward and creating opportunities for deploying intelligent algorithms in networks. However, existing heuristic algorithms perform poorly, while popular reinforcement learning-based routing algorithms suffer from slow convergence and lack of scalability, making their deployment in real-world scenarios nearly impossible. In this study, we propose the AVRO algorithm to optimize SDN routing problems. First, we model the optimization objective of load balancing in the network and propose an improved population initialization algorithm that leverages explicit information on network environment characteristics to accelerate algorithm convergence. Additionally, we introduce an exploration stage to enhance the development of the algorithm and further improve its performance. Finally, we conduct simulation experiments with three network topologies and two traffic intensities, comparing our proposed model against OSPF, RSIR, DDPG, and the original AVOA algorithm. The results show that our model has the highest fitness and the smallest network metric value of maximum link utilization rate. Therefore, our model has superior load-balancing performance.

In the future, we plan to implement our framework in a more realistic network simulator, such as Mininet, using real controllers, such as Floodlight, and simulating bursty situations with high traffic and link breakage to test the effectiveness and practicality of the algorithm. Meanwhile, we will also consider applying heuristic algorithms to deep learning algorithms to combine both advantages.

Authors' contributions

Conceptualization: Junyan Chen and Hongmei Zhang; Methodology: Junyan Chen, Hongmei Zhang and Wei Xiao; Software: Wei Xiao, Jiacheng Zuo and Junyan Chen; Validation: Junyan Chen and Wei Xiao; Formal analysis: Junyan Chen and Hongmei Zhang; Investigation: Junyan Chen and Wei Xiao; Resources: Xinmei Li; Data curation: Xinmei Li; Writing—original draft preparation: Junyan Chen, Wei Xiao and Jiacheng Zuo; Writing—review and editing: Junyan Chen, Hongmei Zhang and Wei Xiao; Visualization: Wei Xiao and Jiacheng Zuo; Supervision: Hongmei Zhang; Project administration: Hongmei Zhang; Funding acquisition: Hongmei Zhang and Junyan Chen. All authors have read and agreed to the published version of the manuscript.

Funding

This work is supported by the National Natural Science Foundation of China (grant numbers 61861013), the major program of Guangxi Natural Science Foundation (grant numbers 2020GXNSFDA238001), and the Middle-aged and Young Teachers' Basic Ability Promotion Project of Guangxi (grant numbers 2020KY05033).

Availability of data and materials

The authors confirm that the data supporting the findings of this study are available within the article.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

All authors confirm that neither the article nor any parts of its content are currently under consideration or published in another journal. The authors agree to publication in the journal.

Competing interests

The authors declare no competing interests.

Received: 11 July 2023 Accepted: 28 January 2024
Published online: 13 February 2024

References

1. Ali J, Rutvij H, Mohannad A, Byeong-hee R (2023) ESCALB: An effective slave controller allocation-based load balancing scheme for multi-domain SDN-enabled-IoT networks. Journal of King Saud University-Computer and Information Sciences 35(6):101566
2. Chen J, Huang X, Wang Y, Zhang H, Liao C, Xie X, Li x, Xiao W (2023) AST-PPO: A proximal policy optimization algorithm based on the attention mechanism and spatio–temporal correlation for routing optimization in software-defined networking. Peer-to-Peer Networking and Applications 16:2039–2057
3. Miguel-Alonso J (2023) A research review of OpenFlow for datacenter networking. IEEE Access 11:770–786
4. Cummaudo A, Vasa R, Grundy J, Abdelrazek M (2022) Requirements of API documentation: a case study into computer vision services. IEEE Trans Software Eng 48(6):2010–2027
5. Ali K, Zafrullah M, Hussain M, Ahmad A (2017) Performance analysis of OSPF and hybrid networks. International Symposium on Wireless Systems and Networks (ISWSN 2017) 2017:1–4.
6. Casas-Velasco D, Rendon O, Fonseca, (2021) Intelligent routing based on reinforcement learning for software-defined networking. IEEE Trans Netw Serv Manage 18(1):870–881
7. Zhuang Z, Wang J, Qi Q, Sun H (2018) Graph-aware deep learning based intelligent routing strategy. Proceedings of the 2018 IEEE 43rd Conference on Local Computer Networks (LCN 2018) 2018:441–444.
8. Shin D, Kim J (2021) Deep reinforcement learning-based network routing technology for data recovery in exa-scale cloud distributed clustering systems. Appl Sci 11(18):8727–8819
9. Wu Z, Pan S, Chen F, Long G, Zhang C, Yu P (2021) A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems 32(1):4–24
10. Yu Y, Si X, Hu C, Zhang J (2019) A review of recurrent neural networks: LSTM cells and network architectures. Neural Computer 31(7):1235–1270

11. Yang S, Tan C, Madsen D, Xiang H, Li Y, Khan I, Choi B (2022) Comparative analysis of routing schemes based on machine learning. Mob Inf Syst 2022:4560072
12. Sutton R, Barto A (2018) Reinforcement learning: An introduction. MIT Press, Cambride
13. Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double Q-learning. AAAI Conference on Artificial Intelligence 2016:2094–2100
14. Rischke J, Sossalla P, Salah H, Fitzek F, Reisslein M (2020) QR-SDN: Towards reinforcement learning states, actions, and rewards for direct flow routing in software-defined networks. IEEE Access 8:174773–174791
15. Luong N, Hoang D, Gong S, Niyato D, Kim D (2019) Applications of deep reinforcement learning in communications and networking: a survey. IEEE Communications Surveys and Tutorials 21(4):3133–3174
16. Wang J, Zhao L, Liu J, Kato N (2021) Smart resource allocation for mobile edge computing: a deep reinforcement learning approach. IEEE Transactions on Emerging and Topics Computing 9(3):1529–1541
17. Fu Q, Sun E, Meng K, Li M, Zhang Y (2020) Deep Q-learning for routing schemes in SDN-based data center networks. IEEE Access 8:103491–103499
18. Bernárdez G, Suárez-Varela J, López A, Wu B, Cabellos-Aparicio A (2021) Is machine learning ready for traffic engineering optimization? IEEE 29th International Conference on Network Protocols (ICNP 2021) 2021:1–11.
19. Chen J, Xiao W, Li X, Zheng Y, Huang X, Huang D, Wang M (2022) A routing optimization method for software-defined optical transport networks based on ensembles and reinforcement learning. Sensors 22:8139
20. Seyed R, Shahram J, Peyman B (2022) A power-efficient and performance-aware online virtual network function placement in SDN/NFV-enabled networks. Comput Netw 205:108753
21. Benyamin A, Farhad S, Seyedali M (2021) African vultures optimization algorithm: a new nature-inspired metaheuristic algorithm for global optimization problems. Comput Ind Eng 158:107408
22. Ali J, Adnan M, Gadekallu T, Jhaveri R, Roh B (2022) A QoS-aware software defined mobility architecture for named data networking. IEEE Globecom Workshops (GC Wkshps) 2022:444–449
23. Zhao L, Bi Z, Lin M, Hawbani A, Shi J, Guan Y (2021) An intelligent fuzzy-based routing scheme for software-defined vehicular networks. Comput Netw 187:107837
24. Rusek K, Varela J, Almasan P, Barlet-Ros P, Cabellos-Aparicio A (2020) RouteNet: leveraging graph neural networks for network modeling and optimization in SDN. IEEE J Sel Areas Commun 38(10):2260–2270
25. Chen J, Wang Y, Huang X, Xie X, Zhang H, Lu X (2022) ALBLP: adaptive load-balancing architecture based on link-state prediction in software-defined networking. Wirel Commun Mob Comput 2022:8354150
26. Farshin A, Sharifian S (2019) A modified knowledge-based ant colony algorithm for virtual machine placement and simultaneous routing of NFV in distributed cloud architecture. Journal of Supercomputing 75:5520–5550
27. Samarji N, Salamah M (2021) A fault tolerance metaheuristic-based scheme for controller placement problem in wireless software-defined networks. Int J Commun Syst 34(4):e4624
28. Samarji N, Salamah M (2022) ESRA: Energy soaring-based routing algorithm for IoT applications in software-defined wireless sensor networks. Egyptian Informatics Journal 23(2):215–224
29. Raouf O, Askr H (2019) ACOSDN-Ant colony optimization algorithm for dynamic routing in software defined networking. The 14th International Conference on Computer Engineering and Systems (ICCES), IEEE:141–148.
30. Isyaku B, Bakar KA, Mohd Zahid MS, Alkhammash EH, Saeed F, Ghaleb FA (2021) Route path selection optimization scheme based on link quality estimation and critical switch awareness for software-defined networks. Appl Sci 11(19):9100
31. Wang X, Chen Y, Zhu W (2022) A survey on curriculum learning. IEEE Trans Pattern Anal Mach Intell 44(9):4555–4576
32. Pandey A, Kulhari A, Shukla D (2022) Enhancing sentiment analysis using Roulette wheel selection based cuckoo search clustering method. J Ambient Intell Humaniz Comput 13(1):629–657
33. Houssein EH, Saad MR, Hashim FA, Shaban H, Hassaballah M (2020) Levy flight distribution: a new metaheuristic algorithm for solving engineering optimization problems. Eng Appl Artif Intell 94:103731
34. Chen J, Wang Y, Ou J, Fan C, Lu X, Liao C, Huang X, Zhang H (2022) ALBRL: automatic load-balancing architecture based on reinforcement learning in software-defined networking. Wirel Commun Mob Comput 2022:3866143
35. Benyamin A, Farhad S, Seyedali M (2021) Artificial gorilla troops optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems. Int J Intell Syst 36(10):5887–5958

Publisher's Note