Mechatronic Systems and Control, Vol. 54, No. 10, 2026
An Open Access Paper

INTELLIGENT AVOIDANCE METHOD FOR LOCAL OPTIMAL SOLUTIONS IN NONLINEAR PROGRAMMING BASED ON REINFORCEMENT LEARNING

Guodong Wang*

Abstract

To maintain a dynamic balance between efficient exploration and exploitation and avoid falling into local optima, this study investigates an intelligent avoidance method for local optimal solutions in nonlinear programming based on reinforcement learning. By transforming the nonlinear programming problem into a deterministic optimisation problem through deterministic descriptions; using long short-term memory networks as the encoder for the policy network and value network in reinforcement learning, extracting temporal features of the solution space within the policy network, estimating state value within the value network, and adjusting the policy network; employing a graph attention model that fuses node and edge features as the decoder for the policy network, combining temporal features to generate a probability distribution for breaking the solution space, and adjusting nonlinear programming decisions; designing a local optimal solution intelligent avoidance reward function for the value network based on the deterministic optimisation problem of nonlinear programming; aiming to maximise the reward function, training the reinforcement learning network by combining policy gradient optimisation algorithms with adaptive Q-learning algorithms, updating the parameters of the policy and value networks to ensure a dynamic balance between exploration and exploitation, thereby achieving intelligent avoidance of local optima in nonlinear programming. Experimental results demonstrate that this method exhibits superior reinforcement learning network training performance, with the optimal learning rate being 1×10^{-3}; in the nonlinear programming problem of vehicle network task offloading, applying this method effectively reduces vehicle energy consumption and resource leasing costs, with maximum energy consumption approximately 100 J and maximum resource leasing cost around 2 Yuan.

* School of General Education, Chongqing Water Resources and Electric Engineering College, Chongqing 402160, China; e-mail: wangguodong_love@163.com
Corresponding author: Wang Guodong

Recommended by Jian Su
(DOI: 10.2316/J.2026.201-0618)

Key Words

Reinforcement learning, nonlinear programming, local optimal solution avoidance, long short-term memory (LSTM) network, graph attention model

1. Introduction

Nonlinear programming problems are an important branch of operations research. These problems require solving the extremum of a multivariate nonlinear function under a set of equality or inequality constraints [1]. They are widely found in fields, such as production management, quality control, and product design, and are also the foundation for achieving artificial intelligence and optimisation. Nonlinear programming problems typically involve an objective function and a set of constraints [2], where both the objective function and constraints may be nonlinear. Unlike linear programming problems, the feasible region and properties of the objective function in nonlinear programming problems are more complex, and multiple local optima may exist [3]. A local optimum is a solution that achieves the optimal value of the objective function within a local region of the feasible region, but it is not the global optimum. In practical engineering decision-making, nonlinear programming problems are widely present in many industrial fields, such as production management, quality control, and product design. Due to the complexity of the feasible region and objective function properties of nonlinear programming, there are easily multiple local optimal solutions. Once the solving algorithm falls into local optima, the result obtained is not the true optimal solution [4], which will directly lead to unreasonable allocation of production resources, difficulty in achieving optimal product quality standards, and inability to achieve the best balance between performance and cost in product design. This seriously affects the efficiency, quality, and economic benefits of industrial production and brings huge losses to enterprises. Therefore, this article emphasises the crucial importance of avoiding falling into local optima in solving these pain points in the industrial

field and improving the scientific and effective decision-making in practical engineering. In recent years, with the rapid development of artificial intelligence, machine learning, heuristic algorithms, and other technologies, new approaches and methods have been proposed to address the problem of local optimal solutions in nonlinear programming. These intelligent methods, which mimic natural phenomena, biological behaviour, or human intelligence, possess strong global search capabilities and adaptability [5], enabling them to explore the feasible region more effectively in complex nonlinear programming problems and increase the probability of finding the global optimal solution.

For example, Escobedo introduced a new direct method for solving sequences of closely related linear equations (SLEs) encountered in nonlinear programming, which is both effective and free from rounding errors. Experiments demonstrate that on fully dense matrices with over one million entries, the method achieves a runtime improvement of over 75 times for exact decompositions. The length of any matrix coefficients generated by the method is polynomial in the input size, without resorting to greatest common divisor operations [6]. The method improves efficiency through exact decomposition, primarily relying on the structure of the linear equation system and lacks dynamic adaptability. Rizk-Allah *et al.* addressed the limitations of heap-based optimisers (HBOs) in easily getting stuck in local optima and failing to reach global solutions by combining HBOs with chaotic search to achieve global optimisation for nonlinear programming problems. Comprehensive results demonstrate that the algorithm effectively handles nonlinear programming problems in optimisation tasks [7]. While chaotic search enhances global exploration, it may result in slow convergence or low search efficiency. Zhang *et al.* developed a value function solution method for two-stage nonconvex integer programming with random right sides based on the equivalence between level set minimal vector sets and non-dominated right-side vector sets. This method can solve pure integer variable problems in both stages. The effectiveness of this method was demonstrated using a nonlinear generalised assignment problem with uncertain capacities [8]. This method is based on the equivalence between the minimal horizontal set and the non-dominated right-side vector set and is not applicable to all nonlinear programming problems. Additionally, its computational complexity grows exponentially with the problem size. Na *et al.* utilised a differentiable exact augmented Lagrange multiplier as the value function and approximated the Newton system using an overlapping time decomposition strategy in each sequence quadratic programming (SQP) iteration. Experiments demonstrate that, under appropriately chosen overlap size and penalty parameters, the approximate search direction remains the descending direction of the augmented Lagrange multiplier. Additionally, the unit step length is a locally accepted approximate search direction [9]. This method approximates the Newton system *via* overlapping time decomposition, but the approximation error affects convergence.

Reinforcement learning maintains a dynamic balance between efficient exploration and exploitation through the interaction between an agent and its environment [10], avoiding getting stuck in local optima and demonstrating superior dynamic exploration and global search capabilities. Additionally, reinforcement learning can adjust strategies online to adapt to environmental changes. Therefore, this study investigates a reinforcement learning-based intelligent avoidance method for local optimal solutions in nonlinear programming, providing theoretical support for nonconvex optimisation. At the model architecture level, long short-term memory (LSTM) networks are used as encoders for reinforcement learning strategy networks and value networks, fully exploring the temporal characteristics of the solution space and providing more comprehensive and accurate information for policy adjustment; Simultaneously using a graph attention model that integrates node and edge features as the decoder of the strategy network, combined with temporal features to generate the probability distribution of solution space breaking actions, making the decision-making process more in line with the complex characteristics of nonlinear programming problems. In the design of the reward function, based on the deterministic optimisation problem of nonlinear programming, the local optimal solution of the value network is carefully designed to intelligently avoid the reward function, effectively guiding the reinforcement learning network to explore towards the global optimal solution. In terms of training algorithms, the combination of policy gradient optimisation algorithm and adaptive Q-learning algorithm ensures that the reinforcement learning network maintains a dynamic balance between efficient exploration and utilisation, achieving precise updates of policy network and value network parameters, providing a new and efficient solution for solving the local optimal solution problem in nonlinear programming.

2. Intelligent Avoidance Method for Local Optima in Nonlinear Programming

2.1 Deterministic Description of Nonlinear Programming Problems

Reinforcement learning algorithms require explicit, well-defined input components, such as the state space, solution space, and action space, for breaking through the problem, and local optimal avoidance reward functions. If a nonlinear programming problem involves uncertainty (such as uncertain parameters), the uncertainty in its objective function and constraints makes it difficult for the input to be directly processed by reinforcement learning models [11]. Through deterministic descriptions, uncertain nonlinear programming problems are transformed into problems with clear, fixed structures (such as definite objective function forms and explicit constraint boundaries), constructing recognisable states, breaking-through actions, and reward rules for reinforcement learning, enabling the algorithm to learn and explore effectively. After the deterministic description of nonlinear programming problems, the objectives and constraints can be precisely defined [12],

allowing reinforcement learning to effectively avoid local optima in nonlinear programming and search for the global optimum with clear optimisation directions and constraint adherence, thereby improving learning efficiency and effectiveness.

Nonlinear programming problems are generally described as

$$\begin{aligned} &\max_{d} F(y,u,\xi) \\ &s.t.\ y = y(d,c), y \in [y_{\min}, y_{\max}], d \in [d_{\min}, d_{\max}], \\ &\quad c \in R \triangleq [c_{\min}, c_{\max}] \end{aligned} \tag{1}$$

where d is the decision vector of the nonlinear programming; y is the output of the nonlinear programming constraints; c is the uncertain parameter; F is the nonlinear objective function; $y = y(d,c)$ represents nonlinear equality constraints containing uncertain parameters; $[y_{\min}, y_{\max}]$ is the value range of y; $[d_{\min}, d_{\max}]$ is the value range of d; and $[c_{\min}, c_{\max}]$ is the value range of c.

For the nonlinear programming problem described by (1), given a decision $d \in [d_{\min}, d_{\max}]$, the variation range of the objective function value $F(d,c)$ can be expressed as $\varphi = [F_{\min}(d), F_{\max}(d)]$, where

$$\begin{cases} F_{\min}(d) = \min\limits_{c \in R} F(d,c) \\ F_{\max}(d) = \max\limits_{c \in R} F(d,c) \end{cases} \tag{2}$$

For any value $\gamma \in \varphi$ within the interval φ, define the risk coefficient β_1

$$\beta_1 = \frac{\gamma - F_{\min}(d)}{F_{\max}(d) - F_{\min}(d)} \tag{3}$$

This β_1 is called the decision risk factor, which indicates that due to the uncertainty of the parameter c, the actual objective function value obtained from the nonlinear programming decision d is greater than or equal to γ. Clearly, $\beta_1 \in [0,1]$. As the value of γ increases, β_1 also increases, meaning the decision-maker faces greater risk. Specifically, when $\gamma = F_{\min}(d)$, there is $\beta_1 = 0$, indicating that the decision-maker is certain to obtain an objective function value greater than or equal to $F_{\min}(d)$; when $\gamma = F_{\max}(d)$, there is $\beta_1 = 1$, indicating that the maximum risk faced by the decision-maker to achieve an objective function value greater than or equal to $F_{\max}(d)$. Transforming the above yields

$$\gamma = \beta_1 F_{\max}(d) + (1-\beta_1) F_{\min}(d) \tag{4}$$

The above expression indicates that when the risk factor is β_1, the maximum objective function value that the nonlinear programming decision d can achieve is γ.

Thus, the objective function of the nonlinear programming problem in (1) can be transformed into the following deterministic optimisation goal:

$$F = \max_{d} \beta_1 F_{\max}(d) + (1-\beta_1) F_{\min}(d) \tag{5}$$

Here, the risk coefficient $\beta_1 \in [0,1]$ represents the degree of risk the decision-maker is willing to bear in pursuit of the optimal objective function value.

Furthermore, considering that the uncertain parameter c causes deviations in the actual objective function value $F(d,c)$ obtained from the decision d, with a maximum deviation of

$$e(d) = F_{\max}(d) - F_{\min}(d) \tag{6}$$

In many cases, decision-makers have specific requirements regarding the magnitude of this deviation [13]. Based on the above objective function, the following objective function can be adopted:

$$\begin{aligned} F = \max_{d} \beta_1 F_{\max}(d) + (1-\beta_1) F_{\min}(d) \\ - M \max(0, e(d) - e_{\max}) \end{aligned} \tag{7}$$

where $e_{\max}$ is a constant (called the maximum allowable deviation), satisfying $0 < e_{\max} < e(d)$, and M is a very large normal number (called the balancing constant). This expression indicates that the decision-maker requires the deviation of the objective function value to be preferably limited within a certain range $e_{\max}$. Even if it exceeds this range, it should be as small as possible.

Considering the nonlinear programming problem in (1), for a given decision $d \in [d_{\min}, d_{\max}]$ at position j, define

$$\begin{cases} y_j^-(d) \triangleq \min\limits_{c \in R} y_j(d,c) \\ y_j^+(d) \triangleq \max\limits_{c \in R} y_j(d,c) \end{cases} \tag{8}$$

In the strict sense of constraints for nonlinear programming problems, the permissible set of solution sets is a feasible decision region Ω. Any decision $d \in \Omega$ in Ω satisfies the following for any $c \in R$:

$$y_{j\,\min} \le y_j^-(d) \le y_j^+(d) \le y_{j\,\max} \tag{9}$$

Through the above discussion, the original uncertain nonlinear programming problem can be transformed into the following deterministic optimisation proposition:

$$\begin{aligned} F = \max_{d} \beta_1 F_{\max}(d) + (1-\beta_1) F_{\min}(d) \\ - M \max(0, e(d) - e_{\max}) \\ s.t.\ ,\ y_{j\,\min} \le y_j^-(d) \le y_j^+(d) \le y_{j\,\max}, \\ d \in [d_{\min}, d_{\max}] \end{aligned} \tag{10}$$

where $y_j^-(d) = \min_{c \in R} y_j(d,c)$; $y_j^+(d) = \max_{c \in R} y_j(d,c)$; $F_{\min}(d) = \min_{c \in R} F(d,c)$;$F_{\max}(d) = \max_{c \in R} F(d,c)$.

Consider a small-scale nonlinear programming sample for a production planning scenario: a factory produces two products A and B. Production of product A requires 2 units of raw material 1 and 1 unit of raw material 2, while production of product B requires 1 unit of raw material 1 and 3 units of raw material 2. The factory currently has 10 units of raw material 1 and 12 units of raw material 2. The profit per product A is 5 yuan, and the profit per product B is 8 yuan. However, there are uncertain factors in the production process, such as fluctuations in the supply of raw materials (corresponding to uncertain parameters). Now we need to determine the quantity of products A and B to be produced

(decision vector), while satisfying the constraints of raw materials (constraint output), to maximise the total profit (objective function). Through the above deterministic description method, this uncertain production planning nonlinear programming problem can be transformed into a deterministic optimisation proposition for processing.

2.2 Intelligent Avoidance of Local Optima Based on Reinforcement Learning

This chapter presents the principles, functions, and interaction logic of LSTM encoder, GAT-NE decoder, and various components of reinforcement learning (such as state space, action space, reward function, *etc.*) clearly through detailed and precise formula derivation and textual explanation. The collaborative process of each module is also fully explained through step-by-step decomposition.

2.2.1 Encoder Design for Reinforcement Learning

The solution space of nonlinear programming is complex (multimodal, non-convex, and high-dimensional) [14]. If the initial solution falls into a local optimum "attraction domain," traditional algorithms (such as gradient descent and local search) are easily trapped and struggle to autonomously escape. Reinforcement learning learns "how to generate better initial solutions" through a policy network, and its exploration mechanism can actively break out of the "cognitive boundary" of local optima. To this end, reinforcement learning is used to perform intelligent avoidance of local optima on the deterministic nonlinear programming problem in Section 2.1, resulting in a set of globally optimal solutions for the intelligent avoidance of local optima in nonlinear programming [15].

The mapped nonlinear programming problem after determination is formulated as a Markov decision process (MDP), with the quadruple defined as $\langle S, B, U^b_{s\widehat{s}}, \theta \rangle$. Among them, S is the state space of the nonlinear programming solution, s representing the current state information of the nonlinear programming solution space, including static information such as the structural features of the problem instance (*e.g.,* $y_{\min}, y_{\max}, d_{\min}, d_{\max}$, and other constraint boundaries); dynamic information, such as the current decision d corresponding to the partial solution state (*e.g.,* the determined decision variable segments, the upper and lower bounds of the current constraint outputs $y_j^-(d), y_j^+(d)$, *etc.*). That is, S contains static constraint information and current solution dynamic information. S is responsible for providing the "environment observation" for the reinforcement learning agent, allowing the agent to understand the current optimisation progress and constraints of the nonlinear programming problem, serving as the basis for decision-making [16]. B is the solution space breaking action space, generated by the policy network. The solution space breaking action b represents an adjustment operation on decision d. It must satisfy the range constraints of decision variables $d \in [d_{\min}, d_{\max}]$ to ensure the feasibility of the solution space breaking action. B enables the agent to explore the solution space by selecting solution space breaking actions, which is a direct means to achieve the optimisation of the nonlinear programming solution [17]. $U^b_{s\widehat{s}}$ is the local optimal avoidance reward function, estimated by the value network. After executing the solution space breaking action b, if the new nonlinear programming solution space state $\widehat{s}$ results in an improved (or more optimal deviation constrained) deterministic objective value, a positive reward is given; otherwise, a negative reward is assigned. The reward can be defined as the difference between the deterministic objective value of the new nonlinear programming solution space state and that of the original solution space state, *i.e.,* $U^b_{ss'} = \Delta[\max_d \beta_1 F_{\max}(d) + (1-\beta_1)F_{\min}(d) - M\max(0, e(d) - e_{\max})]$. $U^b_{s\widehat{s}}$ is responsible for providing the "feedback signal" to the agent, guiding it to learn solution space breaking action sequences that improve the objective and avoid local optima, serving as the core driving force of reinforcement learning optimisation. θ is the decay coefficient, used to balance the weight between current rewards and future long-term rewards, regulating the agent's "foresight," and ensuring it explores globally optimal solutions rather than settling for local improvements.

Using LSTM networks as encoders for the policy network and value network in reinforcement learning, the LSTM encoder in the policy network extracts the temporal dependencies and global features$h(s_t)$ of the nonlinear programming solution space state sequence, that is, it stores the historical decision trajectory through the hidden state $h(s_t)$, capturing the temporal dependencies in exploring the solution space of nonlinear programming problems [18], enabling the policy network to learn the long-term associations of "nonlinear programming solution space state s - solution space escape actions b" (such as remembering features of states that have previously fallen into local optima), providing historical state memory and global constraint representations for the graph attention model (GAT-NE) decoder that fuses node and edge features, and assisting in generating decision adjustments to escape local optima in nonlinear programming problems; in the value network of reinforcement learning, the LSTM encoder estimates the value of the nonlinear programming solution space state, encoding the current state s_t and historical hidden states h_{t-1}, to generate the value of the nonlinear programming solution space state $h(s_t) = \Gamma(s_t)$, evaluating the long-term potential of the current solution to the nonlinear programming problem. By identifying the "plateau" of the value function ($|\Gamma(s_t) - \Gamma(s_{t-1})| < \varepsilon$), it determines whether the system has fallen into a local optimum and adjusts the policy network accordingly, where ε is the judgment threshold. The LSTM controls the cell state through forget, input, and output gates, as described by the following formulas:

$$\begin{cases} X = [h_{t-1}, s_t]^T \\ f(s_t) = \sigma(W_f X + \alpha_f) \\ g(s_t) = \sigma(W_g X + \alpha_g) \\ o(s_t) = \sigma(W_o X + \alpha_o) \\ Z(s_t) = f(s_t) Z(s_{t-1}) + g(s_t) \tanh(W_Z X + \alpha_Z) \\ h(s_t) = o(s_t) \tanh(Z(s_t)) \end{cases} \tag{11}$$

Among them, h_{t-1} is the hidden state of the LSTM at the previous time step, storing encoded information of the historical decision trajectory; $f(s_t)$ is the forget gate; σ is the sigmoid activation function; W_f is the trainable weight matrix for the forget gate; α_f is the bias term for the forget gate; $f(s_t)$ is used to actively forget "invalidated historical experiences" (such as decision patterns that continuously lead to local optima) during the exploration of the solution space, preventing the policy network from repeatedly falling into suboptimal solutions. $g(s_t)$ is the input gate; W_g is the trainable weight matrix for the input gate; α_g is the bias term for the input gate; $g(s_t)$ is responsible for focusing on valuable information in the current state s_t (such as unsaturated decision variables and directions with large gradients of the objective function) to generate effective adjustments to break through the solution space for the policy network. $o(s_t)$ is the output gate; W_o is the trainable weight matrix for the output gate; α_o is the bias term for the output gate; controlling the final output of the LSTM's hidden state $h(s_t)$ *via* $o(s_t)$, ensuring that the encoded information passed to the policy network is "redundancy-free with key exploration directions," and the state value signals passed to the value network are "detectable local optima." $Z(s_t)$ is the cell state update; $Z(s_{t-1})$ is the cell state at the previous time step, storing long-term memory of nonlinear programming solution space exploration; tanh is the hyperbolic tangent activation function; W_Z is the weight matrix for cell state updates; and α_Z is the bias term for cell state updates; at the cell state level, $Z(s_t)$ dynamically updates the "long-term memory" of nonlinear programming solution space exploration by forgetting invalid historical information and integrating current key information, making the nonlinear solution space breaking actions of the policy network and the local optimal detection of the value network more precise.

2.2.2 Decoder Design for Reinforcement Learning

Graph attention model with fusion node and edge features (GAT-NE) as the decoder of the reinforcement learning policy network, combined with the temporal features $h(s_t)$ output by the LSTM encoder, outputs the probability distribution of actions that break the solution space for decision variables in nonlinear programming problems. In the policy network, the GAT-NE decoder is used to drive the exploration of the solution space in nonlinear programming problems [19], mining constraint correlations based on graph structures, and generating actions that escape local optima to break the solution space. The policy network takes the $h(s_t)$ encoded by the LSTM in Section 2.2.1 as input to the GAT-NE decoder, providing historical decision memory and global constraint guidance for generating actions to break the solution space.

The GAT-NE decoder uses the LSTM output $h(s_t)$ as input to compute the context vector k_t, guiding the generation of actions to break the solution space. The first decoding formula is

$$k_t^{(0)} = \begin{cases} h(s_t) + W_h\left(h(s_t)_{d_t} \parallel h(s_t)_{d_{t-1}}\right), t > 1 \\ h(s_t) + \widetilde{v}, \qquad\qquad\qquad\quad t = 1 \end{cases} \tag{12}$$

where $\widetilde{v}$ is a learnable placeholder parameter; d_t is the embedding of the set of decision variables in nonlinear programming, *i.e.,* the local optimal solution avoidance strategy network for nonlinear programming; $h(s_t)_{d_t}$ is the decision variable embedding selected at step t; W_h is the trainable weight matrix; and $h(s_t)_{d_{t-1}}$ is the decision variable embedding selected at step $t-1$.

The context vector $k_t^{(0)}$ is input into a multi-head attention layer to explore the constraint relationships among decision variables in nonlinear programming, generating more precise guidance for actions to break the solution space $k_t^{(1)}$:

$$\begin{cases} \rho = W^{\rho} k_t^{(0)} \\ \kappa_i = W^{\kappa} h(s_t) \\ \nu_i = W^{\nu} h(s_t) \end{cases} \tag{13}$$

$$\mu_{i,t}^1 = \begin{cases} \frac{\rho^T \kappa_i}{\sqrt{\xi_\nu}}, & \text{if } i \notin \{d_{t'} \mid t' < t\} \\ -\infty, & \text{otherwise} \end{cases} \tag{14}$$

where ρ is the query vector (the current decision intent vector in nonlinear programming); ξ_ν is the embedding dimension of the temporal dependencies and global features of the nonlinear programming solution space sequence; κ_i is the key vector of the ith decision variable in nonlinear programming (encoding variable features and constraint relationships); ν_i is the value vector of the ith decision variable (encoding the potential value of variable adjustments); W^{ρ}, W^{κ}, and W^{ν} are trainable weight matrices (learning the matching pattern between queries and variable features); $i = 1, 2, \ldots, m$; m is the total number of decision variables (the number of variables to be optimised in nonlinear programming). $\mu_{i,t}^1$ is the attention coefficient.

The multi-head fusion formula is

$$k_t^{(1)} = W'\left(\Big\|_{n'=1}^{N'} \sum_{i=1}^{m} \left(\sigma\left(\mu_{i,t}^1\right)\right)^{n'} \nu_i^{n'}\right) \tag{15}$$

Among them, N' and n' are the number and index of attention heads; W' is the weight matrix that fuses multi-head features.

Using $k_t^{(1)}$ as input, calculate the probability distribution of the final solution space-breaking actions, directly driving nonlinear programming decision adjustments [20]. The second-layer attention coefficient calculation formula is as follows:

$$\mu_{i,t}^2 = \begin{cases} K \cdot \tanh\left(\frac{k_t^{(1)T} \kappa_i}{\sqrt{\xi_\nu}}\right), & \text{if } i \notin \{d_{t'} \mid t' < t\} \\ -\infty, & \text{otherwise} \end{cases} \tag{16}$$

Among them, K is the scaling factor.

The probability distribution of the solution space-breaking actions is

$$p_{i,t} = \sigma(\mu_{i,t}^2) \tag{17}$$

2.2.3 Nonlinear Programming Local Optimal Solution Intelligent Avoidance Process

The specific steps for intelligently avoiding local optima in nonlinear programming using reinforcement learning are as follows.

Step 1: Encoding phase of the policy network and value network, input the real-time state of the nonlinear programming in Section 2.1 (decision variables d, constraints y, current objective $F(d)$, *etc.*) into the LSTM encoder in Section 2.2.1 after normalisation. Output low-dimensional vectors that capture the temporal dependencies and global features of the solution space of nonlinear programming problems [21]. Within the policy network, they provide historical decision memory and global constraint guidance for the GAT-NE decoder in Section 2.2.2; within the value network, encoding the states of the nonlinear programming solution space is used for value estimation, determining whether the current state has fallen into a local optimum.

Step 2: During the strategy network decoding phase, within the GAT-NE decoder in Section 2.2.2, input the temporal dependencies and global features of the nonlinear programming solution space state sequence encoded by the LSTM in Section 2.2.1. Output the probability distribution $p_{i,t}$ of various actions to adjust the solution space for nonlinear programming decision adjustments, *i.e.,* the probability of adjusting a specific nonlinear programming decision variable. Within the value network, through the policy, the probability distribution of breaking the solution space output by the approximate network $p_{i,t}$, with the auxiliary value network providing a more accurate assessment of the state value in the nonlinear programming solution space, enabling intelligent avoidance of local optima in nonlinear programming.

Step 3: Train the policy network [22], update parameters ϖ. Model the nonlinear programming problem as an MDP and design a local optimal avoidance reward function $U^b_{ss'} = \Delta[\max_d \beta_1 F_{\max}(d) + (1-\beta_1)F_{\min}(d) - M\max(0, e(d) - e_{\max})]$ based on the deterministic description of the objective function in Section 2.1 of the nonlinear programming problem.

The policy network $\pi(b \mid s; \varpi)$ aims for an approximately optimal strategy, with the goal of maximising the long-term cumulative reward function that avoids local optima. The objective function of the policy network is an improvement to the nonlinear programming problem's objective function, as shown below:

$$F(\varpi) = E_{s_t}\left[F(s_t)\right] \tag{18}$$

where E_{s_t} is the expected function.

Through policy gradient optimisation, update the network parameters ϖ, as follows:

$$\nabla_{\varpi} F(\varpi) = E_{s,b}\left[\widehat{Q}_d(s,b)\nabla_\theta \ln d(b \mid s; \varpi)\right] \tag{19}$$

$$\widehat{\varpi} = \varpi + \eta \cdot \nabla_{\varpi} F(\varpi) \tag{20}$$

where $\widehat{Q}_d(s,b)$ is the action value function for breaking the solution space, which includes the improvement to the nonlinear programming problem's objective function in Section 2.1 and the expected reward for avoiding local optima in the future, directly related to the optimisation gains of nonlinear programming [23]. $\widehat{\varpi}$ represents the updated policy network parameters. η is the learning rate.

Step 4: Training the value network, updating parameters ϖ'. Using the solution space state s generated by the policy network and the local optima avoidance reward function $U^b_{ss'}$ as the training set, minimise the temporal difference (TD) error as follows:

$$\widehat{T}(\varpi') = \min\left[\left(U^b_{ss'} + \chi\Gamma_\pi(s', \varpi') - \Gamma_\pi(s, \varpi')\right)^2\right] \tag{21}$$

where χ is the discount factor.

The parameter update formula for the value network is as follows:

$$\widehat{\varpi'} = \widehat{\varpi} - \widehat{\eta} \cdot \widehat{T}(\varpi') \tag{22}$$

where $\widehat{\eta}$ is the learning rate for the value network.

2.3 Reinforcement Learning Training Method for Intelligent Avoidance of Nonlinear Programming Local Optima

To improve the accuracy of intelligent avoidance of nonlinear programming local optima, adaptive Q-learning is used to train the reinforcement learning network in Section 2.2. Integrating adaptive Q-learning into the reinforcement learning network, with the nonlinear programming problem's objective function in Section 2.1 as the core feedback, dynamically adjusts the parameter update logic of the policy and value networks [24], compensating for the shortcomings of policy gradient optimisation methods, and achieving more intelligent exploration of the solution space and local optima avoidance.

Based on the complexity ς of the nonlinear programming solution space, dynamically adjust the solution space breaking action selection strategy (nonlinear programming decision)

$$d(s,b) = \begin{cases} \arg\max_b \widehat{Q}(s,b), & \phi > 1-\varsigma \\ \text{randomselectionB}(s_i), & \phi \le \varsigma \end{cases} \tag{23}$$

where ϕ is the improved variable greedy rate, calculated as follows:

$$\phi = \frac{1}{\sqrt[3]{\tau}} \cdot \frac{\widehat{\xi} \cdot \varsigma}{10 \cdot \widehat{m}} \tag{24}$$

where τ is the number of iterations; $\widehat{\xi}$ is the dimension of the nonlinear programming decision variables; and $\widehat{m}$ is the total number of states in the solution space.

In a certain learning iteration τ, first determine the solution space breaking action selection strategy (nonlinear programming decision) for that round as

$$d(s,b) = \begin{cases} \arg\max_b \widehat{Q}(s,b), & \phi > 1 - \frac{\varsigma}{\sqrt[3]{\tau}} \\ \text{randomselectionB}(s_i), & \phi \le \frac{\varsigma}{\sqrt[3]{\tau}} \end{cases} \tag{25}$$

From the nonlinear programming solution space s_i with probability $\left(1 - \frac{\varsigma}{\sqrt[3]{\tau}}\right)$ execute the solution space

breakthrough action b_i with the maximum $\widehat{Q}(s,b)$ value and transfer to s_{i+1}. The update formula for $\widehat{Q}(s,b)$ is

$$\widehat{Q}(s_i,b_i) \leftarrow \widehat{Q}(s_i,b_i) + \eta[U_{i+1} + \theta \max_b \widehat{Q}(s_{i+1},b_i) - \widehat{Q}(s_i,b_i)] \quad (26)$$

To represent the influence of the learning result on the original $\widehat{Q}(s,b)$ value, as well as the $\widehat{Q}(s,b)$ values corresponding to other solution space breaking actions in $B(s_i)$, it is organised as follows:

$$\widehat{Q}(s_i,b_i) \leftarrow \begin{cases} (1-\eta)\,\widehat{Q}(s_i,b_i) + \eta\left[U_{i+1} + \theta \max_b \widehat{Q}(s_{i+1},b)\right], & s = s_i, b = b_i \\ \widehat{Q}(s_i,b_i), & \text{other} \end{cases} \quad (27)$$

From (27), it can be seen that the $\widehat{Q}(s,b)$ values corresponding to the unselected solution space-breaking actions remain unchanged. The learning result is $U_{i+1} + \theta \max_b \widehat{Q}(s_{i+1},b)$, but affected by the learning rate η. The learning outcome only causes $\eta\left[U_{i+1} + \theta \max_b \widehat{Q}(s_{i+1},b)\right]$ to change the original $\widehat{Q}(s,b)$ value, and the original $\widehat{Q}(s,b)$ value becomes $(1-\eta)\,\widehat{Q}(s_i,b_i)$. After τ iterations of learning, $\widehat{Q}(s,b)$ is updated as

$$\widehat{Q}(s_i,b_i) = (1-\eta)^{\tau}\,\widehat{Q}_i + (1-\eta)^{\tau-1}\,\eta\left(U + \theta\widehat{Q}_{i+1}\right) + (1-\eta)^{\tau-2}\,\eta\left(U + \theta\widehat{Q}_{i+1}\right) + \cdots + \eta\left(U + \theta\widehat{Q}_{i+1}\right) \quad (28)$$

organised as follows:

$$\widehat{Q}(s_i,b_i) = (1-\eta)^{\tau}\,\widehat{Q}_i + \left(U + \theta\widehat{Q}_{i+1}\right)\left[1 - (1-\eta)^{\tau}\right] \quad (29)$$

Learning rate η satisfies $0 < \eta < 1$, then $0 < (1-\eta) < 1$; when the number of iterations τ is large, $(1-\eta)^{\tau} \to 0$. As the initial greedy rate is ς, which already considers the complexity of the nonlinear programming solution space, it can better guide the solution space breaking action selection in the early stage of learning, without increasing learning time due to excessive exploration [25], nor falling into local optima due to insufficient exploration. When the number of iterations is large, the greedy rate tends to 0, and the nonlinear programming problem will stably select the solution space breaking action with the maximum $\widehat{Q}(s,b)$ value to reach the next nonlinear programming solution space state s_{i+1}. Therefore, when s_{i+1} is stable, $\widehat{Q}(s,b)$ will converge $U + \theta \max \widehat{Q}(s_{i+1},b)$, accelerating the learning convergence [26]. By substituting the adaptive Q learned $\widehat{Q}(s,b)$ value into (29), training the reinforcement learning network enables the intelligent avoidance of local optima in nonlinear programming.

The pseudocode for this method is as follows:

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from collections import deque
# ======================
# 1. Problem Definition
# ======================
class NonlinearProgrammingProblem:
def __init__(self, obj_func, constraints, uncertainty_params):
self.obj_func = obj_func # Objective function (deterministic after transformation)
self.constraints = constraints # List of constraint functions
self.uncertainty_params = uncertainty_params # Uncertain parameters (if any)
self.state_dim = len(uncertainty_params) # Dimension of state (problem instance features)
self.action_dim = len(constraints) # Dimension of action (adjustment to decision variables)
def get_initial_state(self):
# Return initial state (e.g., random feasible decision vector)
pass
def step(self, state, action):
# Apply action to state, return next_state, reward, done
next_state = self._apply_action(state, action)
reward = self._compute_reward(state, next_state)
done = self._check_termination(next_state)
return next_state, reward, done
def _apply_action(self, state, action):
# Adjust decision variables based on action (ensure feasibility)
pass
def _compute_reward(self, state, next_state):
# Reward = improvement in objective or constraint satisfaction
current_obj = self.obj_func(state)
next_obj = self.obj_func(next_state)
return next_obj - current_obj # Positive if improved
def _check_termination(self, state):
# Check if convergence or max steps reached
pass
# ======================
# 2. RL Agent with LSTM + GAT-NE
# ======================
class RLAgent:
def __init__(self, state_dim, action_dim):
# LSTM Encoder for state representation
self.lstm = nn.LSTM(input_size=state_dim, hidden_size=128, num_layers=2)
# GAT-NE Decoder for action generation
self.gat_ne = GATNEDecoder(input_dim=128, output_dim=action_dim)
# Policy and Value Networks
self.policy_net = nn.Linear(128, action_dim)
self.value_net = nn.Linear(128, 1)
self.optimizer_policy = optim.Adam(self.policy_net.parameters(), lr=1e-3)
self.optimizer_value = optim.Adam(self.value_net.parameters(), lr=1e-3)
self.memory = deque(maxlen=10000) # Replay buffer
def encode_state(self, state_sequence):
# Encode sequence of states using LSTM
lstm_out, (hidden, cell) = self.lstm(state_sequence)
```

```
    return hidden[-1] # Use last hidden state as representation
    def decode_action(self, encoded_state):
    # Generate action probabilities using GAT-NE
    return self.gat_ne(encoded_state)
    def select_action(self, state, epsilon=0.1):
    # Epsilon-greedy action selection
    if np.random.rand() < epsilon:
    return np.random.randint(self.action_dim)
    else:
    with torch.no_grad():
    encoded = self.encode_state(state.unsqueeze(0))
    action_probs = self.decode_action(encoded)
    return torch.argmax(action_probs).item()
    def update(self, batch):
    # Update policy and value networks using replay buffer
    states, actions, rewards, next_states, dones = batch
    # Encode states
    encoded_states = self.encode_state(states)
    encoded_next_states = self.encode_state(next_states)
    # Compute targets for value network
    with torch.no_grad():
    next_values = self.value_net(encoded_next_states)
    targets = rewards + (0.99 * next_values * (1 - dones))
    # Update value network
    values = self.value_net(encoded_states)
    value_loss = nn.MSELoss()(values, targets)
    self.optimizer_value.zero_grad()
    value_loss.backward()
    self.optimizer_value.step()
    # Update policy network (REINFORCE with baseline)
    advantages = targets - values.detach()
    action_log_probs = torch.log(self.decode_action(encoded_states))
    policy_loss = -(action_log_probs * advantages).mean()
    self.optimizer_policy.zero_grad()
    policy_loss.backward()
    self.optimizer_policy.step()
    # ======================
    # 3. GAT-NE Decoder (Simplified)
    # ======================
    class GATNEDecoder(nn.Module):
    def __init__(self, input_dim, output_dim):
    super().__init__()
    self.attention = nn.MultiheadAttention(embed_dim=input_dim, num_heads=4)
    self.fc = nn.Linear(input_dim, output_dim)
    def forward(self, x):
    # x: encoded state (batch_size, input_dim)
    attn_out, _ = self.attention(x.unsqueeze(0), x.unsqueeze(0), x.unsqueeze(0))
    return torch.softmax(self.fc(attn_out.squeeze(0)), dim=-1)
    # ======================
    # 4. Adaptive Q-Learning
    # ======================
    def adaptive_q_learning(agent, problem, episodes=1000):
    for episode in range(episodes):
    state = problem.get_initial_state()
    done = False
    episode_reward = 0
    while not done:
    # Adaptive epsilon-greedy
    epsilon = max(0.01, 1.0 / (1 + episode / 100))
    action = agent.select_action(state, epsilon)
    next_state, reward, done = problem.step(state, action)
    agent.memory.append((state, action, reward, next_state, done))
    # Update agent
    if len(agent.memory) > 64:
    batch = random.sample(agent.memory, 64)
    agent.update(batch)
    state = next_state
    episode_reward += reward
    print(f"Episode {episode}, Reward: {episode_reward}")
    # ======================
    # 5. Main Execution
    # ======================
    if __name__ == "__main__":
    # Define problem (example: production planning)
    def obj_func(x):
    return 5 * x[0] + 8 * x[1] # Profit from products A and B
    constraints = [
    lambda x: 2 * x[0] + x[1] <= 10, # Raw material 1
    lambda x: x[0] + 3 * x[1] <= 12, # Raw material 2
    lambda x: x[0] >= 0, lambda x: x[1] >= 0 # Non-negativity
    ]
    problem = NonlinearProgrammingProblem(obj_func, constraints, uncertainty_params=[])
    agent = RLAgent(state_dim=4, action_dim=len (constraints))
    # Train with adaptive Q-learning
    adaptive_q_learning(agent, problem, episodes=500)
```

3. Experimental Analysis

A bidirectional double-lane straight urban road with a length of 500 m is constructed using traffic simulation software. On one side of the road, two 5G base stations are evenly distributed, achieving full coverage of the entire segment. Each base station is equipped with a server with varying computing capabilities. Assume that during a certain period, 50 vehicles pass through this segment sequentially, each with at least one computing task to process. Each time slot generates a computing task, with task types following a zipf distribution. The distributed task offloading problem in this cellular vehicle network (C-V2X) is a mixed-integer nonlinear programming problem. Using the method proposed in this paper to perform local optimal solution intelligent avoidance, the global optimal solution is obtained. The objective function maximises the overall benefits of task offloading (including vehicle energy consumption at the user end, resource leasing costs, and network load balancing rate).

To ensure a more standardised and complete description of the experimental environment, the hardware environment used in this simulation experiment is as follows:

Table 1
Simulation Parameters of Task Offloading

Parameters	Parameter Value
The channel allocated by the base station to the vehicle	2
Bandwidth/GHz	3
The channels allocated by satellites to vehicles	70
Bandwidth/GHz	20∼200
Background noise power of satellite communication channel/dB	37,000
Task input data size/MB	6
Satellite orbit altitude/km	0.3∼1.5
Base station antenna height/m	1.9∼4.7
Vehicle calculation frequency/GHz	6∼7
The base station server calculates the frequency in GHz	-35
The satellite server calculates the frequency in GHz	0.5∼2.0
Reference path loss/dB	5
Vehicle signal transmission power/W	45∼75
Channel gain/dB	–110

the CPU adopts Intel Core i7-10700K model, the memory is 32 GB, and the GPU is Nvidia GeForce RTX 3080 model; In terms of software environment, the operating system version is Windows 10, developed based on Python 3.7 and PyTorch 1.11.0, and the simulation platform is Pycharm 2022. The design simulation parameters are shown in Table 1.

The hyperparameter settings for this experiment are shown in Table 2.

When performing local optimal solution avoidance of nonlinear programming using the method proposed in this paper, the effectiveness of the improved variable greedy rate in training the reinforcement learning network is analysed. The analysis results are shown in Fig. 1.

From the analysis of Fig. 1, it can be seen that when applying the improved variable greedy rate training method to the reinforcement learning network proposed in this paper, the local optimal avoidance reward function value is the largest, the fluctuation amplitude is small, and the convergence speed is also the fastest. This is because the improved variable greed rate can dynamically adjust the balance between exploration and utilisation based on the training process, maintain a high exploration ability in the early stages of training, fully explore the environment, and obtain more effective information; As the training progresses, gradually reduce the exploration ratio and increase the proportion of using known effective strategies to accelerate convergence. When the greed rate is fixed at 0.1, its local optimal avoidance reward function has the smallest value, the largest fluctuation amplitude, and the slowest convergence speed. This is due to the fixed low greed rate, which makes the agent overly conservative during the training process, lacking

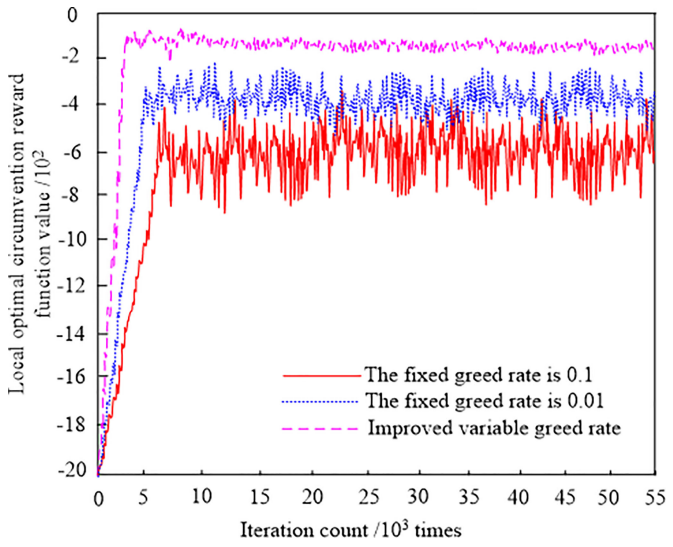


Figure 1. Training effect of the reinforcement learning network.

sufficient exploration and making it difficult to discover better strategies, resulting in poor training effectiveness. When the greed rate is fixed at 0.01, the value of the local optimal avoidance reward function increases, but it is still not high, and the fluctuation amplitude is also large, and the convergence speed is not fast. This is because although the greed rate is lower than 0.1, which reduces blind exploration to some extent, due to weak exploration ability, it still cannot effectively find the global optimal strategy. Experimental results have shown that, based on the analysis of the reward curve in the experimental log,

Table 2
Hyperparameter Setting Table

Hyperparameter	Parameter Values
Batch size	32
Discount factor	0.95
Strategy network learning rate	1×10^{-3}
Value network learning rate	1×10^{-3}
Maximum number of training epochs	1,000
Maximum number of steps per round	200
Target network update frequency	Updated every 10 training rounds
Experience replay buffer size	10,000
Minimum number of experience replay samples	1,000
Initial value of greed rate	0.1
Corruption coefficient of greed rate	0.999
Minimum value of greed rate	0.01

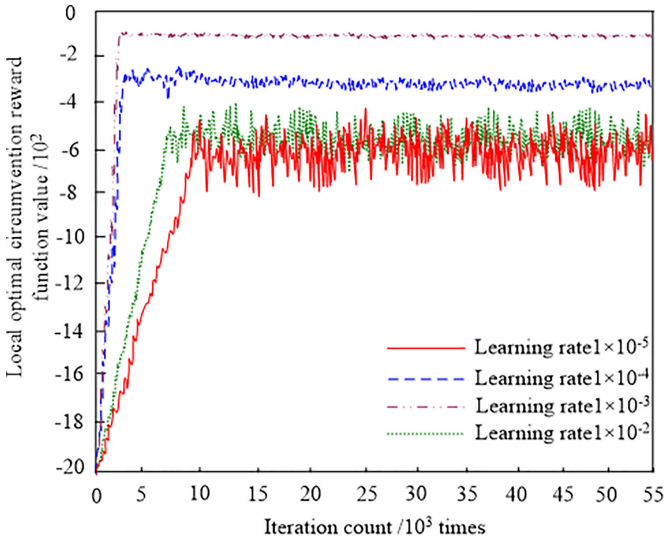


Figure 2. Training effects of reinforcement learning networks at different learning rates.

the method proposed in this paper applies an improved variable greedy rate to train the reinforcement learning network, which can accelerate the network training speed and achieve better network training effects.

Using the proposed method for intelligent avoidance of local optima in nonlinear programming, the learning rate directly affects the training of the reinforcement learning network. By analysing the convergence of the local optimal avoidance reward function under different learning rates, the optimal learning rate is selected. The analysis results are shown in Fig. 2.

According to the analysis of Fig. 2, when the learning rate is 1×10^{-2}, the convergence speed of the algorithm is slow, the reward value for convergence is small, and the fluctuation amplitude of the reward function is large. This is because the larger learning rate leads to excessive weight changes in the reinforcement learning network, and the agent is too aggressive in updating policies, making it difficult to stably learn effective policies, resulting in an unstable training process. When the learning rate is 1×10^{-3}, the algorithm converges the fastest, the reward value for convergence is the largest, and the fluctuation is the smallest. This indicates that the learning rate can update the network weights at an appropriate speed, and the agent can gradually learn the optimal strategy, achieving stable and efficient training. When the learning rate is 1×10^{-4}, the convergence speed of the algorithm slows down, the reward value for convergence decreases, and the fluctuation amplitude increases. This is because the learning rate is too low, the changes in network weights are too small, and each update has limited improvement on the strategy, requiring more time to update network parameters. At the same time, it also reduces the effective utilisation of empirical data, which is not conducive to improving the model's generalisation ability. When the learning rate is 1×10^{-5}, the algorithm converges the slowest, the reward value for convergence is the smallest, and the fluctuation amplitude is the largest. The excessively low learning rate makes the network training process extremely slow, making it difficult to achieve good training results in a reasonable amount of time. Based on the analysis of the reasons, combined with the loss convergence curve in the experimental log, when the learning rate is too high, it means that the weight changes of the reinforcement learning network are also too large, making it difficult for the agent to learn effective strategies; On the contrary, when the learning rate is too low, it means that the weight changes of the reinforcement learning network are too small, which requires more time to update the network parameters and

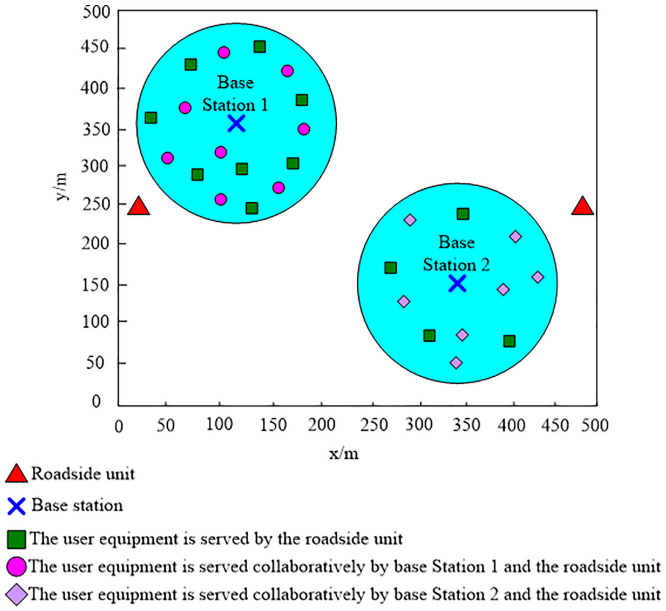


Figure 3. It shows the optimal location distribution results of base stations in the C-V2X network.

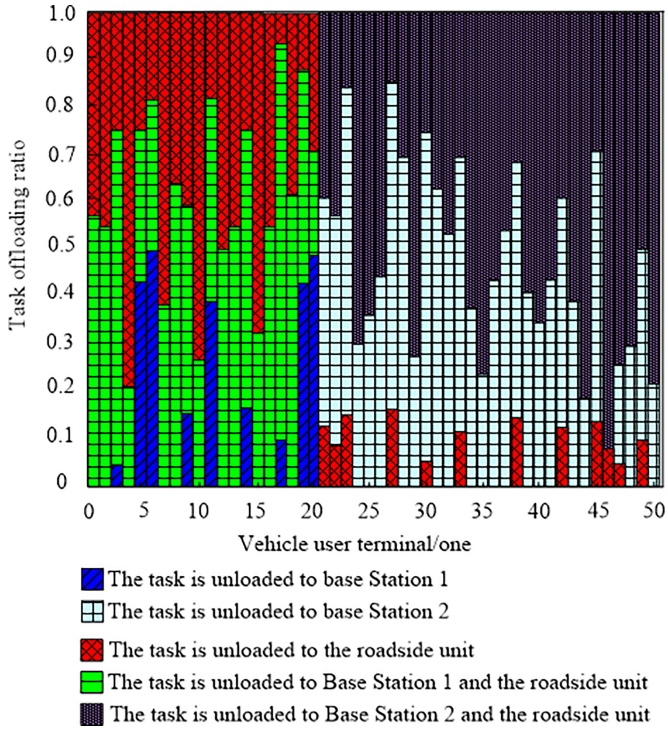


Figure 4. Optimal task offloading strategy of C-V2X network.

also reduces the effective utilisation of empirical data, which is not conducive to improving the generalisation ability of the model. In summary, setting the learning rate to 1×10^{-3} in this article can improve the training effectiveness of reinforcement learning networks.

Using the proposed method to perform local optimal solution intelligent avoidance for the nonlinear programming problem of C-V2X network task offloading, the global optimal solution is obtained, which is the optimal C-V2X network task offloading strategy, including the best base station placement distribution in the C-V2X network and the optimal task offloading strategy. The results of local optimal avoidance are shown in Figs. 3 and 4, where base station 1 covers 20 vehicles, and base station 2 covers 30 vehicles.

From the analysis of Fig. 3, it can be seen that each base station can be accurately deployed at the center of traffic flow hotspots (such as typical vehicle networking business demand concentration points, such as dense intersections of urban roads and high-speed service areas). From the records of vehicle communication within the coverage area of the base station in the experimental log data, it can be seen that in the context of the Internet of Vehicles, areas such as dense intersections of urban roads and high-speed service intervals exhibit high dynamic clustering characteristics in the spatial and temporal distribution of vehicles. There is frequent interaction between vehicles and between vehicles, as well as between vehicles and infrastructure services, which requires strict real-time and reliable requirements for computing offloading. Deploying base stations in such regional centers can minimise the average transmission distance from the vehicle terminal to the base station based on the advantage of geometric center location, enabling the base station to efficiently provide computation offloading services for vehicle user terminals. The light blue circular area in the figure represents the coverage range of the base station. Factors, such as the deployment height and power of the base station, affect its coverage range. Under the reasonable planning of the method proposed in this paper, base stations with location adaptation can achieve better coverage. Based on the records of base station signal strength and vehicle received signal quality in experimental logs, this method intelligently avoids nonlinear local optimal solutions and can determine the optimal position of the base station, providing stable offloading computing services for vehicle users.

From the analysis of Fig. 4, it can be seen that the method proposed in this paper can effectively and intelligently avoid the local optimal solution of the nonlinear programming problem of C-V2X network task offloading, obtain the optimal C-V2X network task offloading strategy, and form regional offloading division of labour. Two base stations cover two traffic hotspot areas, respectively, and base station 1 does not provide offloading services for vehicles within the coverage area of base station 2. From the records of vehicle task offloading in the experimental log, it can be seen that in the experiment, the first 20 vehicles traveling in area 1 were provided with computing offloading services by base station 1, and the last 30 vehicles in area 2 were served by base station 2. This strategy avoids cross-area same-frequency interference from the physical layer through spatial isolation, reduces the probability of resource competition, ensures the signal quality and transmission reliability of the offloading link, and meets the strict communication reliability requirements of the Internet of Vehicles. For large computational tasks, trigger multi-node joint offloading.

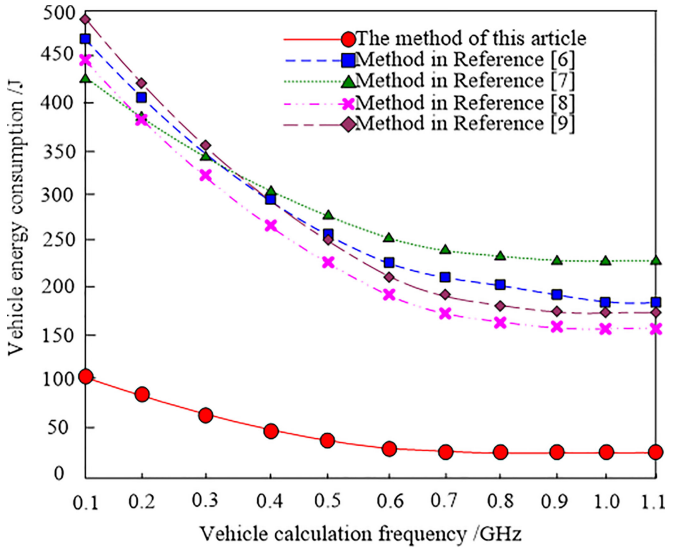


Figure 5. Changes in vehicle energy consumption.

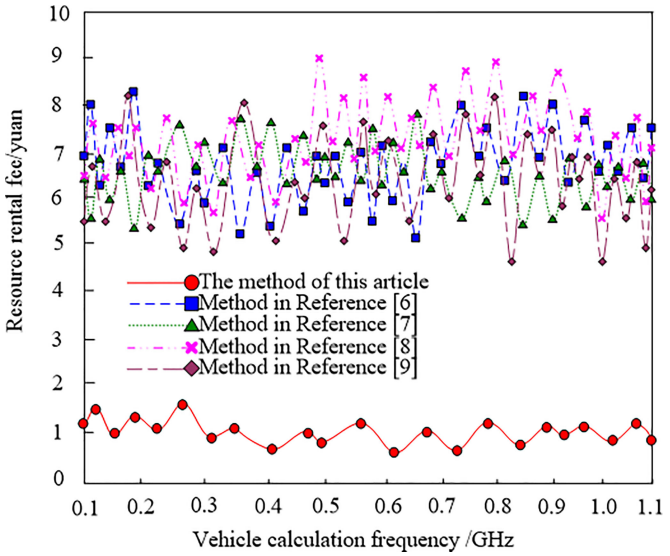


Figure 6. Changes in resource rental fees.

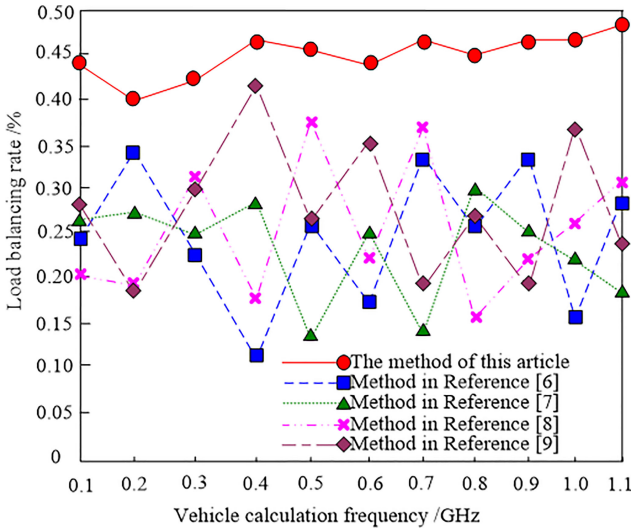


Figure 7. The variation of the load balancing rate.

The roadside unit takes on the near-field task preprocessing by virtue of the low delay characteristics of edge computing; Base stations rely on wide area resource scheduling capabilities to collaboratively complete complex computation offloading. This strategy decomposes large computational tasks into subtasks through task slicing and splitting, dynamically allocates communication resources for task offloading, which not only alleviates the bandwidth transmission pressure of the core network, but also compresses task processing latency through edge node computing power sinking, meeting the collaborative requirements of "low latency high computing power" for vehicle networking services. Based on the records of task offloading latency, resource utilisation, and other indicators in the experimental log, the experiment proves that the optimal task offloading strategy output by this method constructs an offloading system that adapts to the characteristics of C-V2X network through interference avoidance in the spatial domain and heterogeneous collaboration in the computational domain, providing technical support for the large-scale deployment of vehicle networking services with "low interference, low latency, and high resource efficiency".

Using the method in this paper, as well as the methods in [6]–[9], the local optimal solution of the nonlinear programming problem for task offloading in this C-V2X network is intelligently avoided. After applying the above five methods, the changes in vehicle energy consumption, resource leasing costs, and load balancing rate of the C-V2X network are analysed. The results are shown in Figs. 5– 7.

According to the analysis of Fig. 5, with the increase of vehicle computing frequency, the energy consumption of vehicles shows a decreasing trend after applying the above five methods. This indicates that a higher computing frequency means that vehicles can handle more tasks per unit time, and the efficiency of task processing is improved, thereby reducing the additional energy consumption caused by task backlog. At the same time, the increase in computing frequency enables vehicles to allocate resources more reasonably when processing tasks, avoiding excessive occupation or waste of resources, thereby reducing overall energy consumption. From the detailed records of vehicle energy consumption in the experimental log, it can be seen that under different vehicle calculation frequencies, the vehicle energy consumption of our method is significantly lower than the other four methods. The highest vehicle energy consumption of our method is around 100 J, while the highest vehicle energy consumption of the methods in [6]–[9] are around 460, 420, 450, and 490 J, respectively. This significant difference fully demonstrates the excellent performance of our method in reducing vehicle energy consumption. The reason is that in the C-V2X network task offloading nonlinear programming problem, the other four methods are prone to getting stuck in local optima and cannot obtain the globally optimal task offloading strategy. The reinforcement learning network proposed in this article has strong adaptive and exploratory capabilities, which can

Table 3
Results of Ablation Experiment

Model	Training Time Consumption (s)	95% Confidence Interval (s)	VRAM Usage (MB)	95% Confidence Interval (MB)
Baseline model	825	[800, 850]	1250	[1200, 1300]
Ablation model 1	980	[950, 1010]	1420	[1380, 1460]
Ablation model 2	920	[890, 950]	1350	[1300, 1400]
Ablation model 3	880	[850, 910]	1300	[1250, 1350]

continuously interact with the environment, adjust its own strategy based on the reward signals feedback from the environment, intelligently avoid local optimal solutions, dynamically select the optimal task unloading nodes and unloading timing, and thus obtain the globally optimal task unloading strategy. This globally optimal task offloading strategy can effectively reduce the energy consumption of vehicles during task processing.

According to the analysis of Fig. 6, it can be seen from the detailed records of changes in resource leasing costs in the experimental logs at different vehicle calculation frequencies that the method proposed in this paper has the smallest fluctuation range in resource leasing costs, mainly fluctuating between 0 yuan and 2 yuan. The methods proposed in [6]–[9] have relatively high fluctuation ranges in resource leasing costs, fluctuating between approximately 5 yuan and 8.4 yuan, and between 4.9 yuan and 8.2 yuan, respectively. At different vehicle calculation frequencies, the resource leasing cost of this method is significantly lower than the other four methods. This result fully demonstrates that the method proposed in this paper has significant advantages in reducing resource leasing costs and stabilising cost fluctuations. Experimental results have shown that this method effectively reduces resource leasing costs and stabilises cost fluctuations by using reinforcement learning to intelligently avoid local optimal solutions in nonlinear programming problems. However, the methods in [6]–[9] lack global optimisation performance, resulting in significant fluctuations and high costs in resource leasing costs.

From the analysis of Fig. 7, it can be seen that under different vehicle calculation frequencies, the load balancing rate of our method has the smallest fluctuation amplitude and the highest load balancing rate, always above 0.4%, which is significantly higher than the other four methods. At the same time, the load balancing rates of the methods in [6]–[9] have large fluctuations, and the load balancing effect is poor, making it difficult to maintain stable performance in complex and variable vehicle calculation frequency environments. This result fully demonstrates the significant advantages of our method in achieving efficient load balancing. Based on the records of task allocation strategy adjustments in the experimental log, this method adopts reinforcement learning, which can break through the limitations of static task allocation strategies. In the C-V2X network, the vehicle computing frequency will change in real time, and reinforcement learning can perceive these changes in real time and quickly adjust the task allocation strategy to ensure that the load of each vehicle is always in a relatively balanced state, effectively reducing the fluctuation amplitude of load balancing rate and improving the load balancing effect. However, the methods in [6]–[9] cannot adapt to the real-time changes in vehicle calculation frequency, which can easily lead to some vehicles being overloaded while others are overloaded, resulting in significant fluctuations in load balancing rate. Experimental results have shown that the method proposed in this paper can effectively improve the load balancing rate of C-V2X networks.

To further evaluate the independent contributions of each component in the method proposed in this article, ablation experiments were designed. Set the method model in this article as the baseline model; remove the LSTM component and only retain GAT-NE and adaptive Q-learning as ablation model 1; remove the GAT-NE component, retaining only LSTM and adaptive Q-learning as ablation model 2; and remove the adaptive Q-learning component, retaining only LSTM and GAT-NE as ablation model 3. Using training time consumption and memory usage as evaluation indicators, and conducting significance tests using analysis of variance (ANOVA), with a 95% confidence interval provided. Each model is trained in the same experimental environment, using the same simulation parameters and task offloading scenarios. Each model is independently run 10 times to obtain more statistically significant experimental results. The experimental results are shown in Table 3.

According to Table 3 analysis, removing any component will result in a decrease in model performance. Compared with the benchmark model, ablation model 1 (excluding LSTM components) has the longest training time consumption, at 980 s, with a 95% confidence interval of [950, 1010] s. It also has the highest memory usage, at 1420 MB, with a 95% confidence interval of [1380, 1460] MB; Ablation model 2 (removing GAT-NE components) is the second best; The ablation model 3 (removing the adaptive Q-learning component) has a relatively small impact, but still falls short of the baseline model. According to the significance test of analysis of variance (ANOVA), it was found that there were significant differences ($p < 0.05$) in training time consumption and video memory usage between each ablation model and the baseline model, indicating that LSTM, GAT-NE, and adaptive Q-learning components all contribute significantly to the

model performance and work together to achieve better results for the baseline model.

4. Conclusion

Nonlinear programming problems often exhibit characteristics, such as multiple peaks, non-convexity, and high dimensionality, making it easy to fall into local optima and difficult to find the global optimum. Reinforcement learning, as an intelligent decision-making method, possesses strong adaptability and exploration capabilities, enabling it to adjust strategies based on feedback signals to find optimal solutions in complex environments. To this end, this research focuses on an intelligent avoidance method for local optima in nonlinear programming based on reinforcement learning. In practical applications within cellular vehicular networks, task offloading is a complex nonlinear programming problem. Applying the proposed method can provide better solutions for task offloading in cellular vehicular networks and improve network performance.

The innovations of this method are as follows.

1) *LSTM-GAT-NE Encoder–Decoder Structure:* A reinforcement learning network structure with LSTM as the encoder and GAT-NE as the decoder is designed. LSTM can extract the temporal dependencies and global features of the solution space states in nonlinear programming, providing historical decision memory and global constraint guidance for the decoder; GAT-NE can incorporate the output of LSTM, mine constraint correlations based on graph structures, and generate more accurate probability distributions for breaking the solution space deadlock actions, effectively enhancing the ability to avoid local optima.
2) *Adaptive Q-Learning Integrated with Reinforcement Learning:* Adaptive Q-learning is incorporated into the reinforcement learning network to dynamically adjust the strategy for selecting actions that break the solution space deadlock. It adaptively balances exploration and exploitation based on the complexity of the nonlinear programming solution space, improving the intelligence and convergence speed of the reinforcement learning network.

Although current research is mainly based on the specific scenario of task offloading in cellular connected vehicles (C-V2X), given that nonlinear programming problems generally have common characteristics, such as multimodality, non-convexity, and high dimensionality, and the LSTM-GAT-NE encoder–decoder structure and adaptive Q-learning and reinforcement learning fusion method proposed in this paper focus on solving the core problem of intelligent avoidance of local optimal solutions in nonlinear programming, this method has the potential to be transferred to other types of nonlinear programming problems (such as energy scheduling and path optimisation) at the theoretical level. However, its specific applicability still needs to be further verified through experiments in the future.

In addition, there may be some potential risks and limitations when applying the reinforcement learning-based intelligent avoidance method for local optimal solutions in nonlinear programming proposed in this paper. On the one hand, there is a risk of overfitting in the model. If the training data are insufficient or the data distribution is limited, the reinforcement learning network may overfit to specific patterns in the training data, resulting in insufficient generalisation ability in new nonlinear programming problem scenarios and inability to effectively avoid local optimal solutions. On the other hand, in complex nonlinear programming problems, the model may experience convergence oscillations. Due to the multimodal and non-convex nature of the solution space, it is difficult to accurately control the balance between exploration and utilisation in the fusion process of adaptive Q-learning and reinforcement learning. This may lead to frequent fluctuations in the selection strategy for action breaking in the solution space, making the convergence process unstable and affecting the efficiency and accuracy of finding the global optimal solution.

Acknowledegment

This work was supported by the Chongqing Higher Education Society 2023–2024 Higher Education Scientific Research Project: "Research on Digital Practice of Higher Vocational Mathematics Courses Based on OBE Teaching Concept under the Background of New Engineering" (Project Number: cqgj23061B).

References

[1] R. Zhang and Y. Xiao, A nonlinear programming algorithm for finite element limit analysis using feasible arc searching technique, *International Journal for Numerical Methods in Engineering*, *124*(22), 2023, 5102–5119.

[2] E. H. Kassem, Second-order symmetric duality for multiple objectives nonlinear programming under generalizations of cone-preinvexity functions, *Journal of Scientific Computing*, *95*(1), 2023, 95.7.1–95.7.15.

[3] C. Chairatsiripong, D. Yambangwai, and T. Thianwan, New iterative methods for nonlinear operators as concerns convex programming applicable in differential problems, image deblurring, and signal recovering problems, *Mathematical Methods in the Applied Sciences*, *46*(3), 2023, 3332–3355.

[4] K. Halbig, L. Hümbs, F. Rsel, L. Schewe, and D. Weninger, Computing optimality certificates for convex mixed-integer nonlinear problems, *INFORMS Journal on Computing*, *36*(6), 2024, 1579–1610.

[5] L. Zheng and P. Liu, Robust nonlinear decision mapping approach for online bus speed control under uncertainty, *Computer-Aided Civil and Infrastructure Engineering*, *39*(2), 2024, 203–221.

[6] A. R. Escobedo, Exact matrix factorization updates for nonlinear programming, *INFORMS Journal on Computing*, *36*(1), 2024, 245–265.

[7] R. M. Rizk-Allah, I. M. Eldesoky, E. A. Aboali, and S. M. Nasr, Heap-based optimizer algorithm with chaotic search for nonlinear programming problem global solution, *International Journal of Computational Intelligence Systems*, *16*(1), 2023, 1–21.

[8] J. Zhang, A. C. Trapp, and S. Butenko, Solving a class of two-stage stochastic nonlinear integer programs using value functions, *Journal of Global Optimization*, *91*(1), 2025, 129–153.

[9] S. Na, M. Anitescu, and M. Kolar, A fast temporal decomposition procedure for long-horizon nonlinear dynamic programming, *Mathematics of Operations Research*, *49*(2), 2024, 1012–1044.

[10] L. J. Wang, Z. W. Zhu, Y. P. Zhu, and C. Y. Zhu, A multi-policy multi-objective reinforcement learning algorithm based on voting method, *Computer Simulation*, *40*(4), 2023, 341–345, 406.

[11] D. F. Jazzaa and A. J. Abdulqader, Numerical solution for solving fuzzy volterra integral 2nd -kind by using modified decomposition technique, *Journal of Interdisciplinary Mathematics*, *26*(7), 2023, 1583–1590.

[12] D. Skulj, Application of normal cones to the computation of solutions of the nonlinear kolmogorov backward equation, *International Journal of Approximate Reasoning*, *158*, 2023, 108919.1–108919.21.

[13] Y. Jong, Y. Kim, H. Kim, and S. Butenko, A method based on parametric convex programming for solving convex multiplicative programming problem, *Journal of Global Optimization*, *90*(3), 2024. 573–592.

[14] T. Okuno and M. Fukushima, Primal-dual path following method for nonlinear semi-infinite programs with semi-definite constraints, *Mathematical Programming*, *199*(1–2), 2023, 251–303.

[15] T. Bacci, A. Frangioni, C. Gentile, and K. Tavlaridis-Gyparakis, New mixed-integer nonlinear programming formulations for the unit commitment problems with ramping constraints, *Operations Research*, *72*(5), 2024, 2153–2167.

[16] S. Na, M. Anitescu, and M. Kolar, Inequality constrained stochastic nonlinear optimization via active-set sequential quadratic programming, *Mathematical Programming*, *202*(1–2), 2023, 279–353.

[17] A. Gharaei, A. Amjadian, A. Shavandi, A. Amjadian, and M. T. Thai, An augmented lagrangian approach with general constraints to solve nonlinear models of the large-scale reliable inventory systems, *Journal of Combinatorial Optimization*, *45*(2), 2023, ARTN 78–ARTN 114.

[18] J. S. Rodriguez, R. B. Parker, C. D. Laird, B. L. Nicholson, J. D. Siirola, and M. L. Bynum, Scalable parallel nonlinear optimization with pynumero and parapint, *INFORMS Journal on Computing*, *35*(2), 2023, 509–517.

[19] S. Gao, X. Li, and Y. Liu, Deep learning-based semantic segmentation and recognition for agricultural robots, *Mechatronic Systems and Control*, *5*, 2025, 1–9.

[20] H. Zhang, Z. Wen, S. Bi, and W. Zhao, Trilayer stackelberg game for nonlinear systems using adaptive dynamic programming, *Journal of the Franklin Institute*, *360*(3), 2023, 1523–1539.

[21] K. Bestuzheva, A. Chmiela, F. Serrano, S. Vigerske, F. Wegscheider, and S. Butenko, Global optimization of mixed-integer nonlinear programs with SCIP 8, *Journal of Global Optimization*, *91*(2), 2025, 287–310.

[22] J. Zhang, Y. Cheng, S. Cao, and X. Wang, Offline reinforcement learning with reverse diffusion guide policy, *IEEE Transactions on Industrial Informatics*, *20*(10), 2024, 11785–11793.

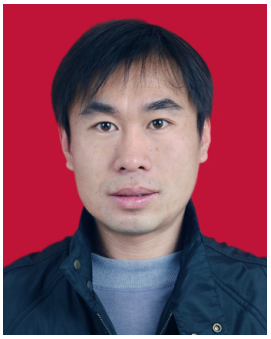
[23] X. Xu, R. Li, Z. Zhao, and H. Zhang, The gradient convergence bound of federated multi-agent reinforcement learning with efficient communication, *IEEE Transactions on Wireless Communications*, *23*(1), 2024, 507–528.

[24] H. Yang, K. Lin, T. Yang, and G. Sun, Score: Simple contrastive representation and reset-ensemble for offline meta-reinforcement learning, *Knowledge-Based Systems*, *309*, 2025, 1.1–1.10.

[25] Y. Zhang and M. M. Zavlanos, Cooperative multiagent reinforcement learning with partial observations, *IEEE Transactions on Automatic Control*, *69*(2), 2024, 968–981.

[26] M. Guoying, S. Yingbo, and W. Huiqin, Multi-agent cooperative reinforcement learning algorithm based on prioritized value network, *Control Engineering of China*, *32*(4), 2025, 691–698.

Biographies



Guodong Wang received the degree in applied mathematics from Jiangsu University in 2007. He is currently a Professor with the General Education College, Chongqing Water and Electric Power Vocational College. His main research areas include nonlinear programming and stability analysis of nonlinear systems.