Ant Colony Algorithm and Fuzzy Neural Network-based Intelligent Dispatching Algorithm of An Elevator Group Control System

Jianchang Liu
Key Laboratory of Integrated Automation of Process Industry, Ministry of Education, Northeastern University
Shenyang, China
liujianchang@ise.neu.edu.cn

Yiyang Liu1,2
1. Robotics Laboratory. Shenyang Institute of Automation, Chinese Academy of Sciences
Shenyang, China
liuyiyang@sia.cn
2. Graduate School of the Chinese Academy of Sciences

Abstract—To improve the performance of elevator group control systems (EGCS), an intelligent dispatching method based on ant colony algorithm and fuzzy neural network is presented. An elevator group control system based on fuzzy neural network adapts to various traffic flow modes. Using ant colony algorithm to optimize the weights of fuzzy neural network before training with BP algorithm can solve the problem that convergence of weights is easy to be trapped in local optimal values when trained just with BP algorithm. This intelligent dispatching algorithm makes the weights of fuzzy neural network more precise and reasonable. These weights greatly affect the performance of an EGCS. The results of simulation show that ant colony algorithm and fuzzy neural network greatly improves the performance of an EGCS. Its average waiting time is obviously shorter than that of the EGCS that is only based on fuzzy neural network.

Keywords-elevator group control; ant colony algorithm; fuzzy neural network; intelligent dispatching

I. INTRODUCTION

An elevator group control system (EGCS) is a kind of complex nonlinear discrete control system that systematically manages three or more elevators in a building in order to efficiently transport passengers. An EGCS consists of hall call buttons, car call buttons, elevators, and a group controller in which there is group control dispatching algorithm, the most important part of an EGCS. If a passenger wants to go to another floor, he or she presses a direction (hall call) button and waits for an elevator to arrive; in this case, the group controller selects the most appropriate elevator in the group, depending on the current positions, moving directions, current passengers' number in every elevator. Then the passenger enters the elevator and presses a floor (car call) button in the elevator. The objective of an EGCS is to provide passengers with high-quality service.

The most important part of an EGCS is the dispatching algorithm, which is also called an elevator group control dispatching algorithm. The algorithm is very complex for the following reasons. First, if an EGCS manages s elevators and dispatches p hall calls to the elevators, the algorithm considers s×p×p cases. Second, the algorithm must consider the hall calls that will be generated in the near future. Third, the algorithm must consider many uncertain factors, such as the number of passengers at the floors where hall calls and car calls are generated. Fourth, it must be possible for a system manager to change the control strategy. Some managers want to operate the system to minimize passenger waiting time while others want to reduce the power consumption.

Many studies have been done and significant progress has been made regarding the algorithm to dispatch hall calls. The methods in [1], [2], [3], [4], and [5] are fuzzy control-based dispatching algorithms regarding not only strong stochastic and discrete features of an EGCS but also the robustness and easy design (without accurate models) of fuzzy control. But many parameters of membership function and fuzzy rules that depend on experience of experts are difficult to confirm. The methods in [6] and [7] adopt neural network to control an EGCS. The studies in [8] and [9] are based on fuzzy neural network (FNN). Fuzzy control can adapt to short-term changes of traffic situations in an EGCS, while the neural network can adapt to long-term changes of traffic situations. When the weights of FNN are trained, the back propagation (BP) algorithm is adopted. This algorithm searches for the optimal values of weights along the gradient of the function, so it is unavoidable to be trapped in local optimal values, and the searching results depend on the initial values, which are randomly set at the beginning of the searching. Many weights in FNN stand for the parameters of membership functions that represent the experience of experts. If these weights are not optimal, that is, if optimal experts experience is not adopted, the performance of EGCS will be greatly affected.

To solve the above problem, this paper presents ant colony algorithm (ACA) and FNN-based elevator group control dispatching algorithm. This intelligent algorithm takes advantage of distributed computing and global search of ACA to confirm the initial values of weights in FNN that represent the parameters of membership functions before the weights are trained by BP algorithm. This intelligent dispatching algorithm helps the weights of FNN avoid being trapped in local optimal values and obviously improves the performance of an EGCS.
II. AN EGCS BASED ON ACA AND FNN

A. Structure of an EGCS based on ACA and FNN

The structure of the EGCS based on ACA and FNN is shown as Figure 1.

![Diagram of EGCS structure](image)

Figure 1. Structure of an elevator group control system based on ant colony algorithm and fuzzy neural network.

In Figure 1, HCWT is the ith elevator's estimated time of arrival to respond to the new hall call (i = 1, 2, 3, ...; e is the number of elevators in the group); Ni is the number of passengers in the ith elevator; AWT is the average waiting time of the ith elevator.

To improve the performance of an EGCS, the dispatching algorithm of this EGCS is based on ACA and FNN, taking the advantage of distributed computing and global search of ACA to confirm the initial values of weights in network before the weights are trained by the BP algorithm. These weights represent the parameters of FNN, such as parameters of membership functions. The performance of an EGCS is measured by several criteria, such as the average waiting time (AWT) of passengers, the average traveling time of passengers, the percentage of passengers waiting more than 60 seconds, and power consumption. In these criteria, the AWT is so important that people pay more attention to it; thus this paper uses AWT as the criterion to measure the performance of this EGCS.

As shown in Figure 1, this EGCS consists of four main modules: simulation of passengers, dispatching algorithm of an EGCS, simulation of elevators in the group, and a computing module. Simulation of passengers generates virtual passengers' information, which includes the arrival time, initial floors, and object floors of new passengers waiting outside the elevators. In the dispatching algorithm, the algorithm based on ACA and FNN calculates the AWT of every elevator in the group, then dispatches the new hall call to the elevator whose AWT is minimal. Simulation of elevators in the group consists of several subprograms, each of which is a virtual elevator and simulates the running process of a real elevator. The computing module is used to calculate the HCWT and Ni of each elevator, according to the initial floors, object floors of new passengers, and information on each elevator's situation.

HCWT and Ni (i = 1, 2, 3, ...; e) are the inputs of dispatching algorithm. Ni is recorded in relevant variable of the ith elevator, while calculating HCWT, is quite difficult, its process is as follows (suppose there are n calls to be responded for the ith elevator before the new hall call is responded by this elevator):

\[ HCWT_i = \sum_{j=1}^{n}(T_j + S_j) \]  (1)

Where

- \( T_j \) — time spent in running process in which the \( i \)th elevator responds to the \( j \)th call that has been dispatched to the elevator before it responds to the new hall call;
- \( S_j \) — time spent in stopping process in which the \( i \)th elevator responds to the \( j \)th call that has been dispatched to the elevator before it responds to the new hall call;

\[ S_j = t_{open} + t_{close} + t_{move}(N_i + N_{out}) \]  (2)

Where

- \( t_{open} \) — time spent in opening the doors;
- \( t_{close} \) — time spent in closing the doors;
- \( t_{move} \) — time that each passenger spends to get on/off the elevators;

\( N_i \) — number of passengers that get on an elevator when the elevator stops;

\( N_{out} \) — number of passengers that get off an elevator when the elevator stops.

Calculating \( T_j \) is very difficult. This variable depends on the distance an elevator needs to travel in the near future. There are different velocity curves corresponding to different distance that elevators will travel, thus, there are many expressions of \( T_j \). They will not be discussed in detail here.

B. ACA and FNN-based dispatching algorithm of an EGCS

AWT is one of the most important criteria to measure the performance of an EGCS. Depending only on HCWT and \( N_i \), AWT is difficult to describe accurately. Thus, this paper takes advantage of FNN to get the values of AWT. Two inputs of FNN are HCWT and \( N_i \), and output is AWT. The algorithm uses FNN to get AWT of every elevator, once a hall call occurs, then dispatches the hall call to the elevator whose value of AWT is minimal. After this, the task of dispatching algorithm is complete. This elevator arranges the hall call newly received and other calls received before, according to the rule of single elevator running, and runs automatically. Other elevators run automatically according to the sequences of respective calls received before.

1) FNN-based dispatching algorithm of an EGCS

FNN-based dispatching algorithm of an EGCS is shown in Figure 2.

The first layer is the input layer, in which the white blank circles represent that there are no disposals to input signals. The number, one, in the rectangles stands for constant input. The weights connecting the rectangles and the second layer are \( w_i \) (or \( w_{ij} \)), while the weights connecting the circles and the second layer are 1.

The second layer is the offset layer containing eight nodes in which there are two inputs, \( x_i \) or \( x_j \) (also: HCWT or \( N_i \)) and \( w_i \) (or \( w_{ij} \)). In each node there is \( x_i - w_i \) (please see equi
The weights connecting the second and third layer are \( w_i \).

The third layer is the S-function layer containing eight nodes, in which there is one input, \( w_i(x_i - w_i) \), and the value of S-function \( \left\{ 1 + \exp\left[-w_i(x_i - w_i)\right]\right\} \) is calculated. The weights connecting the third and fourth layer are 1 or -1.

The fourth layer contains six nodes. There is addition in node two and five, but there are no disposals in other four nodes. The outputs of six nodes are degrees of membership, \( A_i(x_i)_{i=1,2,3,j=1,2} \). The weights connecting the fourth and fifth layer are 1. Now the task to make the inputs fuzzy is complete.

The fifth layer is rule layer containing nine nodes that respectively represent nine rules, in which two inputs are multiplied and then average active degrees are calculated.

The sixth layer containing one node and nine inputs is used to make variables not fuzzy. These inputs are added, and the result is output.

Three fuzzy sets \( A_i, A_j, \; A_k \; (j=1,2) \) are defined for each input variable in the network. For easy definition, we use \( x_i \) and \( x_j \) to respectively represent HCWL and \( N_i \); use \( A_i(x_i)_{i=1,2,3,j=1,2} \) to represent the \( j \_ \text{th} \) fuzzy set of the \( i \_ \text{th} \) input variable, \( x_i \), and the membership function of this fuzzy set. Because BP algorithm is to be used, the membership function must be differentiable. Sigmoid function is adopted here. Then membership of each fuzzy set is respectively:

\[
A_i = \frac{1}{1 + \exp\left[-w_i(x_i - w_i^j)\right]} \quad (3)
\]

\[
A_j = \frac{1}{1 + \exp\left[-w_j(x_j - w_j^i)\right]} \quad (4)
\]

\[
A_k = \frac{1}{1 + \exp\left[-w_k(x_k - w_k^j)\right]} + \frac{1}{1 + \exp\left[-w_k(x_k - w_k^j)\right]} \quad (5)
\]

In the equations, \( w_i \) and \( w_j \) are centers of S-functions, and the pitch of S-function is according to \( w_i \).

Because there are three fuzzy sets for either \( x_i \) or \( x_j \), there are nine rules, whose general formation is:

\[
R_i: \text{if } x_i \text{ is } A_{i1} \text{ and } x_j \text{ is } A_{j1}, \text{ then } y = \bar{y}_i \quad (6)
\]

Training process of network is shown as the following procedures:

First, the algorithm uses ACA to optimize the weights that represent the parameters of FNN (please see section B 2) before training the weights with BP algorithm, because actually BP algorithm converges along the gradient, the weights are unavoidable to be trapped in local optimal values. If the weights that represent the parameters of membership function are not globally optimal, the performance of control will be greatly affected. Therefore it is necessary to optimize the weights with ACA first to get the initial values of weights, then to train the optimized weights with BP algorithm, thus the problem that convergence is easily to be trapped in local optimal values will be avoided effectively.

Second, the algorithm trains the weights with BP algorithm. In the process of propagation forward, input signals are handled through hidden layers, and toward the output layer. If ideal output is not achieved in the output layer, the procedure turns to back propagation. The errors of output are propagated backward along the original path, in which the weights are modified to make the errors of output minimal. Evaluation function of output errors is

\[
E = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \bar{y}_i)^2 \quad 1 \leq i \leq n \quad (7)
\]

Where \( y_i, \bar{y}_i \) respectively represents the expected output vector and actual output vector of the \( t \_\text{th} \) sample \((n \; \text{is the number of samples})\).

From output layer to input layer, use the method of propagation errors back to calculate \( \frac{\partial E}{\partial w} \) of each layer. Assume \( w \) is an adjustable parameter of some layer, the learning rule is

\[
w(t+1) = w(t) - \eta \frac{\partial E}{\partial w} + \alpha \Delta w(t) \quad (8)
\]

\[
\Delta w(t+1) = -\eta \frac{\partial E}{\partial w} + \alpha \Delta w(t) \quad (9)
\]

Where

- \( \tau \) — time to start learning, initial value is 0;
- \( w(0) \) — initial value of weights, calculated by ACA;
- \( \eta > 0 \) — learning rate (0.1 here);
- \( \alpha > 0 \) — momentum factor (0.1 here), which represents the influence that changes of weights last cycle make on that of weights this cycle.

2) Optimizing weights of FNN with ACA

Ant Colony Algorithm (ACA) is presented by Italian scholar M. Dorigo et al.[10,11], and is primarily used to solve the traveling salesman problem (TSP). After achieving excellent performance on TSP, ACA has been adopted to other areas, such as designing large scale integrated circuit, routing in telecommunications network, and vehicle scheduling. Ant Colony Algorithm has been applied in many types of problems successfully, especially in combinatorial optimization problems. These successful applications primarily benefit from the following features of ACA:
ACA combines distributed computing, positive feedback and greedy searching, which provide strong abilities to search optimal solution for ACA. Positive feedback helps the algorithm find optimal solution quickly, distributed computing prevents the algorithm from procecity convergence, and greedy search decreases the searching time;

- ACA has good parallelity;
- ACA has strong robustness. It can be applied to other problems after modifying the model of ACA a little. Failures of some individuals in ACA will not affect the solution of whole problem;
- ACA is easy to be combined with other heuristic algorithms to improve the performance of the algorithm combined.

This paper takes the above advantages to optimize the weights that represent the parameters of FNN, such as that of membership function. This method effectively solves the problem that convergence of weights is greatly affected by initial values of weights, and is also easily trapped in local optimal values. Thus this method is able to help the system achieve optimized FNN with more precise values of weights.

The assumption of optimizing the weights of FNN with ACA is as follows:

Assume there are $m$ weights to be optimized. The first step is to set each weight $w_i (1 \leq i \leq m)$ to $N$ random numbers (not zero), which form a set $\Omega_0$. Ant colony search food from nest, then each ant selects a number in each set $\Omega_0$ (each number stands for a value of one weight), thus each ant selects a group of weights to be optimized. The number of ants is $h$, $\tau_i(\Omega_w)$ is the pheromone of the $j_{th}$ element, $w_i(\Omega_w)$ of the set $\Omega_w (1 \leq i \leq m)$. Every ant selects the element in set independently. Each ant begins searching from the set $\Omega_w$, selects an element from every set, $\Omega_w$, according to the pheromone of each element (please see equation (10)). After selecting elements from all sets, one ant achieves the food resource. The next step is to modify the pheromone of all elements in every set. The two steps are done cyclically, until all the paths selected by ants converge to one path, or the cycle reaches the given cycles times.

The procedures of optimizing the weights of FNN with ACA are as follows:

- Step one, set time $t$ and cyclical times $NC$ to zero, set maximal cyclical times $NC_{max}$, let pheromone of all elements in every set be $\tau_j(\Omega_w)(t)=C$, and $\Delta \tau_j(\Omega_w)=0$, at this time, all ants are in the nest.
- Step two, enable all ants, for ant $k (k=1,..,h)$, select the element of $\Omega_w$ according to rules of selecting paths. Rules of selecting paths: for set $\Omega_w$, ant $k$ selects the $j_{th}$ element according to the following probability function:

$$ \text{Prob}(x_j(\Omega_w)) = \frac{\tau_j(\Omega_w)}{\sum_j \tau_j(\Omega_w)} $$

- Step three, repeat step two, until all the ants achieve the food resource.
- Step four, let $t \leftarrow t+MNC \leftarrow NC+1$, calculate output errors of FNN with weights selected by each ant, record current optimal solution, and update pheromone of every element according to the following rule.

Rule: as time passed, original pheromone decreases gradually, using parameter $\rho (0 < \rho < 1)$ to represent the persistence of pheromone, so $1-\rho$ represents the degree of decrease. After $m$ time units, an ant achieves food resource from the nest, the pheromone of elements in every path must be adjusted by following equation:

$$ \tau_j(\Omega_w)(t+MNC) = \rho \tau_j(\Omega_w)(t) + \Delta \tau_j(\Omega_w) $$

Where $\Delta \tau_j(\Omega_w)$ denotes the pheromone of the $j_{th}$ element $w_j(\Omega_w)$ in set $\Omega_w$ left by the $k_{th}$ ant in this cycle, which can be calculated by the following formula:

$$ \Delta \tau_j(\Omega_w) = \begin{cases} Q \epsilon^k, & \text{ant } k \text{ chooses } w_j(\Omega_w) \smallskip \\
0, & \text{other} \end{cases} $$

Where

- $Q$ —— constant, used to adjust the speed of modifying pheromone;
- $\epsilon^k$ —— output error calculated with the weights selected by the $k_{th}$ ant. It is defined as follows:

$$ \epsilon^k = |O - O_i| $$

Where

- $O$ —— actual output of FNN;
- $O_i$ —— ideal output of FNN.

If the error is smaller, the increase of pheromone will be more.

- Step five, if all the paths selected by ants converge to one path, or the cyclical times $NC \geq NC_{max}$, cycling is finished. Then output the optimal path, which represents the optimized weights. If not, turn to step two.

Through the above process, initial optimized values of the weights that represent the parameters of FNN can be obtained. Then use BP algorithm to train the weights of FNN with these initial values of weights. Thus the train of FNN is complete.

### III. SIMULATION

The parameters of simulation are set as follows:

- The building has 16 floors (0-15), and 4 elevators; the height of each floor is 3 m; the rated velocity of each elevator is 4 m/s; the maximal accelerated/decelerated velocity is 1 m/s²; rate of slope of accelerated velocity is 1 m/s²; the time to open/close the door of elevator is 0.8 s; the rated capacity of
The cumulative average waiting time $T_c$ is defined as:

$$T_c = \frac{\text{number of arrivals}}{\text{time duration}}$$

During this training process, the parameters of FNN and ACA-based EGCS that is optimized by ACA first and then trained by back propagation algorithm converges to $T_{\text{FNN}} = 7.06s$ (shown as Figure 4).

IV. CONCLUSION

In this paper, ACA is adopted to optimize FNN during training process in order to calculate initial values of weights in FNN, then BP algorithm is used to train these weights further. This method is able to solve the problem that convergence of weights through the process of training FNN just with BP algorithm is easily trapped in local optimal values. Thus the method can help EGCS achieve more precise and reasonable parameters of FNN, such as parameters of membership function. The simulation shows that FNN based on ACA has excellent parameters compared with that of FNN just based on BP algorithm so that $AWT$, one of the most important performance criteria, in this method is 17.1% shorter than that of an EGCS just based on FNN. Therefore the intelligent algorithm in this paper reasonably improves the performance of an EGCS.

REFERENCES


