A database-driven Ant Colony Algorithm for PLC networking

Yu Zeng¹, Xiaofang Zhou¹a), Linshan Zhang², Nianrong Zhou², and Gerald E. Sobelman³
¹ State Key Lab. of ASIC and System, Fudan University, Shanghai 200433, China
² YunNan Electric Power Test & Research Institute Group Co., Ltd, Kunming 650000, China
³ Dept. of Electrical and Computer Engineering, University of Minnesota, 200 Union Street SE, Minneapolis, MN55455, USA
a) xiaofangzhou@fudan.edu.cn

Abstract: Relay technology is necessary for Power Line Carrier communication because of the harsh characteristics of the power line channel, including time-variance, strong noise and high attenuation. The Ant Colony Algorithm is a promising approach for implementing relay technology. However, the execution time for the conventional Ant Colony Algorithm is very long. This paper provides a novel approach to speed up the search process using a database together with Dijkstra’s algorithm. The experimental results show that the new algorithm can decrease the convergence time significantly in most types of topologies. In addition, robustness and invulnerability properties of the algorithm are also improved.

Keywords: PLC networking, Ant Colony Algorithm, relay technology, Dijkstra’s algorithm

Classification: Electronic instrumentation and control

References

[1] H. Li: Ph.D thesis China University of Petroleum, Qindao (2010).
[2] H. Li, W. Kensheng and Z. Weixia: WiCom (2009) 1.
[3] C. Huang and Y. Ning: ICAL (2012) 461. DOI:10.1109/ICAL.2012.6308250
[4] J. Heo, K. Lee and H. K. Kang: IEEE Trans. Consum. Electron. 53 (2007) 1410. DOI:10.1109/TCE.2007.4429233
[5] L. Xiaosheng, Z. Yan and Q. Jiajin: CSEE (2006) 76.
[6] W. Lu, Y. Luo and X. Bi: ICCSNA (2010) 81. DOI:10.1109/ICCSNA.2010.5588771
[7] C. Ke, G. Xin and H. Xiaoguang: INDIN (2012) 196.
[8] J. Zhaoting: Master thesis Zhejiang University, Hangzhou (2011).
[9] C. Ke and H. Xiaoguang: Journal of Central South University 44 (2013) 571.
[10] Y. Gang: Powerline Communication Tech (Publishing House of Electronics Industry, Beijing, 2011) 8.
1 Introduction

Power Line Carrier (PLC) technology is a rapidly developing communications approach for smart grids. However, the characteristics of power lines are challenging and include significant non-idealities due to time-variance, strong noise and high attenuation, with the result that the one-hop communication distance is limited.

Relay technology is a key methodology for overcoming these limitations. Several relay algorithms have been put forward, including the cluster-based routing algorithm [1, 2, 3] and the flooding algorithm [4]. However, these methods do not dynamically adapt to changes in the communications environment. The Ant Colony Algorithm was proposed for PLC routing because of its ability to adapt to the changing environment [5]. However, the Ant Colony Algorithm is time-consuming and can become trapped into paths that are only locally optimal. Various methods have been introduced to improve the performance of this algorithm [6, 7, 8]. One interesting idea [7, 8, 9] is to combine the Ant Colony Algorithm with the Genetic Algorithm (ACGA). However, its convergence time is still relatively long.

This paper introduces a novel Ant Colony Algorithm for PLC networking, in which a database and Dijkstra’s algorithm are employed. The database is used to store paths that have been found. This gives additional information about the connections of nodes that can be used to search for the globally optimal path with Dijkstra’s algorithm.

The remainder of this paper is organized as follows: In section 2, the conceptual framework of the algorithm is analyzed. In section 3, the algorithm is presented in detail. In section 4, the results of simulation experiments are presented. Finally, our conclusions are given in section 5.

2 Conceptual framework

The topology of a power line network is usually a tree structure or a fishbone structure, and the latter can be considered as one branch of a tree structure [5]. A typical topology is as follows:

![Fig. 1. One typical physical topology of PLC network.](image-url)
In Fig. 1, node 0 is the concentrator node which is responsible for collecting and delivering information. The other nodes represent the meters. To collect from and deliver information to the other nodes, the concentrator node must know the path to every node and each node must also know the path to the concentrator node. In the Ant Colony Algorithm, ants start from the concentrator node and choose the next node according to a certain strategy until they find the target node. The ACGA uses the genetic algorithm to obtain new valid paths from existing valid paths, which can expand the solution space. In the crossover process of the genetic algorithm, a common node shared by two valid paths has special significance. This is due to the constraints of the communication distance. The following example graph illustrates the situation:

![Graph showing two paths](image)

In Fig. 2, consider what happens if we crossover the two paths at a random point. For example, suppose we cut path A between node 2 and node 3, cut path B between node 5 and node 8, and then we connect the first part of path A with the second part of path B and connect the second part of path A with the first part of path B. Then, two new paths are obtained, which can be labeled as ‘0-1-2-8-9-10-6’ and ‘0-7-5-3-4-5-6’. However, these new paths may not be valid ones because we do not know whether node 2 and node 8 can communicate with each other, or whether node 3 is within the communication distance of node 5. The only way to guarantee obtaining new valid paths is to cut the two paths at their common node 5. In this way, we can obtain two valid paths, ‘0-1-2-3-4-5-8-9-10-6’ and ‘0-7-5-6’. Here, of course, the path ‘0-7-5-6’ is the most optimal one among the four paths.

From the above example, we can see the importance of common nodes in obtaining new valid paths. Moreover, the paths also convey information about the topology, because adjacent nodes in the paths can communicate with each other. Thus, if we want to obtain new paths, we need to find at least one common node other than the concentrator node and the target node. We call such a node a ‘nontrivial common node’, whereas the term ‘common node’ could also include the concentrator node or the target node. The entire networking process can be divided into two stages. Stage I is when no nontrivial common node has been found. When at least one nontrivial common node has been found, the process enters stage II. The goal in stage I is to find at least two valid paths as soon as possible, because we can find a nontrivial common node only after we have at least two valid paths. The goal in stage II is to optimize the path using the nontrivial common nodes that have been found. These two stages are independent of each other and we use different optimization approaches in each stage. In stage I, we record the node numbers that have been visited and have the ants be more likely to visit nodes that have not been previously visited. This will cause the ants to visit more nodes in an iteration and
thus be more likely to find the target node. In stage II, we use a database to store the paths found in Stage I in order to make full use of the topology information inherent in the paths. A connected graph and Dijkstra’s Algorithm are also used in Stage II. The entire algorithm will be elaborated in the following section.

3 The ACAD algorithm

For convenience in presenting the algorithm, we first define several terms:

Definition 1: Hop. This is the number of intermittent segments between two nodes. For example, if two nodes are adjacent, hop is one.

Definition 2: Communication distance. This is the distance, in terms of the number of segments, to the farthest node one node can communicate with.

Definition 3: Ant’s life. This is the maximum number of hops one ant can reach.

Definition 4: Connection. If node A and node B exist in a certain path, the connection of node A and node B refers to the portion of the path between node A and node B. For example, in a path ‘0-1-2-3-4-5’, the connection of node 1 and node 5 is ‘2-3-4’.

Definition 5: Connection length. This is the number of the nodes in a connection. For example, the connection length of ‘2-3-4’ is three.

Definition 6: Valid path. This is a path that connects the concentrator node with a target node where the path is within the specified limit on the number of hops.

We now present our novel Ant Colony Algorithm using a Database (ACAD). The algorithm is divided into the following sequence of steps.

3.1 Initialization

The following quantities must be initialized: 1). pheromone of the paths; 2). a database; 3). a connection table; 4). a table of the occurrence number of each node in the valid paths; 5). a table of the number of visits of each node. For simplicity, we call the table of the occurrence number the ‘occurrence table’ and we call the table of the visit times of each node the ‘visit table’.

3.2 Select the next node

One ant starts out and chooses the next node according to the following rules: If less than two valid paths have been found since the beginning, which means that the algorithm is in stage I, the ant will choose the next node according to the following formulas:

\[
P_{ij} = \begin{cases} 
\frac{r_j^i}{(1 + \text{visit table}(j))^\beta} & j \in \text{allowed} \\
0 & j \notin \text{allowed} 
\end{cases} 
\]

In the above formulas, \( \beta \) is a parameter used to adjust the time to find the target node. The larger \( \beta \) is, the more likely the ant will be to choose an unvisited node. However, experiments found that if \( \beta \) is too large, the possibility of finding the global optimal path is sharply decreased. \( P_{ij} \) is the transition probability from node
i to node j. $\tau_{ij}$ is the pheromone between node i, where the ant is, and node j, which is one of the nodes that are within the communication distance of node i. visit_table(j) is the corresponding value in the visit table of the node j. $q_0$ ($0 < q_0 < 1$) is a parameter we set. $q$ ($0 < q < 1$) is a random number we generate. If $q \leq q_0$, we choose the node j with the maximum value $\tau_{ij}^0/(1 + \text{visit}_\text{table}(j)^0)$. Otherwise, the probability of each allowed node is calculated according to the formula (2): the higher $p_{ij}$ is, the more likely the ant will be to choose node j. When fewer than two valid paths are found, the more times one node has been visited, the less likely that it will be visited again. This makes the ant more likely to choose the nodes that have never been visited and to find the target node. The goal is to find nontrivial common nodes as soon as possible.

If more than at least one nontrivial common node has been found, which means the algorithm is in stage II, the ant will choose the next node according to the following formulas:

$$ j = \begin{cases} \arg \max_{j \in \text{allowed}} \{\tau_{ij}^n\} & q \leq q_0 \\ \text{using formula (4)} & q > q_0 \end{cases} $$

$$ p_{ij} = \begin{cases} \frac{\tau_{ij}^n}{\sum_{j \in \text{allowed}} \tau_{ij}^n} & j \in \text{allowed} \\ 0 & j \notin \text{allowed} \end{cases} $$

The parameters in these two formulas have the same meanings as those in formulas (1) and (2).

### 3.3 Continue searching

The ant moves to the selected node, and chooses the next node according to the same rules as in section 3.2. If the node has been visited, the ant will not visit it again. The ant moves until it finds the target node. The hops that an ant can go through have an upper limit, i.e. the ant’s life, in order to avoid occupying the channel for too long. If the ant does not find the target node within the ant’s life, the ant will die and another ant will start out. Also, occasionally the ant will go to a node for which all the nodes it communicates with have been visited. In this case, the ant will also die. If the ant finds the target node successfully, it will update the local pheromone on its path according to the following rule [5, 7]:

$$ \tau_{ij} = (1 - \xi)\tau_{ij} + \xi\tau_0 $$

Where $\xi$ is the local update volatilization coefficient and $\tau_0$ is the initial pheromone.

If the ant does not find the target within the ant’s life, the pheromone will not be updated, and the next ant will start out. One iteration is completed after all the ants have found the target node or have died.

### 3.4 Obtaining the connected graph

After one iteration, if fewer than two valid paths have been found, the corresponding value in the visit table of every node that has been visited in this iteration is increased by one. If at least one valid path is found in this iteration, the occurrence table is used. The occurrence table records the occurrence number of each node in the valid paths. The corresponding value in the occurrence table of each node in the
valid path will be increased by one. If the corresponding value of one node is larger than one, it is a common node. Notice that when there is only one valid path in one iteration, we may also find a common node as long as the value of the node in occurrence table is more than one. That is due to the fact that in the first several iterations, there is a high probability that only one valid path has been found.

A database stores all the paths in the order that they were found. For instance, after each iteration, all the paths found in this iteration will be stored into the database, following the paths found in the last iteration.

Then the connection table will be updated according to the database. The connection table stores the shortest connection of every pair of connected common nodes. For every pair of connected common nodes, we search for connections of them in all the paths in the database in order to find the shortest one. The shortest connection is entered into the connection table. The connection table is updated after each iteration. If we put all the common nodes that have been found together and connect every pair of common nodes with lines which represent the shortest connections of them, a connected graph is obtained. The following example graph illustrates this process:

![Graph](fig3.png)

**Fig. 3.** The process for constructing a connected graph.

In Fig. 3, there are five valid paths in the database. The nodes with the same number are the same common nodes. We put all the common nodes: 0,1,2,3 and T together and include lines which represent their connections. This results in a connected graph.

### 3.5 Dijkstra’s algorithm

As mentioned earlier, our objective is to find the globally optimal path as quickly as possible. The connected graph is used precisely for this purpose. Though we may not be able to directly obtain the globally optimal path in this graph, we can find at least a locally optimal path. These locally optimal paths will lead the ants to find the globally optimal path more rapidly. In this graph, the locally optimal path is equivalent to the path with the minimum sum of connection lengths and the number of common nodes in it. Dijkstra’s algorithm is then applied. For a given node, Dijkstra’s algorithm is used to find the path with lowest cost (i.e. the shortest
path) between that node and every other node. By using Dijkstra’s algorithm, we can find the most optimal path from the concentrator node to the target node in the connected graph. We then replace the lines with the connections according to the connection table and the complete path is thereby generated.

3.6 Global pheromone update

After one iteration, if the locally optimal path is found, the global pheromone is updated according to the following formulas:

\[
\tau_{ij} = (1 - \rho)\tau_{ij} + \rho \Delta \tau_{ij} \\
\Delta \tau_{ij} = \begin{cases} 
K \cdot (L_{bs})^{-1} & (i, j) \in \text{iterative optimal path} \\
0 & \text{otherwise}
\end{cases}
\]

where \( \rho \) is the global update volatilization coefficient, \( L_{bs} \) is the length of the iterative optimal path and \( K \) is the total pheromone. If no valid path is found, the global pheromone will be updated according to the formulas:

\[
\tau_{ij} = (1 - \rho)\tau_{ij} + \rho \Delta \tau_{ij} \\
\Delta \tau_{ij} = \begin{cases} 
K \cdot (L_{bs})^{-1} & (i, j) \in \text{the latest optimal path} \\
0 & \text{otherwise}
\end{cases}
\]

3.7 Completing the iterations

All the ants start out again to complete another iteration until the number of the iterations reaches the maximum iteration limit we have set initially.

4 Results of the simulation experiments

Several computational experiments were carried out to demonstrate the performance of ACAD including the convergence time, robustness and invulnerability. The results of ACGA in [7] are also presented here for comparison. However, since the success rate for finding the optimal path with ACGA is not available in [7], only the success rate of ACAD is listed in the following tables. The convergence time of ACGA given in [7] is also included here for comparison.

In these experiments, the following parameters are used: (1) An ant’s life is 7 hops, meaning that the ant can visit up to 7 nodes, including the concentrator node and the target node. (2) The communication distance of each node is set to 3. In other words, if two nodes have more than two other nodes (or more than 3 edges) between them, they cannot directly communicate with each other. All of the experimental results shown in following tables are the averages of 1,000 simulations.

In the experiments, the number of iterations is used to represent the convergence time for the following reasons. First, the convergence time consists of two parts: the operation time of the database and the time of communication. The time of communication is the time that the searching frames are transmitted among the nodes, which is proportional to the number of iterations. However, the operation time is trivial compared with the time of communication. The following estimation is provided to support this claim. In ACAD, the time complexity of the operation time is proportional to the number of nodes in the topology and the square of the
number of iterations. As is shown in Table II, the number of iteration grows slowly with an increase in the number of nodes. Here we are assuming that the number of iterations is proportional to the number of nodes. Then, the time complexity of ACAD is $O(n^3)$, where $n$ is the number of nodes. In our Matlab program, the average time for one simulation is about 0.313 seconds when the number of nodes is sixty. The concentration nodes, which are used to do the calculations, would typically include an embedded processor, so we estimate that the actual computation time would be approximately of the same order of magnitude. Usually the number of nodes in a topology is less than 400 [10]. Thus, we estimate that the operation time will be less than approximately $0.313 \times (400/60)^3$ seconds or about 93 seconds. On the other hand, the time of communication is proportional to the number of nodes in the topology and the number of iterations. So the time complexity of communication is $O(n^2)$. According to the test results in [8], communication from one node to another takes about five seconds. That is to say, it takes an ant about five seconds to move from one node to another. Because the globally optimal path in the example topology is 4, each ant moves at least four times. As there are about 6.663 iterations on average and ten ants in each iteration, the communication takes about $6.663 \times 10 \times 4 \times 5 = 1332.6$ seconds. Thus, in a topology of 400 nodes, the time is at most $1332.6 \times (400/60)^2 = 59226.67$ seconds. (Note that this time is of the same order of magnitude as the total time reported in [8].) From the above estimation, we can see that the operation time is very small compared with the time of communication, so it can be ignored. The time of communication is proportional to the number of iterations, so the number of iterations can be used to represent the convergence time.

### 4.1 Convergence time test

This experiment uses the topology of Fig. 1. The results of the experiment include the number of iterations each algorithm needs to converge and its success rate. The number of iterations is used to represent the convergence time as mentioned above. The success rate is the rate that the ants can find the globally optimal path. In this experiment, the parameter $\beta$ equals 2. Table I here shows the results of the experiment.

| Algorithm | Global Optimal path | Number of iterations | Success Rate |
|-----------|---------------------|----------------------|--------------|
| ACAD      | 0-19-41-60          | 6.6630               | 100%         |
| ACGA      | 0-19-41-60          | 27                   | Not available|

From these results we can see that ACAD converges much more rapidly than ACGA. The convergence time is improved by 75.3% and the success rate is one hundred percent. Besides, in this simulation, the maximum iteration number of ACAD is 25, even less than the average iteration number of ACAG.

To examine the superiority of ACAD in other topologies, we applied both ACAD and ACGA to one hundred randomly generated topologies, all of which have 60 nodes. The Fig. 4 below shows the convergence time of the two algorithms when they are applied to these one hundred topologies.
From the graph we can see that the convergence time of ACAD is generally much shorter than for ACGA. To illustrate the superiority of ACAD, the ratios between the convergence time of ACAD and that of ACGA are also calculated. The results are shown in the Fig. 5.

In the pie chart, each sector represents an interval value of the ratios between the convergence time of ACAD and that of ACGA. For example, the dark blue sector with “44%” above it means that in 44 percent of the 100 tests, the convergence time of ACAD is shorter than 20% of that of ACGA. From the pie chart we can see that, the convergence time of ACAD is shorter than the convergence time of ACGA in 99% of the tests. Besides, in 63% of the tests, the convergence time of ACAD is shorter than 40% of the convergence time of ACGA. So the pie chart shows that ACAD is effective for most of the topologies.

The following table contains the results of other experiments using topologies having different numbers of nodes.

| Number of nodes | $\beta$ | Number of iterations | Success Rate |
|-----------------|---------|----------------------|--------------|
| 70              | 3       | 11.1219              | 96.8%        |
| 80              | 3       | 11.3779              | 99.5%        |
| 100             | 6       | 12.3996              | 99.7%        |
\( \beta \) can be adjusted to balance the convergence time and the success rate. As we can see from the table, the convergence time does not grow rapidly as the number of nodes increases from 70 to 100, which is a significant advantage of ACAD. The increase in convergence time from 60 nodes to 70 nodes is due to the fact that the convergence time in ACAD is directly associated with the number of hops in the globally optimal path. In the topology of 60 nodes, the hops of the globally optimal path are 4, while in the topologies of 70, 80, or 100 nodes the hops are 5. The success rate here is not one hundred percent because in some experiments suboptimal paths rather than globally optimal paths were found.

### 4.2 Robustness test

Robustness here means adaptability to changes in the environment, e.g. changes in communication distance due to the time-variance of the PLC channel. In this experiment, the physical topology of Fig. 1 is also used, but the communication distance of each node in the figure is selected randomly.

In the Table III, No. represents the number of each node and D represents the communication distance. For example, the table shows that the communication distance of node 2 is 5, which means that the node 2 can directly communicate with nodes that are at most 5 segments away.

| No. | D  | No. | D  | No. | D  | No. | D  | No. | D  |
|-----|----|-----|----|-----|----|-----|----|-----|----|
| 1   | 21 | 2   | 21 | 2   | 31 | 4   | 41 | 5   | 51 |
| 2   | 51 | 2   | 32 | 3   | 32 | 5   | 42 | 1   | 52 |
| 3   | 4  | 13  | 4  | 33  | 4  | 43  | 1  | 53  | 5  |
| 4   | 5  | 14  | 4  | 34  | 4  | 44  | 2  | 54  | 4  |
| 5   | 1  | 15  | 3  | 35  | 2  | 45  | 1  | 55  | 1  |
| 6   | 5  | 16  | 4  | 36  | 3  | 46  | 1  | 56  | 3  |
| 7   | 1  | 17  | 4  | 37  | 1  | 47  | 3  | 57  | 1  |
| 8   | 2  | 18  | 4  | 38  | 2  | 48  | 4  | 58  | 4  |
| 9   | 1  | 19  | 2  | 39  | 4  | 49  | 2  | 59  | 3  |
| 10  | 1  | 20  | 4  | 40  | 3  | 50  | 2  | 60  | 1  |

The results of the experiments are shown in Table IV.

| Algorithm | Global Optimal path | Number of iterations | Success Rate |
|-----------|---------------------|----------------------|--------------|
| ACAD      | 0-8-41-60           | 9.6708               | 96.6%        |
| ACGA      | 0-8-41-60           | 13                   | Not available|

As can be seen from Table IV, both of the algorithms can find the globally optimal path ‘0-8-41-60’ when the communication distance is changed due to noise and attenuation. ACAD improves the convergence time by 25.6%.
4.3 Invulnerability test

The physical topology may also be changed due to the loss of certain nodes. In this experiment, we assume that node 19 in Fig. 1 is non-functional.

As can be seen in Table V, when this node is lost, both ACAD and ACGA can still find the globally optimal path: '0-8-41-60'. ACAD takes about 10.6639 iterations times on average to find the global optimal path, which is 23.8% faster than ACGA.

5 Conclusion

This paper presents a new kind of ant colony algorithm called ACAD for use in PLC networking. ACAD finds the globally optimal path much faster than previous approaches by taking advantage of information in paths that have been already obtained. The results of the experiments show that, compared with ACGA, the convergence time of ACAD is significantly reduced. In addition, the robustness and invulnerability of the algorithm have been improved. The results also show that ACAD works well for a range of different network topologies.

Acknowledgments

This paper is supported by China Southern Grid YunNan Electric Power Test & Research Institute Project K-YN2012-477, Shanghai Municipal Commission Of Economy and Information project Software & IC Industry Specific grant number 130303, and State Key Lab of ASIC & System grant No. 11MS003 and 12GF002.