Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 5055019, 8 pages
https://doi.org/10.1155/2022/5055019

Research Article

A New Way to Model the Wireless Sensor Network Maintenance Job

Ziqi Wei[1] **and Mike MacGregor**[2]

[1]*School of Software, Tsinghua University, Beijing, China 100084*
[2]*Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2E8*

Correspondence should be addressed to Ziqi Wei; ziqi2@ualberta.ca

Received 24 June 2021; Revised 29 December 2021; Accepted 4 January 2022; Published 25 April 2022

Academic Editor: Ning Wang

Copyright © 2022 Ziqi Wei and Mike MacGregor. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

For the maintenance of an urban wireless sensor network, the staff's travel route greatly affects the whole network's response time. Every time the network reports an error, the staff needs to find the best route to minimize the time spent on the way to the error point. The difficulty of the problem is that although the entire network fails, the error point remains unclear. In this paper, the staff's route planning is modeled as an NP-complete problem, MWLP (Minimum Weighted Latency Problem). It is a problem of finding the best route for a moving agent to satisfy multiple customers' different demands as much as possible. To solve the problem, we propose a heuristic algorithm which borrows the idea from a biological computing model called P_system. In the proposed algorithm, different classic heuristics work as separate "membranes" to accomplish their own jobs. They also collaborate under some mechanism to search for a better result. We designed the cell's structure to balance the different heuristics' time consumption and searching capacity. With this design, all the heuristics can cooperate properly in the proposed heuristic algorithm. To enhance the algorithm's efficiency, we also introduced a way to run it in parallel. The numerical experiments show that the proposed algorithm is very competitive compared with classic heuristic algorithms and helps eliminate the whole network delay as well.

1. Introduction

Koutsoupias et al. described the following situation in 1996: a treasure is hidden in one certain cave in a mountain area. An adventurer is trying to find it. However, the mountain area is full of different caves, and they all look alike. The good news is that the adventurer has a treasure map. Every cave's position and the probability of whether the treasure is placed inside it are marked on the map. Now, the adventurer is planning his search path. His goal is to minimize the distance he will travel before finding the treasure. The problem the adventurer is facing is called the Graph Searching Problem (GSP) in [1]. Wu renamed it MWLP in [2]. Two years later, Sitters proved it to be NP-hard in [3].

MWLP is formulated as follows: given n vertices $v_1, \cdots, v_n$ of a weighted (both edge and vertex) graph G. The vertices' weights are represented by $w(v)$. MWLP is looking for a tour π, starting at a fixed vertex s and visiting every other vertex of G. For this tour, the sum of all the vertices' weighted latencies (first arrival times of all the vertices $d_\pi(s, v)$ multiplied by their weights $w(v)$) must be minimized. MWLP is presented as

$$\min_{\pi} \sum_{v \in G} w(v) d_\pi(s, v). \tag{1}$$

The vertices' weights are sometimes noted as the probability of finding a treasure and limited to the interval $(0, 1]$. In this case, MWLP is formulated as follows:

$$\min_{\pi} \sum_{v \in G} pr(v) d_\pi(s, v). \tag{2}$$

By simply taking the sensor's error probability as $pr(v)$, MWLP can help to model the mobile agent's route planning problem in a wireless sensor network. To minimize the expression above, we shall minimize the time maintenance staff uses to find the failed sensor. Compared to TSP (Travelling Salesman Problem) and MLP (Minimum Latency Problem), MWLP pays more attention to the nodes' requirements in the network, which makes it more suitable for this environment.

MWLP can be seen as the vertex-weighted generalization of the Minimum Latency Problem (MLP), which is widely studied and proved to be NP-hard in [4]. Koutsoupias et al. also proposed a way to convert MWLP into MLP in polynomial time in [1]. With that approach, any algorithm designed for MLP can be used to solve MWLP. However, in practical cases, because of the shortcomings of the conversion process (weight expression form and graph structure are limited, as we explained in [5]), this method is inefficient. Wu proposed a DP (dynamic programming) algorithm to solve some special MWLPs in [2]. To the best of our knowledge, besides this, neither exact nor approximate algorithms designed for MWLP have been proposed. Though it can find the global optimum in a relatively short time, Wu's method still has the drawbacks of all DP algorithms.

Heuristic algorithms are a fast developing research field during the past several years. Different heuristic algorithms are designed for MLP, such as [6–9]. In recent years, using composite heuristic algorithms to solve NP-hard problems is becoming a trend. For example, Ha et al. proposed a metaheuristic combining tabu search (TS) and Variable Neighbourhood Search (VNS) to solve MLP in [10]. They used VNS to find the next feasible solution and TS to avoid searching one solution multiple times. Lenin et al. proposed a metaheuristic for a problem very similar to MLP which combined TS and simulated annealing (SA) in [11]. Their algorithm added the tabu list into an SA algorithm. The experimental results showed that the composite algorithm produced better results compared to either SA or TS alone. In [6], Ahmed proposed a hybrid genetic algorithm which combines the genetic algorithm (GA) and local search to solve MLP and got good results. Koutsoupias' conversion method can be used to apply these algorithms to MWLP. However, the problem scale will expand dramatically. It will cause both time and space consumption to increase exponentially, which makes a heuristic algorithm designed for MWLP necessary. In [5], we studied the heuristics used to solve MWLP. Five classic heuristic algorithms including TS, SA, GA, Particle Swarm Optimization (PSO), and ant colony optimization (ACO) were all redesigned to solve MWLP and tested effectively.

In this paper, we propose a heuristic algorithm based on our previous research. We borrowed the idea from P_system and optimized the five classic heuristics to embed them into the P_system structure. We also applied two ACO methods with different pheromone update mechanisms. We evaluated our approach with the widely used dataset for graph theory problems, TSPLib. The experiments show that this metaheuristic yields greatly improved results compared to the classic heuristics.

The rest of this paper is organized as follows. Section 2 introduces the heuristic based on P_system. Experimental results are shown in Section 3, and Section 4 gives the conclusion.

2. The Algorithm Based on P_system

P_system (also known as membrane computing) is a computational model proposed by Păun in [12]. Păun is a computer scientist whose research field is biological computing. Inspired by the biological cell's structure, he proposed the P_system model to simulate the process that a cell uses to handle compounds in a circulation system consisting of multilayer membranes. Thanks to the advantage of its inherent structure, many other algorithms can easily be embedded into a P_system, so that many metaheuristic algorithms based on P_system have been proposed recently.

For instance, Niu et al. proposed a metaheuristic algorithm based on the structure of P_system and used ant colony optimization to solve the Capacitated Vehicle Routing Problem (cVRP) in [13]. Another heuristic algorithm, which is also based on the structure of P_system, was given by Yan et al. in [14]. They used SA to communicate between different membranes to solve the weight optimization problem for case-based reasoning. In the following, we will introduce P_system and then go to the details of the proposed composite algorithm.

2.1. Introduction of P_system. Similar to other natural computation models, P_system is inspired by an existing process in the natural environment. It simulates the chemical processes that occur between different layers of the membranes within a cell. During processing, the cell is considered a frame. The chemical compounds within it are the subjects that are involved in the process. When passing through a membrane, the compounds are involved in some chemical reactions with other compounds or some prestored catalysts inside the membrane. Through iterations of this process, new compounds are produced and then involved in subsequent processing. The final output is produced after some stopping criterion is met.

In general, a P_system consists of three parts: the hierarchical structure of membranes, a multiset to represent the objects, and the evolutionary rules. The hierarchical structure of membranes defines the computing architecture. It limits the sequence and affiliation of different membranes. The object multiset represents the developing state during the computing process. The evolutionary rules define the method P_system uses to develop the results. The hierarchical membrane structure guarantees that the architecture can handle multiple heuristics in one algorithm.

The composite heuristic algorithm based on P_system has attracted much attention for two reasons. The first is that the computing model's structure is very suitable for combining multiple algorithms. The second is that the no-free-lunch (NFL) theorem guarantees its computational capacity.

The famous NFL theorem was proposed and proven by Wolpert and Macready in [15]. It proves that if an algorithm performs well on a certain class of problems, then it necessarily pays for that with degraded performance on the set of all remaining problems. That is, different algorithms have specific advantages and disadvantages when solving one certain problem. Composite algorithms can synthesize the advantages of different algorithms, which in theory gives them better searching ability than the individual components considered separately. People designed various mechanisms such as diversification in

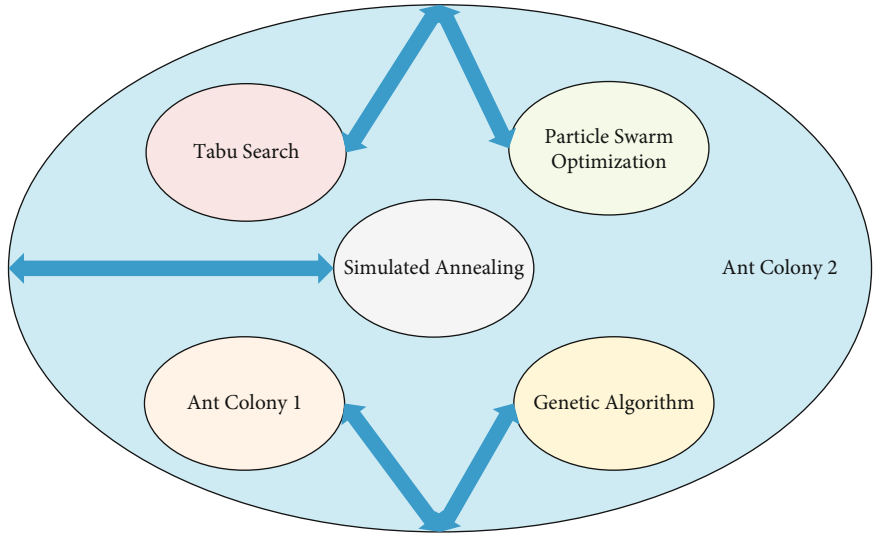

Figure 1: Illustration of the algorithm based on P_system.

TS and mutation in GA to avoid heuristic algorithms being trapped in local optima. However, being trapped in local optima is still inevitable because of the limitations of single heuristics. Composite algorithms can avoid their limitations.

2.2. Heuristic Based on P_system. Based on P_system, we designed a composite heuristic algorithm to solve MWLP. The algorithm consists of six different heuristic algorithms. They are PSO, TS, SA, GA, and two different ACOs. The two ACOs update their pheromone data in different ways. Though one of them produces better results (noted as ACO2 in this paper) than the other (noted as ACO1) in most circumstances when they are used alone to solve MWLP, as shown in [5], we choose to use both of them in the composite algorithm because of the NFL theorem. The parameters of the heuristics are the same as in [5]. The algorithm's main structure is shown in Figure 1.

As shown in the illustration, the "cell" consists of five inner membranes and an outer membrane. The five inner membranes are independent of each other, whereas they are all connected to the outer membrane. The inner membranes conduct PSO, TS, SA, GA, and ACO1, respectively, and the outer membrane runs ACO2. Previous experiments showed that ACO2 produces the best results among the six heuristic algorithms when they are running alone (same iteration number) to solve MWLP. Therefore, we consider ACO2 as the most suitable algorithm for the outer membrane.

When using this metaheuristic to solve MWLP, the mobile agent's route is stored in vertex arrays. Ten arrays are included in a group. The groups are considered the compounds to be transmitted between membranes. At the beginning, a group of vertex arrays is generated by using a greedy algorithm. This group is used as the initial solutions of the inner membranes. After the five optimization processes (inner membranes) are performed, each of them outputs a group of vertex arrays, consisting of the ten best solutions found by each heuristic. The five groups are combined as the initial solutions of the outer membrane. After the outer membrane finishes its optimization process, it outputs a group of the ten best solutions. This group of solutions is returned to the inner membranes as their initial solutions. At this time, the algorithm comes back to the starting state. We call this an iteration. The algorithm will iterate a predetermined number of times until the outer membrane outputs the final solution. The procedure's pseudocode is shown in Algorithm 1.

```
procedure P_system
  ini_seq = GreedyAlg(weighted_map, priority_seq)
  best_val = ComputeValue(ini_seq)
  iteration_number ⟵ 0
  while iteration_number < iteration_time do
    ac1_best_seq = ACO1_Alg(ini_seq)
    sa_best_seq = SA_Alg(ini_seq)
    ts_best_seq = TS_Alg(ini_seq)
    pso_best_seq = PSO_Alg(ini_seq)
    ga_best_seq = GeneticAlg(ini_seq)
    new_seq = combination of the five seqs above
    ac2_best_seq = ACO2_Alg(new_seq)
    temp_val = ComputeValue(ac2_best_seq)
    if temp_val < best_val then
      best_val = temp_val
    end if
    ini_seq = ac2_best_seq
    iteration_number + +
  end while
  return best_val
end procedure
```

Algorithm 1: P_system algorithm

2.3. Parallel Technique. It is obvious that the heuristic contains relatively many subalgorithms that make its computing scale large. Due to this, the time consumption of the P_system is its main drawback. However, thanks to the use of the membrane computing model, the whole optimization procedure

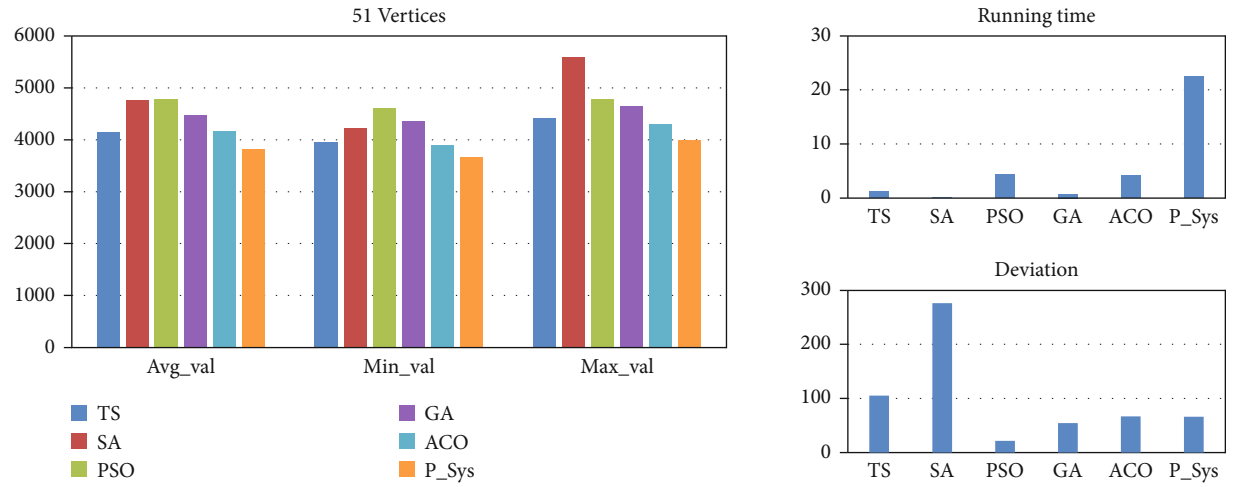

Figure 2: Experimental results of the map with 51 vertices.

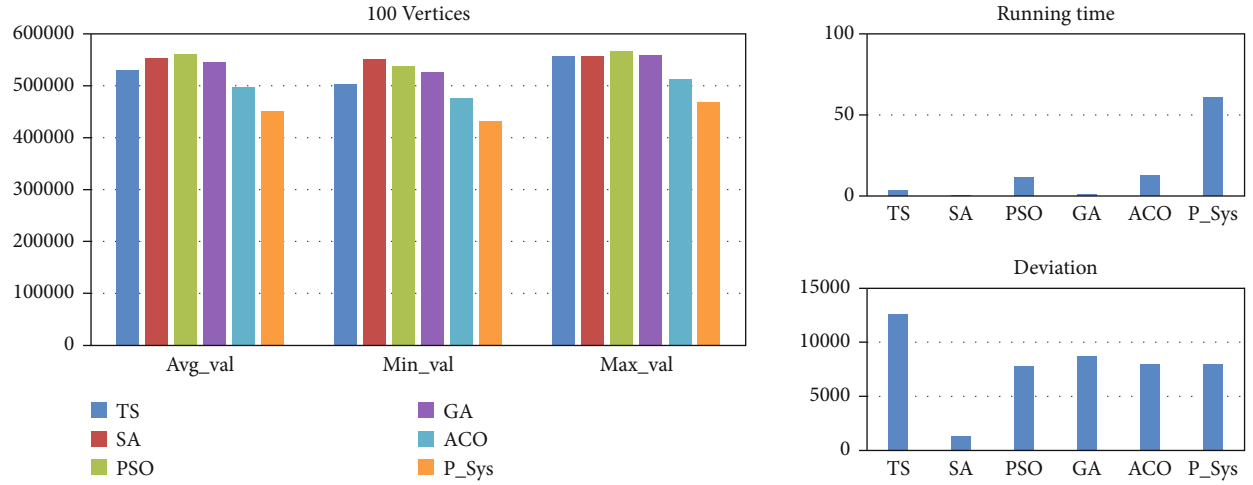

Figure 3: Experimental results of the map with 100 vertices.

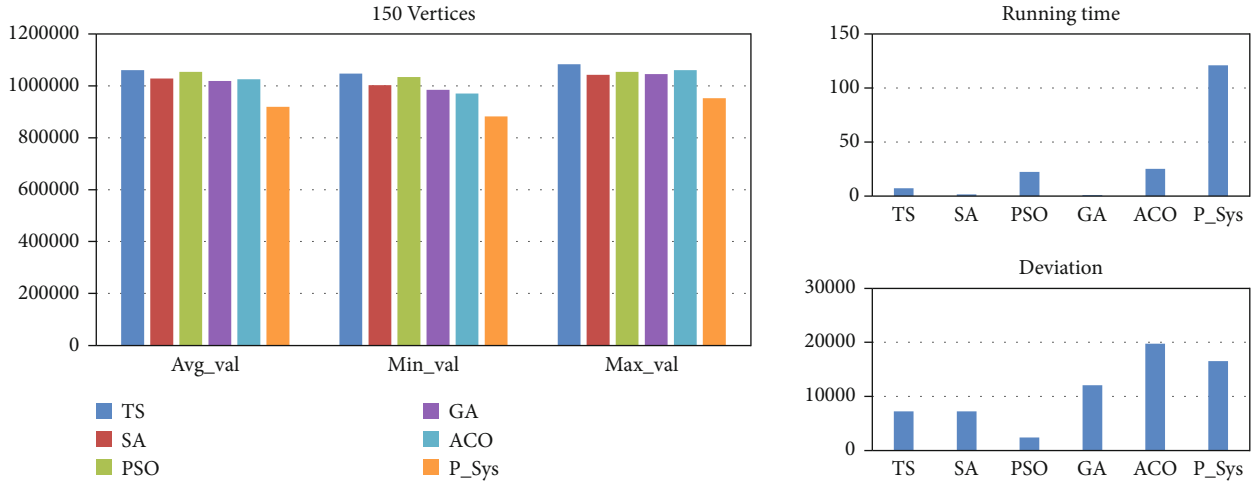

Figure 4: Experimental results of the map with 150 vertices.

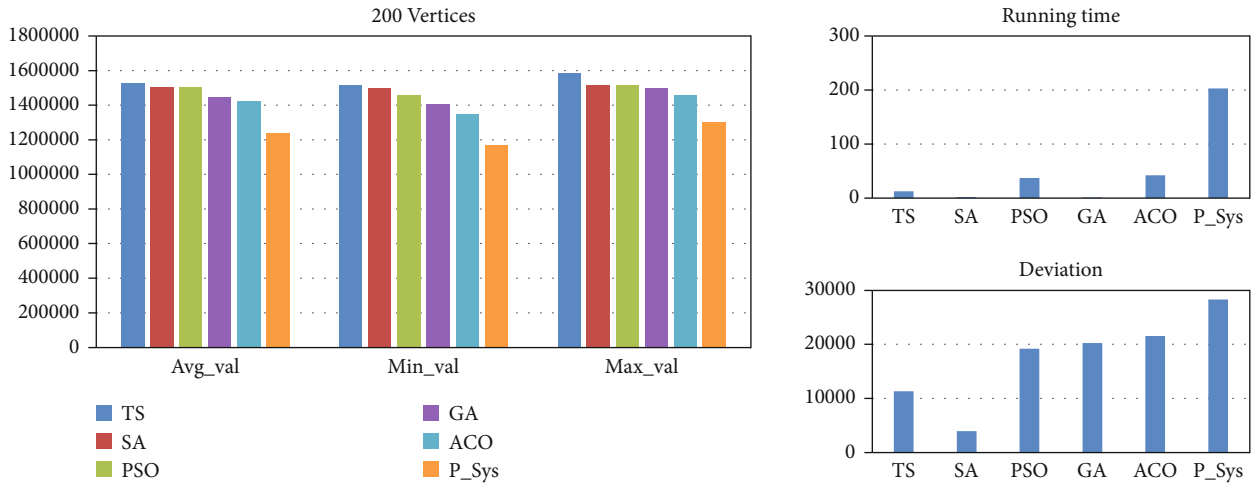

Figure 5: Experimental results of the map with 200 vertices.

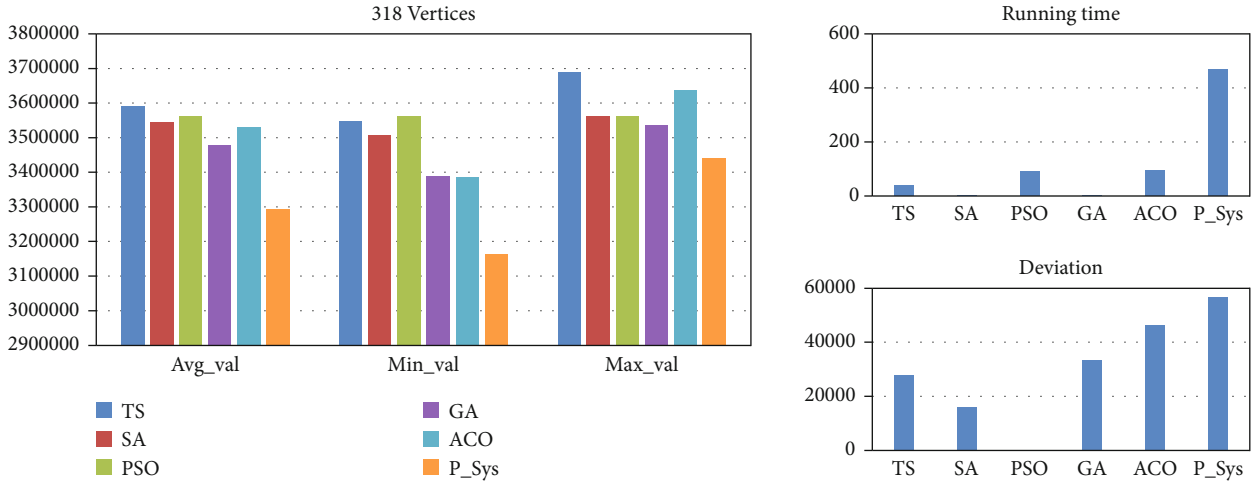

Figure 6: Experimental results of the map with 318 vertices.

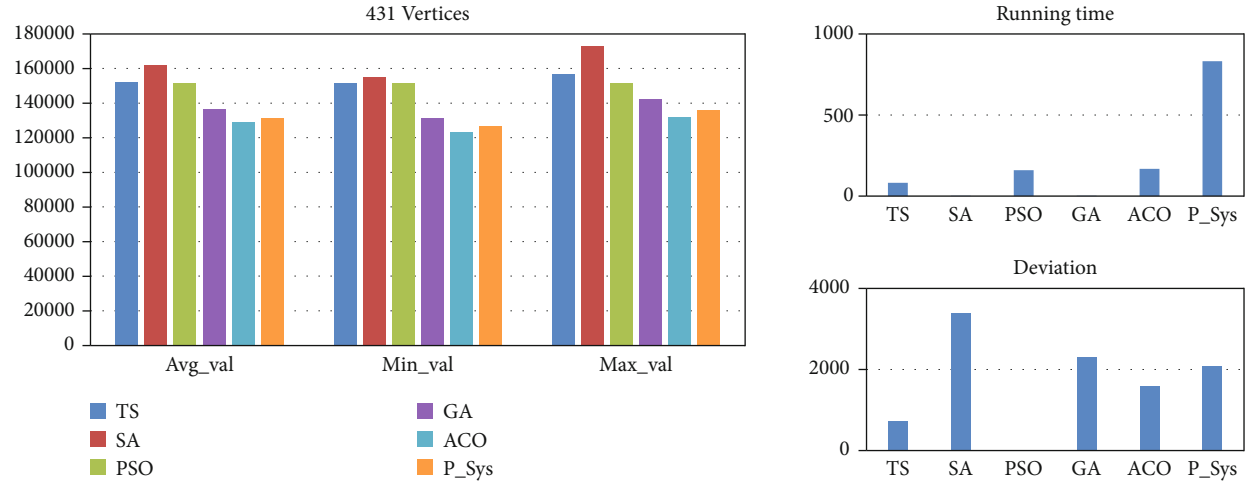

Figure 7: Experimental results of the map with 431 vertices.

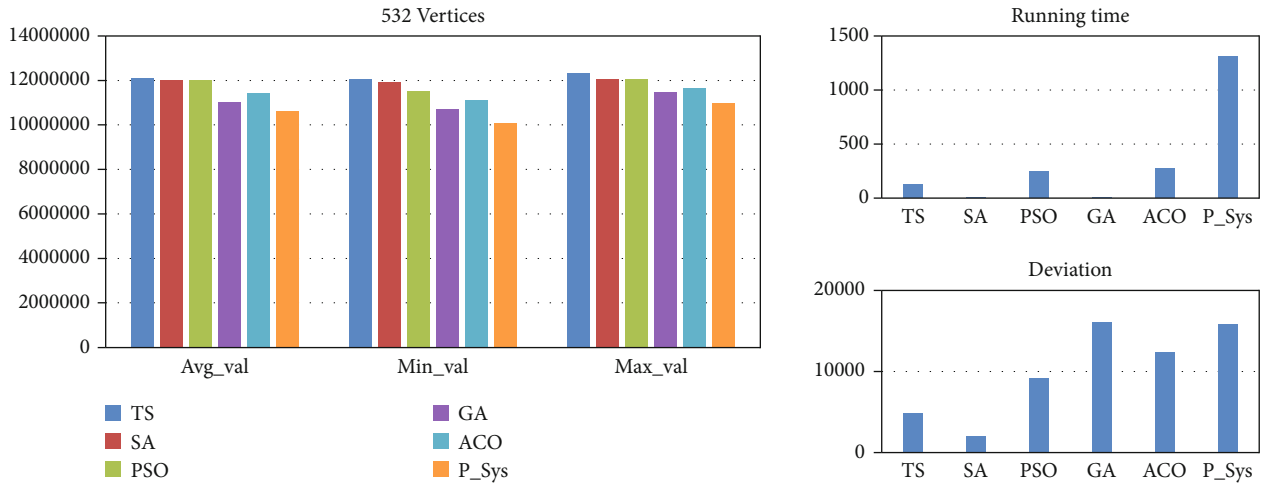

Figure 8: Experimental results of the map with 532 vertices.

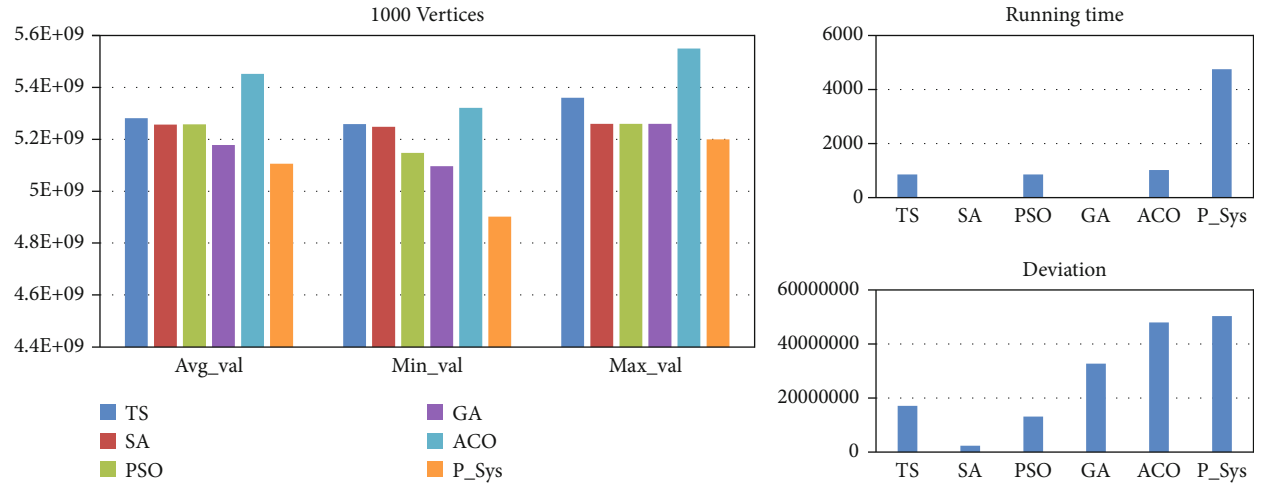

Figure 9: Experimental results of the map with 100 vertices.

Table 1: Average outputs.

Map	ACO	P_system	P_system improvement
eil51	$4.16E+03$	$3.83E+03$	7.93%
kroB100	$4.98E+05$	$4.52E+05$	9.24%
kroB150	$1.06E+06$	$9.19E+05$	13.3%
kroB200	$1.42E+06$	$1.24E+06$	12.68%
lin318	$3.53E+06$	$3.29E+06$	6.8%
gr431	$1.29E+05$	$1.31E+05$	-1.6%
att532	$1.14E+07$	$1.06E+07$	7.02%
dsj1000	$5.45E+09$	$5.11E+09$	6.24%

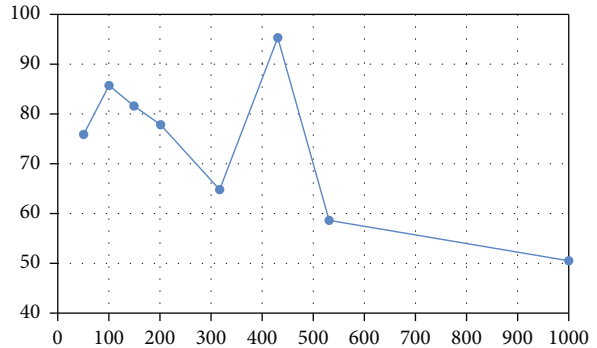

Figure 10: Iteration number at which ACO found the best result.

can be designed to run in parallel. The five inner membranes are mutually independent. None of them needs to use the others' outputs.

Based on the analysis above, we designed a parallel procedure. The main program creates five threads corresponding to the five inner-membrane heuristics. They are allocated to different CPU cores to run at the same time. When a thread completes, it sends its result to the main program and terminates. The main program waits until the five threads are all finished. Then, the main program combines the 5five threads' outputs as the outer

Table 2: Experimental results of ACO and the proposed P_system.

Map	ACO			P_system		
	Worst result	Best result	Variance	Worst result	Best result	Variance
eil51	$4.31E+03$	$3.90E+03$	$4.44E+03$	$3.99E+03$	$3.66E+03$	$4.41E+03$
kroB100	$5.13E+05$	$4.77E+05$	$6.39E+07$	$4.70E+05$	$4.32E+05$	$6.42E+07$
kroB150	$1.06E+06$	$9.70E+05$	$3.90E+08$	$9.52E+05$	$8.82E+05$	$2.72E+08$
kroB200	$1.46E+06$	$1.35E+06$	$4.63E+08$	$1.30E+06$	$1.17E+06$	$2.83E+04$
lin318	$3.64E+06$	$3.39E+06$	$2.15E+09$	$3.44E+06$	$3.16E+06$	$3.22E+09$
gr431	$1.32E+05$	$1.23E+05$	$2.51E+06$	$1.36E+05$	$1.26E+05$	$4.30E+06$
att532	$1.17E+07$	$1.11E+07$	$1.52E+10$	$1.10E+07$	$1.01E+07$	$2.49E+10$
dsj1000	$5.55E+09$	$5.32E+09$	$2.29E+15$	$5.20E+09$	$4.90E+09$	$2.52E+15$

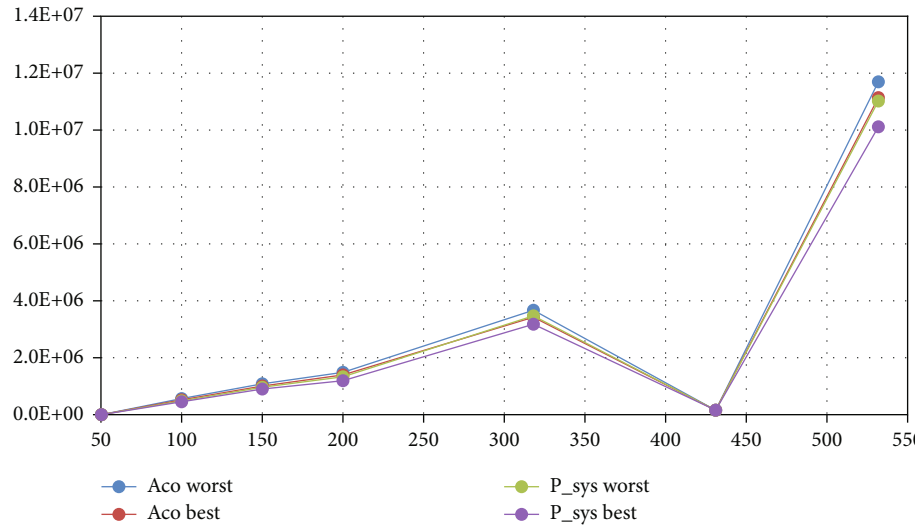

Figure 11: Best and worst results for both algorithms.

membrane's input. In other words, by running the sixth to the tenth line in the pseudo-code at the same time, some parts of the original algorithm can run in parallel. By using this strategy, the composite algorithm's running time drops impressively. The results of our experiments will be given in the fourth section.

3. Experimental Environment and Results

As in [5], the experiments were all run on a small-scale server. It is equipped with two Intel(R) Xeon(R) E5-2670 v3 @ 2.30 GHz CPUs (each has 8 cores) and 64 GB memory. The operating system is Windows® Server 2012. We used MATLAB® R2017b.

The same as the experiments in [5], we applied ACO (the outer membrane in the proposed algorithm), SA, TS, GA, and PSO as the competitive comparison for this paper. The algorithms' parameters are all the same as we set in [5]. Considering that the proposed heuristic is relatively powerful, we determined to test it in a wider range from 51 vertices to 1000 vertices. Rather than randomly generating the test map, we choose instances from TSPLib. They are *eil51*, *kroB100*, *kroB150*, *kroB200*, *lin318*, *gr431*, *att532*, and *dsj1000*, respectively. The vertices' weights are allocated randomly. We coded our P_system to run in parallel to save time. As in [5], all the problem instances are tested by the algorithms for 100 times. The outputs and running time for every map are all stored for later analysis.

We use ACO (which performed the best among the classic heuristics) and P_sys algorithms as representatives to illustrate the performance of the P_sys algorithm. The results of all the tested algorithms are shown from Figures 2–9. Table 1 shows the average outputs and running time for both algorithms to search the maps 100 times. For all but one instance, the output from the P_system was better than that from ACO by at least 6%. The performance of this proposed P_system is so good that we do not need to divide the problems into different scale groups. The P_system is able to find a better solution than ACO except for map gr431. For this map, we recorded the iteration number at which ACO found its best value and stopped improving. Then, for 100 runs on the full set of maps, we calculated the average iteration number at which the best value was found on each map. The values are shown in Figure 10.

The horizontal axis is the number of vertices in the map, and the vertical axis is the average iteration number at which the best value was found.

Figure 10 shows that the iteration number at which the best value was found decreases as the number of vertices increases—except in the case of map gr431. That is, as the scale of the problem increases, ACO tends to be trapped in a local optimum earlier. That may occur because as the problem scale increases, different local optima tend to get further away from each other. When ACO finds a local optimum, the algorithm is unable find a second local optimum that is far away. This leads us to hypothesize that the exceptional behaviour for map gr431 in Table 1 is because the map was relatively small.

To test this hypothesis, we captured the best and worst results for both ACO and our P_system (see Table 2 and Figure 11). These results show that the optimal value found for map gr431 is quite low, nearly matching the value for map eil51, which has 51 vertices. That is, map gr431 seems exceptional compared with the other maps in TSPlib, in terms of the results on this problem. For a map like this, we may not need an algorithm like the proposed P_system, which is able to search every possible direction of the search space. A simpler algorithm like ACO works better in this case. So in the case of map gr431, the proposed P_system did not have an advantage from using more heuristics. Instead, because of the existence of those additional heuristics, ACO did not perform the best in its task as an outer membrane.

4. Conclusion

In this paper, we propose a new way to model the urban WSN's maintenance job. A heuristic based on the structure of P_system is also designed to help solve MWLP. The proposed heuristic combines several well-known heuristics: TS, SA, GA, PSO, and ACO. Our experimental results show that this metaheuristic is more powerful than ACO in most cases, especially those with large numbers of vertices. Considering that our previous results given in [5] show that ACO gives the best results of all the individual heuristics used in our P_system, we can say that the proposed P_system has been demonstrated to be the most powerful approach overall as it yields better results. One exception has been observed in this paper where the proposed P_system performed slightly worse than ACO. Improving performance in this case is the priority in our future research. One possible solution may be finding a method to let the inner membranes cooperate with each other to empower the P_system to select better results.

Data Availability

The experimental results and the maps we used in the experiment will be found through the following link: https://drive.google.com/drive/folders/1XKDtCPHpJj2aAhiEYO3Wa4bhSIMFTJBU?usp=sharing.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (No. 62102058), and part of the job was done when Ziqi Wei was pursuing for his PhD degree in the University of Alberta, which was supported by a grant from the China Scholarship Council.

References

[1] E. Koutsoupias, C. Papadimitriou, and M. Yannakakis, "Searching a fixed graph," *In International Colloquium on Automata, Languages, and Programming, pages*, vol. 1099, 1996.

[2] B. Y. Wu, "Polynomial time algorithms for some minimum latency problems," *Information Processing Letters*, vol. 75, no. 5, pp. 225–229, 2000.

[3] R. Sitters, "The minimum latency problem is NP-hard for weighted trees," *Lecture Notes in Computer Science*, vol. 2337, pp. 230–239, 2002.

[4] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, "On the minimum latency problem," in *Proc 26thSymp Theory of Computing STOC*, p. 9, 1994.

[5] Z. Wei and M. H. MacGregor, "Heuristic approaches for minimum weighted latency problem," in *2016 4th International Conference on Machinery, Materials and Information Technology Applications*, pp. 552–556, Atlantis Press, 2017.

[6] Z. H. Ahmed, "The minimum latency problem: a hybrid genetic algorithm," *IJCSNS International Journal of Computer Science and Network Security*, vol. 18, no. 11, pp. 153–158, 2018.

[7] H. B. Ban and N. D. Nghia, "Improved genetic algorithm for minimum latency problem," in *Proceedings of the 2010 Symposium on Information and Communication Technology*, pp. 9–15, 2010.

[8] T. Dewilde, D. Cattrysse, S. Coene, F. C. R. Spieksma, and P. Vansteenwegen, "Heuristics for the traveling repairman problem with profits," *Computers and Operations Research*, vol. 40, no. 7, pp. 1700–1707, 2013.

[9] M. M. Silva, A. Subramanian, T. Vidal, and L. S. Ochi, "A simple and effective metaheuristic for the minimum latency problem," *European Journal of Operational Research*, vol. 221, no. 3, pp. 513–520, 2012.

[10] B. B. Ha and N. N. Duc, "A meta-heuristic algorithm combining between tabu and variable neighborhood search for the minimum latency problem," *Fundamenta Informaticae*, vol. 156, no. 1, pp. 21–41, 2017.

[11] K. Lenin, B. R. Reddy, and M. Suryakalavathi, "Hybrid tabu search-simulated annealing method to solve optimal reactive power problem," *International Journal of Electrical Power and Energy Systems*, vol. 82, pp. 87–91, 2016.

[12] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[13] Y. Niu, S. Wang, J. He, and J. Xiao, "A novel membrane algorithm for capacitated vehicle routing problem," *Soft Computing*, vol. 19, no. 2, pp. 471–482, 2015.

[14] A. Yan, H. Shao, and Z. Guo, "Weight optimization for case-based reasoning using membrane computing," *Information Sciences*, vol. 287, pp. 109–120, 2014.

[15] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.