# Solving Traveling Salesman Problems with Ant Colony Optimization Algorithms in Sequential and Parallel Computing Environments: A Normalized Comparison

G. Dong, W. Li, J. Shen, Y. Wang, X. Fu, and W. W. Guo

***Abstract*—In recent years some comparative studies have explored the use of parallel ant colony optimization (ACO) algorithms over the traditionally sequential ACOs to solve the traveling salesman problem (TSP). However, these studies did not take a systematical approach to assess the performance of both algorithms on a comparable ground. In this paper, we aim to make a comparison of both the quality of the solutions and the running time as a result of the application of a sequential ACO and a parallel ACO to Eil51, Eil76 and KroA100 on a normalized and thus, comparable ground. Our study reaffirmed that the parallel algorithm is superior in computing efficiency over the sequential algorithm, particularly for larger TSPs. We also found that such a comparison could be meaningless if the size of the TSPs keeps increasing. We revealed that the worst solution among 10 repeated runs obtained from the parallel ACO was still better than the best solution among 10 repeated runs obtained from the sequential ACO, though both did not reach the global optimal solution within 300 iterations. The proposed parallel ACO has a very high consistency because at least one best solution was found within an error of 0.5% to the global optimal solution in every three repeats for all three cases.**

***Index Terms*—Ant colony optimization, traveling salesman problem, parallel computing, sequential computing.**

## I. Introduction

The traveling salesman problem (TSP) is described as: given a list of cities and their pairwise distances, find the shortest possible tour that visits each city exactly once and then returns to its original city [1]. TSP is a classic problem in combinatorial optimization called NP-complete problems, which is hardly solvable efficiently by any analytical algorithm [2]. Hence, a variety of heuristic algorithms have been devised to produce approximate solutions good enough for application in TSPs [3], [4]. Genetic algorithm (GA) and ant colony optimization (ACO) and their variations are among the heuristic algorithms that have been widely used in solving various cases of TSPs [5]-[10].

ACO is a multi-agent system stimulating the searching actions of some ant species, i.e., ants deposit pheromone on their paths in order to mark the favorable trail that can be followed by other members of the colony. The shortest route is eventually taken by all the ants to transport food from the source to the nest the most efficiently. ACO simulates such a process using a number of artificial ants to build-up the shortest route as the solution to a given optimization problem [6], [11]-[13]. Different ACO algorithms have been proposed and produced satisfactory outcomes in solving various TSPs, but many ACO algorithms exhibit stagnation at the local optima and slow convergence. Even more challenging in sequential ACO algorithms is the direct conflict between improving stagnation at the local optima and accelerating convergence. Avoiding local stagnation requires adding other disturbing operations into an ACO algorithm, but such an action inevitably prolongs the convergence [8], [9]. Given the multi-agent nature of ACOs, a few recent studies have explored the likelihood of efficiently applying parallel ACO algorithms to solve TSPs [14]-[19].

Among these cases of parallel ACO experiments, Tan et al. in [14] focused their study on exploring how much speed-up could be made in running a parallel Max-Min Ant System (MMAS) implemented in MapReduce to solve Oliver30, St70, Tsp225 and Rat783 by increasing the number of processors in the computer cluster. This study showed that more processors should bring higher speed-ups for all four cases. However, this study did not present any statistical detail on the quality of the solutions and the average running time resulting from both the sequential and parallel MMASs on the same TSP. Wu et al. reported an experiment comparing both the quality of the solution and the average running time from running a sequential ACO and MapReduce-based parallel ACO to solve Gr666 [15]. From the outcomes of this sole TSP, it indicated that the parallel ACO performed better than the sequential ACO in both the quality of the solution and computing efficiency. Unfortunately, the comparison was based on data from this single case. Like [14], the study reported in [16] compared the outcomes of parallel ACOs with different number of ants and clusters of computers by assuming that both sequential and parallel ACOs have the same accuracy in the quality of their solutions for the selected TSPs. In [17], some outcomes from a parallel MMAS based on Spark MapReduce applied to five cases of TSPs were reported. However, it focused on reaching the global optimal solution with a random number of ants assigned to the different cases without rationalization. It also did not present any statistical details for further comparison. As all efforts on exploring parallel ACO algorithms for solving TSPs or alike in distributed and/or cloud environments are still in their infant

Manuscript received January 22, 2018; revised March 3, 2018.

G. Dong is with College of Computer and Information Engineering Computing Science, Inner Mongolia Agricultural University, Hohhot, China (e-mail: donggf@imau.edu.cn).

W. Li, Y. Wang, W. W. Guo are with School of Engineering and Technology, Central Queensland University, North Rockhampton QLD 4702, Australia (e-mail: w.li@cqu.edu.au, y.wang2@cqu.edu.au, w.guo@cqu.edu.au).

J. Shen is with School of Information Systems and Technology, University of Wollongong, Wollongong, NSW 2522, Australia (e-mail: jshen@uow.edu.au).

stages, gaps in the above-mentioned studies are understandable, and only through making more efforts these and new gaps in this emerging area can be bridged.

In this paper, we try to fill some of these gaps by making a comparison of both the quality of the solutions and the running time resulting from applying a sequential ACO and a MapReduce-based parallel ACO to three cases of TSPs (Eil51, Eil76 and KroA100) on a normalized and thus, comparable ground. Firstly we aimed to quantify the relevant differences in the running time between the sequential and parallel ACOs for different-sized TSPs, so as to minimize the dependence on the computing environments exhibited in previous studies based on a comparison of the absolute running time. Secondly we aimed to answer the seemingly contradictory assumption or opinion on the quality of the solutions to TSPs obtained from both the sequential and parallel ACOs in the previous studies. Lastly, incorporating the detailed outcomes of this study with those from previous studies, we discuss the balance between the accuracy and consistency of parallel ACOs for TSPs in an effort on guiding the future studies in this emerging area of research and applications.

In the next section of the paper we briefly describe the processes of the metaheuristic ACO algorithm for TSPs with necessary modifications in applications. The MapReduce framework was then introduced to convert a traditional sequential ACO into a parallel ACO. Our simulation settings are described, and simulation results of using both ACO algorithms used to solve Eil51, Eil76 and KroA100 are presented. Based on these results, comparisons and discussions are then made with the outcomes of previous studies. Conclusions are drawn in the the last section of this paper.

## II. Sequential Metaheuristic ACO Algorithm for TSPs

Since the 1990s, many ACO algorithms have been devised to solve various optimization problems including TSPs, such as the Ant System (AS), Max-Min Ant System (MMAS) and Ant Colony System (ACS) [11]-[13]. These attempts can be generalized as a metaheuristic algorithm for TSPs, which is shown in Algorithm 1.

```
Set parameters, initialize pheromone trails
    while termination condition not met do
        Construct AntSolutions (AS) for TSP
        Apply LocalOptimalOperator (optional)
        Update Pheromones
    endwhile
Terminate with AS as the solution
```

Algorithm 1. The metaheuristic ACO algorithm used for TSPs.

After initialization, the metaheuristic algorithm iterates over three phases; in each iteration, a number of solutions are constructed by the ants; these solutions are then improved through a local optimal operator and finally, the pheromone is updated.

Different mathematical models have been proposed to control this metaheuristic process, but the backbone is around the following main controllers.

- In construction of a solution, the ants select the next city to be visited through a stochastic mechanism. The probability for ant *k* in city *i* to move to the next city *j* is determined by

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in S_k} \tau_{il}^\alpha \cdot \eta_{il}^\beta} & \text{if } j \in S_k \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $S_k$ is the intersection of the candidate list of city *i* and the set of cities that ant *k* has not visited yet, $\alpha$ and $\beta$ control the relative importance of the pheromone versus the heuristic information $\eta_{ij} = \frac{1}{d_{ij}}$, which depends on the distance $d_{ij}$ between cities *i* and *j*, and $\tau_{ij}$ is the pheromone associated with the path joining cities *i* and *j*.

- For the next iteration, the pheromone values $\tau_{ij}$ are updated by all the *m* ants that have built a solution at the end of current iteration by

$$\tau_{ij}(n+1) \leftarrow (1-\rho)\cdot\tau_{ij}(n) + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{2}$$

where $\rho$ is the evaporation rate and $\Delta\tau_{ij}^k$ is the quantity of pheromone laid on the edge (*i*, *j*) by ant *k* expressed as

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ visited edge } (i,j) \text{ in its tour,} \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where *Q* is a constant and $L_k$ is the length of the tour constructed by ant *k*.

The probability-based stochastic controller (1) has been proven prone to stagnation at local maxima when being used alone. Therefore, even though stated as optional, various local optimal operators have been incorporated with the basic ACO so as to further improve the performance of the ACOs [8]-[10]. However, the more sophisticated the local optimal operator, the slower the whole ACO process. Particularly for sequential ACOs, a sophisticated local optimal operator may lead to a higher quality of solution to TSPs, but it prolongs the process significantly, especially for large TSPs.

As our main goal is to compare the performance between the sequential and parallel ACO algorithms used for TSPs, large TSPs and sophisticated local optimal operators were not selected in this study to avoid the obvious bias against sequential ACO algorithms. In fact, Eil51, Eil76 and KroA100 were selected because these cases are within the range that sequential ACO algorithms have performed best [8], [9].

As ACOs are prone to local maxima, as-observed in many existing studies [9], [10], [19], we make a small change to the basic ACO for TSPs in Algorithm 1 to deal with such stagnation. In the probability-based stochastic controller (1), instead of directing ant *k* in city *i* to next city *j* according to the highest probability $p_{ij}^k$, we make a pool of three potential cities with the three highest probability $p_{ij}^k$ and direct ant *k* in city *i* to next city *j* by a random pick from the pool. Such a simple modification to the basic ACO should diversify the search performed by subsequent ants during the iteration to some extent, which should reduce the likelihood for several ants to produce identical solutions in the same iteration, hence improving the stagnation at the local maxima. We call this modified algorithm the Random on Best Probability ACO or RBPACO. The flowchart of this sequential RBPACO is presented in Fig. 1.

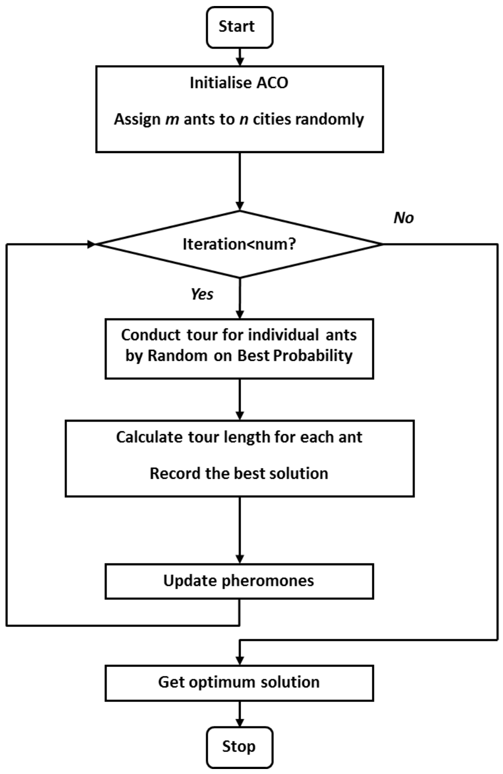
Start
Initialise ACO
Assign *m* ants to *n* cities randomly
Iteration<num?
No
Yes
Conduct tour for individual ants by Random on Best Probability
Calculate tour length for each ant
Record the best solution
Update pheromones
Get optimum solution
Stop

Fig. 1. The flowchart of the sequential RBPACO algorithm used for TSPs.

## III. MapReduce Framework and the Parallel ACO Algorithm for TSPs

MapReduce is a distributed computing framework proposed by Google in 2004 [20], [21]. MapReduce consists of two major procedures called the *Map Stage* and the *Reduce Stage*, which are connected by other linking services, as illustrated in Fig. 2.

In the beginning, the input is split into a number of parallel parts (or Splits), which are then fed to the Map process that performs business processing such as sorting, filtering and relisting, assisted by linking services (or Shuffle) to generate concurrent Map tasks. All independent Map tasks are transferred to the *Reduce Stage* for further processing, including merging the results with the same key carried from Map tasks, and outputting the final result.

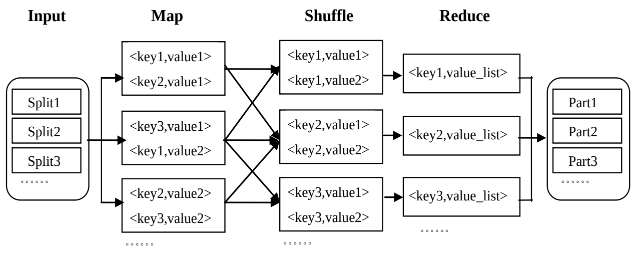

Fig. 2. The operational flowchart of the MapReduce framework.

The sequential RBPACO algorithm used for TSPs can be converted to a parallel RBPACO algorithm fitting to the MapReduce framework. Fig. 3 shows the operational flowchart of the converted parallel RBPACO algorithm in MapReduce. Different tasks are performed at each of these steps.

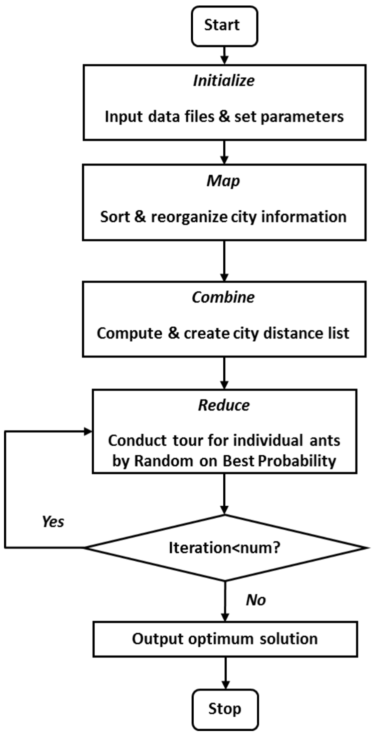
Start
*Initialize*
Input data files & set parameters
*Map*
Sort & reorganize city information
*Combine*
Compute & create city distance list
*Reduce*
Conduct tour for individual ants by Random on Best Probability
Yes
Iteration<num?
No
Output optimum solution
Stop

Fig. 3. The flowchart of the parallel RBPACO algorithm used for TSPs in MapReduce.

- In the *initialize* step, the specified data files are fetched into the systems and all the required parameters for the ACO-TSP are initialized. This is a part of the *Map Stage*.
- The *Map* step decomposes the information of the involved cities and ties each city with its coordinates. This is also a part of the *Map Stage*.
- The *Combine* step computes distances between any two cities and creates the sorted city lists for individual cities. This is also a part of the *Map Stage*.
- The *Reduce* step coordinates the execution of the RBPACO on concurrent processors for individual ants until the end of the entire iteration with the optimum output, which also marks the end of the *Reduce Stage*.

## IV. Experimental Settings and Simulation Results

Both the sequential and parallel RBPACO algorithms were implemented using Java with JDK1.6. The parallel RBPACO was built on Hadoop-1.2.1 as a virtual cloud computing platform for concurrent processing. The hardware was a Lenovo R680 server with 4 ×8 CPU cores and a total memory of 1024 GB.

The three TSPs, Eil51, Eil76 and KroA100, are selected from TSPLIB [22]. The parameters used for the individual problems are listed in TABLE I. Note that for leveling the ground for comparison, we deliberately set most parameters the same and use an ant/city load factor of 1 in all cases. This should ensure that exactly one ant starts at one city during the parallel processing, hence to eliminate the unevenness created by using different load factors for the different cases in the previous studies.

TABLE I: THE PARAMETERS USED FOR SIMULATING RBPACO FOR TSPS.

| TSP | $n$ | $m$ | $\alpha$ | $\beta$ | $q_0$ | $\tau_0$ | $\rho$ | $Q$ | $m/n$ load |
|---|---|---|---|---|---|---|---|---|---|
| Eil51 | 51 | 51 | 1 | 2 | 0.5 | 0.1 | 0.1 | 100 | 1 |
| Eil76 | 76 | 76 | 1 | 2 | 0.5 | 0.1 | 0.1 | 100 | 1 |
| KroA100 | 100 | 100 | 1 | 2 | 0.5 | 0.1 | 0.1 | 100 | 1 |

For both the sequential and parallel RBPACOs, the maximum number of iterations was set to 300 for all three cases, and each case was repeated 10 times for self-verification and assessing the quality of the solutions later. All the ants were placed in the same city at the beginning of the execution for the sequential RBPACO, whereas one ant was placed in each city to start the parallel processing. The statistical results of our simulations for both algorithms are given in Table II and Table III. Note the relative errors of the actual solutions were normalized with their corresponding known global optimal values.

TABLE II: THE SIMULATION FOR THE SEQUENTIAL RBPACO (10 RUNS PER CASE)

| TSP | Solution (relative error %) | | | Time (*ms*) | | | Global optimal |
|---|---|---|---|---|---|---|---|
| | Best | Worst | Mean | Best | Worst | Mean | |
| Eil51 | 436 (2.3%) | 452 (6.1%) | 442 (3.8%) | 3243 | 3293 | 3267 | 426 |
| Eil76 | 555 (3.2%) | 579 (7.6%) | 565 (5.1%) | 10344 | 10735 | 10474 | 538 |
| KroA100 | 22390 (5.2%) | 23387 (9.9%) | 22796 (7.1%) | 23354 | 23354 | 23354 | 21282 |

TABLE III: THE SIMULATION FOR THE PARALLEL RBPACO (10 RUNS PER CASE)

| TSP | Solution (relative error %) | | | Time (*ms*) | | | Global optimal |
|---|---|---|---|---|---|---|---|
| | Best | Worst | Mean | Best | Worst | Mean | |
| Eil51 | 427 (0.2%) | 435 (2.1%) | 431 (1.2%) | 2686 | 2810 | 2704 | 426 |
| Eil76 | 540 (0.4%) | 551 (2.4%) | 545 (1.3%) | 5692 | 5818 | 5748 | 538 |
| KroA100 | 21296 (0.1%) | 22165 (4.1%) | 21576 (1.4%) | 11687 | 11744 | 11707 | 21282 |

## V. COMPARISON AND DISCUSSION

### A. *Computing Efficiency*

All the previous studies show that parallel ACOs are faster than sequential ACOs for TSPs [14]-[17]. Although there were mismatched settings between the sequential and parallel ACO algorithms in these studies, the general trend points to the fact that the larger the TSP, the more efficient the parallel computing. The outcome from our study not only reaffirms this general trend, but also allows a quantitative comparison to be made on a levelled ground between the two algorithms.

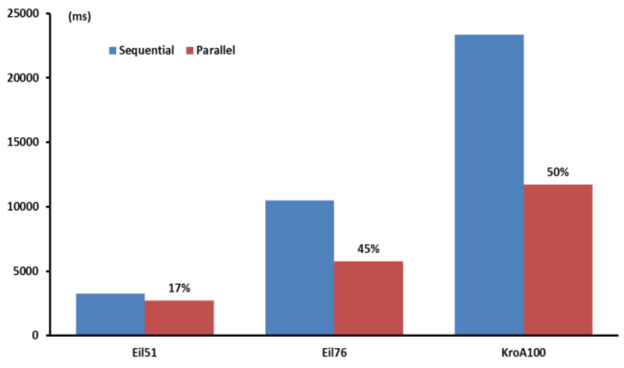

Fig. 4. Aomparison of the computing cost between the sequential and parallel RBPACOs (time saved shown as a percentage).

For Eil51, the smallest TSP among the three cases, the parallel ACO saved only 17% of computing time over the sequential ACO (Fig. 4). This small difference is likely due to the extra overhead required by the *MAP Stage* in the parallel paradigm. By increasing the size of the TSP to 76 (Eil76), a 45% saving in the computing time over the sequential ACO was achieved by the parallel ACO, and a 50% saving in the computing time was even recorded for a TSP size of 100 in KroA100. However, if we notice that the computing time of the sequential ACO for KroA100 was fixed at 23354 ms after 300 iterations for each run (TABLE II), such a comparison for larger TSPs seems meaningless. This indicates that larger TSPs may be beyond the reach of (some) sequential ACOs in terms of the computing efficiency.

### B. *Quality of Solutions*

Some previous studies simply assume that the quality of the solutions from obtained both sequential and parallel ACOs are similar [14], [16] whereas others showed parallel ACOs produced better solutions than sequential ACOs, at least among the best solutions [15], [17]. Even in the latter cases, the comparison was based on the mismatched settings between the sequential and parallel ACO algorithms.

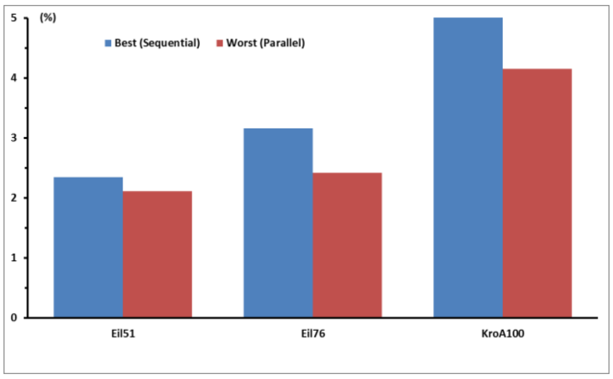

Fig. 5. A comparison of the relative accuracy of the solutions obtained from the sequential and parallel RBPACOs used for TSPs.

Our results confirm that the parallel RBPACO produced consistently better solutions than the sequential one did, though both did not reach the global optimal solution in 300 iterations used for Eil51, Eil76 and KroA100 (TABLES II & III). From the 10 repeated runs, the best solutions from the parallel algorithm used for Eil51, Eil76 and KroA100 are all within an error of 0.5% to the global optimal solutions, respectively and even the worst ones are around 4% or less, with an average error of less than 1.5% (TABLE III). On the contrary, the best obtained from the sequential algorithm is with an error of about 2.3% for the smallest TSP (Eil51) and the worst is around 10% for the largest TSP (KroA100) (TABLE II). The most convincing fact is that the best solutions to the three TSPs obtained from the sequential ACO are still worse than the worst solutions to the same cases obtained from the parallel algorithm (Fig. 5).

In the parallel ACO, each of the cities has one ant to search its own path and thus, all the ants are more likely to go through a much larger variety of routes, a case of from-many-to-many, when compared with the situation of the sequential ACO, where all the ants begin their search from the same city, a case of from-one-to-many. Thus, the parallel algorithm should, in theory, return a better solution than or at least the same as the sequential one can does under the comparable conditions.

### C. *Accuracy and Consistency of the Parallel ACO*

In our experiments, the known global optimal solution to any of the three cases was not reached in 10 repeated runs of the parallel RBPACO within 300 iterations each run. The best solution for Eil51 is within an error range of 0.2-2.1% to the global optimal with an average error of 1.2%; that for Eil76 is within an error range of 0.4-2.4% to the global optimal with an average error of 1.3%; that for KroA100 is within an error range of 0.1-4.1% to the global optimal with an average error of 1.4%. In KroA100, the second worst solution has an error of 1.9%, meaning the worst solution with an error of 4.1% may be an exception (Fig. 6). Nonetheless, the average errors of these three cases show high consistency at around 1.3%, regardless of the size of the TSP. In our cases, we found that at least one best solutions within an error of 0.5% to the global optimal solution was reached in every three repeats for all three cases, a very high level of consistency.

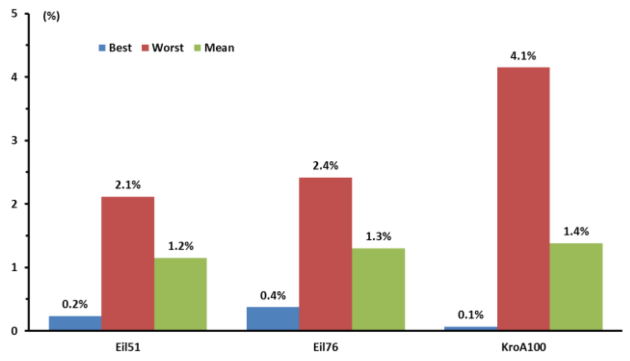

Fig. 6. The relative accuracy of the solutions obtained from the parallel RBPACO.

We indeed tried to simulate Eil51, Eil76 and KroA100 using the parallel RBPACO with 800 iterations and found the global optimal solutions occasionally. However, the running time was more than doubled over that found for 300 iterations. Other studies have reported that the global optimal is reached for several TSPs using various hybrid algorithms combining ACO with GA [8], [9], but the running time will be significantly increased. In real applications, the decision becomes if it is worth of spending more time in the hope of getting the global optimal over quickly finding a near-best solution to a given problem.

## VI. Conclusions

It is reaffirmed that the parallel algorithm is superior in computing efficiency over the sequential algorithm, particularly for larger TSPs. However, such a comparison may become meaningless if the size of TSPs keeps increasing. This is because large TSPs may be beyond the feasibility of applying sequential ACO algorithms in solving these problems. It is confirmed that the parallel ACO produces more accurate solutions than the sequential ACO, though both did not reach global optimal solutions within 300 iterations. The more convincing fact is that the worst solution among the 10 repeated runs obtained from the parallel ACO is still better than the best solution obtained from the 10 repeated runs in the sequential ACO. In 10 repeated runs, the mean best solutions of these three cases are around 1.3% to their global optimal solutions regardless of the size of the TSP. At least one best solution is within an error of 0.5% to the global optimal solution in every three repeats for all three cases, which is remarkably consistent.

## References

[1] E. L. Lawer, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, *The Traveling Salesman Problem*, New York, USA: Wiley, 1985.

[2] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications*, Berlin, Heidelberg: Springer-Verlag, 1994.

[3] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498-516, 1973.

[4] C. Rego, D. Gamboa, F. Glover, and C. Osterman, "Traveling salesman problem heuristics: Leading methods, implementations and latest advances," *European Journal of Operational Research*, vol. 211, pp. 427–441, 2011.

[5] S. Chatterjee, C. Carrera, and L. A. Lynch, "Genetic algorithms and traveling salesman problems," *European Journal of Operational Research*, vol. 93, pp. 490-510, 1996.

[6] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, pp. 29-42, 1996.

[7] L. Hu, J. Liu, C. Liang, F. Ni, and H. Chen, "A Phoenix++ based new genetic algorithms involving mechanism of simulated annealing," *International Journal of Distributed Sensor Networks*, vol. 2015, Article ID 806708, 8 pages, 2015.

[8] G. Dong and W. W. Guo, "Cooperative ant colony system and genetic algorithm for TSPs," *Lecture Notes in Computer Science*, vol. 6145, pp. 597-604, 2010.

[9] G. Dong, W. W. Guo, and K. Tickle, "Solving the traveling salesman problem using cooperative genetic ant systems," *Expert Systems with Applications*, vol. 39, pp. 5006-5011, 2012.

[10] Y. Yan, H. S. Sohn, and G. Reyes, "A modified ant system to achieve better balance between intensification and diversification for the traveling salesman problem," *Applied Soft Computing*, vol. 60, pp. 256-267, 2017.

[11] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53-66, 1997.

[12] T. Stutzle and H. Hoos, "MAX-MIN ant system," *Future Generation Computer Systems*, vol. 16, pp. 889-914, 2000.

[13] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization: artificial ants as a computational intelligence technique," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 28-39, 2006.

[14] Q. Tan, Q. He, and Z. Shi, "Parallel Max-Min ant system using MapReduce," *Lecture Notes in Computer Science*, vol. 7331, pp. 182-189, 2012.

[15] H. Wu *et al.*, "MapReduce-based ant colony optimization," *Computer Integrated Manufacturing Systems*, vol. 18, pp. 503-509, 2012.

[16] A. Mohan and G. Remya, "A parallel implementation of ant colony optimization for TSP based on MapReduce framework," *International Journal of Computer Applications*, vol. 8, no. 1, pp. 9-12, 2014.

[17] L. Wang, Y. Wang, and Y. Xie, "Implementation of a parallel algorithm based on a Spark cloud computing platform," *Algorithms*, vol. 8, pp. 407-414, 2015.

[18] E. N. Kencana, I. Harini, and K. Mayuliana, "The performance of ant system in solving multi traveling salesmen problem," *Procedia Computer Science*, vol. 124, pp. 46-52, 2017.

[19] S. G. Ozden, A. E. Smith, and K. R. Gue, "Solving large batches of traveling salesman problems with parallel and distributed computing," *Computers and Operations Research*, vol. 85, pp. 87-96, 2017.
[20] D. Jeffrey and G. Sanjay, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107-113, 2008.
[21] L. Ralf, "Google's MapReduce programming model-revisited," *Science of Computer Programming*, vol. 70, no.1, pp. 1-30, 2008.
[22] TSPLIB. [Online]. Available: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/

**Gaifang Dong** recently completed her PhD degree at Inner Mongolia Normal University. She also received her BS and MS degrees both in computer science from Inner Mongolia Normal University and Guizhou University in China in 2002 and 2005, respectively. She is currently an Associate Professor in the College of Computer and Information Engineering of the Inner Mongolia Agricultural University. Her research interests include computational intelligence, bioinformatics computing and parallel computing.

**Wei Li** received BS, MS and PhD degrees in computer science from Harbin University of Science & Technology, Harbin Institute of Technology and the Institute of Computing Technology of Chinese Academy of Sciences, China in 1986, 1989, and 1998, respectively. He is currently a Senior Lecturer in information technology with the School of Engineering & Technology, Central Queensland University, Australia. He has been a peer reviewer for a number of international journals and a program committee member of 30 international conferences in his research domain. His research interests include dynamic software architecture, P2P volunteer computing and multi-agent systems.

**Jun Shen** was awarded a Ph.D. in 2001 at Southeast University, China. He held positions at Swinburne University of Technology in Melbourne and University of South Australia in Adelaide before 2006. He is an Associate Professor in the School of Computing and Information Technology at University of Wollongong in Wollongong, NSW, Australia. He is a senior member of three institutions: IEEE, ACM and ACS. He has published more than 120 papers in prestigious journals (including IEEE Transactions) and conferences (for example, IEEE Big Data) in CS/IT areas, in particular on computational intelligence topics. His expertise includes Web services, cloud computing and learning technologies including MOOC. He has been Editor, PC Chair, Guest Editor and PC Member for numerous journals and conferences published by IEEE, ACM, Elsevier and Springer. A/Prof Shen is also a current member of ACM/AIS Task Force on Curriculum MSIS 2016.

**Yucang Wang** works at Central Queensland University Australia. He obtained a Bachelor of physics in 1987, a Master's degree in 1992 and Ph.D. in 1998 in geophysics. He has post-doc experience from 1998 to 2000 in the Institute of Mechanics, Chinese Academy of Sciences. From 2001 to 2008, he worked in the Earth System Science Computational Centre (ESSCC), University of Queensland, Australia. He was the major physical developer of Esys-Particle code, an open source code. From 2009 to May 2014, he worked in the Earth Science and Resource Engineering, Commonwealth Science and Industry Research Organization (CSIRO), Australia. He has been developing and using the Discrete Element Model (DEM) in the past 20 years.

**Xueliang Fu** is a Professor in algorithms of graph theory at Inner Mongolia Agricultural University in China. He received the MS and Ph.D. in software engineering from Dalian University of Technology, China in 2005 and 2008, respectively, and a BS in computer science in 1992. His research interests include domination number of graph and intelligent computation. He has published about 50 papers in international journals and conference proceedings.

**William W. Guo** is a Professor in applied computation and mathematics education at Central Queensland University Australia. He received a Ph.D. from the University of Western Australia in 1999, Master's of science in 1991 and Bachelor of engineering in 1982 in China. His research interests include computational intelligence, mathematics education, data and image processing, modelling and simulation. He has published over 100 papers in international journals and conference proceedings, two mathematics textbooks and co-edited four special issues in international journals. He has supervised multiple Ph.D. students and served as a keynote speaker at many international conferences and regional events. He has abundant experience in leadership and academic governance through his services as Dean/Deputy Dean of School, University Academic Board and many committees. He is a member of IEEE, ACM, ACS and Australian Mathematics Society (AUSTMS).