MSC 05C10, 03D15, 68W10 DOI: 10.14529/mmp200409

PNACO: PARALLEL ALGORITHM FOR NEIGHBOUR JOINING HYBRIDIZED WITH ANT COLONY OPTIMIZATION ON MULTI-CORE SYSTEM

***W.B. Yahia*[1], *M.W. Al-Neama*[2], *G.E. Arif*[1]**
[1]Tikrit University, Tikrit, Iraq
[2]Mosul University, Mosul, Iraq
E-mails: warifb@gmail.com, mwneama@uomosul.edu.iq, ghasanarif@tu.edu.iq

One of the most interesting and relevant approaches for solving optimization problems are parallel algorithms that work simultaneously with a large number of tasks. The paper presents a new parallel algorithm for NACO that is a hybrid algorithm that consists of the Ant Colony Optimization method combined with the Neighbour Joining method to get accurate and efficient results when solving the Traveling Salesman Problem. Through carrying out comprehensive experiments using a wide variety of real dataset sizes and the multi-core system, the practical results show that the developed program outperforms NACO in terms of execution time and consumed storage space. Availability and implementation: source codes in MATLAB 2017 are publicly available at Internet[1].

Keywords: ant colony optimization; neighbour joining method; traveling salesman problem; parallel algorithm; multi-core system.

Introduction

One of the most interesting and relevant methods for solving optimization problems are parallel algorithms that work simultaneously with a large number of current solutions. In 2020, Warif, Al-Neama, and Ghassan presented a new hybrid method to solve Traveling Salesman Problem (TSP), which is resulted from combining Ant Colony Optimization (ACO) and Neighbour Joining method (NJ), and called the new method Ant Colony Optimization and Neighbour Joining to solve TSP (NACO) [1].

The main idea of the NACO method is to reduce the distance matrix generated by the cities which the salesman must to visit. The NACO tries to merge the cities by constructing a phylogenetic tree and then generating a new matrix called LEAF1 which represents the first stage of the joined leaves. When the salesman reaches any city within this matrix, he is moved directly to the next corresponding city without doing any computations. In NACO, runtime and storage space are the most consuming phases appear when the program performes re-computations of the moving probability on each edge.

Also, visited cities are removed from the distance matrix and the Roulette Selection function that responsible for choosing the next city is computed. The MATLAB profile of NACO shows that the Roulette Selection function computation typically exhausts up to 45% of the execution time [1].

In NACO, the complexity of the searching optimal solution is of the order $O(n^2(n-\omega))$. Recently, diverse software approaches have been presented to reduce the consumed time and space in the ACO method. Such approaches include parallel processing [2], and usage of easily accessible accelerator technologies such as GPU [3].

From this point of view, NACO algorithm has two obstacles.

The first one is space complexity especially when dealing with the big number of cities, which would require the occupancy of capacity storage for most of the programs. As an example, finding the shortest path with a dataset that includes (10k) cities requires several gigabytes of the hard disk, which can not be provided by the most computer resources.

The computational load of Roulette Selection function calculations increases greatly as the number of ants increases. Actually, sometimes, even the best algorithms fail in to obtain good accuracy results and deal with these complexities efficiently at the same time.

Therefore, this paper presents a new parallel algorithm for the NACO method to overcome the mentioned obstacles without altering its accuracy. Our aim is to produce a superlative method over existing ones, especially in execution time.

NACO algorithm has good accuracy with reducing the memory requirement and acceptable accelerating execution time. Nowadays, the best solution to increase the speed-up of NACO method is considered to be the parallelization widespread programming method that allows multiple independent processes that share the same resources to be executed concurrently at less time. A platform with a multi-core system is used to implement this program.

The paper is organized as follows. A brief summary of parallel computing is presented. Then, explanations of the independent procedures of NACO are given. Next, we propose the optimization algorithm called PNACO for the NACO computation using parallelism on multi-cores. Finally, we present the comparative study of the parallel proposed algorithm and the sequential proposed algorithm.

1. Parallel Computing using Multi-Core System

Parallel computing is the simultaneous use of multiple-core system to solve a computational problem. A problem is divided into tasks that can be solved concurrently. Each task is divided into a sequence of instructions (threads). The instructions (threads) are executed simultaneously on different processors. However, the choice of computer platform often significantly influences on the result [4].

As regards advantages [5], parallel programming is able to
1) decrease the execution time required to get the result and to solve large problems;
2) use non-local resources on a local area network, or even Internet when local computing resources are insufficient.

On the other hand, the pivotal disadvantage of parallel programming is the communication time required for simultaneity and transferring of data between processors. Fig. 1 shows the parallel programming execution [6, 7].

1.1. Parallel Algorithm

Algorithm design is the most significant stage in problem solving. The complexity of today's applications coupled with the widespread use of parallel computers led to great interest in the parallel algorithms.

A parallel algorithm is a method to solve a given problem designed to be performed on a parallel computer. There exist different levels of parallelism that can be presented in parallel algorithms as follows [4]. The first one is the lowest level of parallelism called Instruction-level parallelism, where instructions are executed simultaneously. The second one called data-level parallelism is the execution of the same operation(s) on all the

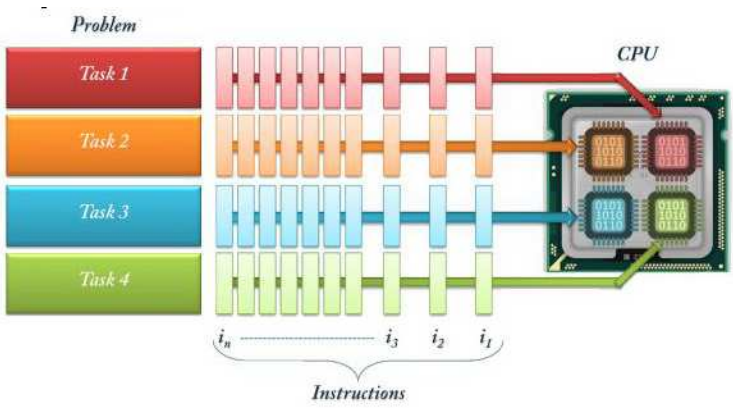

Fig. 1. Parallel programming execution

data elements at the same time. The third one called task-level parallelism uses tasks to implement parallelization. A simple definition is the following: a task is a list of instructions along with input data that produces an optional output, e.g. the result of executing the instructions.

The use of task-level parallelism to do multiple tasks in parallel on the same dataset is contrary to data-level parallelism. For example, it is clear that the tasks to find the maximum and minimum values on the same dataset are possible to execute in parallel.

1.2. Measurements

The run-time, speedup, and efficiency are the common timescale units to perform measurements for the proposed parallel program. Parallel run-time is the elapsed time for complete computation of the best tour, including all comparison, updating, and all operations. The ratio between the parallel execution times of the two involved programs is called the parallel speedup and given by the following equation:

$$S = \frac{T_{seq}}{T_{par}}, \tag{1}$$

where T_{seq} is the running time of the sequential program and T_{par} is the running time of the parallel program. The ratio of the corresponding parallel speedups of the number of cores is called parallel efficiency and given by the following equation:

$$E = \frac{S}{P}.$$

Taking into account (1), the efficiency has the form:

$$E = \frac{T_{seq}}{P.T_{par}}.$$

2. Parallel of NACO Algorithm

To speed up the algorithm, it is advisable to run ants in separate execution threads in parallel, but here it should be taken into account that a large number of ants compared available multi-core processors will create an execution queue that will slow down the

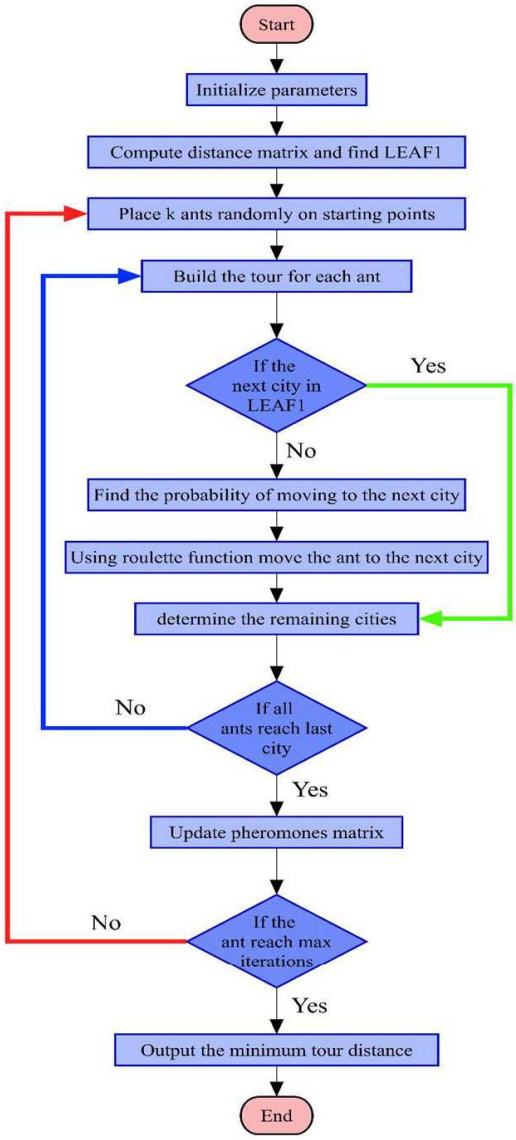

Fig. 2. The Flow chart of NACO

operation of the algorithm. In NACO algorithm, there exist three computational loops that the entire algorithm counts on completely as shown in Fig. 2.

Therefore, any modification of these loops will affect the efficiency of the algorithm directly. In Fig. 2, red color shows the first computational loop, which is considered to be the main loop of the algorithm; however, it is not possible to modify this loop and apply the parallel loop on it, because the components inside the loop are not independent. The second loop, which is shown in green color in Fig. 2, is the interior loop of the algorithm; however; it is also not possible to modify this loop and apply the parallel loop on it. Indeed, this loop is created for moving an ant from a city to another depending on the Roulette Selection function where its values differ depending on the placing ants randomly in the beginning of the algorithm. In other words, the lines inside this loop are not independent.

Finally, the third computational loop, which is shown in blue color in Fig. 2, is the loop to be modified. In fact, this loop is in charge of finding k tours for k ants independently. Therefore, applying the parallel loop is possible here.

Optimization uses the parallel tools library in MATLAB which is intended for systems with shared memory. It is easy to see that solving problems related to the accuracy of algorithms in the described way entailed significant losses in the efficiency of using loops. The independencies of routes from each other with a large number of iterations are precisely the conditions in which it is advisable to use optimization by means of parallel programming.

In MATLAB, the parallel tools library is suitable for this purpose: the (parfor) allows the algorithm to distribute the operations between threads whose number $Nthreads$ depends on the number of processor cores of the computer on which the algorithm runs.

First, initialize all parameters for the PNACO including distance computations, number of cities, alpha, beta, ... Then, the task parallelism process to the tour construction is based on the movement of ants which are run in parallel looking for the best tour. The construction of the tour and update of the pheromone loop are carried out until they reached the convergence criterion. The procedure of ants' movement is distributed on the available processors (P) and implemented based on task parallelism on a multi-core system.

To run PNACO algorithm in MATLAB using (parfor), identify each ant as a thread, then each thread is distributed among the available processors (P) equally. All threads assigned to each ant include an ant memory (list of all visited cities, and so on) and its movement.

Algorithm and Fig. 3 briefly summarize the process developed by each ant.

Algorithm 3.1: PNACO Algorithm

Input: cities number, ants number, iteration and ACO parameters $\rho, \alpha, \beta, \zeta_0, Q$

Output: Min of (local solution)

```
1  Compute distance matrix
2  Compute LEAF1 matrix using NJ algorithm
3  for t ← 1 to iteration do
4      parfor m ← 1 to ants number do
5          for n ← 2 to cities number do
6              if i ∈ LEAF1matrix then
7                  go to j
8              else
9                  compute P and chose j
10             store the tour h and compute the cost (h)
11         localsolution ← Min(cost(h))
12     store local solution
13     update pheromone matrix
14 globalsolution ← Min(localsolution)
```

Algorithm in MATLAB

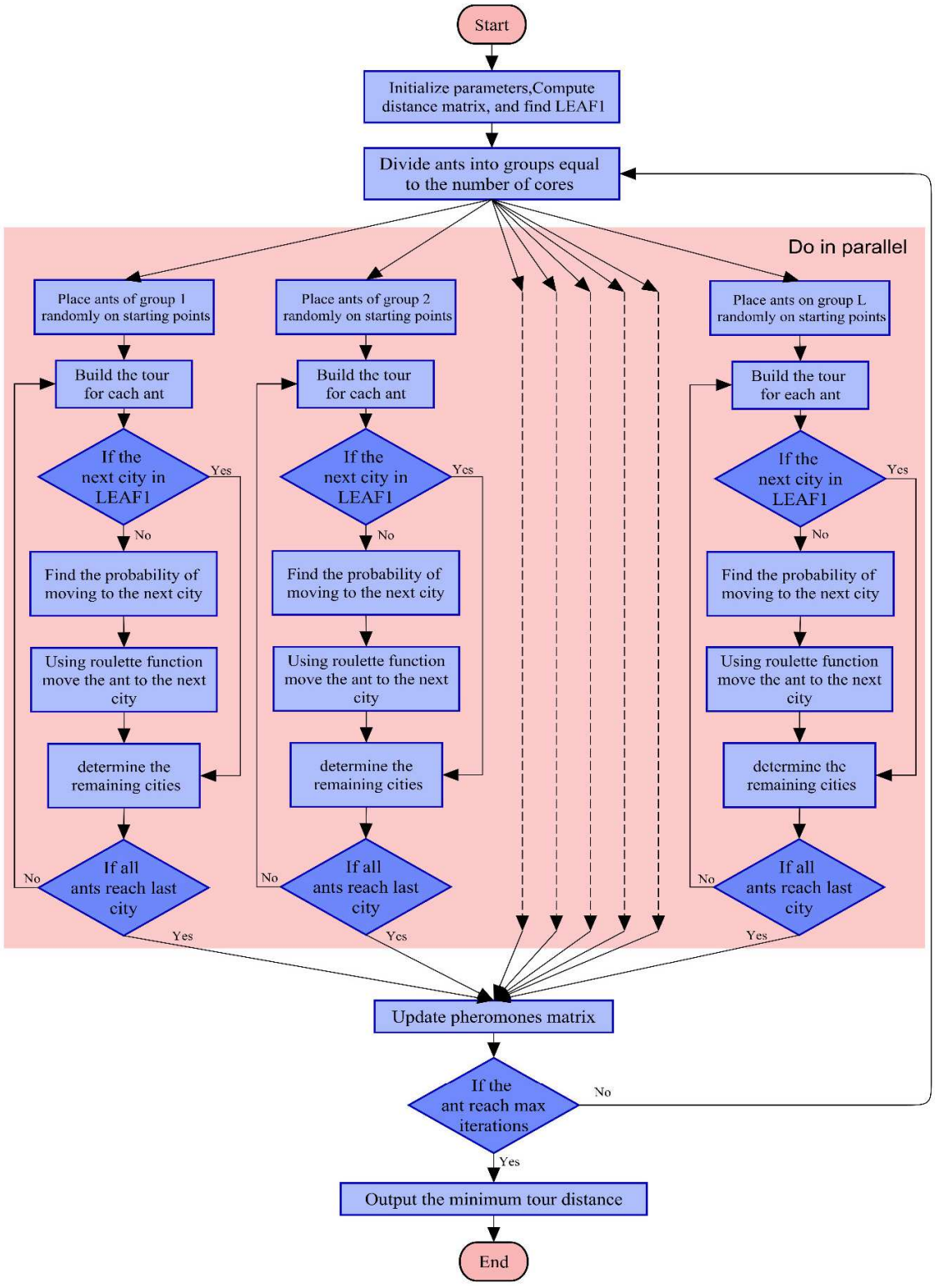

Fig. 3. PNACO flow chart

3. PNACO Performance Evaluation

This section presents and discusses the results obtained when measuring the performance of the proposed PNACO algorithm. The algorithm is implemented in

MATLAB2017 [8]. The performance of the conceived parallel implementation of the proposed algorithm was extensively evaluated using different performance measurements.

We present the evaluation methodology used to study correctly the results obtained by the described solution. The improvements achieved due to introduced optimizations for the multi-core system are intensively explored. The proposed hybrid paradigm agrees very well with the characteristics of a multicore system. The resulted program has such an advantage of fine-grained parallelism as a loop level, in which each (parfor) spawns a team of threads to occupy the multi-core processors when encountering parallel sections of code using parallel tools.

The given below implementations of all the experiments were obtained using the following information (Devises and Software):

- HP Laptop with 8GB of RAM and Intel core i7-8565U CPU,
- 64-bit windows operating system,
- MATLAB R2017a (Version 9.2) [8].

The proposed algorithm (PNACO) was tested on many datasets of TSP. In addition, the results of PNACO was compared with ACO and NACO. The datasets that used in experiments can be found at the website[2].

All the parameters that are shown in Table 1 were taken from the range of the standard parameters without any changes [9].

Table 1

PNACO, NACO and ACO parameters setup

Parameter name and symbol	**Value**
Density of pheromones Q	100
Coefficient of Pheromone evaporation ρ	0,1
Data gathering factor α	1
Indicative prediction factor β	5
Number of iterations i	200
Number of ants k	Equal to the number of cities

The datasets used in the experiments are dantzig42, att48, eil51, eil76, eil101, pr107, bier127, krob200.

3.1. PNACO Execution Time

The analysis and the results of available datasets are listed below. Experiments were executed on a 4 cores processor. The number of cities controlled the number of computations that required for a tour. The run-time (sec.) of the PNACO results against ACO and NACO are listed. Table 2 presents the time execution of ACO, NACO, and PNACO.

The number of cities has a large influence on the performance of PNACO. For the datasets (eil51, att48, and dantzig42), PNACO shows a slight low performance comparing

[2]http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/

Table 2

The execution time of ACO, NACO, and PNACO

Datasets	ACO runtime (sec.)	NACO runtime (sec.)	PNACO runtime (sec.)
dantzig42	7,92	7,59	19,61
att48	11,07	10,45	17,88
eil51	12,27	11,56	17,95
eil76	38,65	29,09	22,33
eil101	60,65	57,08	25,27
pr107	72,47	66,37	26,70
bier127	102,62	95,14	27,70
krob200	319,68	279,66	29,88

with the other algorithms. On the other hand, PNACO shows a significant improving in the performance when the number of cities is more than 76. Fig. 4 shows Relation of execution time and the number of cities.

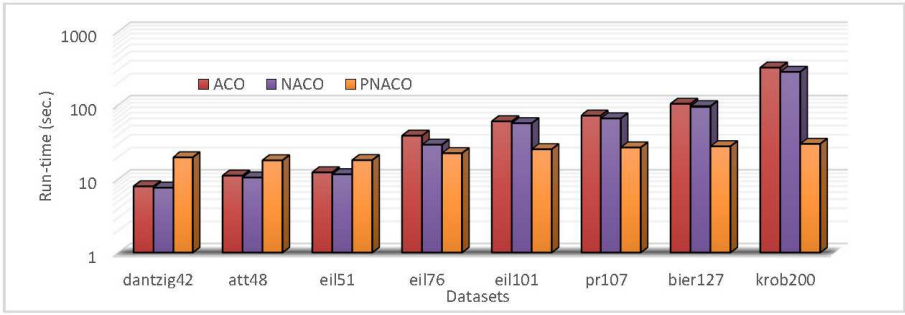

Fig. 4. Relation of execution time and the number of cities

Formula (1) is used to calculate the speedup of PNACO. The highest speedups appeared when PNACO is implemented on (krob200). Table 3 shows the speedup of PNACO comparing with ACO and NACO, in addition, Fig. 5 illustrates the comparison.

Table 3

PNACO speed-up
against ACO and NACO

Datasets	Speed-up	
	ACO	NACO
dantzig42	0,403824	0,387251
att48	0,619099	0,584352
eil51	0,683247	0,643822
eil76	1,731141	1,303013
eil101	2,400588	2,259155
pr107	2,713752	2,48558
bier127	3,704387	3,434294
krob200	10,70072	9,360971

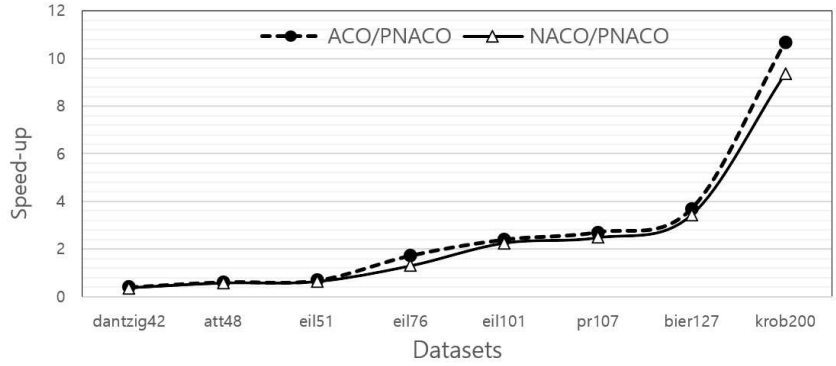

Fig. 5. Speed-up comparison between PNACO and both NACO and ACO

3.2. PNACO Efficiency

The criterion of efficiency is one of the most important criteria when it comes to the standard comparison of the competent algorithm. In the efficiency comparison test, PNACO shows an obvious superiority comparing with ACO and NACO. In fact, the efficiency of PNACO reached up to 2,66 and 2,32 comparing with ACO and NACO, respectively. Table 4 shows the full comparison between the proposed algorithm and competed algorithms.

Table 4
Efficiency comparisons between PNACO and both ACO and NACO

Datasets	ACO	NACO
dantzig42	0,10	0,095
att48	0,155	0,145
eil51	0,17	0,16
eil76	0,435	0,325
eil101	0,6	0,565
pr107	0,68	0,62
bier127	0,925	0,86
krob200	2,675	2,32

3.3. PANCO and CPU Performance

The CPU performance of NACO is illustrated in Fig. 6. On the other hand, the optimal performance of PANCO, with a full CPU usage up to 100%, is illustrated in Fig. 7.

4. The Complexity Analysis of PNACO

In ACO and NACO, the Big O-notation are $O(n^3)$ [10] and $O(n^2(n-\omega))$, respectively. Assume that the numbers of iterations, cities, and ants are all equal, and ω is positive and represents the number of rows in the matrix LEAF1. In PNACO, it is clear that

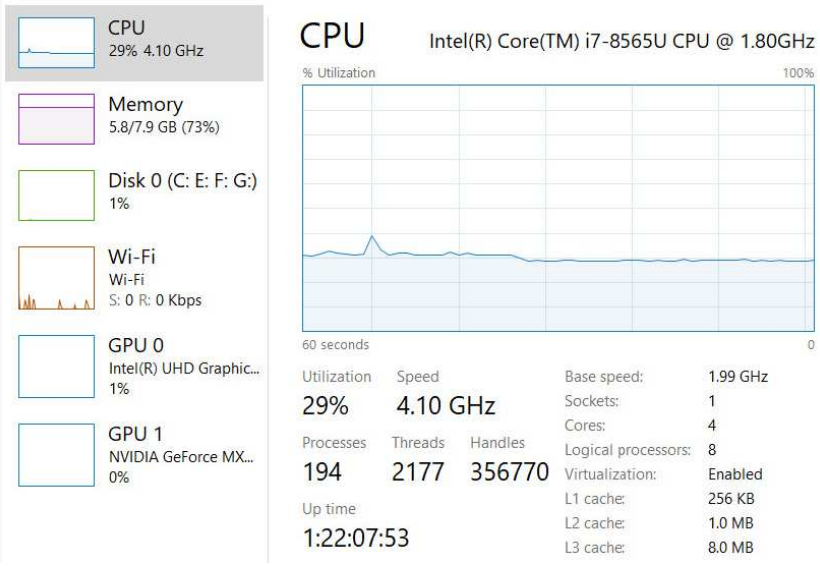

Fig. 6. The performance of CPU when implementing the NACO algorithm

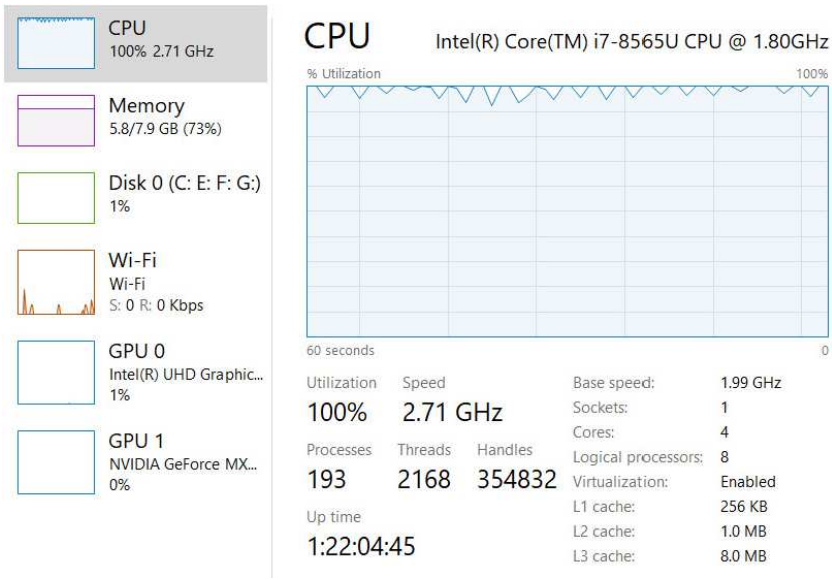

Fig. 7. The performance of CPU when implementing the PNACO algorithm

the number of ants is divided by the number of cores. Therefore, the Big O-notation of PNACO is of the order $O(n(n^p)(n-\omega))$, where p represents the number of cores. As shown in Fig. 8, this is a great development in improving program implementation time.

Conclusion

In the paper, we propose a new parallel NACO algorithm, which is targeted at the multi-cores system and accomplishes TSP computation through the efficient improvement of the overall processing time. This is due to the new suggested partitioning and scheduling approaches, integrated with (parfor) function in parallel tool in MATLAB, and the availability of more cores.

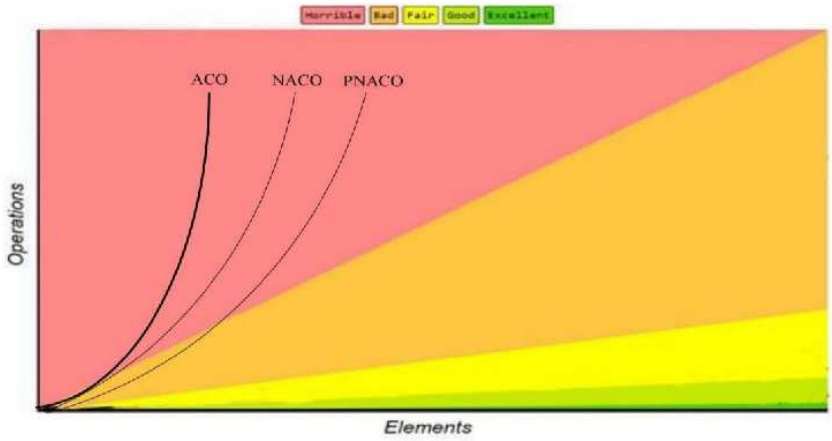

Fig. 8. The implementation time improvement of PNACO against ACO and NACO

The NACO and PNACO methods were implemented on an HP computer with 8GB of RAM and Intel Core i7-8565U CPU, 64-bit Windows operating system, and tested on the common benchmark (TSPLIB) datasets. As a result, PNACO program outperforms NACO at 9,4-fold speedup and ACO at 10,7-fold speedup when comparing on the dataset (krob200). This accomplishment is due to the perfect hybrid partitioning approaches. In addition, PNACO supreme efficiency is up to 2,66 and 2,32 with respect to the ACO and NACO, respectively, for the big cities number.

Acknowledgement. *The authors would like to thank Mosul University and Tikrit University for their computations.*

References

1. Yahia W.B., Al-Neama M.W., Arif G.E. A Hybrid Optimization Algorithm of Ant Colony Search and NeighbourJoining Method to Solve the Travelling Salesman Problem. *Advanced Mathematical Models and Applications*, 2020, vol. 5, no. 1, pp. 95–110.
2. Hsu-Chih Huang. SoPC-Based Parallel ACO Algorithm and Its Application to Optimal Motion Controller Design for Intelligent Omnidirectional Mobile Robots. *IEEE Transactions on Industrial Informatics*, 2012, vol. 9, no. 4, pp. 1828–1835.
3. Yi Zhou, Fazhi He, Yimin Qiu. Dynamic Strategy Based Parallel Ant Colony Optimization on GPUs for TSPs. *Science China Information Sciences*, 2017, vol. 60, no. 6, article ID: 068102, 12 p. DOI: 10.1007/s11432-015-0594-2
4. Al-Neama M.W., Naglaa M.R., Fayed F.G. An Improved Distance Matrix Computation Algorithm for Multicore Clusters. *Journal of Biomedicine and Biotechnology*, 2014, article ID: 406178, 13 p. DOI: 10.1155/2014/406178
5. Kotenko I.V., Saenko I.B., Kushnerevich A.G. Architecture of the Parallel Big Data Processing System for Security Monitoring of IOT Networks. *SPIIRAS Proceedings*, 2018, vol. 59, pp. 5–30.
6. Al-Neama M.W., Naglaa M.R. *A Study of Parallel Algorithms for Multiple Sequence Alignment.* Ph.D. Thesis, Al-Azhar University, 2014.

7. Nikiforov V.V., Shkirtil V.I. Estimation of Blocking Factor for Tasks in Real-Time Systems with Multi-Core Processors. *SPIIRAS Proceedings*, 2013, vol. 27, pp. 93–106.

8. *The Math Works, Computer Software.* Available at: https://www.mathworks.com (accessed 13.11.2020).

9. Peng Li, Hua Zhu. Parameter Selection for Ant Colony Algorithm Based on Bacterial Foraging Algorithm. *Mathematical Problems in Engineering*, 2016, article ID: 6469721, 10 p. DOI: 10.1155/2016/6469721

10. Yongbo Yuan, Kai Wang, Le Ding. A Solution to Resource-Constrained Project Scheduling Problem: Based on Ant Colony Optimization Algorithm. *Ninth International Conference on Hybrid Intelligent Systems*, Shenyang, China, 2009, vol. 1, article ID: 10891196, 13 p. DOI: 10.1109/HIS.2009.92

Received August 27, 2020

УДК 519.172.2+519.174+510.5 **DOI: 10.14529/mmp200409**

PNACO: ПАРАЛЛЕЛЬНЫЙ ГИБРИДНЫЙ АЛГОРИТМ ОБЪЕДИНЕНИЯ СОСЕДЕЙ В СОЧЕТАНИИ С ОПТИМИЗАЦИЕЙ КОЛОНИИ МУРАВЬЕВ НА МНОГОЯДЕРНОЙ СИСТЕМЕ

***В.Б. Яхия*[1], *М.В. Аль-Нима*[2], *Г.Э. Ариф*[1]**
[1]Университет Тикрита, г. Тикрит, Ирак
[2]Мосульский университет, г. Мосул, Ирак

Одними из наиболее интересных и актуальных подходов к решению задач оптимизации являются параллельные алгоритмы, которые работают одновременно с большим количеством задач. В этой статье представлен новый параллельный алгоритм для NACO, т.е. гибридный алгоритм, который состоит из метода оптимизации колонии муравьев в сочетании с методом объединения соседей для получения точных и эффективных результатов при решении задачи коммивояжера. Результаты, полученные на практике при проведении всесторонних экспериментов с использованием большого количества реальных наборов данных и многоядерной системы, показали, что разработанная программа превосходит NACO с точки зрения времени выполнения и потребляемого дискового пространства. Доступность и реализация: исходные коды в MATLAB 2017 размещены в открытом доступе в сети Интернет.

Ключевые слова: оптимизация колонии муравьев; метод объединения соседей; задача коммивояжера; параллельный алгоритм; многоядерная система.

Вариф Яхия, Университет Тикрита (г. Тикрит, Ирак), warifb@gmail.com.

Мохаммед Ваджид Аль-Нима, PhD, Мосульский университет (г. Мосул, Ирак), mwneama@uomosul.edu.iq.

Гассан Эззулддин Ариф, PhD, глава отдела математики, Университет Тикрита (г. Тикрит, Ирак), ghasanarif@tu.edu.iq.

Поступила в редакцию 27 августа 2020 г.