Hindawi
IET Computers & Digital Techniques
Volume 2023, Article ID 9915769, 14 pages
https://doi.org/10.1049/2023/9915769

Research Article

A Fast Fully Parallel Ant Colony Optimization Algorithm Based on CUDA for Solving TSP

Zhi Zeng[1], **Yuxing Cai**[1], **Kwok L. Chung**[1], **Hui Lin**[2] **and Jinwei Wu**[3]

[1]*School of Computer Science and Engineering, Huizhou University, Huizhou, Guangdong 516007, China*
[2]*College of Resources and Environment, Beibu Gulf University, Qinzhou, Guangxi 535011, China*
[3]*School of Mathematics and Statistics, Huizhou University, Huizhou, Guangdong 516007, China*

Correspondence should be addressed to Kwok L. Chung; klchung@hzu.edu.cn and Hui Lin; linhui@bbgu.edu.cn

Received 5 June 2023; Revised 29 August 2023; Accepted 20 September 2023; Published 31 October 2023

Academic Editor: Roger Woods

In view of the known problems of parameter sensitivity, local optimum, and slow convergence in the ant colony optimization (ACO), we aim to improve the performance of the ACO. To solve the traveling salesman problem (TSP) quickly with accurate results, we propose a fully parallel ACO (FP-ACO). Based on the max–min ant system (MMAS), we initiate a compensation mechanism for pheromone to constrain its value, guarantee the correctness of results and avoid a local optimum, and further enhance the convergence ability of ACO. Moreover, based on the compute unified device architecture (CUDA), the ACO is implemented as a kernel function on a graphics processing unit (GPU), which shortens the running time of massive iterations. Combined with the roulette wheel selection mechanism, FP-ACO has powerful search capabilities and is committed to obtaining better solutions. The experimental results show that, compared with the effective strategies ACO (ESACO) that runs on CPU, the speed-up ratio of the proposed algorithm reaches 35, and the running time is less than that of the max–min ant system-roulette wheel method-bitmask tabu (MMAS-RWM-BT) that runs on GPU. Furthermore, our algorithm outperforms the other two algorithms in the speed-up ratio and less runtime, proving that the proposed FP-ACO is more suitable for solving TSP.

1. Introduction

Traveling salesman problem (TSP) is a typical NP complete problem in the field of combinatorial optimization, which is easy to describe but difficult to solve. The number of possible paths and cities increases exponentially, making it very difficult to solve. So far, there is still no perfect solution to address this problem. With the rapid development of computer hardware technology, exploring TSP solutions has become available. TSP is essentially the search for a minimum-weight Hamiltonian cycle in a weighted complete undirected graph. Therefore, the problem of Hamilton cycles in graph theory has great significance in academic research and practical applications. Suppose that we have a graph $G = (V, E)$, where V is the vertex set and E is the edge set. A path p meets the conditions that it contains all the vertices of G, and the edges on p are a subset of E. Starting from one of the vertices of G, traversing with path p, and finally back to itself. If a path like p exists, this path is a Hamilton cycle of G. It is usually necessary to find a minimum Hamilton cycle (a Hamilton cycle with a minimum sum of weights) in a graph. A given graph with N vertices has $(N\text{-}1)!/2$ different Hamilton cycles and there is no simple and efficient algorithm to acquire a Hamilton cycle [1]. In 1992, Dorigo [2] proposed a famous algorithm called ant colony optimization (ACO), and his experiment [3] showed that the length of the output result was close to the shortest path using ACO. By converting the TSP into an iterative probabilistic optimization problem, a likely shortest path will come out after hundreds or thousands of iterations.

The algorithm flowchart of the ACO is shown in Figure 1.

The main idea of the ACO is to transform a minimum Hamilton cycle problem into a probability problem of iterative optimization. In general, solving TSP with ACO contains five steps:

S1: Randomly placing ants on different initial cities.

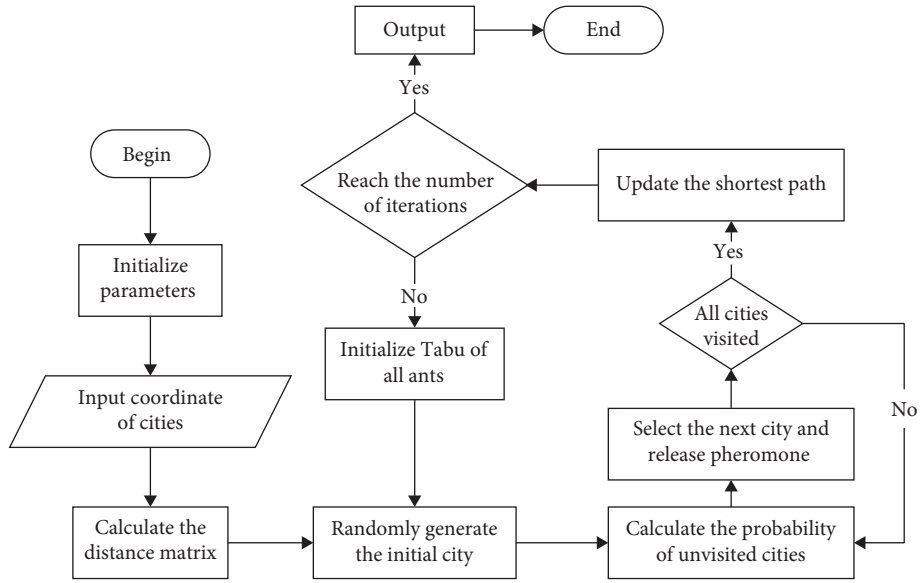

Figure 1: Flowchat of the ACO.

S2: Calculating the probabilities of selecting unvisited cities, as shown in Equation (1).

$$p_{ij}(t) = \left(\tau_{ij}(t)\right)^{\alpha} \times \left(1/d_{ij}\right)^{\beta} / \sum_{j=1}^{n} \left(\tau_{ij}(t)\right)^{\alpha} \times \left(1/d_{ij}\right)^{\beta}. \quad (1)$$

S3: Choosing a city for the next exploration until all the cities are visited.

S4: Releasing pheromone on the chosen path.

S5: Repeating steps S1–S4 until finishing all iterations.

As shown in Equation (1), $p_{ij}(t)$ represents the probability of selecting a path from node i to j at t condition, $\tau_{ij}(t)$ represents the value of the pheromone on a path from node i to j at t condition, α represents the importance degree of the pheromone, d_{ij} is the distance from node i to j, and β represents the importance degree of the distance.

Nowadays, ACO has been applied in various realms such as pricing transmission rights [4], predicting turbine engine vibration [5], and path planning and localization for mobile robot [6, 7]. Solving the TSP is the most representative application of ACO [8]. TSP is a classic problem in which a salesman travels from one city to all cities and returns to the first city. Cities apart from the first city can be visited only once. An effective approach to TSP should be that the average length of the output result is close to the shortest length and the speed is fast meanwhile. The emergence of ACO provides an optional algorithm for solving TSP; however, we found it remains some problems.

First, the quality of results length of the output result is affected by the parameters of ACO greatly. Research [3, 9, 10] tries different values of the parameters according to the datasets showing that the ACO displayed different results on each dataset because of the values of the parameters. To make the average length of the result close to the shortest length on each dataset or to ensure the output result is the best solution as much as possible, it is necessary to set parameters at reasonable ranges after massive debugging, which is an expenditure of massive time.

Besides, ACO methods are known to easily get stuck in local optima. During the execution, one path will be selected by ants frequently. For example, if ants constantly walk along with path $p1$, the pheromone on $p1$ will be very high. If there is a shorter path $p2$, ants will not take it because the higher value of the pheromone brings a higher probability of choosing $p1$. After several iterations, ACO will adopt $p1$ as the best solution even though it is not the shortest path.

Another problem is the low execution speed of ACO. Assume that N is the number of cities, M is the number of ants, and I is the number of iterations. For each iteration, operations of all the ants include updating of the pheromone matrix whose size is $N \times N$. The time complexity of solving TSP in ACO is $O(I \times M \times N^2)$, which is close to $O(N^4)$. In terms of speed optimization of ACO, it is crucial to improve the operations in every iteration.

For the local optima and the high complexity problems that exist in the ACO, improvements are mainly divided into the idea of the algorithm and the platform where the ACO runs. The architecture of computers is improving gradually and the need for advanced technologies has increased, and, thus, the requirement for the speed of data processing is becoming higher. People need to acquire data and results of the programs executed within a shorter time [11]. Due to multiple rounds of nested loops in ACO, which leads to a slow convergence speed, the efficiency of solving TSP is not high. Efficient optimization of ACO is a research topic. Jeff et al. [12] proposed a Gbeam-ACO using a greedy searching strategy. Yan and Jia [13] devised an IFC-based ACO for planning paths in a Web3D environment more quickly. Zhang and Ge [14] proposed a hybrid ant colony algorithm for the solution of the vehicle routing problem with soft time

windows (VRPSTW). Li and Qin [15] proposed an improved ant colony algorithm based on spark solving the vehicle routing problem with a time window. These improvements are relevant to the idea of the algorithm. By improving some operations of ACO or combining ACO with other algorithms like genetic algorithms (GA) [16], the improved ACO has a better performance. However, only the CPU is responsible for executing these algorithms, bringing the excessive load to the CPU. Improving the advanced CPU architecture for better performance is a hard nut to crack, and, therefore, a limitation of algorithm improvement remains.

Since the appearance of the graphics processing unit (GPU), its applications have gradually expanded. In the past, GPU played a role in graphic processing. As the scale of data becomes massive, GPU is a critical tool for high-performance computing [17, 18]. Several research works are related to accelerating ACO's runtime using GPU, the hardware foundation of compute unified device architecture (CUDA). Nie et al. [19] proposed a joint inversion algorithm based on a GPU parallel ant colony algorithm and the traditional least-squares inversion method for the fast imaging of 3D resistivity inversion. Their experiments showed that the ACO based on GPU has a more accurate precision and faster runtime in 3D resistivity inversion imaging of the tunnel. To solve the track correlation problem more effectively, Gao et al. [20] gained a better result by using a GPU-based parallel ACO. In solving quadratic assignment problems (QAPs) [21], Tsutsui and Fujimoto [22] proposed a MATA method that showed a promising speedup compared with the CPU computation mode. These achievements have been successful instances of using GPU to accelerate the execution of ACO. Programs executed on GPU are in parallel, which provide a feasible method of improving algorithms by allocating some complex operations to the GPU for fast processing.

In this research, we make improvements on the ACO for solving TSP faster with higher accuracy. Our contributions are as follows:

(1) To accelerate the execution of ACO, we profoundly consider a fully parallel method at the thread level and implement ACO with the highest degree of parallelism that all the iterations are finished using one kernel function. Considering a hardware acceleration method, we implement the ACO on CUDA, in which the ACO runs in parallel. We analyze the strategy of setting block size for high degree parallelism and illustrate the parallel model of the proposed fully parallel-ACO (FP-ACO).

(2) To alleviate the local optima problem, we come up with a novel pheromone compensation mechanism based on the max–min ant system (MMAS), making the value of the pheromone with a minimum of a decimal that can be represented on a 64-bit PC and a maximum that is penalized by previous iterations.

(3) To make the length of the output result close to the shortest length as much as possible, we combine ACO with the roulette wheel selection mechanism to enhance the ability of searching for a better solution.

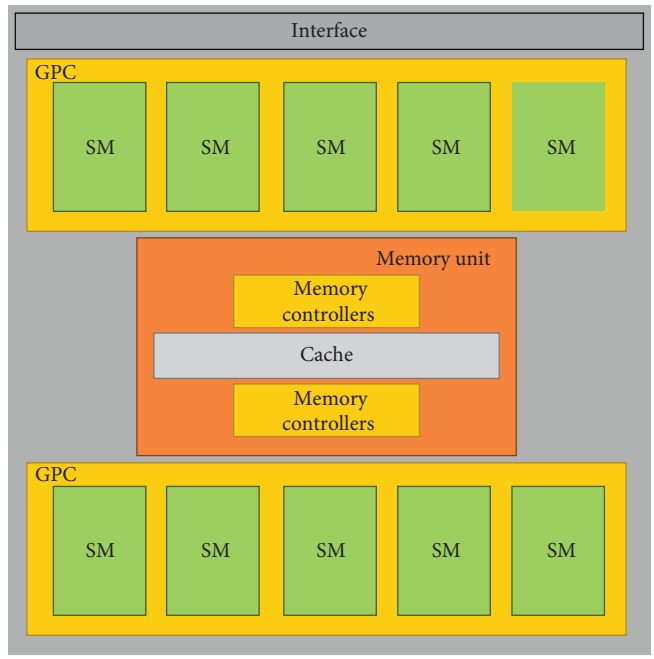

Figure 2: Architecture of a Pascal-based GPU. It contains an interface connected with peripheral component interconnect express (PCI-e), a memory unit with caches and controllers, and several graphics processing clusters (GPCs) with streaming multiprocessors (SMs). GPC is a basic unit for carrying out computational tasks. GPCs work under the control of the thread engine, which dispatches tasks and the SMs are responsible for executing specific operations in parallel.

With the CUDA platform, the proposed pheromone compensation, and the roulette searching mechanism, the improved ACO has the advantage of fast speed and high accuracy. The rest of this article is as follows. In Section 2, we illustrate the improved ACO—fully parallel method of ACO designated as FP-ACO, the organization of threads, the updating mechanism of pheromone based on the MMAS, and the process of the roulette wheel selection mechanism. Section 3 presents the analyses and summary of the experimental results using different datasets. Finally, conclusions are drawn in Section 4.

2. Proposed Methods

2.1. Fully Parallel Model Based on CUDA. CUDA, proposed by NVIDIA in 2006, refers to compute unified device architecture. GPU, which usually performs matrix computations for speeding up data processing, is the core of this architecture. The improved ACO here uses a Pascal-based GPU [23] with a capability of 6 GB. Figures 2 and 3 show architecture diagrams of the GPU and the streaming multiprocessor (SM), respectively.

A kernel function is needed to drive the GPU to work. Execution of a kernel function is on GPU, and Figure 4 describes the execution process.

A magnitude of data input means an instruction requires more SP to process data, which explains why the FP-ACO consumes more hardware resources.

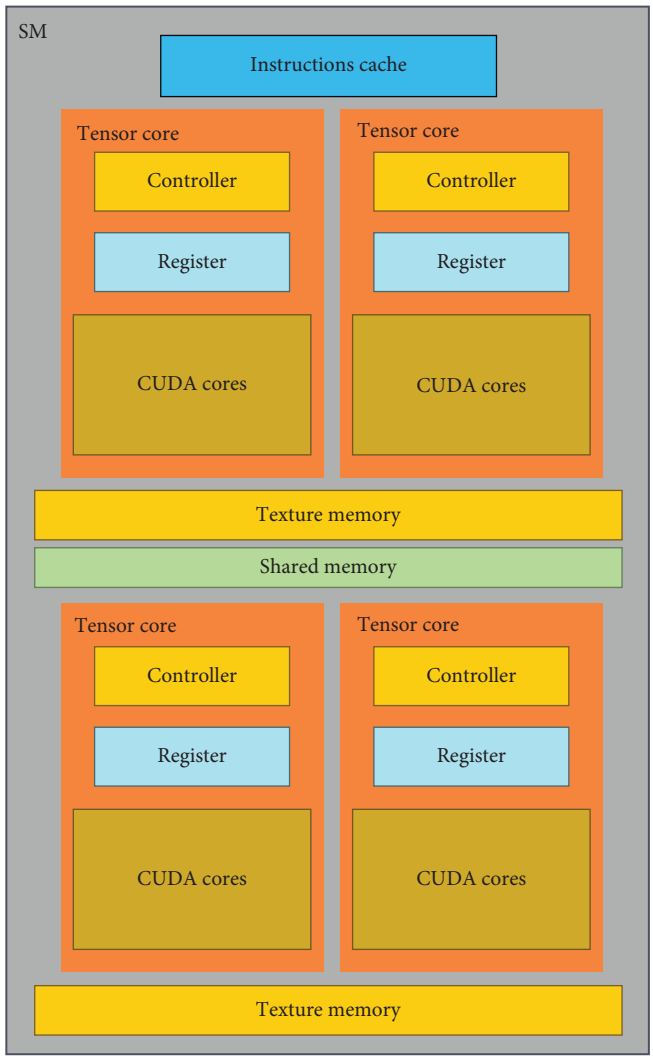

FIGURE 3: Architecture of SM. SM is the unit that conducts specific operations in the form of instructions that constitute a kernel function. It consists of an instructions cache for storing instructions, several tensor cores for dispatching and executing instructions under the controller of each tensor core, the texture memory and the shared memory. Each tensor core has its controller contains the wrap scheduler and the dispatch unit, register, and plenty of CUDA cores.

Research [28–30] shows that determining an appropriate parallel model is vital to the performance of the GPU. As shown in Figure 4, we can infer that when a kernel function runs on GPU, that is a process of the corporation of the grids, the blocks, and the threads. Thus, the primary work is to coordinate the executing units, which aims at suiting our algorithm for high degree parallelism. First, the block size should be considered. According to the architecture of SM, as shown in Figure 3, and the suggestion of setting block size proposed in [31], we set the block size is 128, which enables 128 threads to be concurrent per block. A highest parallel degree is under the situation that all the iterations can be implemented as one kernel function. The finishing of the kernel function means the result is available. Suppose I is the number of iterations and m is the number of ants, theoretically, the number of blocks B_N is determined, as shown in Equation (2):

$$B_N = I \times \mathrm{m}/128. \tag{2}$$

Within each block, the execution of each thread represents the operation of one ant. Figure 5 shows the diagram of threads allocation, and Figure 6 shows design of the parallel model—the kernel function.

Attributed to the concurrency of executing a kernel function, some synchronization mechanisms are necessary to ensure the correctness of data processing. Among the data in ACO, the pheromone matrix maintains itself throughout iterations. If a_i is computing probabilities of selecting other cities while a_{i-1} has not finished releasing pheromone or a_i is utilizing the pheromone matrix after being updated by a_{i+k}, that means a wrong result for a_i (judging from Equation (1)). To address this problem, a CUDA function__syncthreads(), which can synchronize threads within one block to the calling location, is called to guarantee the correctness of calculation and updating procession. The kernel function and the FP-ACO flowchat are shown in Algorithm 1 and Figure 7, respectively.

2.2. Pheromone Compensation. Unlike the conventional as literature [3], Macro Dorigo proposed an Ant System model utilized in Ant Colony Optimization, only which consider the updates of pheromone without considering the threshold of pheromone. While in 2000, Thomas and Holger [32] first proposed that the MMAS adopts an improved mechanism of updating pheromone. In MMAS, only the pheromone on the path whose length is shorter than the last iteration will be updated. The pheromone concentration is between the thresholds that are feedback by the result of previous iterations.

During the experiments, it was found that the results were significantly affected by the range of the pheromone. However, the setting of the range of pheromone should follow the distance matrix according to the input data. This issue is mainly caused by data precision [33]. When using the CUDA C to program on a 64-bit PC, the minimum of a positive double-type variable is 10^{-15}. If the value of a variable is smaller than 10^{-15}, the compiler will set it to 0, which causes an inaccurate calculation. We did the following analysis.

Assuming D is the distance matrix calculated by the coordinate of cities, and d_{ij} is the element of D. To ensure the correctness of the probability matrix P, as shown in Equation (1), Equation (3) is needed:

$$(\tau(t))^{\alpha} \times \left(1/d_{ij}\right)^{\beta} > 10^{-15}. \tag{3}$$

Hence, the minimum of pheromone $\tau_{\min}$ can be expressed, as shown in Equation (4):

$$\tau_{\min} = 10^{-15/\alpha} \times \left(d_{ij}\right)^{\beta/\alpha}. \tag{4}$$

In the matrix P, if p_{ij} (an element of P) reaches $\tau_{\min}$, the ratio of $(\tau(t))^{\alpha}$ and $\left(d_{ij}\right)^{\beta}$ is 10^{-15}, which means the value of pheromone is too low and will cause the value of the matrix P strongly depends on $\left(1/d_{ij}\right)^{\beta}$. To maintain the possibility of a

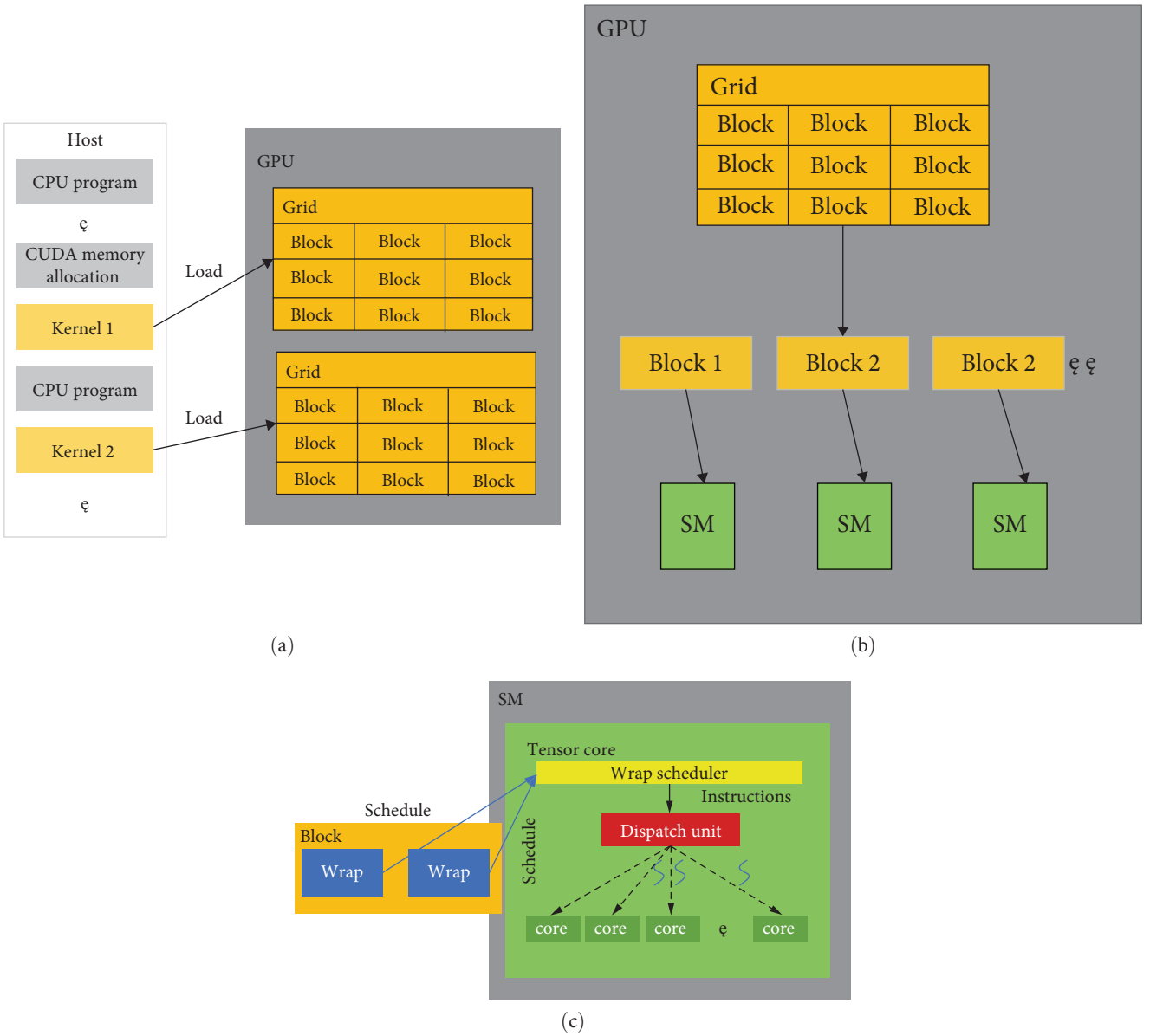

FIGURE 4: Execution process of the kernel function. When executing a CUDA program, the kernel function is loaded on the GPU and perceived as a grid (Figure 4(a)). Before that, the programmer can set the dimension of a grid that consists of several blocks. Then, the controller of GPU will identify and allocate the blocks to SMs. In Figure 4(b), one block runs on one SM until the instructions within this block are finished. The controller of the GPU will determine which SM block runs on. Figure 4(c) shows the scheduling of threads. In SM, the wrap scheduler will schedule threads in one block wraps by wraps (a wrap consists of 32 threads), an architecture called single instruction multiple threads (SIMT) [24], and then send instructions to the dispatch unit. The dispatch unit will dispatch threads to the CUDA cores and execute the instructions. More details about the execution of threads can be referred to in [25–27]. (a) Kernel functions are loaded from host to GPU. (b) A grid is divided into several blocks and allocated to SMs. (c) Controller of SM dispatches threads to each CUDA core for executing.

city that the ants will choose for a shorter result, we made compensation for the pheromone on each path. Supposing K is an appropriate compensation factor, the range of the pheromone can be limited, as shown in Equation (5):

$$10^{-k} \leq \frac{(\tau(t))^{\alpha}}{(d_{ij})^{\beta}} \leq 10^{k}. \tag{5}$$

If $\tau(t)$ reaches $\tau_{\min}$, we set it to $\left((d_{ij})^{\beta}/10^{k}\right)^{1/\alpha}$ according to Equation (5). In the MMAS, $\tau_{\max}$ is dynamically updated by the newfound solution. However, this has not addressed the local optimum problem caused by frequently choosing one path. We made a limitation of the maximum of pheromone using the Gaussian function [34].

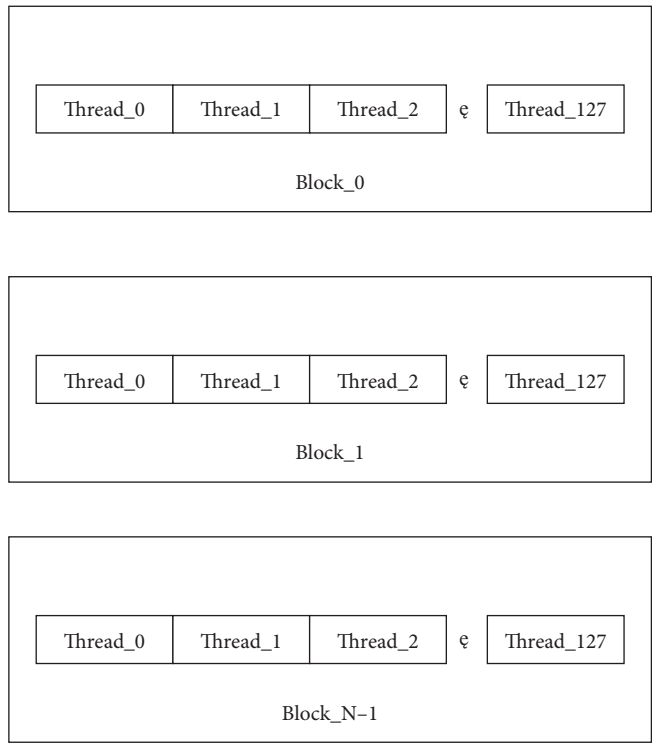

Figure 5: Threads allocation diagram.

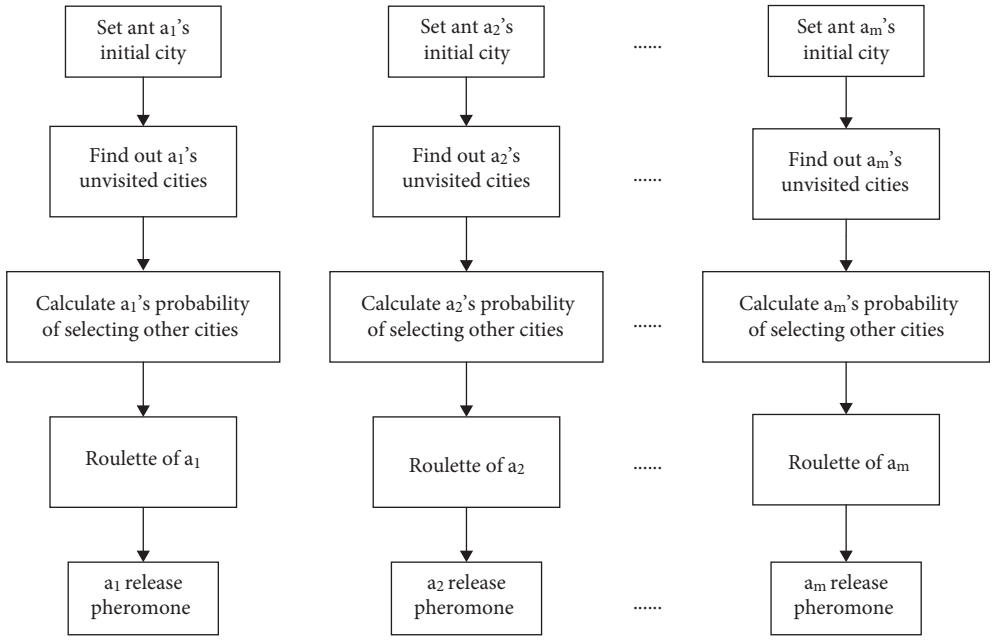

Figure 6: Parallel model of the kernel function.

$$\tau_{\max} = \left(\frac{1}{1-\rho} \times f(s^{\text{opt}})\right) \times \exp(-a), \tag{6}$$

where a is the number of ants that successfully find out a path with a length equal to the current optimum. If a shorter path has been found, we set a to 0 and then continue counting. Algorithm 2 describes the process of the compensation mechanism.

2.3. Roulette Selection Mechanism. Before improvement, the selection process of ACO usually chooses the next city with the maximum probability among unvisited cities, which placed ACO into a dilemma that is missing a better solution.

```
Begin:
  idx ← blockDim.x × blockIdx.x + threadIdx.x
  idy ← blockDim.y × blockIdx.y + threadIdx.y
  thread_id ← (gridDim.x * blockDim.x) × idy + idx
  Ini_City ← random (int) [1-N]
  for i:1 to N
    for j:1 to i
      //update visited cities
    end for
    for j:1 to N
      flag ← 0
      for k:0 to N-i//Find out unvisited cities
        if visited [k] == k
          flag ← 1
        end if
      end for
      if flag == 0
        unvisited [unvisited_index++] = j
      end if
    end for
    __syncthreads();
    for i:1 to unvisited_index
      //Calculating probability of selecting unvisited cities
      and cumulative probability
    End for
    //Roulette
  End for
  L ← 0
  deta_tao [N][N] ← 0
  R[N] ← 0
  for i:1 to N
    R [i-1] ← visited city [i-1]
  end for//Record the nodes on the shortest path
  //calculating length of the shortest path
  L ← distance [[R [N-1]-1] [R [0] -1]]
  for i:1 to N-1
    L ← L + distance [ R[i]-1]] [R[i + 1]-1]
  end for
  __syncthreads();
  //Pheromone update using compensation mechanism
End.
```

ALGORITHM 1: Kernel function <<<B_N, 128>>>.

Cities with low selected probability do not attract the ants even though these cities are a part of a shorter path. The adoption of roulette [35] is to enhance the search capability of ACO instead of considering a city with the maximum probability. A cumulative probability matrix is needed. Supposing V_{sum} is the cumulative probability matrix, which can be expressed, as shown in Equation (7):

$$V_{\text{sum}_i} = \begin{cases} p_1, i = 1 \\ V_{\text{sum}_{i-1}} + p_i, i > 1 \end{cases}, \tag{7}$$

where Pi is the probability of selecting the ith unvisited city. Then, generating a random number R in range [0,1]. The next selected city C is decided, as shown in Equation (8):

$$C = \begin{cases} V_{\text{sum}_i}, V_{\text{sum}_{\min}} \le V_{\text{sum}_{i-1}} \le R \le V_{\text{sum}_i} \\ V_{\text{sum}_{\min}}, R < V_{\text{sum}_{\min}} \end{cases}, \tag{8}$$

where $V_{\text{sum}_{\min}}$ is the minimum of the cumulative probability matrix. Algorithm 3 shows the roulette selection mechanism.

For example, the probabilities of selecting five cities are shown in Table 1.

After computing the cumulative probability of these five cities, the results are summarized in Table 2.

If R is 0.1, ants will select city 1, while R is 0.5, city 3 is the target. From this testing result, city 2 with the maximum selected probability is not a fixed choice. The roulette selection mechanism has a better searching capability that brings ACO more possible cities for consideration.

3. Experiments and Results

We choose the datasets from the TSPLIB [36] with the number of cities ranging from 14 to 99. Each dataset with 100 repetitions under the conditions that the number of ants n was 200, α was 2, β was 5, ρ was set at 0.5, and 2,000 iterations. In each experiment, the average length of the result and the average solution runtime of a dataset were recorded after implementing datasets by using the proposed FP-ACO in this article, the max–min ant system-roulette wheel method-bitmask tabu (MMAS-RWM-BT) [37] and the effective strategies ACO (ESACO) [38]. Three experiments were conducted to reveal the performance of the improved ACO. The first experiment used the datasets Eil51, Eil76, and Rat99 to test the overall performance of the improved ACO. Table 3 shows the environments where all the experiments were carried out, whereas Table 4 shows the result of the first experiment. In the first experiment, compared with the GPU-based MMAS-RWM-BT and the CPU-based ESACO, the improved ACO showed satisfactory performance on datasets Eil51 and Eil76. In these two datasets, the results of the FP-ACO are shown to be the best. Its runtime was much shorter than the ESACO and closed to the MMAS-RWM-BT, especially on the dataset Eil51, it took only one-third of the runtime of the MMAS-RWM-BT. However, as the number of cities increased, the acceleration effect of the FP-ACO was lessening. To focus on its performance on TSP with cities less than 51, the second experiment was conducted using datasets Att48, Dantzig42, Oliver30, Ulysses22, Ulysses16, and Burma14. Table 5 shows and compares the corresponding results.

In terms of the quality of the solution, the FP-ACO always had a shorter length than the other two algorithms. Apart from this, the speed-up ratio was excellent in the case

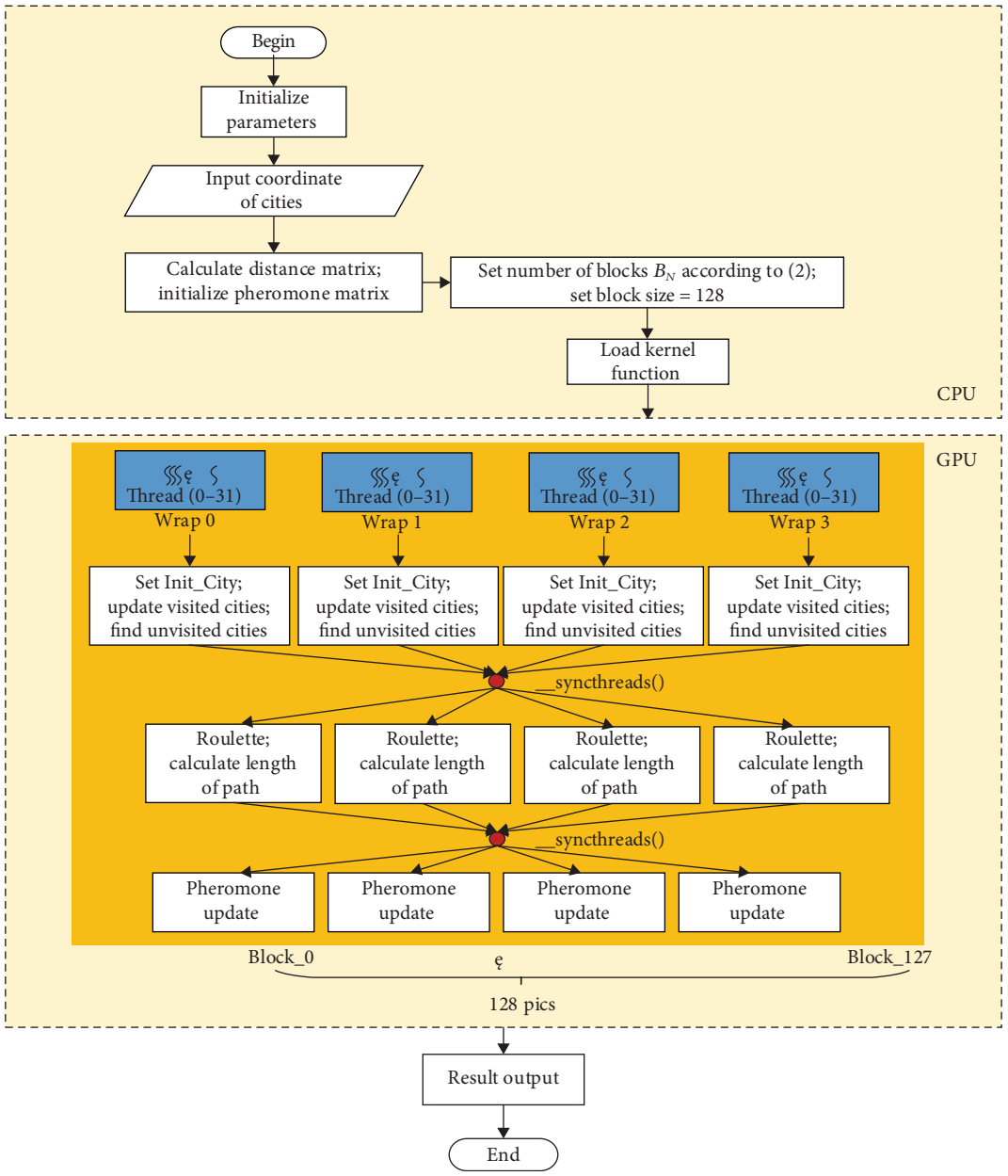

FIGURE 7: Flowchat of the proposed FP-ACO algorithm.

of the number of cities being less than 51. As the number of cities decreases, the growth of the speed-up ratio increases rapidly, as shown in Figure 8. The speed-up ratio is computed as the ratio between the runtime of the two methods. The figure reveals that the speed-up ratio was constantly increasing with decreasing number of cities. Moreover, the smaller the number of cities was, the steeper the curve, showing that compared with the MMAS-RWM-BT (blue solid curve), the acceleration effect of the FP-ACO gradually became outstanding. Despite the slope of the curve lessening when the number of cities decreased from 16 to 14, the speed-up ratio curve (red-dashed curve), with respect to ESACO, increased with an upper bound of 35. Under the situation where the number of cities was less than 42, the proposed FP-ACO can maximize the effect of acceleration, which illustrates that the FP-ACO is more suitable for solving TSP when the number of cities is less than 51.

The third experiment was to explore the performance of the proposed FP-ACO in solving TSP with the number of cities greater than 51. Datasets Brazil58, St70, Gr96, and Lin105 were employed for this experiment. The results are listed and compared, as shown in Table 6.

The result shows that in the datasets Brazil58, St70, Gr96, and Lin105, the runtime of the improved ACO was

```
Begin:
  a ← 0
  released pheromone matrix deta_tao [n][n] ← 0
  //using nodes matrix R in Algorithm 1
  for i:1 to n
    deta_tao [R [i-1]-1] [R [i]-1] ← deta_tao [R [i-1]-1] [R
    [i]-1] + Q
    //Q is the pheromone release coefficient
  end for
  deta_tao [R [n-1]-1] [R [0]-1] ← deta_tao [R [n-1]-1] [R
  [0]-1] + Q
  for i:1 to n
    for j:1 to n
      tao [i-1] [j-1] ← tao [i-1] [j-1] * (1-ρ) + deta tao [i-1] [j-1]
      //pheromone compensation
      if tao [i-1] [j-1] ≤ (10^(-15/α) * (D [i-1] [j-1]) ^(β/α))
        //D is the distance matrix
      then
        tao [i-1] [j-1] ← ((D [i-1] [j-1]) ^β/10^k)^(1/α)
      end if
      //constrain maximum of pheromone
      if tao [i-1] [j-1] ≥ (1/(1-ρ) * L) * exp (-a)
        //L is the length of current shortest path
      then
        tao [i-1] [j-1] ← (1/(1-ρ) * L) * exp (-a)
      end if
    end for
  end for
   //update shortest path
  if L ≤ Best_Path_value
    Best_Path_value ← L
    a ← 0
   else if L = = Best_Path_value
        a ← a + 1
      end if
  end if
End.
```

Algorithm 2: Compensation mechanism.

```
Begin:
  accumulative matrix V_csum [0] ← probability matrix P [0]
  for i:2 to number of unvisited cities
  V_csum [i] ← V_csum [i-1] + P [i]
  end for//calculating accumulative matrix
  flag ← -1
  for i:1 to number of unvisited cities
    temp ← random [0,1]
    if V_csum [i] ≥ temp
      flag ← i
    else
      continue
    end if
  end for
  if flag ! = -1
    selected city ← unvisited city [i-1]//selecting the ith city
  else
    selected city ← unvisited city [number of unvisited cities -1]
  end if
End.
```

Algorithm 3: Roulette selection mechanism.

Table 1: Selected probabilities of given five cities.

City index	Selected probability
1	0.15
2	0.3
3	0.25
4	0.275
5	0.025

the shortest. As the number of cities increased, the runtime was gradually closed to the MMAS-RWM-BT. When the scale of input data was greater than 96, the speed-up ratio between the FP-ACO and the ESACO declined. When the number of cities exceeded 70, the quality of the solution also weakened. There is a problem with the FP-ACO. During the third experiment, the memory footprint of the GPU was only 46% when executing the dataset Lin105, showing that the utilization of GPU memory was not appropriate. Consequently, the next improvement is optimizing the memory allocation and thread scheduling.

In summary, compared with the MMAS-RWM-BT and the ESACO, the proposed FP-ACO can maintain the shortest runtime in solving TSP where the number of cities is less than 105. The runtime of these three algorithms on the datasets is shown in Figure 9, whereas the overall solution results are compared, as shown in Figure 10.

Additionally, the solution results were vulnerable to the setting of the parameters α, β, ρ, Q, and the number of ants n. There is no fixed value but to adjust them according to the input data, which is a time-costly work when using an ACO-based algorithm to solve problems. Overtopping of α will place the algorithm into a local optimum, whereas β leads to a nonconvergence. A feasible way is to set $\beta = 5$ initially and then try different values of other parameters to obtain a better solution result.

Table 2: Result of cumulative probability.

City index	Selected probability	Cumulative probability
1	0.15	0.150
2	0.3	0.450
3	0.25	0.700
4	0.275	0.975
5	0.025	1.000

Table 3: Hardware environments of each experiment.

Operating system	CPU	GPU	RAM capacity
Win10, 64-bit	Intel(R) Core (TM) i5-8500 3.00 GHz	Pascal-based architecture, 6 GB	8 GB

Table 4: Results of the first experiment.

Dataset	Algorithm	Solution result	Runtime (s)
Eil51	MMAS-RWM-BT	435.18	0.489
	ESACO	434.63	1.356
	FP-ACO	**431.55**	**0.174**
Eil76	MMAS-RWM-BT	571.81	**0.526**
	ESACO	570.92	1.512
	FP-ACO	**568.53**	0.527
Rat99	MMAS-RWM-BT	1,349.22	1.383
	ESACO	**1,349.19**	1.734
	FP-ACO	1,483.77	**1.108**

Bold values depict our experiment is better than other algorithms in different dataset.

Table 5: Results of the second experiment.

Dataset	Algorithm	Solution result	Runtime (s)
Att48	MMAS-RWM-BT	35,877.76	0.368
	ESACO	35,811.11	0.858
	FP-ACO	**35,790.08**	**0.149**
Dantzig42	MMAS-RWM-BT	751.84	0.304
	ESACO	698.09	0.796
	FP-ACO	**694.54**	**0.104**
Oliver30	MMAS-RWM-BT	437.91	0.204
	ESACO	425.77	0.494
	FP-ACO	**425.76**	**0.043**
Ulysses22	MMAS-RWM-BT	76.19	0.152
	ESACO	**76.17**	0.387
	FP-ACO	**76.17**	**0.019**
Ulysses16	MMAS-RWM-BT	74.62	0.121
	ESACO	74.52	0.302
	FP-ACO	**74.22**	**0.009**
Burma14	MMAS-RWM-BT	31.18	0.112
	ESACO	31.06	0.235
	FP-ACO	**30.98**	**0.007**

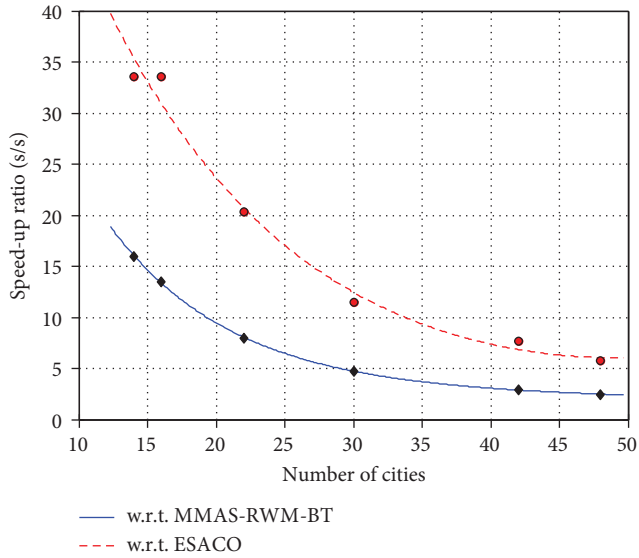

Figure 8: Speed-up ratio versus number of cities.

Table 6: Results of the third experiment.

Dataset	Algorithm	Solution result	Runtime (s)
Brazil58	MMAS-RWM-BT	25,649.397	0.497
	ESACO	25,746.118	1.395
	FP-ACO	**25,438.407**	**0.256**
St70	MMAS-RWM-BT	731.357	0.501
	ESACO	730.319	1.483
	FP-ACO	**730.316**	**0.424**
Gr96	MMAS-RWM-BT	554.821	1.289
	ESACO	552.683	1.659
	FP-ACO	**553.528**	**1.069**
Lin105	MMAS-RWM-BT	15,479.219	1.539
	ESACO	14,812.105	1.801
	FP-ACO	**15,377.992**	**1.376**

Bold values depict our algorithm sometimes is better than other algorithms in different datasets in third experiment.

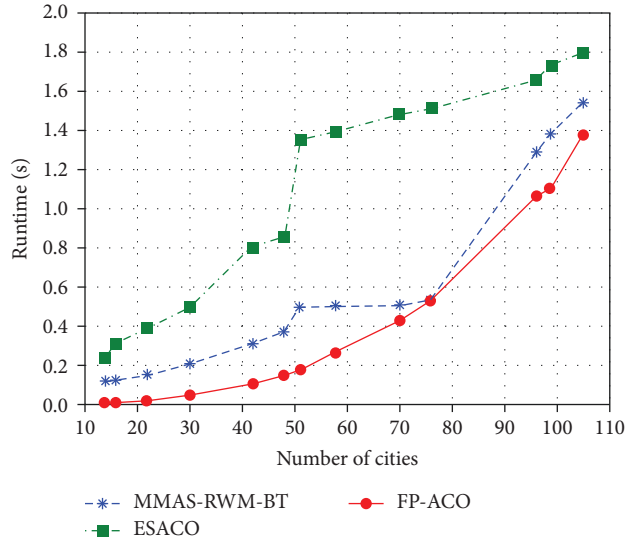

Figure 9: Runtime comparison of MMAS-RWM-BT, ESACO, and proposed FP-ACO.

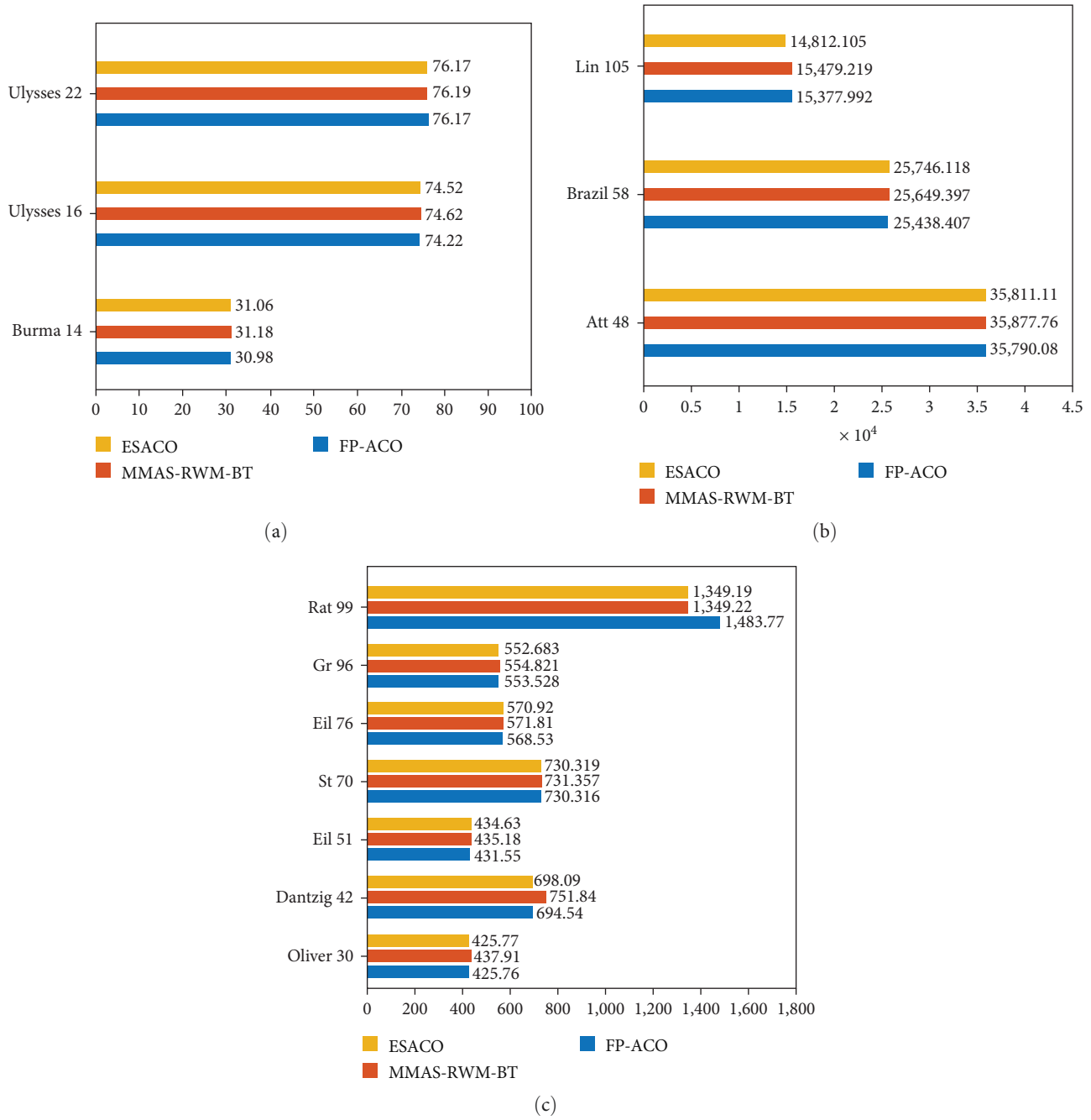

Figure 10: Results comparison of different datasets on MMAS-RWM-BT, ESACO, and proposed FP-ACO. (a) Datasets Burma14, Ulysses16, and Ulysses 22. (b) Datasets Att48, Brazil58, and Lin105. (c) Datasets Oliver30, Dantzig42, Eil51, St70, Eil76, Gr96, and Rat99.

4. Conclusion

This article presents a fast ACO algorithm based on CUDA denoted as FP-ACO that accelerates the runtime in solving TSPs. Combined with the pheromone compensation method and the roulette selection mechanism, the solution quality improved as well. The proposed FP-ACO can update the pheromone according to the precision of input data and the feedback of the current result. Meanwhile, it can make good use of GPU resources. To test the performance of the proposed method, three experiments were conducted utilizing datasets from TSPLIB. In each experiment, datasets were implemented using the FP-ACO, the GPU-based MMAS-RWM-BT, and the CPU-based ESACO. Compared with the MMAS-RWM-BT, the acceleration effect of the FP-ACO constantly exhibited better results with a decreasing number of cities. When using FP-ACO, the entire ACO is loaded on GPU, perceived as several blocks of threads, and dispatched to CUDA cores for executing in parallel while the MMAS-RWM-BT just allocates partial operations on GPU. If the memory of the GPU is large enough, the runtime of the FP-ACO only contains the parallel executing time, as we

know, which is much faster than that of the CPU. Actually, the constraint of memory capacity leads to the failure of finishing ACO in one iteration. Consequently, the unfinished parts of the ACO must wait for other idle CUDA cores to execute, which is a large consume of time. That accounts for why the number of cities increases, the speed-up ratio degrades, and even the runtime becomes larger than the MMAS-RWM-BT. Compared with the ESACO, the acceleration effect was found to be more obvious, where its speed-up ratio reaches 35. In the case that the number of cities is less than 99, the quality of the results attained by FP-ACO is better than the two other methods.

In summary, the results showed that the proposed FP-ACO is suitable for solving TSPs faster in the case that the number of cities is less than 100. Apart from the application in TSP, practical problems similar to TSP can be solved by FP-ACO. For instance, the arrangement of traveling to a tourist site. People can use FP-ACO to acquire the shortest path to the places of their interest and the application in express delivery. An express delivery company can attain the shortest delivery path for saving costs. In the real domains, the constituent plots of a problem are relatively in small quantity, which meets the condition that the FP-ACO can show its best performance.

However, some problems exist. As shown in Equation (5), the setting of k remains a difficulty. The experiments also inspired the next step is to optimize the arrangement of threads and the utilization of GPU memory, so that it can quickly solve large-scale TSPs.

Data Availability

Due to the research focus on using a parallel ACO algorithm upon CUDA to solve TSP, the dataset availability from the previously reported TSPLIB data was used to support this study and is available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/. These prior studies (and datasets) are cited at relevant places within the text as references [3, 9, 10, 36–38].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Conceptualization, Y.X. Cai and H. Lin; methodology, Y.X.Cai and J. Wu; formal analysis, Z.Z., Y.X. Cai, H. Lin, and J.W.; investigation, Z.Z., K.L. Ch, H. Lin, and J. Wu; data curation, Y.X. Cai and H. Lin; writing—original draft preparation, Y.X. Cai and K.L. Ch; writing—review and editing, Z.Z. and K.L. Ch.; supervision, Z.Z.; funding acquisition, Z.Z. All authors have read and agreed to the published version of the manuscript.

Acknowledgments

This research was funded by the key projects of Guangdong Provincial Department of Education of China (no. 2022ZDZX3030), Science and Technology Project of Huizhou City of Guangdong Province of China (no. 2016X0423038), and partly by the project of 2020 Rural Commissioner of Provincial Science and Technology Department of China (2020; no. 409), the National Natural Science Foundation of China (no. 42261024), Beibu Gulf University High-level Talents Research Start-up Fund Project (no. 2021KYQD03), and Marine Science Guangxi First-Class Subject, Beibu Gulf University (no. DTA004).

References

[1] Z. H. Gao and Z. Chen, "Path-operation matrices of graph for solving Hamilton cycles and other path problems," *Journal of Huazhong University of Science and Technology (Natural Science Edition)*, vol. 49, no. 2, pp. 32–36, 2021.

[2] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, pp. 29–41, 1996.

[3] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," in *Proc. ECAL*, pp. 134–142, Paris, France, 1991.

[4] K. S. Sameer, K. T. Ruppa, and T. Parimala, "Pricing transmission rights using ant colony optimization," in *Proc. GECCO '11*, pp. 809-810, Association for Computing Machinery, New York, NY, USA, 2011.

[5] E. AbdElRahman, W. Brandon, E. J. Fatima, H. James, and D. Travis, "Optimizing LSTM RNNs using ACO to predict turbine engine vibration," in *Proc. GECCO '17*, pp. 21-22, Association for Computing Machinery, New York, NY, USA, 2017.

[6] Q. Zhang, "*Path planning and location for mobile robot*," Ph.D. dissertation, Dept. Astro., Harbin Institute of Technology, Harbin, China, 2014.

[7] L. Chen, Y. Su, D. Zhang, Z. Leng, Y. Qi, and K. Jiang, "Research on path planning for mobile robots based on improved ACO," in *2021 36th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pp. 379–383, 2021.

[8] L. H. Tao, Z. N. Ma, P. T. Shi, and R. F. Wang, "Dynamic ant colony genetic algorithm based on TSP," *Machinery Design & Manufacture*, vol. 12, no. 346, pp. 147–149 154, 2019.

[9] Z. Ma, "*Research on the improvement of ant colony algorithm and its application in TSP*," M.S. thesis, Dept. Comp. Eng., Qingdao University of Technology, Qingdao, China, 2016.

[10] Y. J. Xiang, "Research on parameter setting in ant colony algorithm—take TSP as an example," *Modern Information Technology*, vol. 4, no. 22, pp. 95–98+102, 2020.

[11] W. J. Bian, "Research on the development of computer technology under the background of digital information age," *Science & Technology Information*, vol. 19, no. 36, pp. 4–6, 2021.

[12] H. Jeff, O. Suely, E. S. David, and W. Laura, "GBeam-ACO: a greedy and faster variant of Beam-ACO," in *Proc. GECCO '20*, pp. 1434–1440, Association for Computing Machinery, New York, NY, USA, 2020.

[13] F. T. Yan and J. Y. Jia, "IFC-based improved ACO for multi-agent path planning in Web3D mountain environment," in *Proc. ICCAE '17*, pp. 99–103, Association for Computing Machinery, New York, NY, USA, 2017.

[14] Y. Zhang and B. Ge, "VRP with soft time window and its hybrid ant colony algorithm," *Journal of Chifeng University (Natural Science Edition)*, vol. 37, no. 7, pp. 9–12, 2021.

[15] Y. Y. Li and G. Qin, "Solving vehicle routing problem with time window based on spark's improved ant colony algorithm," *Computer Systems & Applications*, vol. 28, no. 7, pp. 9–16, 2019.
[16] F. Dahan, W. Binsaeedan, M. Altaf, M. S. Al-Asaly, and M. M. Hassan, "An efficient hybrid metaheuristic algorithm for QoS-aware cloud service composition problem," *IEEE Access*, vol. 9, pp. 95208–95217, 2021.
[17] B. Y. Zhou, Q. K. Chen, L. P. Gao, and C. Qin, "Image matching algorithm based on CUDA," *Computer Engineering and Applications*, vol. 51, no. 12, pp. 165–170, 2015.
[18] D. Zlotrg, N. Nosović, and A. Huseinović, "Utilizing CUDA architecture for improving application performance," in *2011 19th Telecommunications Forum (TELFOR) Proceedings of Papers*, pp. 1458–1461, 2011.
[19] L.-C. Nie, X.-X. Zhang, B. Liu et al., "A study on resistivity imaging in tuznnel ahead prospecting based on GPU joint inversion," *Chinese Journal of Geophysics*, vol. 60, no. 12, pp. 4916–4927, 2017.
[20] Y. Gao, X. Chen, Y. T. Wang, and M. J. Wu, "Improved ant colony solution algorithm accelerated by GPU in track correlation," *Journal of Northwestern Polytechnical University*, vol. 34, no. 3, pp. 514–519, 2016.
[21] P. Merz and B. Freisleben, "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 337–352, 2000.
[22] S. Tsutsui and N. Fujimoto, "ACO with tabu search on a GPU for solving QAPs using move-cost adjusted thread assignment," in *Proc. GECCO '11*, pp. 1547–1554, Association for Computing Machinery, New York, NY, USA, 2011.
[23] Pascal Architecture Whitepaper, "NVIDIA Corp." 2021, https://www.nvidia.com/de-de/data-center/resources/pascal-architecture-whitepaper/.
[24] H. T. Bai, "*Research on high performance parallel algorithms based on GPU*," Ph.D. dissertation, Dept. Comp. Sci. and Tech., Jilin University, Jilin, China, 2010.
[25] J. A. Robinson, S. V. Vrbsky, X. Hong, and B. P. Eddy, "Analysis of a high-performance TSP solver on the GPU," *ACM Journal of Experimental Algorithmics*, vol. 23, no. 1, pp. 1–22, 2018.
[26] J. Fu, "*Study and realization of parallel ant colony optimization based on GPU*," M.S. thesis, Dept. Wuhan Digital Engineering Institute, China Ship Research and Development Academy, Beijing, China, 2011.
[27] A. Filippou, D. A. Karras, and I. Tsatrafyllis, "On the detailed design issues of a multi-agent architecture based on CUDA threads towards efficiently Simulating WSN Systems," in *2015, 23rd Telecommunications Forum Telfor (TELFOR)*, pp. 349–352, 2015.
[28] S. Larisa, E. Murali, P. H. Lin, and C. H. Liao, "Data placement optimization in GPU memory hierarchy using predictive modeling," in *Proc. MCHPC '18*, pp. 45–49, Association for Computing Machinery, New York, NY, USA, 2018.
[29] M. U. Ashraf, F. Alburaei Eassa, A. Ahmad Albeshri, and A. Algarni, "Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems," *IEEE Access*, vol. 6, pp. 23095–23107, 2018.
[30] Y. Torres, A. Gonzalez-Escribano, and D. R. Llanos, "Llanos, uBench: exposing the impact of CUDA block geometry in terms of performance," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1150–1163, 2013.
[31] "CUDA C++ Best Practices Guide," NVIDIA Corp., 2022, https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html.
[32] S. Thomas and H. H. Holger, "MAX–MIN ant system, future gener," *Computing Systems*, vol. 16, no. 8, pp. 889–914, 2000.
[33] S. Prata, *Data and C, in C Primer Plus*, pp. 60–88, Posts & Telecom Press, Beijing, China, 6th edition, 2016.
[34] Z. Zhang, S. Xia, Y. Cai, C. Yang, and S. Zeng, "A soft-YoloV4 for high-performance head detection and counting," *Mathematics*, vol. 9, no. 23, Article ID 3096, 2021.
[35] Q. Huang, Y. Xu, Y. Chen, H. Zhang, and F. Min, "An adaptive mechanism for recommendation algorithm ensemble," *IEEE Access*, vol. 7, pp. 10331–10342, 2019.
[36] G. Reinelt, "TSPLIB—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.
[37] R. Skinderowicz, "Implementing a GPU-based parallel MAX–MIN ant system," *Future Generation Computer Systems*, vol. 106, pp. 277–295, 2020.
[38] H. Ismkhan, "Effective heuristics for ant colony optimization to handle large-scale problems," *Swarm and Evolutionary Computation*, vol. 32, pp. 140–149, 2017.