An Approximate Algorithm for the Steiner Tree Problem based on Ant Colony Algorithm

JINGWEI JIA and HAIMING LONG

ABSTRACT

Ant colony optimization (ACO) is a well-known heuristic intelligent method which is used to solve combinatorial optimization problem and can obtain high-quality results in a reasonable execution time. Steiner Tree Problem (STP) is a classic NP-complete problem. The computational complexity of the exact algorithm increases exponentially with the increment of the size of the problem. In this paper, we propose an ant colony optimization algorithm with data parallelism to solve the STP. The simulation results show that the algorithm has good performance in result quality.

INTRODUCTION

Steiner tree problem is a classical NP-complete problem [1], which can be described as giving an undirected graph \( G = (V, E, c) \) where \( V \) is the set of nodes in the graph, \( E \) is the edge, \( c \) represents the non-negative weight of the graph, i.e., \( c: E \rightarrow R^+ \). \( A \) is the set of terminal nodes \( T \), where \( T \subseteq V \). The minimum steiner tree problem (MSTP) in graph is to find a subtree that connects all the terminal nodes \( T \) and make the sum of the weights of all edges in the subtree is minimal.

The ant colony algorithm is proposed by Dorigo M et al. in 1992 to solve the traveling salesman problem (TSP). It can effectively solve the various combinatorial optimization problems. Before 2006, Gutjahr W J [2], Yang W [3] et al. have made some research on the convergence of ACO, and explored the validity of the algorithm in theory.

The structure of this paper is as follows. Part II summarizes the application of ant colony optimization algorithm in steiner tree problem in recent years. Part III introduces the concrete realization of the ant colony optimization algorithm on the GPU. Part IV introduces the results and analysis of the algorithm in various tests. Part V sets out the conclusions and future work.

RELATED WORK

In recent years, many researchers have applied the ant colony optimization algorithm to the MSTP. XuHong, WangHua et al. [4] proposed an ant colony optimization algorithm based on forest growth to solve the STP. In this algorithm, each step of the ant only chose one edge from candidate edge set to make the current forest grow further until all of the forest is connected into a tree and contains all the terminal nodes. Prossegger M, Bouchachia A et al. [5] used the divide-and-conquer
strategy to solve the problem by dividing the original problem into several sub-problems. The algorithm combines the spectral clustering and ant colony optimization algorithms. For the larger problem, although the running time of the algorithm is significantly reduced when increases the number of clusters, but the solution will be worse. In the algorithm proposed by Prasad S et al. [6], the number of ants is set to be the same as the number of terminal nodes. Each ant starts from a terminal node and finds the shortest path from the terminal node to the source node, and finally merge into a steiner tree.

Although many researchers have done a lot of research on the study of GPU-based ant colony optimization algorithms, most of the problem models solved by the algorithm are travelling salesman problem (TSP). In this paper, we propose an ant colony optimization algorithm with data parallelism to solve steiner tree problem.

ANT COLONY OPTIMIZATION ALGORITHM

Procedure of Ant Colony Optimization Algorithm

This section will describe how the algorithm to construct a steiner tree. The procedure of algorithm is as follow:

**Step 1: Initialization**

At the beginning of the algorithm, a node of the terminal node $A$ will be randomly selected as the starting node, the set $S$ is the set of nodes that have been visited, and $S$ is initialized to the starting node. The amount of pheromone on each edge of the graph is initialized to $\tau_0$.

**Step 2: State Transition Rule**

We add an unselected node $j$ adjacent to the node in $S$ to the set $S$, where $j$ is a node in the set of candidate node:

$$j = \begin{cases} \arg\max_{i \in S, k \in C} [\tau_{i,k}]^\alpha [\eta_{i,k}]^\beta, & q \leq q_0 \\ \text{otherwise} & \end{cases}$$

(1)

$\tau_{i,k}$ is the size of the pheromone value associated with this edge $(i, k)$. $\eta_{i,k}$ is a heuristic function that shows the feasibility from the current node $i$ to the node $k$.

The parameters $\alpha, \beta$ determine the importance of $\tau_{i,k}$ and $\eta_{i,k}$.

$q$ is a number randomly generated in $[0, 1]$, $q_0$ is a parameter, and $0 \leq q_0 \leq 1$.

$$p(i,j) = \frac{[\tau_{i,j}]^\alpha [\eta_{i,j}]^\beta}{\sum_{k \in \text{cand.set}} [\tau_{i,k}]^\alpha [\eta_{i,k}]^\beta}, \quad \text{where } \eta_{i,k} = \frac{1}{c(i,k) + d(i,z)}$$

(2)

The algorithm add the new node to the set $S$ until all the target nodes are found. We generate the final steiner tree by tracing the path of the target node in turn.
Figure 1. Gunrock's two steps to turn the current frontier into a new frontier.

**Step 3: Pheromone Update Rule**

For the edges on the Steiner tree, the pheromone update to form a positive feedback mechanism for useful information, the formula is as follows:

\[ \tau_{i,j} = (1 - \rho) \cdot \tau_{i,j} + \rho \cdot \Delta \tau_{i,j}, \quad \text{where} \quad \Delta \tau_{i,j} = \frac{1}{Q} \]  

(3)

Where \( \rho \) denotes the pheromone attenuation coefficient for controlling the pheromone, and \( 0 < \rho < 1 \). \( \Delta \tau_{i,j} \) represents the increment of pheromone at edge \((i,j)\) in this cycle. In the global update, \( Q \) is the global optimal solution; in the local update, \( Q \) is the solution of current iteration. The algorithm repeats the process of Step 2 ~ 3 until the convergence condition is met.

**The Detail of Data Parallelism**

The implementation of ant colony algorithm parallelization is mainly in two aspects: (i) parallel selection of candidate node set \( C \) (ii) selection of next node \( j \).

**SELECTION OF CANDIDATE NODE SET**

For the selection of candidate set \( C \), we use a programmable high-performance graphics processing library—Gunrock [7] [8]. When selecting candidate set \( C \), we used Gunrock’s two steps to do this, Advance and Filter.

**Advance** This step generates a new frontier by accessing the current frontier’s adjacency. Frontier can be composed of adjacent edges or nodes, Advance can enter and output any form of frontier. In this article, we use the nodes frontier.

**Filter** The input of this step is the output of the step Advance, which compresses the nodes that have already been accessed by using the efficient segmentation node of the filter. Filtering the Frontier after Filter is the candidate node set \( C \) we need.

**SELECTION OF NEXT NODE**

After the candidate set \( C \) is selected, we will need to select the next node to be accessed by the transition probability formula. The random number \( q \) is generated using the XORWOW pseudo-random number generator in the cuRand library.

If \( q \leq q_0 \), here we use the parallel reduction method to calculate the choice. Assume \( N \) is the size of the candidate set, \( p \) is the number of threads within a thread block, \( p \) threads select their own maximum from \( N \) elements in parallel, and then select the global maximum and maximum index from the local maximum.
1. **PROCEDURE** select_next_node(curr_set)
2. \( N \leftarrow \text{size(cand_set)} \)
3. \( q \leftarrow \text{pseudo-random number generator} \)
4. if \( q \leq q_0 \) then
5. \( \text{next_node} \leftarrow \argmax_{e \in \text{cand_set}} \text{pher}(e) \cdot \text{heur}(e) \)
6. else
7. \( \text{next_node} \leftarrow \text{proportional_selection(cand_set)} \)
8. end if
9. curr_set \( \leftarrow \) next_node

Figure 2. Pseudo-code of the algorithm to choose the next node.

If \( q > q_0 \), we use the formula (2) to select the next node, this method is also called roulette select method. Skinderowicz R [9] based on the characteristics of the thread bundle proposed a reduction algorithm, the maximum size of the calculation is limited to 32. This is because, in the CUDA architecture, the number of threads in a thread bundle is 32. So, here we also use this reduction method, we only select the candidate nodes in the 32 elements of the probability of the calculation.

**EXPERIMENT AND RESULT ANALYSIS**

The proposed algorithm was implemented in C++ through an Intel(R) Core(TM) i3-2100 3.10GHz with 1GB RAM, GTX650 GPU, Ubuntu16.04 operating system. In order to verify the effectiveness of the algorithm, this paper uses the B, D, and E class graph in SteinLib Library as test cases. Koch T, Martin A et al. [10] summarize the known optimal solutions and the best solutions of various topological graphs.

The algorithm in this paper is compared with the ACOF algorithm [4] and the Novel-ACO algorithm [6] on the B class graph. In Table I, Optimal is the current or known optimal solution of the test case. The negative value in Difference indicates that the experimental results of this algorithm is better than other algorithms, Positive values in contrast. The blank part of the table indicates that the algorithm does not test the corresponding graph.

From the table I we can see that in the B class topology, the algorithm proposed in this paper can always get the optimal solution on the result, which is better than the ACOF algorithm and the Novel-ACO algorithm, and the number of nodes in the graph and the number of edges increased when the effect is more obvious. In Table 2, We can see that the proposed algorithm is superior to the KMB algorithm in the results. When the degree in the graph is small, the proposed algorithm is similar to the running time of the KMB algorithm. However, when the degree in the figure is large, we can see that the algorithm is faster than the KMB algorithm.
TABLE I. EXPERIMENTAL RESULT AND COMPARISON ON B.

<table>
<thead>
<tr>
<th>Test B</th>
<th>V</th>
<th>E</th>
<th>Optimal</th>
<th>PACO</th>
<th>ACOF</th>
<th>Novel ACO</th>
<th>Difference ACOF</th>
<th>Difference Novel ACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>b04</td>
<td>50</td>
<td>100</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b05</td>
<td>50</td>
<td>100</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b06</td>
<td>50</td>
<td>100</td>
<td>122</td>
<td>123</td>
<td>122</td>
<td>125</td>
<td>1</td>
<td>-2</td>
</tr>
<tr>
<td>b07</td>
<td>75</td>
<td>94</td>
<td>111</td>
<td>111</td>
<td>111</td>
<td>111</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b11</td>
<td>75</td>
<td>150</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>b12</td>
<td>75</td>
<td>150</td>
<td>174</td>
<td>174</td>
<td>175</td>
<td>179</td>
<td>-1</td>
<td>-5</td>
</tr>
<tr>
<td>b13</td>
<td>100</td>
<td>125</td>
<td>165</td>
<td>165</td>
<td>172</td>
<td>172</td>
<td>-7</td>
<td>-7</td>
</tr>
<tr>
<td>b14</td>
<td>100</td>
<td>125</td>
<td>235</td>
<td>235</td>
<td>242</td>
<td>236</td>
<td>-7</td>
<td>-1</td>
</tr>
<tr>
<td>b16</td>
<td>100</td>
<td>200</td>
<td>127</td>
<td>127</td>
<td>134</td>
<td>132</td>
<td>-7</td>
<td>-5</td>
</tr>
</tbody>
</table>

TABLE II. EXPERIMENTAL RESULT AND COMPARISON ON D, E.

<table>
<thead>
<tr>
<th>Test</th>
<th>V</th>
<th>E</th>
<th>T</th>
<th>Optimal</th>
<th>Result</th>
<th>Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PACO</td>
<td>KMB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PACO</td>
<td>KMB</td>
</tr>
<tr>
<td>d18</td>
<td>1000</td>
<td>25000</td>
<td>167</td>
<td>223</td>
<td>242</td>
<td>252</td>
</tr>
<tr>
<td>d19</td>
<td>1000</td>
<td>25000</td>
<td>250</td>
<td>310</td>
<td>336</td>
<td>350</td>
</tr>
<tr>
<td>d20</td>
<td>1000</td>
<td>25000</td>
<td>500</td>
<td>537</td>
<td>539</td>
<td>542</td>
</tr>
<tr>
<td>e13</td>
<td>2500</td>
<td>12500</td>
<td>417</td>
<td>1280</td>
<td>1344</td>
<td>1376</td>
</tr>
<tr>
<td>e14</td>
<td>2500</td>
<td>12500</td>
<td>625</td>
<td>1732</td>
<td>1790</td>
<td>1834</td>
</tr>
<tr>
<td>e15</td>
<td>2500</td>
<td>12500</td>
<td>1250</td>
<td>2784</td>
<td>2843</td>
<td>2859</td>
</tr>
<tr>
<td>e18</td>
<td>2500</td>
<td>62500</td>
<td>417</td>
<td>564</td>
<td>582</td>
<td>658</td>
</tr>
<tr>
<td>e19</td>
<td>2500</td>
<td>62500</td>
<td>625</td>
<td>758</td>
<td>808</td>
<td>817</td>
</tr>
<tr>
<td>e20</td>
<td>2500</td>
<td>62500</td>
<td>1250</td>
<td>1342</td>
<td>1362</td>
<td>1359</td>
</tr>
</tbody>
</table>

CONCLUSION

This paper presents an ant colony optimization algorithm with data parallelism to solve STP. The algorithm uses the Gunrock graph processing library and the correlation function in CUDA to implement the data parallelism in the path search phase. In this paper, we test the topology of B, D, and E in SteinLib Library. The simulation results show that the proposed algorithm is better than the related ACOF, Novel-ACO and KMB algorithm. In future research, we can study the algorithm from multiple target nodes, and further increase the parallelism of the algorithm in order to achieve better performance.

REFERENCES


