A Genetic Approach for Solving Minimum Routing Cost Spanning Tree Problem

Quoc Phan Tan

Abstract—Minimum Routing Cost Spanning Tree (MRCT) is one of spanning tree optimization problems having several applications in network design. In general case, the problem is proved to be NP-hard. The paper uses genetic (GA) approach to solve MRCT problem. Computational experiment results show that GA approach outperforms current approximation algorithms.

Index Terms—Routing Cost Spanning Tree, Genetic Algorithm, Metaheuristic Algorithm, Spanning Tree Optimization Problem.

I. MINIMUM ROUTING-COST SPANNING TREE PROBLEM

In this section, we are going to represent some main terms related to MRCT problem, traditional approaches and their drawbacks.

Given $G = (V,E,w)$ is an undirected connected graph having non-negative edge weights (costs); in which $V$ is the node set, $E$ is the edge set, $w$ is the cost matrix. Suppose $T$ is a spanning tree in $G$, the routing cost of $T$, denoted by $C(T)$, is the total routing costs of all vertex pairs in $T$, in which the routing cost of a vertex pair $(u,v)$ in $T$, denoted by $d_T(u,v)$, is the sum over edge costs on the path connecting vertex $u$ and vertex $v$ in $T$. So, by definitions, we have:

$$C(T) = \sum_{u,v \in V} d_T(u,v)$$

(1)

The problem requirement is to to find the one having minimum routing cost among all possible spanning trees in $G$.

Computing spanning tree routing cost of the one having $n$ nodes in MRCT problems by definition occupies $O(n^3)$ time. However, by the definition of “routing load” below we could compute spanning tree routing cost within linear time.

Given a spanning tree $T$ having edge set $E(T)$. If remove an edge $e$ from $T$, $T$ is then separated into 2-subtrees of $T_1$ and $T_2$ having the node set of $V(T_1)$ and $V(T_2)$ respectively. Routing load of $e$ is defined as follows: $l(T,e) = 2 |V(T_1)| |V(T_2)|$.

The formula (1) is then equivalent to formula (2) as follows:

$$C(T) = \sum_{e \in E(T)} l(T,e)w(e)$$

(2)

The MRCT problem is proved to be of NP-hard class. Edge weights and spanning tree topology are two factors affecting on spanning tree routing cost. The spanning tree topology affects highly on the graphs in which the bias of edge weights is not too high.

Constructing a minimum routing cost spanning tree is equivalent to constructing a spanning tree so that the average length of vertex pairs is at least. The problem plays important role in applications of network system building. Specifically, peer to peer network is an example in which the ability of data transfer and all node priorities are equal (the problems origin and its applications are available in [1][2])

Up to now, there are several approximation methods used for MRCT problem based on approximation algorithms and metaheuristic algorithms. Typical examples of approximation approaches are Wong algorithm [1], Add algorithm [3], Campos algorithm [5]. These approximation algorithms could not find out high quality solutions but get time advantages and could assure solution quality when applied into MRCT problem.

II. GENETIC ALGORITHM

When using genetic algorithm for optimizing problems, there are primary points that need to be considered: The first is to find solution representation or find a suitable data structure to represent individuals (from now, the term ‘individual’ and ‘solution’ are used interchangeably); the second is population initialization; The third is to determine a fitness-function which plays a role as median; The fourth is to define proper genetic operators; The fifth is to determine clearly genetic parameters like population size, usage-probability for specific genetic operators, number of generations/loops [6][8],…

The following is simple GA scheme.

- $t = 0 // t$ is the number of generations.
- Establish the initial population $P(t)$.
- Compute the fitness for each individual belonging to $P(t)$
- while (not meet termination criteria) {
  - $t = t+1$;
  - Perform cross-over $Q(t)$ from $P(t-1)$.
  - Perform mutation $R(t)$ from $P(t-1)$.
  - Compute fitness value for each individual in $P(t)$.
  - Perform selection $P(t)$ from $P(t-1) \cup Q(t) \cup R(t) \cup P(t)$.
}

Termination Condition

GA gets stopped when the algorithm reaches a fixed number of generations.

III. APPLY GENETIC ALGORITHM INTO MRCT PROBLEM

This section is for proposing GA to solve MRCT. We first represent some related issues when applied GA into MRCT.
A. Spanning Tree Encoding

Spanning tree encoding can affect the performance of algorithm. An encoding method is considered good if it meets: Ability of utilizing spanning tree features, having reasonable complexity of encoding and decoding process, reasonable time as well as memory occupation for performing genetic operators, ensuring the inheritance of spanning tree individual [4].

At present, there are some common spanning tree encodings like: binary-based encoding, edge-based encoding, node-based encoding. These encoding methods do not work on spanning tree edge.

Binary encoding: Each spanning tree is encoded into a string of n-1 bits, where i-th bit is 0 or 1 depending on the i-th edge participates into the spanning tree corresponding to the string.

Edge-based encoding: Each spanning tree is encoded into a string of n-1 integers, where each integer is an edge index participating into encoded spanning tree.

Node-based encoding: Each spanning tree is encoded into n-1 integers, where the i-th integer is a head index of the spanning tree travel following breadth first search (or deep first search) and tail indexes are fixed in range [2.. n].

Decoding for above encodings is simple.

Example: Given a graph in Fig. 1

![Fig. 1. Spanning tree encoding example](image)

Edges are defined as follows $e_1=\{v_1, v_2\}$, $e_2=\{v_1, v_3\}$, $e_3=\{v_2, v_3\}$, $e_4=\{v_2, v_4\}$, $e_5=\{v_1, v_4\}$, $e_6=\{v_4, v_5\}$, $e_7=\{v_5, v_6\}$. Select a spanning tree having edge set of $\{e_1, e_2, e_5, e_6, e_8\}$, we then have 3 encoded string including binary encoding \{110011010\}, edge-based encoding \{1,2,5,6,8\}, node-based encoding \{1,1,3,3,4\}.

To the problems affected by edge weights. Edge-based encoding should be the most appropriate method. The proposed algorithm in the section uses edge-based encoding.

B. Establishing Initial Population

Population individuals are distributed so that they could explore at most in search space in order to have population diversification. This is main factor to population-based metaheuristics. If the algorithm does not meet this, final obtained results is poor.

Initial population is created commonly by heuristic or random method or both.

Although population size changes through generations, for simply we fixed it in implementation. Population size is an important parameter in population-based metaheuristics as usual; In general, if population size is too small then the algorithm converges slowly; if population size is over size then the algorithm occupies too much time for processing a generation. Initial population diversification, as well as individual selection, is affected by population size.

Since spanning tree routing cost depends upon edge weights and spanning tree topology. So we used below way to construct initial population:

- Each individual is a shortest path tree started at each vertex of graph (find SPTs by Dijkstra).
- Each individual is a spanning tree constructed by graph travel using breadth first search started at each vertex of graph.
- Each individual is created from edges by Prim-based algorithm: incoming edge selection for inserted into T does not work on edge weights that select randomly an edge incident to selected edge set so that no cycle exists in T.

The first 2 methods could construct spanning tree individuals with the size not exceeding the number of vertices that satisfies edge weight factor and spanning tree topology. To Prim-based algorithm, the algorithm could construct spanning tree without size limitation. This guarantees population diversification but does not ensure solution quality when n is significant.

C. Fitness

The prominent level of individuals in population is one among criteria to determine their fitness towards median.

Apparent, the best individual in current generation could also get stuck in next generations and a trivial individual in current generation could have high potential leading to good result on contrary. However, in general, the prominent individuals in current generation have higher probability than poor individuals could lead to optimal result. Thus, we still treat solution quality as the most basic factor to determine individual fitness. Generally, individual fitness is also the probability that gets individual selected or crossed-over or performed mutation for reproduction in next generations.

In the following, we are representing 2-methods commonly used to determine individual fitness.

1) Fitness-Criterion

Objective-function is a function used to estimate solution quality (used to estimate individual fitness and be treated as estimate function). With each individual in population, we compute objective function value by formula (2).

- Suppose pop-size is population size.
- Each i-th individual is labeled $t_i$. Fitness level $f(t_i)$ of each individual ($i=1..pop-size$). In which $f(t_i)$ is the objective-function, where $i=1..n-1$ and $t_i$ are edges of spanning tree.
- Compute the total fitness-level of population:
  \[
  F = \sum_{i=1}^{pop-size} f(t_i)
  \]

- Compute selection-probability of each individual $t_i$: $p_i = f(t_i)/F$
- Compute position-probability $q_i$ of each individual $t_i$:
  \[
  q_i = \sum_{j=1}^{i} p_j, (i = 1..pop-size)
  \]

2) Fitness-Rank

Calculation of fitness-criterion as above is indeed effective
in the population of individuals having the fitness under relative homology. If in the case that there exists an individual having fitness value over-high or over-low, isolated from the others then the individuals in next-generations will be pulled towards this special individual and lead to local inheritance.

The method of determining fitness-rank is a key to remove this local inheritance issue. This method does not work on objective function value, it only follows individual orders after sorting by objective function values.

D. Genetic Operators
1) Individual selection

There are many principles to select individuals for next generations. Herein, we are only mentioning 2-ways: the first is Roulette wheel selection strategy, the second is individual selection following its fitness-rank.

We are select starting prominent- individuals for new population by turning Roulette Wheel by pop-size times, each time of selecting an individual from current population follows the principle: generate a real number r in range of [0..1], if r < q, then select individual vi; if not, select individual vi, where q < r ≤ q, (2 ≤ i ≤ pop-size). Obviously, there are individuals selected many times; the better individual, the more copies it has.

2) Individual selection by fitness-rank

Population is sorted by fitness-rank corresponding to objective-function value, and next generations constructed from the individuals having the most fitness.

3) Cross-over operator

One of vital factors in GA is cross-over probability pc. At each individual in new population, we generate randomly a number r in range of [0..1], if r < pc, then select that individual for reproduction (if the number of selected individuals for reproduction is an odd, it is able to remove/add 1-individual to make a pair).

4) Cross-over probability

Herein, we select cross-over probability pc=0.25 and so we hope that, in average, there are 25% of individuals participated in cross-over process. Starting from 2-individuals T1 and T2, after a cross-over, we have 2- new individuals T1' and T2; then we replace T1 and T2 with T1' and T2 in the same generation. In other words, we could compare the fitness function value of these 4- individuals and select 2-best individuals for inheritance towards next generations.

To genetic algorithm there are some cases that GA is unable to find the global best solution, since the algorithm converges quickly towards local solutions. Premature convergence is the matter of GA as well as others. If premature-convergence occurs too quickly, then useful population data will be loss.

In the following, we are proposing 1 cross-over operators that are really effective in experiments for MRCT problem.

• Given 2-spanning tree individuals T1 and T2 needing to perform cross-over.

• Suppose X is a sub-set of edges in T1, suppose Y is a sub-set of edges in T2.

• The result of cross-over T1 and T2 procreates 2-child individuals T1' and T2; under the principle as follows: T1' = X ∪ T2' and T2' = Y ∪ T1.

E. Mutation Operator

1) Mutation probability

Mutation affects on bits (corresponding to individual gens or spanning tree edges). Herein, we could assign mutation-probability p = 0.01, so we hope that there are 1% of the total number of bits (the number of edges in population) in average will perform mutation. Each bit has the same mutation probability; thus with each bit in population, we could generate randomly a real number r in range of [0..1]. If r < p, let’s perform mutation on it.

In the following, we are presenting a mutation operator which is quite efficient via experiments (for MRCT problem, we proposal the mutation- probability pm which is in range of [0.01..0.03]).

2) Mutation operator

Constructing a spanning tree T neighboring to T could be done by 2-ways as follows: the first ways is to remove randomly a certain edge – e from T and then find an edge e’ so that T = T – e ∪ e’ is a valid spanning having value better or equivalent to the value of T. The second way is to remove randomly a certain edge – e from T then find an edge e’ so that e’ has common vertex with e and T = T – e ∪ e’ is a valid spanning tree having the value better or equivalent to the value of T. If it is unable to find out the edge e’ corresponding to e, we then find randomly another e to start over replacement process.

IV. Experiments

To GA algorithm, we used cross-over probability pc = 0.3, mutation probability pm = 0.03 and population size pop-size = n in implementations (the individuals of initial population were created by shortest-path trees or generated randomly) and the maximum number of generations for inheritance was 1000.

Since the initial population we selected based on Wong algorithm or generated randomly or in both ways.

This section is going to compare experimental results of proposed algorithms with Wong, ADD, CAMPOS.

A. Experimental System

All proposed algorithms were implemented in C++ under DEV CPP compiler on the computer powered by a 2.26 Ghz processor and 4 GB RAM.

We first conducted experiments on general graphs, took the obtained results into consideration on some special graphs such as homogeneous graphs, graphs with uniform edge distribution and graphs with non-uniform edge distribution.

Experiment data were generated randomly. The graph size we used in experiments has the number of nodes in range [20..200] and the number of edges in range [50..2400]. The routing costs obtained by the algorithms in experimental table are displayed as ½ of the value obtained from the formula (2).

B. General Graph

1) Test-case building

General graphs G = (V,E,w) were generated as follows: we first constructed randomly a spanning tree of n = |V| nodes
and \( n - 1 \) edges then inserted randomly other \( m - (n - 1) \) valid edge; all edge weights of graphs are random integers in range \([1..2500]\).

2) Experimental results

Table I shows experimental results through 15 test and compares the obtained results of the metaheuristic with the results of WONG, ADD, CAMPOS.

<table>
<thead>
<tr>
<th>Test</th>
<th>WONG</th>
<th>ADD</th>
<th>CAMPOS</th>
<th>GENETIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3408</td>
<td>5358</td>
<td>3725</td>
<td>3408</td>
</tr>
<tr>
<td>2</td>
<td>8760</td>
<td>15670</td>
<td>8569</td>
<td>8552</td>
</tr>
<tr>
<td>3</td>
<td>29915</td>
<td>51472</td>
<td>30063</td>
<td>29799</td>
</tr>
<tr>
<td>4</td>
<td>14784</td>
<td>41531</td>
<td>15026</td>
<td>14784</td>
</tr>
<tr>
<td>5</td>
<td>40242</td>
<td>101790</td>
<td>47270</td>
<td>39957</td>
</tr>
<tr>
<td>6</td>
<td>185248</td>
<td>249124</td>
<td>254654</td>
<td>182970</td>
</tr>
<tr>
<td>7</td>
<td>1145919</td>
<td>4350050</td>
<td>1420996</td>
<td>1140406</td>
</tr>
<tr>
<td>8</td>
<td>3178505</td>
<td>11375802</td>
<td>3765028</td>
<td>3150636</td>
</tr>
<tr>
<td>9</td>
<td>3374998</td>
<td>17056890</td>
<td>3865164</td>
<td>3360541</td>
</tr>
<tr>
<td>10</td>
<td>5485453</td>
<td>23448926</td>
<td>6303738</td>
<td>5474075</td>
</tr>
<tr>
<td>11</td>
<td>1384422</td>
<td>7064096</td>
<td>1483611</td>
<td>1373560</td>
</tr>
<tr>
<td>12</td>
<td>2964078</td>
<td>16402988</td>
<td>3611100</td>
<td>2964078</td>
</tr>
<tr>
<td>13</td>
<td>5311194</td>
<td>19142486</td>
<td>5903426</td>
<td>5266341</td>
</tr>
<tr>
<td>14</td>
<td>6605587</td>
<td>27951407</td>
<td>8284471</td>
<td>6584499</td>
</tr>
</tbody>
</table>

GA algorithm returns better results over the result of Add, Campos in entire 15 tests; to Wong, GA does better in 13 tests and 2 tests with equivalent results.

C. Homogeneous Graph

1) Test-case building

Homogeneous graphs were generated as follows: we first chose a random value as a homogeneous value for edges, suppose \( \Delta \in [1..2500] \). Then constructed a random spanning tree of \( n = |V| \) nodes and \( n - 1 \) edges, and finally inserted randomly other \( m - (n - 1) \) valid edges. All edges in G were attached with a positive value \( \Delta \pm \mu \) in which \( \mu \) is a small integer.

2) Experimental results

Table II shows the experimental results through 15 tests under homogeneous graphs to experience the effectiveness.

<table>
<thead>
<tr>
<th>Test</th>
<th>WONG</th>
<th>ADD</th>
<th>CAMPOS</th>
<th>GENETIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>72818</td>
<td>70494</td>
<td>106012</td>
<td>72042</td>
</tr>
<tr>
<td>17</td>
<td>98150</td>
<td>102307</td>
<td>131853</td>
<td>97523</td>
</tr>
<tr>
<td>18</td>
<td>144751</td>
<td>148596</td>
<td>189264</td>
<td>142652</td>
</tr>
<tr>
<td>19</td>
<td>342910</td>
<td>341099</td>
<td>433157</td>
<td>336357</td>
</tr>
<tr>
<td>20</td>
<td>474658</td>
<td>566614</td>
<td>509018</td>
<td>465002</td>
</tr>
<tr>
<td>21</td>
<td>560523</td>
<td>562463</td>
<td>690849</td>
<td>551476</td>
</tr>
<tr>
<td>22</td>
<td>520997</td>
<td>549681</td>
<td>636465</td>
<td>518830</td>
</tr>
<tr>
<td>23</td>
<td>117392</td>
<td>117676</td>
<td>158210</td>
<td>116872</td>
</tr>
<tr>
<td>24</td>
<td>329528</td>
<td>340386</td>
<td>405366</td>
<td>328528</td>
</tr>
<tr>
<td>25</td>
<td>6461300</td>
<td>7127108</td>
<td>7133250</td>
<td>6372634</td>
</tr>
<tr>
<td>26</td>
<td>3183150</td>
<td>3369136</td>
<td>3813918</td>
<td>3173676</td>
</tr>
<tr>
<td>27</td>
<td>1345428</td>
<td>1393342</td>
<td>1774572</td>
<td>1327592</td>
</tr>
<tr>
<td>28</td>
<td>2242772</td>
<td>2414476</td>
<td>2699136</td>
<td>2230890</td>
</tr>
<tr>
<td>29</td>
<td>879674</td>
<td>885659</td>
<td>1268719</td>
<td>873849</td>
</tr>
<tr>
<td>30</td>
<td>2952464</td>
<td>2929879</td>
<td>3782156</td>
<td>2928933</td>
</tr>
</tbody>
</table>

GA outperforms Add, Campos in entire 15 tests; to Wong, GA does better in 12 tests and gets equivalent results in 3 tests.

D. Edge Distribution Factor

1) Test-case building

Graphs with uniform edge distribution are the graphs in which node degrees are equivalent or insignificant difference. Graphs with uniform edge distribution we used were generated as follows: we first determined a parameter \( r = 2 \times \lfloor m/n \rfloor + 1 \) called as the average number of edges of a node then constructed a random spanning tree of \( n = |V| \) and \( n - 1 \) edges so that all node degrees did not exceed \( r \), we next inserted randomly other \( m - (n - 1) \) valid edges and assured that every node degree did not exceed \( r \), edge weights in the graph were generated randomly in range \([1..2500]\).

2) Experimental results

Table III shows experimental results through 15 tests under graphs with uniform edge distribution.

<table>
<thead>
<tr>
<th>Test</th>
<th>WONG</th>
<th>ADD</th>
<th>CAMPOS</th>
<th>GENETIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>166232</td>
<td>441740</td>
<td>174190</td>
<td>165052</td>
</tr>
<tr>
<td>32</td>
<td>242436</td>
<td>1110048</td>
<td>262110</td>
<td>241096</td>
</tr>
<tr>
<td>33</td>
<td>552034</td>
<td>2330760</td>
<td>658246</td>
<td>551083</td>
</tr>
<tr>
<td>34</td>
<td>754534</td>
<td>5906702</td>
<td>852214</td>
<td>751180</td>
</tr>
<tr>
<td>35</td>
<td>1591536</td>
<td>11757760</td>
<td>2015229</td>
<td>1586588</td>
</tr>
<tr>
<td>36</td>
<td>1931829</td>
<td>14010963</td>
<td>2036603</td>
<td>1927643</td>
</tr>
<tr>
<td>37</td>
<td>193600</td>
<td>1386026</td>
<td>242028</td>
<td>193492</td>
</tr>
<tr>
<td>38</td>
<td>7111108</td>
<td>36043876</td>
<td>7705378</td>
<td>7100349</td>
</tr>
<tr>
<td>39</td>
<td>2308062</td>
<td>10356986</td>
<td>2735218</td>
<td>2302596</td>
</tr>
<tr>
<td>40</td>
<td>5640618</td>
<td>19951434</td>
<td>7332050</td>
<td>5640618</td>
</tr>
<tr>
<td>41</td>
<td>935328</td>
<td>3294241</td>
<td>1271022</td>
<td>933538</td>
</tr>
<tr>
<td>42</td>
<td>2259932</td>
<td>8717570</td>
<td>2813048</td>
<td>2249706</td>
</tr>
<tr>
<td>43</td>
<td>879674</td>
<td>885659</td>
<td>1268719</td>
<td>873849</td>
</tr>
<tr>
<td>44</td>
<td>2952464</td>
<td>2929879</td>
<td>3782156</td>
<td>2928933</td>
</tr>
</tbody>
</table>

GA outperforms Add, Campos in entire 15 tests.

E. Non-Uniform Edge Distribution

1) Test-case building

Graphs with non-uniform edge distribution \( G = (V,E,w) \) were generated as follows: we first selected random \( k \) nodes \((\lfloor n/2 \rfloor + 1 \leq k \leq n - 1)\) then assigned each node an integer \( r \in \{1, 2\} \) indicates that node degrees can not bigger than \( r \); we next constructed a spanning tree and its edges as we did in general graphs. Note that in the case of not being able to construct enough \( m \) edges, we would repeat the process (this kind of graph is named as asymmetric graph).

2) Experimental results

Table IV shows experimental results through 15 tests under graphs having non-uniform edge distribution.
GA outperforms Add, Campos in entire 15 tests; to Wong, GA does better in 12 tests and gets equivalent results in 3 tests.

### TABLE V: SUMMARIZED TABLE OF TESTS (60 TESTS)

<table>
<thead>
<tr>
<th>GA</th>
<th>WONG</th>
<th>ADD</th>
<th>CAMPOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better</td>
<td>Quantity</td>
<td>%</td>
<td>Quantity</td>
</tr>
<tr>
<td>Worse</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### V. CONCLUSIONS AND FUTURE WORKS

This paper has proposed genetic algorithm for solving MRCT. The algorithm was implemented and experimented in a bundle of tests. Experimental results show that proposed algorithm could get stable performance and return high quality solutions over against some current approximation algorithms. However, improving performance of cost and time should be more taken into consideration. That will certainly be included in our next studies.

#### REFERENCES


