Green Routing: An investigation of bicriterion shortest path problems

SHI Chao, Zhang Chenzi, and CHAU Sze Yiu, *Member, IEEE*

***Abstract*— Among all algorithms, Dijkstra's algorithm is the most widely used one to solve routing problems. In link-state routing algorithms, Dijkstra's algorithm attempts to find out a path from the source to the destination with the shortest latency or the least number of hops. However, the advantage of the short latency or fewer hops might be compromised by the large amount of power consumption. Since the world nowadays is focusing more on energy saving, the paper attempts to find out a path with the smallest latency while at the same time not exceeding the energy cap. Two categories of algorithms will be discussed, namely Martin's algorithm and genetic algorithm. The advantages and the disadvantages of the two algorithms will also be included. In the last part, a new method of generating random graph is proposed. Graphs generated by using this method are extremely hard to compute, and we use these graphs to simulate the worst case in the real world situation.**

***Index Terms*— Shortest path problem, Dijkstra's algorithm, Martin's algorithm, Genetic algorithm, Random graph generation, Algorithm analysis**

I. Introduction

Dijkstra's algorithm is widely used to find the shortest path in network routing problems. However, in most cases, it takes into consideration only one dimension of cost. For example, latency is one of the most important factors to consider in a network problem. The total latency for the data communication from the source to the destination is calculated by adding up the latency segment by segment. This method can guarantee that the path with the shortest latency is found but other obscure conditions of the network may be neglected. Much of previous research work of bicriterion optimization deals such kind of problems.

To make the case easier, the cost number is restricted to two in our study. The first cost is defined as latency or the number of hops. The second cost is power consumption. The problem with the Dijkstra's algorithm is that possibly we end up with a routing path with little latency but unaffordable power consumption, because the high speed of the network usually needs the support of high power. Since the world nowadays is emphasizing more on energy saving, it is our goal to limit our power consumption below a certain level. Therefore, we are going to find out a route with the lowest latency and satisfying the energy cap at the same time. This kind of route is called "green path" or "green route" and the process of discovery is called "Green Routing".

II. An Example for Green Routing

Fig. 1 and Table 1 show the difference between a traditional shortest path and a green path. The nodes are labeled with number {1, 2…6} and the cost of each node link is represented as an ordered pair (x, y), in which x is the latency and y is the power consumption. Unit of x is second and unit of y is kWh. All the links are full-duplex. Node labeled with 1 is the source and node labeled 6 is the destination.

The shortest path when considering only latency and found by Dijkstra's algorithm is Path 1 which is 1→2→5→4→6. But path 1 has too much power consumption. The latency of path 2 is only one second longer than that of path 1, but the power consumption of path 2 is more than three times that of path 1, which makes path 1 appears to be not so optimal.

If we add energy consumption cap which is 20kWh, than path 2 is the optimal path then. If the maximum power consumption cap is even lower, say 10 kWh, and the optimal is path 5.

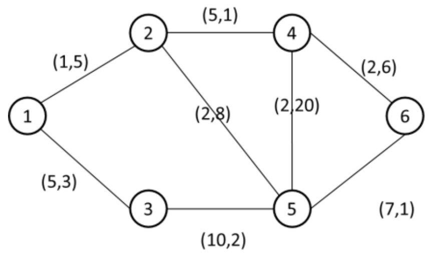

Fig. 1. Network Example.

TABLE I
UNITS FOR MAGNETIC PROPERTIES

Path 1	1→2→5→4→6	(7,39)
Path 2	1→2→4→6	(8,12)
Path 3	1→2→5→6	(10,14)
Path 4	1→2→4→5→6	(15,27)
Path 5	1→3→5→6	(22,6)
Path 6	1→3→5→4→6	(19,31)
Path 7	1→3→5→2→4→6	(24,20)

Manuscript received April 10, 2012. This work was supervised by Dr Henry Chan.

SHI Chao, student of Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR. (phone number: (+852) 92094248; e-mail: chris19891128@gmail.com).

Zhang Chenzi, student of Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR. (phone number: (+852) 51358275; e-mail: china.zhangchenzi@gmail.com).

CHAU Sze Yiu, student of Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR. (phone number: (+852) 62019472; e-mail: 09140841d@polyu.edu.hk).

III. Define the problem in math

The real world network can be abstracted to be a graph. Let $G(N,A)$ denotes a directed network which is composed of a finite set $N=\{1,2....n\}$ of nodes and a finite set $A \subset N \times N$ of m arcs which represents the set of directed edges. Each arc can be denoted as an order pair of (i, j).

One can assign a weight to an edge. In a generalized case, an edge can contain r dimensions. Let $c_{i,j}^{k}$ where $(i,j) \in A$ and $1 \leq k \leq r$ represents the kth cost of arc (i, j). If the dimension of costs is 2, then the problem can be named as bicriterion optimization problem. In order to make the optimization problem harder to compute and more similar to the real world case, the two costs are inversely related.

There are two nodes in the graph s and t where $s \in N$ and $t \in N$. The purpose of the algorithm is to find out a path p of length l which is denoted as a sequence of adjacent paths $\{(s,p_1),(p_1,p_2)...(p_{l-1},t)\}$. The cost of the path $c(p)$ is an ordered pair represented as:

$$(c^1(p), c^2(p))(1)$$
$$c^1(p) = \sum_{(i,j)\in p} c^1{}_{i,j} \tag{1}$$
$$c^2(p) = \sum_{(i,j)\in p} c^2{}_{i,j}$$

We are interested in minimizing the first coordinate of the path cost while at the time satisfying the condition that the second coordinate is no larger than a certain number *MAX*. The solution of the problem is to find a path p_{green} while there does not exist a path p_{other} such that

$$c^1(p_{other}) < c^1(p_{green}) \text{ and } c^2(p_{green}) \leq MAX \tag{2}$$

In the previous work by Martins (1984) and B. Smith (1989), a path p which can satisfy the optimization of the entire criterion ($k \in \{1....,r\}$) acts as the ultimate goal of the algorithm. However, unfortunately it is impossible to find such a solution which can simultaneously optimize all the costs in most real cases. A path p is *non-dominated* if there does not exist another path p_{other} satisfying the condition that $c^k(p_{other}) < c^k(p_{green})$ for all $k \in \{1....,r\}$ and there is at least one k such that $c^k(p_{other}) \neq c^k(p_{green})$. In the case of cost dimension equals to 2, the cost of all the non-dominated solution paths can be depicted in the Fig. 2. The x-axis represents the first cost while the y-axis represents the second cost.

IV. Proof of NP-Complete

In a very similar way as [7], we can prove this problem is a NP-hard problem by showing that 0-1 knapsack problem is polynomial-time reducible to routing problem.

The proof is as follows: 0-1 knapsack problem is defined as: there are n items, each of them has a weight $w[i]$ and a value $v[i]$, the goal of this problem is to select some of these items so that the total value is maximized while the total weight is less than a certain value k.

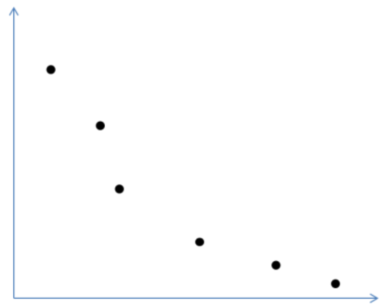
Fig. 2. Distribution of non-dominated paths.

We already know that general 0-1 knapsack problem is a NP-complete problem, the only thing we need to do is to show that every case for 0-1 knapsack can be transformed to a routing problem and the output of the routing problem is the output of 0-1 knapsack problem. For each 0-1 knapsack problem, we can build a graph like Fig. 3. This graph starts with a node *S* and ends with a node *T*.

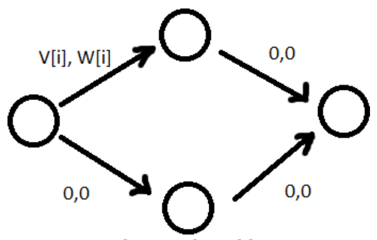

Fig. 3. Graph representing 0-1 knapsack problem

For each 0-1 knapsack problem, we can build a graph in Fig.3 correspond to it. This graph starts with a node S and a node T.

Then for each item in the 0-1 knapsack problem:

(1) $v[i]$ (routing problem) = -1 * $v[i]$ (0-1 knapsack problem);

(2) $w[i]$ (routing problem) = $w[i]$ (0-1 knapsack problem);

(3) The edges are directed, one can only go from left to right.

The path taking upper way is equivalent to the solution that 0-1 knapsack chooses this item; the path taking lower way is equivalent to the solution that does not choose this item. Literally, the graph in Fig 3 is equivalent to a link between two network nodes. By substituting each link in the network topology with the graph above, we can well convert the routing problem into a network of 0-1 knapsack problems.

The solution of this routing problem is a solution which aims to minimize sum($v[i]$) and sum($w[i]$) < k. We can find that is the solution of 0-1 knapsack problem, which is the maximum value of - sum($v[i]$) with constraint that sum of weight must less than k. The selected items are these with path go through upper part of the sub-graph.

It is shown any 0-1 knapsack problem input can be transformed in polynomial time to a routing problem. If the routing problem is not NP-hard, that is the entire routing problem cases can be solved in polynomial time, then 0-1 knapsack also belongs to P (polynomial) which is not true. So we have proved routing problem is NP-hard.

V. Martin’s algorithm

Martin’s algorithm is a label setting algorithm which aims to solve the bicriterion optimization problem. Martin’s algorithm can help to find all the non-dominated paths from the source to the destination. In martin’s algorithm, the structure of the label is as follows:

(*cost 1, cost 2, owner ,parent, parent index*)

The first two attributes represents the cost of one path from source to the owner of the label. The owner of the label is the node where the label is set. The attribute parent is the node from which the label is generated. Since each node may have several labels, we use parent index to differentiate them.

Labels are categorized into two classes, temporary and permanent. A temporary label may represent a dominated path and be supplanted by other labels. A permanent label represents a non-dominated path from the source to the owner of the node. The attribute parent index is the index of the permanent label for a node.

The following part shows a revised Martin’s algorithm

```
Step 0:
    Assign the temporary label [0 ,0 ,s ,null ,null ] to node s.
Step 1
    (1) If the set of temporary labels is empty go to step 4.
    (2) Otherwise, select the lexicographically smallest label.
    (3) If the label’s owner is t, go to step 4
    (4) Otherwise, set this label as a permanent one.
Step 2
    While some node $j \in N$ exists, such that $(i, j) \in A$
    (a) Let $(c^1(p_{s,j})+c^1_{i,j}\ ,\ c^2(p_{s,j})+c^2_{i,j}\ ,\ j\ ,\ i\ ,\ l)$ be the new temporary label for node j.
    (b) If $c^2(p_{s,j})+c^2_{i,j} > MAX$ , go to Step 3 directly
    (c) Among all the temporary labels of node j, delete all labels representing a dominated path from s to j.
Step 3
    Return to step 1.
Step 4
    Find the label with the smallest first cost belonging to t and use backtracking to find the path from s to t.
Step 5
    Stop the execution of the algorithm
```

An example explaining how the algorithm works is shown in Fig. 4. Node 1 is the source and Node 4 is the destination. The cap for the second cost is 10. Table 2 shows the flow of the algorithm. Each cell is divided into two parts. The upper part is permanent labels and the lower part is temporary labels in the lexicographically order.

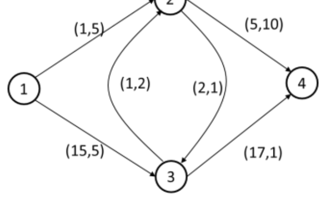

Fig. 4. Martin’s example

TABLE II
Martin’s Labels in each iteration

	1	2	3	4
1				
	(0,0,1,-1,-1)			
2	(0,0,1,-1,-1)			
		(1,5,2,1,0)	(15,5,3,1,0)	
3	(0,0,1,-1,-1)	(1,5,2,1,0)		
			(3,6,3,2,1) (15,5,3,1,0)	(6,15,4,2,1) Deleted
4	(0,0,1,-1,-1)	(1,5,2,1,0)	(3,6,3,2,1)	
		(4,8,2,3,1) Deleted	(15,5,3,1,0)	(20,7,4,3,1)
5	(0,0,1,-1,-1)	(1,5, 2,1,0)	(3,6, 3,2,1) (15,5,3,1,0)	
		(16,7,2,3,2) Deleted		(20,7,4,3,1) (32,6,4,3,2)
6	(0,0,1,-1,-1)	(1,5, 2,1,0)	(3,6, 3,2,1) (15,5,3,1,0)	(20,7,4,3,1)
				(32,6,4,3,2)

VI. Complexity analysis

Before proceeding to discuss the complexity of Martin’s algorithm, a new factor which has a great influence on the computational complexity is going to be introduced. AL represents the average number of labels for each node. The value of AL is highly dependent on the mechanism of the random graph generator. AL is defined as:

$$AL = \frac{TL}{n} \qquad (3)$$

TL is the total number of permanent labels of all nodes generated before the computation ends and n is the number of nodes. For simplicity, we can assume that during the computation process, the number of the temporary and permanent labels can maintain a fairly steady level of $O(AL)$ for each node.

In step 1, there are $O(n)$ labels in the priority queue, the cost of retrieving the smallest element and reconstruct the heap will be $O(\log n)$. The cost of inserting into the heap is also $O(\log n)$.There are altogether $O(n{\cdot}AL)$ temporary labels and all of them will ultimately become permanent and there should be $O(n{\cdot}AL)$ times of polling from as well as inserting into the priority queue. All the operations of step 1(1) all takes $O(n{\cdot}AL{\cdot}\log n)$.

Step 2(a) and Step 2(b) take constant time and we can have two different mechanisms of storing the set of temporary labels as well as the permanent labels. One is to store them in a lexicographical order in an array. Another is to store them in a binary tree with the left leaf with smaller lexicographical value than the root and the right leaf with larger lexicographical value. The complexity of managing the list of temporary labels mainly comes from the effort of finding the dominated labels and deleting them from set. The average case and worst case complexity for the two operations are summarized below in table 3 & table 4. If we implement the binary search tree storage, the total cost of computation after combining all the factors will be $O(n\cdot AL\cdot \log n + m\cdot AL\cdot \log(AL))$.

TABLE III
AVERAGE CASE COMPARISON

	Search for *ix(nx)* and *iy(ny)*	Insert new path and delete the dominated path
Array	log(*AL*)	*AL*
BST	log(*AL*)	log(*AL*)

TABLE IV
WORST CASE COMPARISON

	Search for *ix(nx)* and *iy(ny)*	Insert new path and delete the dominated path
Array	log(*AL*)	*AL*
BST	*AL*	log(*AL*)

VII. GENETIC ALGORITHM

This problem has been shown to be NP-hard, so we do not intend to find a solution that can solve this problem in polynomial time. Firstly, there is a group of “travelers” that contain some genes (randomly generated) to guide go through the whole graph. The first group of travelers is called the first generation. Each traveler starts the trip based on the genes. Some of them can get to the destination point and some cannot. For those travelers who can reach the destination, each has two scores, namely accumulated distance (C1) and accumulated energy consumption (C2).

The next generation is generated from the travelers having reached the terminate point based on the scored they get. As for the fitness function, the ones with short distance and energy consumption have high possibilities to be chosen to go to next generation. Also there is a very small possibility for mutations; crossover is conducted on the best individual to form new generations. The iterations stops until there is little improvement or the running time overpasses the threshold.

A. Genetic Representation

We store the graph in an adjacency list. Each vertex’s outgoing edge has a unique number, which is the subscript of that edge stored in the adjacency list. We have a list *a*[0], *a*[1] … *a*[n], in which *a*[*i*] is the number indicate which edge should choose if now in that node. So array contains the whole gene that helps the traveler go through from S to T.

B. Fitness Function

The fitness function is analog to survival criteria in the natural environment, which evaluates if the solution is good or not. As at beginning, a valid solution (fulfill the upper bound) is very hard to find, so we do not set the upper bound limit as a concrete condition, the result is that some travelers may exceed that limit when getting to *T* node, but these solutions are also accepted with a very large penalty compared with the those satisfying the requirement. The actual function is as:

$$\mathrm{F}(c_1, c_2) = \begin{cases} c_1 + c_2 \times R & \text{if } c_2 > \mathrm{k} \\ c_1 & \text{if } c_2 \le \mathrm{k} \end{cases} \tag{4}$$

C. Crossover

In order to generate the next generation’s traveler, two travelers will be selected based on some algorithms. They may be selected randomly, or based on the possibility related to the fitness function. After selecting two travelers, the new gene $A'[i]$ is generated by duplicating the gene of either party of the parent. Suppose traveler A has fitness function value *Fa* and traveler B has fitness function value *Fb*, then there are the following probabilities:

$$\mathrm{P}(A'[i]{=}Aa[i]) = \frac{Fb}{Fa+Fb}$$
$$\mathrm{P}(A'[i]{=}Ab[i]) = 1 - \mathrm{P}(A'[i]{=}Aa[i]) = \frac{Fa}{Fa+Fb} \tag{5}$$

D. Mutation

After the crossover, there is a very small possibility (*Pm*) that a segment of gene will be changed. The mutation is the source of gene diversity and also has the risk of instability. If the Pm too small, then the whole generation evolves too slow, and on the other hand if the Pm too large, then the generations may become unstable, the new features have no time to be selected out and stay stable.

E. Tuning

The key part of genetic algorithm is tuning the parameters to make the algorithm grow fast and stay stable. Besides, the size of each generation, the fitness functions as well as the crossover order etc need to be tuned carefully. Most of the parameters are set by experience and also by testing.

VIII. GENERATE RANDOM GRAPH

In this part, a new method for generating a random graph will be discussed. Graph generated using this method will take $O(k^n)$ time to compute. In this graph generating mechanism, nodes are added to the graph layer by layer. Suppose we have *L* layers and the layers are numbered in the sequence of {1, 2 … *L*}.The first layer only contains the source node, the second layer contains two nodes and the last layer contains the destination node. The rest of the layers which contain k nodes are called middle layers. Fig. 5 shows a generalized case of the graph structure. Every middle layer in the graph has three nodes. The adjacent middle layers form a complete bipartite graph. Layer 1 and layer 2 are connected with two links with the cost of (p,q) and (q,p) respectively. All the nodes in layer L-1 has links pointing to the destination which have the same cost of (1,1). For convenience, we force p to be smaller than q, namely p<q. The following paragraph explains how to generate the cost for each link between the middle layers.

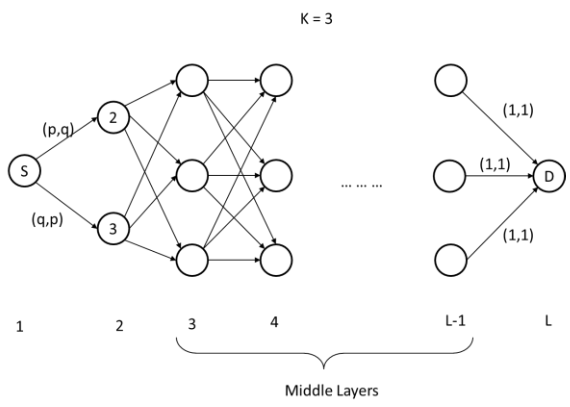

Fig. 5. Generalized graph structure

```
For layer $i$ ← 2 to $L$-2
    $b1$←Random() + $k^{i-1}$ + Pos();
    $b2$←Random() + $k^{i-1}$ + Neg();
    For each node $x$ in layer $i$
        For each node $y$ in layer $i+1$
            Suppose $y$ is the $j$th node in layer $i$+1
            Let $dif = k^{i-2}$;
            The cost from $x$ to $y$ is ($b1$- ($j$-1)·$dif$, $b2$ + ($j$-1)·$dif$)
        End
    End
End
```

A graph which implements the algorithm above is shown in Fig. 6. For clarity, on each layer, only one pair of links is drawn. c1 to c4 are random numbers and the function of Pos() and Neg() are set to be zero in this current case.

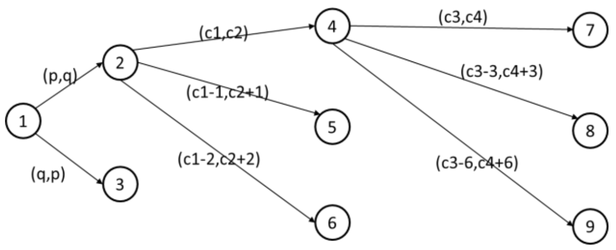

Fig. 6. A graph example

TABLE V
TEMPORARY LABELS GENERATED FOR EACH LAYER

1	2	3	4	
0,0	p,q	p+c1,q+c2	p+c1+c3,q+c2+c4	q+c1+c3,p+c2+c4
		p+c1-1,q+c2+1	p+c1+c3-3,q+c2+c4+3	q+c1+c3-3,p+c2+c4+3
	q,p	p+c1-2,q+c2+2	p+c1+c3-6,q+c2+c4+6	q+c1+c3-6,p+c2+c4+6
		q+c1,p+c2	p+c1+c3-1,q+c2+c4+1	q+c1+c3-1,p+c2+c4+1
		q+c1-1,p+c2+1	p+c1+c3-4,q+c2+c4+4	q+c1+c3-4,p+c2+c4+4
		q+c1-2,p+c2+2	p+c1+c3-7,q+c2+c4+7	q+c1+c3-7,p+c2+c4+7
			p+c1+c3-2,q+c2+c4+2	q+c1+c3-2,p+c2+c4+2
			p+c1+c3-5,q+c2+c4+5	q+c1+c3-5,p+c2+c4+5
			p+c1+c3-8,q+c2+c4+8	q+c1+c3-8,p+c2+c4+8

One can observe from table 5 that on layer 1, there is only 1 label (0,0) and on layer 2, there are 2 labels namely(p,q) and (q,p). These two labels can generate 6 labels on the third layer, which is divided into two groups with the first group generated by (p,q) and second by (q,p). On layer 4 we have altogether 18 labels divided into 6 groups. If we observe layer 3, it can be easily observed that the subset of labels representing a path passing node 2 is non-dominated and the same applies for the case passing node 3. We only need to consider whether the set is still all non-dominated if we merge these two subsets together.

We must make sure that (p+c1, q+c2) and (q+c1-2, p+c2+2) are non-dominated, this is equivalent to the condition that q-p > 2. If we apply the same analysis on layer 4 we will have q – p > 2+ 2·3 = 8.To generalize the relationship to L layers, we have the following formulas. If there are L layers, then layer 3 to layer L-1 are middle layers. The relationship between p and q should satisfy that

$$\begin{gathered} p+\sum_{n=0}^{n=L-4}(k-1)\times k^{n}<q \\ q-p>\sum_{n=0}^{n=L-4}k^{n+1}>\frac{k^{L-2}}{k-1} \end{gathered} \qquad (6)$$

The labels for the destination will have size O(k^l) and In conclusion, the total labels for the destination can be exponential size.

$$L=O(\frac{n}{k})\text{ then }O(k^{L})=O(k^{n/k})=O(k^{'n})\quad\text{where }\ k^{'}=k^{\frac{1}{k}} \quad (7)$$

However, the graph generated in this way is not random enough. The following strategies are proposed in order to add randomness to the graph.

(1) Each node contains different number of nodes

(2) Not every pair of nodes on the adjacent layer will be connected. The possibility that they are connected is *prob*.

In the experiment, k is set to be 4 while k' is set to be 2. *prob* is set to be 0.8. The experiments start from the case where there are 4 layers to 19 layers. For each layer number, 50 random graphs are generated. Table 6 shows the average number of total permanent labels and average running time for the test cases.One can observe that as the number of the layers increases, especially more than 10 layers, the number of total permanent labels and running time nearly doubles as the layer increases by one.

TABLE VI
EXPERIMENT DATA

Layer Num	Permanent Label Num	Run time	Layer Num	Permanent Label Num	Run time
4	5.6	0.25	12	495.2	2.1
5	9.4	0.30	13	1091.5	10.7
6	16.4	0.27	14	2320.4	26.4
7	30.5	0.13	15	3857.1	47.7
8	58.4	0.22	16	8241.0	115.9
9	92.3	0.35	17	17393.9	233.0
10	208.8	0.85	18	37759.6	2274.8
11	252.8	0.85	19	85082.5	14864.0

IX. EXPERIMENT ON MARTIN'S AND GA

In the first group, the graphs used are generated using the algorithm described in section 11. The number of the nodes for the middle layer should be within the range of [2,4]. The probability of assigning an edge is 0.8. The energy cap is about 70% of the energy cost for the path with the smallest latency. The number of nodes increases by 5 each time. For accuracy, for each node number, 20 graphs are generated. The value of computing time is the average of the computing time for the 20 graphs, while the result column only takes that of the first graph. The result comparison is recorded in table 7.

In the second group, an average case which may appear much often in the real life will be analyzed. Anders J. V. Skriver (2000) offers a way to generate a graph which a node will only be connected to neighbors. It is defined that for a node with index *i*, it will only be connected to the nodes which has the index lying in the range of [max(1,*i*-*span*),min(*n*,*i*+*span*)]. In addition, the probability that an edge will be established is *prob*. In the experiment below, *span* is set to be 40 while *prob* is set to be 0.2. The result is shown in Table 8.

TABLE VII
STATISTICS FOR MARTIN'S AND GA

	Martin's		GA	
n	Result	Time	Result	Time
10	94 98	< 1	95 97	1000
15	257 92	< 1	257 92	1000
20	614 788	< 1	614 788	1000
25	9868 8762	< 1	9868 8762	1000
30	105932 159276	15	105932 159276	2000
35	433468 620746	62	433468 620746	2000
40	31812188 35306180	406	Not Found	Nil
45	89484016 178963122	1203	89484016 178963122	2000
50	3433742547 861235252	87125	Not Found	2000

TABLE VIII
STATISTICS FOR MARTIN'S AND GA

	Martin's		GA	
n	Result	Time	Result	Time
100	46 130	< 1	47 35	5000
300	85 289	47	115 289	16000
600	148 662	172	174 354	32000
900	178 872	391	308 492	47000
1200	235 1849	1312	455 576	63000
1500	273 1371	2016	541 863	79000
1800	369 1644	4422	609 750	97000
2100	418 2635	7360	901 982	112000
100	46 130	< 1	47 35	5000

The comparison of the martin's algorithm and GA will be carried out in the following aspects.

Solution: Martin's algorithm can guarantee the optimal solution is found. While the solution of the genetic algorithm will be as close to optimization as possible with the increasing number of generations, it still cannot guarantee optimization. Sometimes, GA might have trouble even coming up with a solution.

Runtime: For most of the random generated cases, Martin's algorithm will terminate in polynomial time, because most of the labels can be eliminate by others and therefore making the number of the temporary labels on a steady level. On the other hand, genetic algorithm is good at NP-hard cases, which have non-polynomial number of labels. GA is also good at large scale average cases. Besides, Marin's algorithm is suitable for small size graph because the cost of traversing all the labels by brute force in is no so high. But GA might take some unnecessary time to form a convergence.

Space: Space can be considered as the most serious problem for martin's algorithm. In cases where there are labels with an exponential size, it is nearly impossible to store all of them while GA does not have such a problem. However, for the average cases where there are polynomial numbers of labels, space is not a problem.

X. CONCLUSION

From this paper, it can be learned that latency is not the only important factor to consider in a routing problem. It is also necessary to take the power consumption into account. To solve such kind of "Green Routing" problem, a revised Martin's algorithm and genetic algorithm is proposed. Martin's algorithm uses a brute force way to search each possible while genetic algorithm implements the natural evolution which can at least get satisfactory solutions in any given time interval.

A method which can generate a random graph that takes exponential time to compute is introduced and such kind of graph works in a worst case analysis. Martin's algorithm is not scalable and works best for small-scale network and fairly well for average real world cases. But the computing time and memory increase exponentially in an extreme bad case where there are non-polynomial numbers of labels. The advantage of genetic algorithm is that it can deal with the NP graph problem, but it does not guarantee that the solution is optimal and the computing time for a small-scale graph is longer than that of Martin's.

REFERENCES

[1] J.B. Smith, D. Shier. An empirical investigation of some bicriterion shortest path algorithms European. Journal of Operation Research, Vol.43, 216-224 (1989).

[2] E.L. Ulungu, J. Teghem. Multi-objective Combinatorial Optimization Problems: A Survey. Journal of Multi-Criteria Decision Analysis, Vol.3, 83-104 (1994).

[3] M. Caramia, P.D. Olmo. Multi-objective Management in Freight Logistics Increasing Capacity, Service Level and Safety with Optimization Algorithms, Chapter 2(2008).

[4] E. Queiros, V. Martins. On a multicriteria shortest path problem (1983) European Journal of Operation Research (1984).

[5] E. Queiros, V. Martins. On a special class of bicriterion path problems European Journal of Operation Research (1984).

[6] A.J.V. Skriver, K.A. Andersen. A label correcting approach for solving bicriterion shortest-path problems Computers and Operations Research, Vol.27, 507-524 (2000).

[7] A.J.V. Skriver. A Classification of Bicriterion Shortest Path (BSP) Algorithms Asia-pacific journal of operational research, Vol.17, 199-212 (2000).