COMPARISONS OF YEN'S AND A MODIFIED GENERALIZED FLOYD *K*-SHORTEST-PATH ALGORITHMS

Huey-Kuo Chen
Professor
Department of Civil Engineering,
National Central University, Taiwan
32054
Tel: +886-3-4227151 ext 4115, Email:
ncutone@cc.ncu.edu.tw

Cheng-Yi Chou
Graduate Student
Department of Civil Engineering,
National Central University, Taiwan
32054
Tel: +886-3-4227151 ext 4144, Email:
monson@trans.cv.ncu.edu.tw

Abstract : Yen's loopless *K*-shortest-path algorithm has been widely used for networks that are comprised of negative link costs. However, it's computational time is usually lengthy, especially for large networks. The generalized Floyd (GF) algorithm, on the other hand, is efficient due to arithmetic operations of matrices for *k* shortest paths but at the high risk of generating looped paths.
In this paper, the GF algorithm is modified to avoid generating looped paths by applying node recurrence check and then compared with Yen's *K*-shortest-path algorithm by means of numerical examples. The results show that the modified generalized Floyd algorithm (MGF) is superior in terms of computation time, but inferior in terms of memory requirement. The computational complexity analysis also support the experimental results.

1. INTRODUCTION

K-shortest-path algorithms are for each O-D pair to find k shortest paths by which many real transportation network problems such as disaster rescue and/or evacuation, freight transportation, origin-destination estimation, are applicable. In recent years, Yen's loopless K-shortest-path algorithm was perhaps the most frequent one being applied in the area of transportation planning. A comparison between Yen's loopless K-shortest-path algorithm and several other algorithms has been made with respect to number of additions, number of comparisons, and memory requirement (Yen, 1971) as shown in Table 1. Yen's algorithm outperforms in all three performance measures, however, its computational complexity indicates that the computation time is proportional to the fourth order of the total number of nodes in a network. For real urban networks which are usually large in the number of nodes, this computational requirement is rather demanding and thus is not acceptable. Reducing the computation time by lowering the magnitude of order with respect to the total number of nodes is a thrust. A hybrid of Clarke's and Yen's methods is such an attempt (Perko, 1986).

Table 1: computational complexities of K-shortest-path algorithms

Algorithm	Approximate Number of Necessary			Ratio of Other Algo's Operations to Yen's Operation
	Additions	Comparisons	Memory Address	
Yen's	KN^4	KN^4	$N^2 + KN$	1
Pollack's	N^{K+2}	N^{K+2}	$N^2 + KN$	N^{K-2} / K
Bock, Kantner and Haynes'	$\sum_{i=1}^{N-2} \binom{N-2}{i} i!$	$\sum_{i=1}^{N-2} i! Log_2 i!$	$N^2 + KN$	Very large
Clarke, Krikorian and Ransans'	Difficult to specify			
Sakarovitch	Difficult to specify			

Source: Yen (1971)

In addition to the K-shortest-path algorithms compared by Yen (1971), a lot more K-shortest-path algorithms can be found (or readily induced) in the area of network flow optimizations. Notable examples are double sweep, generalized Floyd, and generalized Dantzig algorithms (Evans and Minieka, 1992) and a generalization of Tabourier (1973). All four algorithms employ arithmetic operations of matrices to find k shortest paths, which is generally efficient. However, in their original forms, undesired looped paths may be generated during the solution procedure. To retain computational efficiency pertaining to the arithmetic operations of matrices and in the meantime avoid generating undesired looped paths, a modification for the above mentioned four K-shortest-path algorithms is needed. For the purpose of illustration, only the generalized Floyd algorithm is modified and then makes a comparison with Yen's algorithm.
In the following, Yen's algorithm is presented in Section 2. A modified generalized Floyd (MGF) algorithm is introduced in Section 3. Comparisons of Yen's and MGF algorithms are made and numerical examples provided in Section 4. Finally, conclusions and suggestions are given in Section 5.

2. Yen's Algorithm

Yen's K-shortest-path algorithm generates k shortest paths by means of selecting the loopless shortest path from a candidate list (L_2). The paths in the candidate list (L_2) are deduced from the incumbent shortest paths that already stored in the shortest path set L_1. The algorithmic procedure iterates between adding the k-st shortest path into L_1 and generating new candidates into L_2, till all k shortest paths have been found. Upon termination, the first k shortest paths are guaranteed. The time-dependent extension of Yen's algorithm (Chen and Feng, 1998) can be summarized as follows:

Yen's Algorithm

For each O-D pair rs and time interval k, do the following.

Step 0: Initialization.
Let $\kappa = 1$. Search for a shortest path over the time-space network based on the temporarily fixed link travel times $\{\bar{c}_a(t)\}$. If we have less than K and more than

one paths, we assign any arbitrary one of these paths to be $p_{\kappa}^{rs}(k)$ and store it in $L_1^{rs}(k)$ (the list of K shortest paths); the rest of these paths are stored in $L_2^{rs}(k)$ (the list of candidates for (κ+1)st shortest paths). Otherwise, if we have only one such path, it is $p_{\kappa}^{rs}(k)$ which to be stored in $L_1^{rs}(k)$.

Step 1: Routes Augmentation.

Step 1.1: Let $\kappa=\kappa+1$.

Step 1.2: For each of $i=1,\cdots,\text{bnode}(s)$, where $i(t)\in p_{\kappa-1}^{rs}(k)$, do the following:

Step 1.2.1: For each of $j=1,2,\cdots,\kappa-1$, check if the subpath consisting of the first i nodes of $p_{\kappa-1}^{rs}(k)$ in sequence coincide with the suopath consisting of the first i nodes of $p_j^{rs}(k)$ in sequence. If so, temporarily set $c_a(t)=\infty$, $a=(i,\text{anode}(i))\in p_j^{rs}(k)$; otherwise, make no changes.

Step 1.2.2: Apply a shortest path algorithm to find the shortest path from node $i(t)$ to node s, allowing it to pass through those nodes that are not yet included in the path. Note that the subpath from 1 to $i(t)$ is $R_{i(t),\kappa}^{rs}(k)$, the root of $p_{i(t),\kappa}^{rs}(k)$; and the subpath from $i(t)$ to s is $S_{i(t),\kappa}^{rs}(k)$, the spur of $p_{i(t),\kappa}^{rs}(k)$. Note $R_{i(t),\kappa}^{rs}(k)\cap S_{i(t),\kappa}^{rs}(k)=\{i(t)\}, i(t)\in p_{\kappa}^{rs}(k)$.

Step 1.2.2: Find $p_{i(t),\kappa}^{rs}(k)$ by joining $R_{i(t),\kappa}^{rs}(k)$ and $S_{i(t),\kappa}^{rs}(k)$. Then add $p_{i(t),\kappa}^{rs}(k)$ to $L_2^{rs}(k)$. Note that it is necessary to store only the K-k+1 shortest $p_{i(t),\kappa}^{rs}(k)$'s in $L_2^{rs}(k)$.

Step 1.2.3: Place the time-space link costs that temporarily set to infinity by their original values.

Step 1.3: Find from $L_2^{rs}(k)$ the path(s) that have the minimum path travel time. If the path(s) found plus the path(s) that already in $L_1^{rs}(k)$ exceed K, we are done. Otherwise, denote this path (or an arbitrary one, if there are more than one such paths) by $p_{\kappa}^{rs}(k)$ and move it from $L_2^{rs}(k)$ to $L_1^{rs}(k)$--leaving alone the rest of the paths in $L_2^{rs}(k)$. Then go to Step 1.1.

3. MODIFIED GENERALIZED FLOYD ALGORITHM

The idea underlying the generalized Floyd algorithm is identical to that underlying the Floyd algorithm (Floyd, 1962), except that addition is replace by generalized addition and minimization is replaced by generalized minimization (Evans and Minieka, 1992). These operations are performed not on single numbers but on sets of k distinct numbers that represent the lengths of paths. As mentioned above, this algorithm is computationally fast but may generate looped paths during the solution procedure. To avoid generating looped paths and in the meantime retain its computational efficiency, the original generalized Floyd algorithm has to include additional checks on node recurrences, hereafter referred to as the modified generalized Floyd (MGF) algorithm. The steps of the MGF algorithm may be stated as follows:

MGF Algorithm

Step 0: Initialization.

Let h=0. Initialize a path cost matrix $\mathbf{C}^h$ and a path trace matrix $\mathbf{P}^h$

$$\mathbf{C}^h = \begin{bmatrix} (c_1^{11}, c_2^{11}, \ldots, c_K^{11}) & (c_1^{12}, c_2^{12}, \ldots, c_K^{12}) & \cdots & (c_1^{1n}, c_2^{1n}, \ldots, c_K^{1n}) \\ (c_1^{21}, c_2^{21}, \ldots, c_K^{21}) & (c_1^{22}, c_2^{22}, \ldots, c_K^{22}) & \cdots & (c_1^{2n}, c_2^{2n}, \ldots, c_K^{2n}) \\ \vdots & \ddots (c_1^{ij}, c_2^{ij}, \ldots, c_K^{ij}) \ddots & & \vdots \\ (c_1^{n1}, c_2^{n1}, \ldots, c_K^{n1}) & (c_1^{n2}, c_2^{n2}, \ldots, c_K^{n2}) & \cdots & (c_1^{nn}, c_2^{nn}, \ldots, c_K^{nn}) \end{bmatrix} \quad (1)$$

$$\mathbf{P}^h = \left[(p_1^{o1}, p_2^{o1}, \ldots, p_K^{o1})(p_1^{o2}, p_2^{o2}, \ldots, p_K^{o2}) \cdots (p_1^{o2}, p_2^{o2}, \ldots, p_K^{o2}) \right] \quad (2)$$

Each vector $\mathbf{c}^{ij}$ contains k shortest paths directing originating from node i to node j. In the beginning, the link cost between nodes i and j is entered as the first component, c_1^{ij}, and the rest K-1 elements are temporarily filled with infinitive. Similarly, each vector $\mathbf{p}^{ij}$ contains K elements denoting the intermediate nodes that precede node j from i for the current K shortest paths. If the k-st shortest path has not generated, then the corresponding element is temporarily filled with node number i, which will be updated later.

Step 1: Recursive Operations.

Let $h = h + 1$. Perform the following generalized additions and minimizations.

$$\mathbf{P}^h = \left[\mathbf{P}^{h-1} \odot \mathbf{C} \right] \oplus \mathbf{P}^{h-1} \quad (3)$$

If shorter loopless paths are found, update the matrices **C** and **P** accordingly. Otherwise, continue.

Step 2: Termination Check.

If two successive matrices $\mathbf{c}^{ij}$ and $\mathbf{c}^{ij}$ are identical, then stop with optimal solution $\mathbf{P}^h$. Otherwise, go to Step 1.

In equation (3), generalized additions and minimizations are performed. Assume $\mathbf{a} = (a_1, a_2, a_3, \ldots, a_K)$ and $\mathbf{b} = (b_1, b_2, b_3, \ldots, b_K)$, the generalized arithmetic operators, $\oplus$ and $\otimes$, are defined respectively as follows:

$$\mathbf{a} \oplus \mathbf{b} = \min{}_K \{a_i, b_i \colon i = 1,2,3,\ldots,K\} \quad (4)$$

$$\mathbf{a} \otimes \mathbf{b} = \min{}_K \{a_i + b_j \colon i, j = 1,2,3,\ldots,K\} \quad (5)$$

Given $\mathbf{a} = \{1,3,8\}$ and $\mathbf{b} = \{3,5,16\}$, we obtain:

$$\mathbf{a} \oplus \mathbf{b} = \min{}_3 \{1,3,8,3,5,16\} = \{1,3,3\}$$

$$\mathbf{a} \otimes \mathbf{b} = \min{}_3 \{1+3, 1+5, 1+16, 3+3, 3+5, 3+16, 8+3, 8+5, 8+16\} = \{4,6,6\}$$

To demonstrate how the MGF algorithm works, a simple network shown in Figure 1 is taken for illustration.

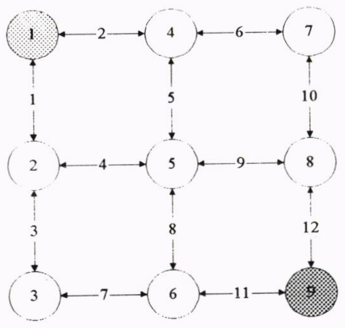

Figure 1: Test Network 1

Upon exploitation of the MGF algorithm, four types of information shown in Figure 2 must be recorded at each node. The first information identifies O-D pair *r-s*. The second information tags every node with different node number. The third information stores the new subroute cost from origin *r* to node *a*. The fourth information labels the *k* shortest subroute costs, denoted by cost 1 through cost *k*.

r - s	
node a	
new cost	
cost 1	*cost 2*

Figure 2: Node Information

With the above four types of node information, a *K*-shortest-path spanning tree is generated from each origin *r* toward each destination *s* as shown in Figure 3. At each intermediate node *a*, if the ***new subroute cost*** is loopless and lower than any of the incumbent *k* shortest subroute costs, then replaces the highest subroute cost with this new subroute cost. The occurrence of a loop can be avoided if the succeeding node of the current node has not encountered before. This spanning process continues till the destination has been reached.

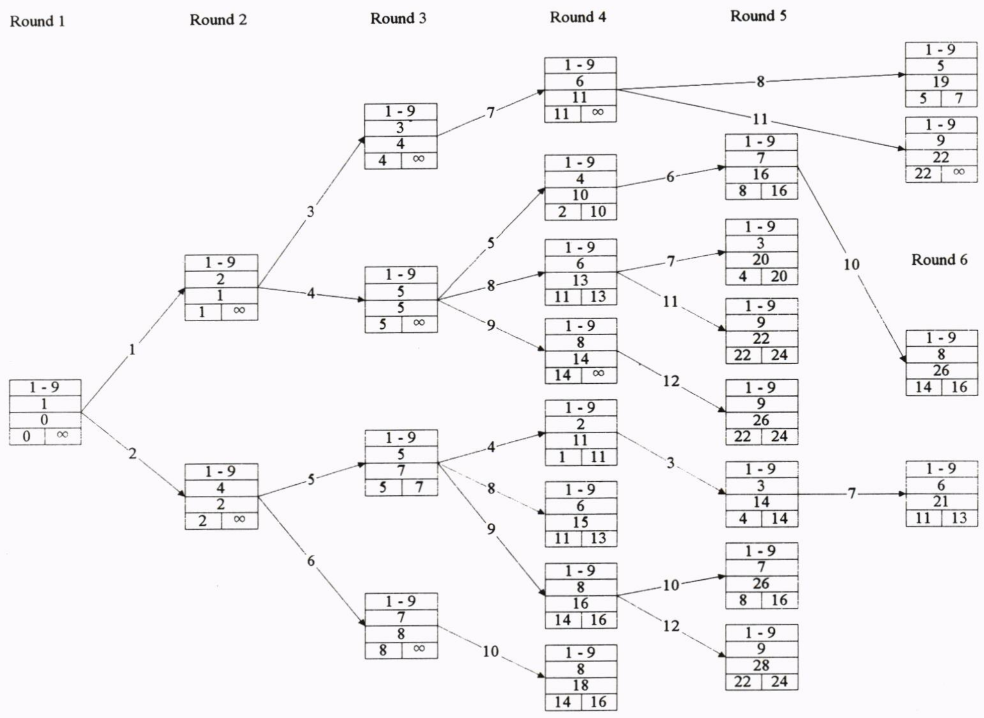

Figure 3: *K*-Shortest-Path Spanning Tree Associated with the MGF Algorithm

4. COMPARISON OF YEN'S AND MGF ALGORITHMS

4.1 Computational Complexities

For a network with N nodes, computational complexities for Yen's and MGF K-shortest-path algorithms can be summarized in Table 2. Yen's algorithm requires less computer memory but the MGF algorithm is superior in terms of computational efficiency.

Table 2: Computational Complexities for Yen's and MGF *K*-shortest-path Algorithms

Algorithm	Approximate Number of Necessary			Ratio of Other Algo's Operations to Yen's Operation
	Additions	Comparisons	Memory Address	
Yen's	$4KN^4$	KN^4	$2N^2+2KN$	1
MGF	$3KN^3(K+N)$	$K^2N^3+KN^4$	N^3K	$\frac{4}{5}+\frac{4K}{5N}$

Source: this study.

4.2 Numerical Examples

To validate the computational complexities of Yen's and the MGF algorithms. Three networks with different topologies are taken for testing. The first two networks are hypothetical whereas the third network is realistic. The first test network plotted in Figure 4 is in a grid shape. The incidence of two adjacent nodes is determined by random sampling, with average connection rate 0.75. The second test network shown in Figure 5 is constructed by connecting two consecutive nodes. The linkage of two nodes not adjacent in node numbers is determined randomly. The last test network shown in Figure 6 represents the Chungli-Pincheng urban area in Taoyuan County, Taiwan.

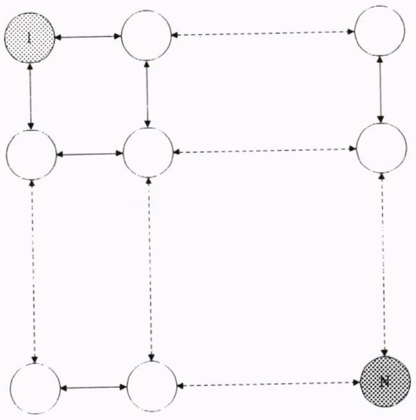

Figure 4: Test Network 2

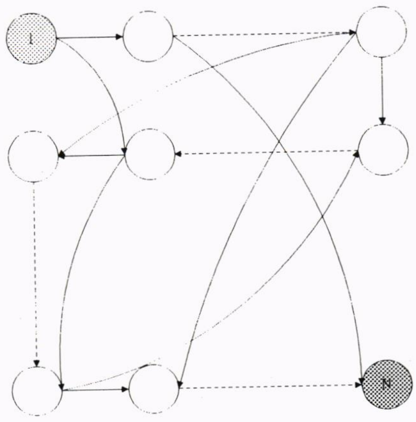

Figure 5: Test Network 3

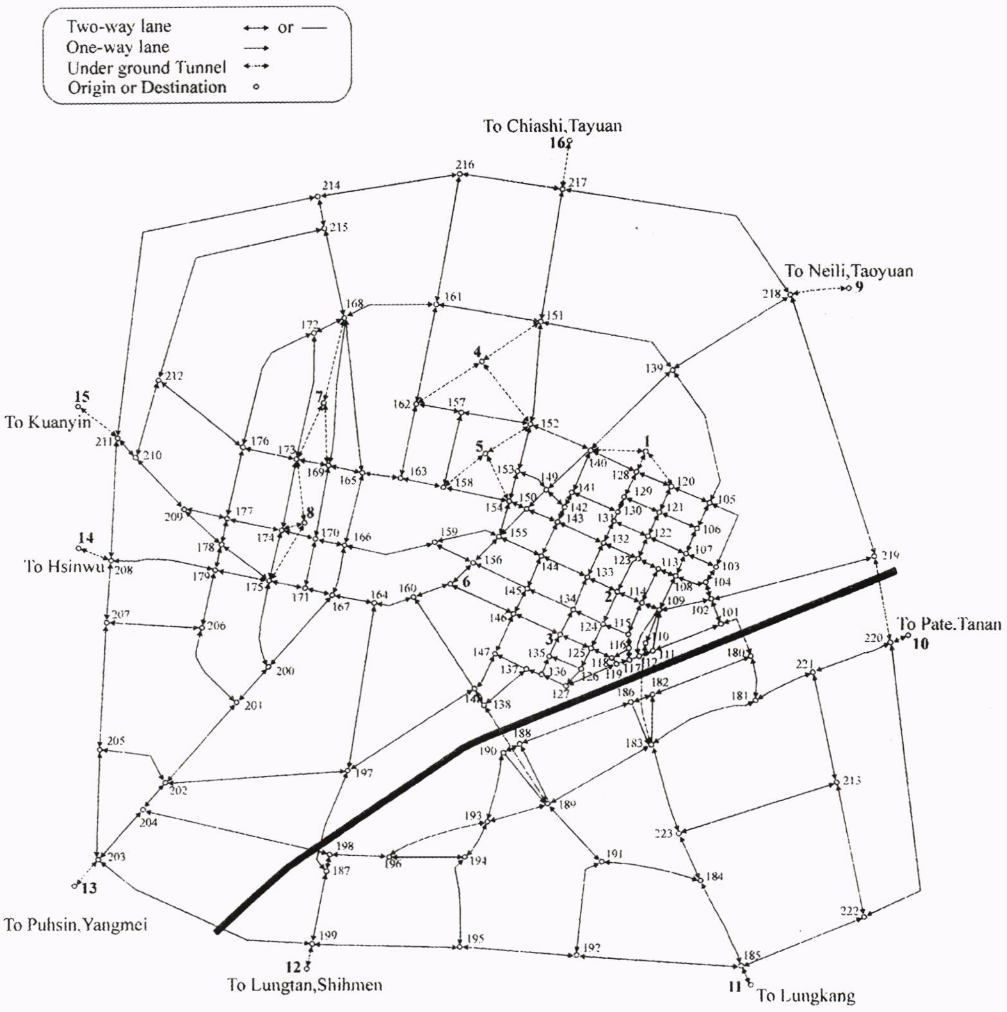

Figure 6: Test Network 4

A computer program coded with Borland C++ 5.01 was executed on a Pentium II 266 personal computer equipped with 128M RAM. The computational efficiency is analyzed as with number of shortest paths (K), number of nodes (N), and network density.

(1) Number of shortest paths (*K*)

Suppose test network 3 consist of 308 links and 100 nodes, the computational times with different number of shortest paths are plotted in Figure 7. The computational times are linearly proportional to the number of shortest paths for both Yen's and MGF algorithms (see Table 3) and the latter accounts for about 15%~30% of the former.

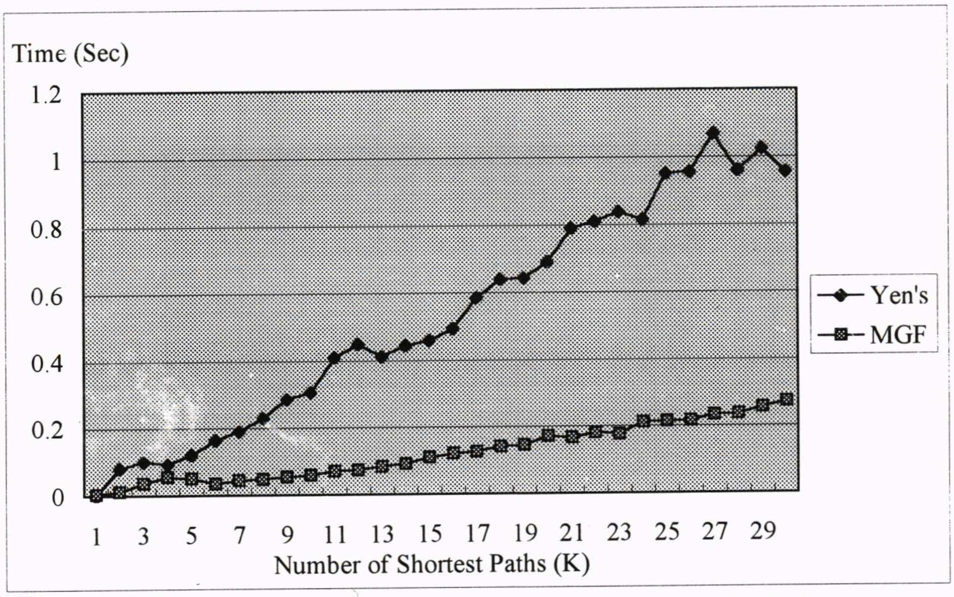

Figure 7: Computational Times versus Number of Shortest Paths

Table 3: Regressions of Computational Times in terms of Number of Shortest Paths

Algorithm	Regression	R-Squared
Yen's	$Y=0.0262*K^{1.088}$	0.9749
MGF	$Y=0.0074*K^{1.011}$	0.9213

(2) Number of Nodes (*N*)

Suppose test network 3 are duplicated 18 times with different network sizes, in the range of 20 to 500 node numbers. The computational times of generating 30 shortest paths for 18 networks are plotted in Figure 8. The MGF algorithm obviously outperforms and the superiority gets more pronounced as the number of nodes get larger. While the computational times are proportional to the square of the number of nodes for Yen's algorithm, the linear relationship is derived for the MGF algorithm, see Table 4 .

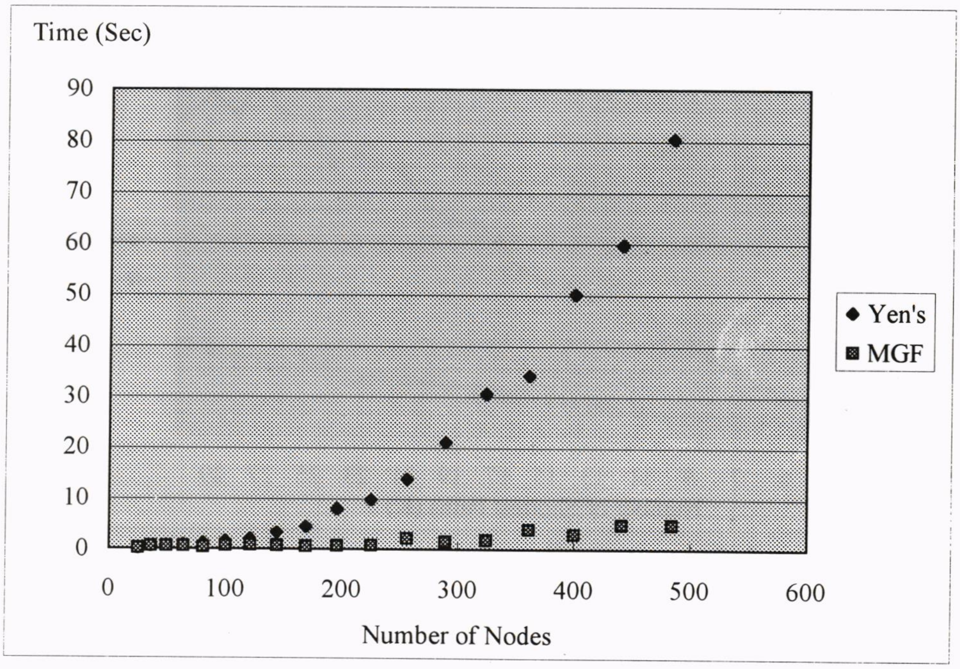

Figure 8: Computational Times versus Number of Nodes

Table 4: Regressions of Computational Times in terms of Number of Nodes

Algorithm	Regression	R-Squared
Yen's	$Y=0.001585*nodes^{2.075}$	0.9709
MGF	$Y=0.01258*nodes^{0.8978}$	0.7710

(3) Network Density

Suppose test network 3 contain 100 nodes, the computational times of generating 10 shortest paths for different connection rates, ranging from 0.0 to 0.3, for both Yen's and MGF algorithms are plotted in Figure 9.

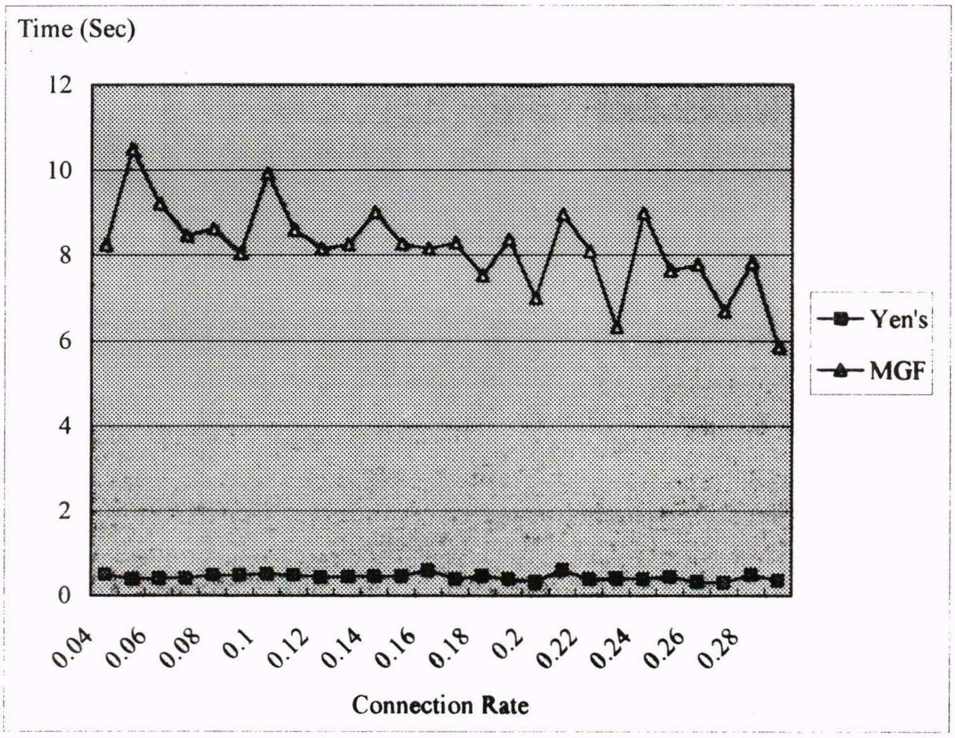

Figure 9: Computational Time versus Connection Rate

Higher connection rates do not necessarily require higher computational times which contradicts our intuitive expectation. Moreover, the MGF algorithm performs inferior in this experiment. Evidently, no conclusive evidence can be drawn at this point.

(4) Real Network Testing

A real network, Chungli-Pincheng urban area, is taken for testing. This network consists of 435 links and 138 nodes, and link costs are realized by their link lengths. Due to computer capability, only one O-D pair (from Lungkang to Taoyuan) is examined with different number of shortest paths, ranging from 1 to 110. The relationship between computational time and the number of shortest paths is plotted for both the MGF and Yen's algorithms in Figure 10. The computational times monotonically increase as the number of shortest paths increases. The MGF performs better by requiring about 70%～90% less computational times.

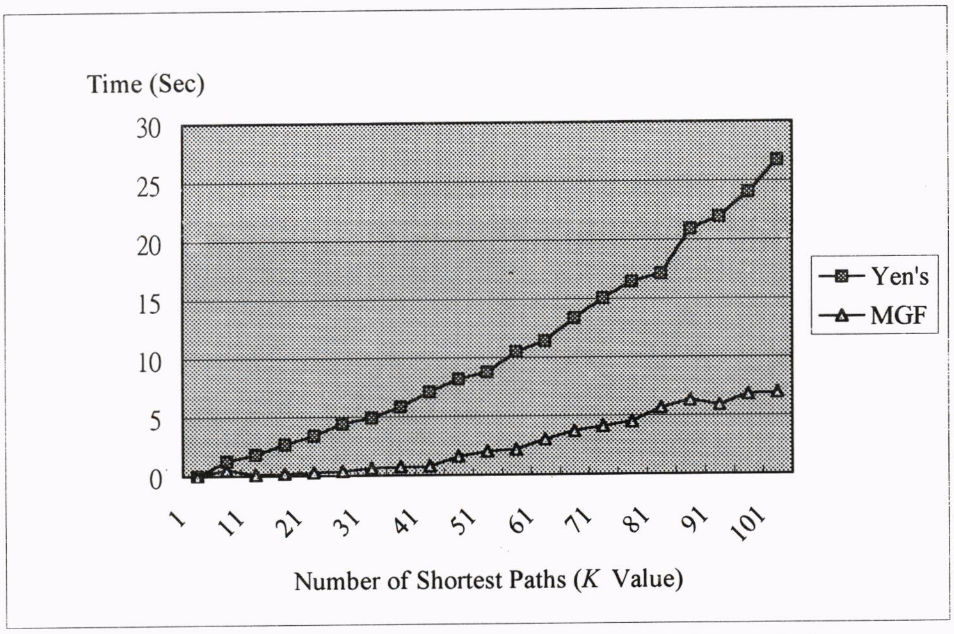

Figure 10: Computational Times versus Number of Shortest Paths for Chungli-Pincheng City

5. CONCLUSION AND SUGGESTIONS

In this paper, a generalized Floyd algorithm is modified to avoid the occurrence of looped paths and in the mean time to retain its computational efficiency. From the theoretical point of view, the proposed MGF algorithm needs more computer memory but requires fewer computational time as compared with Yen's algorithm. Experiments conducted on three test networks basically justified these points. More specifically, the computational times required by the MGF are shorter for a grid network and a real network, and the superiority becomes more pronounced as the network size gets larger. To be conclusive, more tests on different types of networks are essential. It is also worth noting that the high overlapping ratio among *K* shortest paths are generally not desired and needs to be tackled in the future.

REFERENCES

1. Chen H.K. and Feng G. 1998. Heuristics for the Stochastic/Dynamic User-Optimal Route Choice Problem, **The European Journal of Operational Research(Accepted)**. **(SSCI)**
2. Evans J.R. and Minieka E. 1992. **Optimization Algorithms for Networks and Graphs (2 Edition)**, Marcel Dekker. Inc., New York.
3. Floyd R.W. 1962. Algorithm 97, Shortest Path, Commun. **ACM**, **5**, p.345.
4. Papadimitrion L.H. and Steiglitz K. 1982. **Combinatorial Optimization Algorithm and Complexity**. Prentice Hall Inc., New Jersey. 164-166.
5. Perko A. 1986. Implementation of Algorithms for *K* Shortest Loopless Paths. **Networks**, Vol. **16**, 149-160, John Wiley & Sons, Inc.
6. Shier D. R., 1974, Computational Experience with an Algorithm for Finding the *k* Shortest Paths in a Network, **J. R. Natl. Bur. Std., 78B**, 139-165.
7. Tabourier Y. 1973. All Shortest Distances in a Graph-- An Improvement to Dantzig's Inductive Algorithm, **Discrete Mathematics 4**, 83-87, North-Holland Publishing Company.
8. Yen J.Y. 1971. Finding the *K* Shortest Loopless Paths in a Network. **Management Science**. Vol. **17**. 712~716.

NOTATIONS

Symbols used in this paper are summarized as follows:

C_k^{ij}	: an element of matrix **C**, representing the κst shortest path cost between nodes i and j
$\mathbf{C}$	: the link cost matrix
K	: predetermined number of shortest paths to be found
$L_1^{rs}(k)$	: the list of κ-shortest paths
$L_2^{rs}(k)$	: the list of candidates for (κ+1)st shortest paths
$p_\kappa^{rs}(k)$	: the κst shortest path from origin r to destination s during time interval k
p_k^{ij}	: the cost for the κst shortest path between nodes i and j
$\mathbf{P}^h$	: the path trace matrix at iteration h
$R_{i(t),\kappa}^{rs}(k)$	: the subpath from 1 to $i(t)$ on which overlaps occur with the first i nodes of $p_{\kappa-1}^{rs}(k)$
$S_{i(t),\kappa}^{rs}(k)$	: the subpath from $i(t)$ to s during the search procedure for κst shortest path, the spur of $p_{i(t),\kappa}^{rs}(k)$