Network-Tree Model and Shortest Path Algorithm

Guozhen Tan[1], Xiaojun Han[1], and Wen Gao[2]

[1] Department of Computer Science and Engineering
Dalian University of Technology, 116024, China*
gztan@dlut.edu.cn
[2] Institute of Computing Technology
Chinese Academy of Sciences, 100080, China

Abstract. For the first time, this paper proposes the Network-Tree Model (NTM), route optimization theory and wholly new, high performance algorithm for shortest path calculation in large-scale network. We first decompose the network into a set of sub-networks by adopting the idea of multi-hierarchy partition and anomalistic regional partition, and then construct the NTM and the Expanded NTM. The Network-Tree Shortest Path (NTSP) algorithm presented in this paper narrows the searching space of the route optimization within a few sub-networks, so it greatly reduces the requirements for main memory and improves the computational efficiency compared to traditional algorithms. Experiment results based on grid network show that NTSP algorithm achieves much higher computational performance than Dijkstra algorithm and other hierarchical shortest path algorithms.

Keywords: nNetwork-tree model, shortest path algorithm, large-scale network, route optimization algorithm

1 Introduction

The shortest-path problem (SP) is a key problem of many fields [1] such as computer science, transportation engineering, communication engineering, system engineering, and operation research. Especially, the computing efficiency is the bottleneck problem of route optimization for large-scale network. Furthermore, the main memory requirement also is an important factor as to implement the algorithm [2].

As known, there are two kinds of algorithms for shortest path computing, that one is label-setting algorithm and the other is label-correcting algorithm, with the representation of the classic Dijkstra algorithm and Bellman-Ford algorithm respectively. All the other algorithms are the variations of these two kinds. People have widely realized that all these algorithms cannot meet the requirements of many applications when computing the shortest paths of large-scale network.

* This work was supported in part by Grand 99025 of Ministry of Education and Grand 9810200104 of Liaoning Science Foundation, China.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 537–547, 2003.

Partitioning the network and constructing new network model have become a fast and effective methods for improving computations in large-scale networks [3], which have been widely applied in many fields such as Intelligent Transportation System (ITS), Graphic Information System (GIS) and route protocols of network communication. Level graphs and associated approximate shortest path algorithm were presented in [4,5,6], which significantly improved the computing efficiency of path optimization. However, the level graph model cannot be abstracted from any given graph, and the papers only provide the approximate shortest path algorithm with the errors that cannot be overlooked. Hierarchical Encoded Path View (HEPV) model can be found in [2,7], and a path-query algorithm based on this model was proposed to precompute and store the shortest path of each sub-network for later on-line query. It did not provide significant improvement for route optimization of multi-level large-scale network. Furthermore, the method requires a large overhead main memory for storing the precomputed path information, which makes it impractical to use for calculating the shortest path in a large-scale network.

For the first time, we present the new network-tree model and the associated network-tree shortest path (NTSP) algorithm to solve effectively the storage requirement and efficiency problem of large-scale network optimization. The prominent advantages of network-tree model and NTSP algorithm compared to the previous ones can be founded in the following four aspects: (1) We can construct the network-tree model from any network; (2) The regional partition and hierarchical partition method of network-tree model is very useful to store the network date both hierarchically and distributively; (3) The concept of tree expresses clearly the internal relation among the sub-networks. (4) The route optimization can be constrained within a few sub-networks, which can greatly reduce the searching scope and significantly improve the computing efficiency. So the main memory requirements of our algorithm are very limited.

The remainder of this paper is organized as follows: In Sect. 2, we present the definitions of both the network-tree model and expanded network-tree model, and the optimization theorem. In Sect. 3, we present the preprocessing procedure and NTSP algorithm. The experiment result will be presented in Sect. 4. Section 5 is the conclusions.

2 Network-Tree Model

Network-Tree Model is the definitional combination and expansion of network and tree. Any network can be divided into a number of sub-networks, and each sub-network will be constructed to one node of network-tree. Specially, the node of network-tree is called macro-node to distinguish from the node of original network.

2.1 Definition of Network-Tree Model

Let the network-tree be $N\_Tree = (T, H)$, where T is the macro-node set of N. If only one macro-node exists in T, $H = \phi$, otherwise, H is a binary relation

defined on T. There is only one macro-node m_r called the root of N_Tree. If $T - \{m_r\} \neq \phi$, there exists a division of $T_1, T_2, \cdots, T_n (n > 0)$, with $\forall j \neq k, 1 \leq j, k \leq n, T_j \cap T_k = \phi$. For $\forall i, 1 \leq i \leq n$, the only macro-node $m_i \in T_i$ satisfies $< m_r, m_i > \in H$. As the counterpart of the $T - \{m_r\}$ division, there exists the only division $H_1, H_2, \cdots, H_n$ of $H - \{< m_r, m_1 >, \cdots, < m_r, m_n >\}$, with $\forall j \neq k, 1 \leq j, k \leq n, H_j \cap H_k = \phi$. For $\forall i, 1 \leq i \leq n$, H_i is a binary relation defined on T_i, and (T_i, H_i) is also a network-tree defined above, and we call it the sub-tree of the root macro-node m_r.

Next, we will present the method of network division and network-tree model construction procedure from any network.

2.2 Construction of Network-Tree Model

Let the network be $N = (V, A, W)$, where V is a finite set of nodes, $A \in N \times N$ is a finite set of arcs, $W = \{w(a) | a \in A\}$, w is the mapping function of $A \to R^+$. First, we assign levels to all arcs according to properties in the original network, that is, we create a mapping function for arc set A.

Definition 1. $\forall a \in A, \exists h \in I, I = \{1, 2, \cdots, k\}$, there exists a mapping function $f : A \to I$, which makes $f(a) = h$, where h is the level of arc a. And $\forall h \in I$, which makes $f^{-1} = \{a | a \in A, f(a) = h\}$ as a non-null space. Let the arc set of level h be A_h, that is, $A_h = f^{-1}(h)$.

Generally, let A_1 be the highest level arc set. The more importance in rank in the network, the smaller value of arc level.

Based on the arc level, we create the network-tree $N\_Tree = (T, H)$ from the network $N = (V, A, W)$. Let the A_1 induced sub-network of N be the root macro-node m_1 of the network-tree, $m_1 \in T$. If $N - \{m_1\} \neq \phi$, we divide the sub-network $N - \{m_1\}$ into n_1 non-null sub-networks $T_{11}, T_{12}, \cdots, T_{1n_1}$ by A_1, and for $\forall p \neq q$, $1 \leq p, q \leq n_1$, we get $T_{1p} \cap T_{1q} = \phi$. Then $\forall i, 1 \leq i \leq n_1$, abstract the highest-level arcs from T_{1i} to make the arc set A^{1i}, let the A^{1i} induced sub-network of T_{1i} and the node intersection set of T_{1i} and m_1 be the ith son macro-node m_{1i} of m_1, that is $m_{1i} \in T$, $< m_1, m_{1i} > \in H$. If $T_{1i} - \{m_{1i}\} \neq \phi$, we continue to divide the sub-network $T_{1i} - \{m_{1i}\}$ into n_{1i} non-null sub-networks $T_{li1}, T_{1i2}, \cdots, T_{1in}$ by A^{1i}, and for $\forall p \neq q$, $1 \leq p, q \leq n_{1i}$, we get $T_{1ip} \cap T_{1iq} = \phi$. Then $\forall j$, $1 \leq j \leq n_{li}$, abstract the highest-level arcs from T_{1ij} to make the arc set A^{lij}, let the A^{lij} induced sub-network of T_{1ij} and the node intersection set of T_{1ij} and m_{1i} be the jth son macro-node m_{1ij} of m_{1i}, that is $m_{1ij} \in T$, $< m_{1i}, m_{1ij} > \in H$. Follow the step above to continue the construction until we get the network-tree model $N\_Tree = (T, H)$.

We get the following definitions and properties of NTM.

Definition 2. In network-tree model $N\_Tree = (T, H)$, any macro-node $m_c \in T$, if there exists the parent macro-node $m_p \in T$, that is, $< m_p, m_c > \in H$, the node set of $m_c \cap m_p$ is called the connecting node set of m_c, denoted by R_{m_c}. Obviously, the connecting node set of the root macro-node is null.

Definition 3. The mapping function $\sigma : V \to T$, for $\forall v \in V$, there exists the only macro-node $m_i \in T$, $v \in V(m_i)$ and $v \notin R_{m_i}$, then let $(v, m_i) \in \sigma$, denoted by $\sigma(v) = m_i$, and we call σ the mapping from node of original network to macro-node of network-tree.

Definition 4. The mapping function $\varphi : V \to I$, $\forall v \in V$, let $\varphi(v)$ denote the level of node v, with the level number of the macro-node $\sigma(v)$ in the network-tree being the node's level number. We assign the level number of root macro-node to 1.

Property 1. $\forall u, v \in V$, let $T_{\sigma(u)}$ be the sub-tree which root is $\sigma(u)$, and m_p be the parent macro-node of $\sigma(u)$, with m_p being non-null. If $\sigma(v) \notin T_{\sigma(u)}$, and there exists the path $\pi = \pi_1 = (v_0 = u, v_1, v_2, \cdots, v_n = v)$ or $\pi = \pi_2 = (v_0 = v, v_1, v_2, \cdots, v_n = u)$ between u and v in the original network, then $\exists v_i$, $0 < i \leq n$, with $v_i \in R_{\sigma(u)} \subseteq V(m_p)$, and if $\pi = \pi_1$, the path from u to v_i must be in $T_{\sigma(u)}$, and if $\pi = \pi_2$, the path from v_i to u must be in $T_{\sigma(u)}$.

Property 1 shows the special characteristics of path in network-tree, which can be directly extracted from the construction procedure of network-tree.

2.3 Theory of Path Optimization

Let $sp_{N_0}(u, v)$ denote the shortest path from u to v in network N_0, and $l(\pi)$ denote the distance of path π.

Definition 5. Network-tree model $N\_Tree = (T, H)$, if for each sub-tree T' of N_Tree, with macro-node m_r being the root of T', $\forall u, v \in V(m_r)$, $l(sp_{m_c}(u, v)) = l(sp_{T'}(u, v))$, thus N_Tree is called Expanded Network-Tree Model.

Definition 6. Network-tree model $N\_Tree = (T, H)$, let the transitive closure of macro-node m be $B_m = \{m_p | (m_p \in T) \wedge (m_p = m \vee (\exists m_q \in B_m)(< m_p, m_q > \in H))\}$.

In Expanded Network-Tree Model, we get the following optimal theorem.

Theorem 1. In the Expanded Network-Tree Model, $\forall s, t \in V$, let $B_{\sigma(s)}$, $B_{\sigma(t)}$ be the transitive closures of macro-nodes $\sigma(s)$ and $\sigma(t)$ respectively, and then there must be $l(sp_N(s, t)) = l(sp_{B_{\sigma(s)} \cup B_{\sigma(t)}}(s, t))$, that is, at least one shortest path from s to t in original network N exists in the sub-networks contained by $B_{\sigma(s)}) \cup B_{\sigma(t)}$.

Proof. Let one of shortest paths from s to t in original network N be $sp_N(s, t) = (v_0 = s, v_1, v_2, \cdots, v_n = t)$, so we only need to prove if $sp_N(s, t)$ does not exist in the sub-networks of $B_{\sigma(s)}) \cup B_{\sigma(t)}$, we always can find the path π from s to t in the $B_{\sigma(s)}) \cup B_{\sigma(t)}$, with $l(\pi) = l(sp_N(s, t))$.

We discuss the $sp_N(s, t)$ repeatedly under the following three cases:

Case 1: when $\varphi(s) > \varphi(t)$, let $T_{\sigma(s)}$ be the sub-tree which root is $\sigma(s)$, and m_p be the parent macro-node of $\sigma(s)$, obviously $m_p \neq \phi$ and $\sigma(t) \notin T_{\sigma(s)}$. According to the Property 1, there must be $\exists v_i$, $0 < i \leq n$, with $v_i \in R_{\sigma(s)} \subseteq V(m_p)$, and

the path from s to v_i exists in $T_{\sigma(s)}$, so $l(sp_{T_{\sigma(s)}}(s, v_i)) = l(sp_N(s, v_i)))$. As $s, v \in \sigma(s)$, and according to the definition of expanded network-tree model, $l(sp_{\sigma(s)}(s, v_i)) = l(sp_{T_{\sigma(s)}}(s, v_i))$. Therefore, $l(sp_{\sigma(s)}(s, v_i)) = l(sp_N(s, v_i))$. Let $\pi = sp_{\sigma(s)}(s, v_i) + sp_N(v_i, t)$, and then $l(\pi) = l(sp_N(s, t))$. $\sigma(s) \in B_{\sigma(s)}$, so $sp_{\sigma(s)}(s, v_i)$ exists in $B_{\sigma(s)}$. Let $s = v_i$, and we continue to discuss the sub-path $sp_N(v_i, t)$ until all sub-paths of π exist in $B_{\sigma(s)} \cup B_{\sigma(t)}$.

Case 2: when $\varphi(s) < \varphi(t)$, Similar as the case 1, there must be $\exists v_i$, $0 < i \leq n$, with $v_i \in R_{\sigma(t)} \subseteq V(m_p)$, and the path from v_i to t exists in $T_{\sigma(t)}$. Let $\pi = sp_N(s, v_i) + sp_{\sigma(t)}(v_i, t)$, with $l(\pi) = l(sp_N(s, t))$, and $sp_{\sigma(t)}(v_i, t)$ exists in $B_{\sigma(t)}$. Let $t = v_i$, and we continue to discuss the sub-path $sp_N(s, v_i)$ until all sub-paths of π exist in $B_{\sigma(s)} \cup B_{\sigma(t)}$.

Case 3: when $\varphi(s) = \varphi(t)$, if $\sigma(s) \neq \sigma(t)$, discuss the path as case 1. And if $\sigma(s) = \sigma(t)$, that is, both s and t locate in the same macro-node, there exist two case: **A**.When $\sigma(s)$ is not the root macro-node of N_Tree, if $l(sp_{\sigma(s)}(s, t) = l(sp_N(s, t))$, let $\pi = sp_{\sigma(s)}(s, t)$, with π being in the $B_{\sigma(s)}$, satisfying the proof requirements, and quit the discussion. Otherwise, there must be $\exists v_i$, $o < i < n$, with $v_i \in R_{\sigma(s)}$, and $sp_N(s, v_i)$ exists in the $\sigma(s)$, that is, $l(sp_{\sigma(s)}(s, v_i)) = l(sp_N(s, v_i))$. Let $\pi = sp_{\sigma(s)}(s, v_i) + sp_N(v_i, t)$, and continue to discuss the rest path $sp_N(v_i, t)$ as case 2. **B**.When $\sigma(s)$ is the root macro-node of N_Tree, then $\pi = sp_{\sigma(s)}(s, t)$, obviously $l(\pi) = l(sp_N(s, t))$, and π exists in $B_{\sigma(s)}$, satisfying the proof requirements, and quit the discussion.

Stated as above, the path π will exist in the sub-networks contained by $B_{\sigma(s)}$ and $B_{\sigma(t)}$ after a finite number of discussions, with $l(\pi) = l(sp_N(s, t))$. The theorem of route optimization builds theoretical bases for high performance shortest path computing of large-scale network, and we only need to search a few macro-nodes when computing the shortest path of Expanded NTM.

3 Shortest Path Algorithm of Expanded Network-Tree Model

3.1 Preprocessing Procedure

We have presented the construction procedure from general network to network-tree in the previous section, but the constructed network-tree cannot be guaranteed to be an expanded network-tree. Therefore we need a preprocessing procedure to create the expanded network-tree from network-tree. First we present the definition of virtual arc.

Definition 7. In network model $N = (V, A, W)$, $\forall u, v \in V$, let the virtual arc from u to v be $\overline{a_{uv}} = (u, v, sp_N(u, v))$, with $w(\overline{a_{uv}}) = l(sp_N(u, v))$, that is, the virtual arc is constructed by the shortest path between the corresponding two nodes.

Preprocessing procedure will begin with the bottom level macro-nodes in the network-tree, and process all the macro-nodes from bottom to top. For any two connecting nodes in a macro-node m, if the shortest path between these two

nodes in m is less than the corresponding shortest path in the parent macro-node of m, add the virtual arc between these two nodes in m to the parent macro-node of m. We present the following recursive procedure, where the function $Djik(u, v, N_0)$ denotes that we invoke the $Dijkstra$ algorithm to calculating the shortest path from u to v in the network N_0.

Preprocessing Algorithm:
/* Input: the macro-node m*/
[1] : **for** each son macro-node m_s of m **do**
[2] : Preprocess (m_s);
end for;
[3] : **if** the parent macro-node m_p of m not null **then**
[4] : **for** $\forall u, v \in R_m$**do**
[5] : **if** $Djik(u, v, m) < Djik(u, v, m_p)$, **then**
[6] : $new\_arc = (u, v, Djik(u, v, m))$, $m_p = m_p \cup \{new\_arc\}$;
end if; end for; end if; end.

So we can invoke the preprocessing procedure to convert any network-tree to the expanded network-tree, with the root macro-node of the network-tree being the input macro-node. Obviously, the preprocessed network-tree is sure to be the expanded network-tree.

3.2 Network-Tree Shortest Path (*NTSP*) Algorithm

There should be two steps in the $NTSP$ algorithm: computing the transitive closure B that the optimal path exists in and searching the optimal path in B.

We present the following $NTSP$ algorithm, where the network model is the expanded network-tree N_Tree, and $d[v]$ is the label of the shortest distance from starting node s to the node v, and $\pi(v_i)$ represents the shortest path from starting node s to node v_i, and *open* is the set of temporarily-labeled node during the searching procedure, and *close* is the set of permanently-labeled node.

$NTSP$ Algorithm.
/* Input: Starting node s and destination node t; Output: The shortest path from s to t */
[1] Initialization:
$open = \{s\}$, $closed=\phi$; $\forall v \in N\_Tree$, $d[v]=\infty$, $d[s]=0$;
$\forall i \in N\_Tree$, $\pi_i = Null$; $B = \phi$, $m_s = \sigma(s)$,$m_t = \sigma(t)$;
[2] **while** $m_s \neq \phi$ **do** $B=B+m_s$, $m_s = m_s.parent$;
[3] **while** $m_t \neq \phi$ **do** $B=B+m_t$, $m_t = m_t.parent$;
[4] **while** $open \neq \phi$ **do**
[5] $v_c = \min_{d[v]}\{v | v \in open\}$;
[6] $open = open - \{v_c\}$, $closed = closed + \{v_c\}$;
[7] **if** $v_c = t$ **then break**;
[8] **for each** arc $a_{v_c v}$ **in** B emanating from v_c **do**
[9] **if** $d[v] > d[v_c] + w(a_{v_c v})$ **then**
[10] $d[v] = d[v_c] + w(a_{v_c v})$;

[11] **if** $v \notin open$ **then** $open = open + \{v\}$;
end if; **end for**; **end while**;
[12] **if** $d[t] = \infty$ **then** $\pi(t) = null$;
[13] **return** $\pi(t)$, and algorithm terminates.

$NTSP$ algorithm computes the transitive closure at step [2], [3], and B contains the transitive closures of both macro-node $\sigma(s)$ and $\sigma(t)$. The searching procedure always be constrained in B, satisfying the requirements of Theorem 1, so at least one shortest path from s to t is sure to exist in the B. As the label-marked searching technology is adopted, the algorithm will be able to find the shortest path from s to t in B that is also the shortest path in original network according to Theorem 1.

4 Experimental Results

4.1 Network-Tree Based on Lattice Structures

We present the following procedure to construct the network-tree N_Tree based on lattice structures, with $k(k \geq 2)$ being the level number of the network-tree, l_h and b_h are integers, $1 \leq h \leq k$, and $w_1 < w_2 < \cdots < w_{k-1} < w_k$.

I. Construct a $\prod_{h=1}^{k} l_h \times \prod_{h=1}^{k} b_h$ rectangular lattice, and place the lattice in the first quadrant of the Cartesian plane, with the lower left corner corresponding to $(0, 0)$, and the lattice and Cartesian coordinates match;
II. The nodes of the lattice correspond nodes of the network; sides of the lattice correspond to the bi-directional arcs of the network. The following sub-procedure constructs the network-tree N_Tree and assigns weight value to the arcs:

Step1: Abstract the nodes which abscissa can be divided exactly by $\prod_{i=2}^{k} l_i$ or which ordinate can be divided by $\prod_{i=2}^{k} b_i$, and all the arcs between these nodes to construct the root macro-node m_r of N_Tree. Let the weight of all these arcs be w_1. The rest network is divided into $l_1 \times b_1$ sub-networks by m_r, and let the parent macro-node of these sub-networks be m_r, and process all these sub-networks as step2, with initially $h = 2$.

Step2: For each sub-network N_{sub}, if $h = k$, then let the whole sub-network N_{sub} be a macro-node m of N_Tree. Otherwise, in the sub-network N_{sub}, abstract the nodes which abscissa can be divided exactly by $\prod_{i=h+1}^{k} l_i$ or which ordinate can be divided by $\prod_{i=h+1}^{k} l_i$, and all the arcs between these nodes to construct a macro-node m. Let the parent macro-node of N_{sub} be the parent macro-node of m, and the weight of all the arcs in m be w_h. When $h \neq k$, the sub-network $N_{sub} - \{m\}$ is divided into $l_h \times b_h$ smaller sub-networks, and let the parent macro-node of all these new smaller sub-networks be m. Until all the sub-networks are processed, then continue to the step3.

Step3: Let $h = h + 1$, and if $h > k$, then go to the step4. Otherwise, go to step2 to process all the new smaller sub-networks.

Step4: The network-tree construction is completed.

For example, shown as Fig. 1, we create a network-tree based on lattice structures with $k = 3$, $l_1 = b_1 = 2$, $l_2 = b_2 = 3$, $l_3 = b_3 = 4$, $w_1 = 2$, $w_2 = 3$ and $w_3 = 5$.

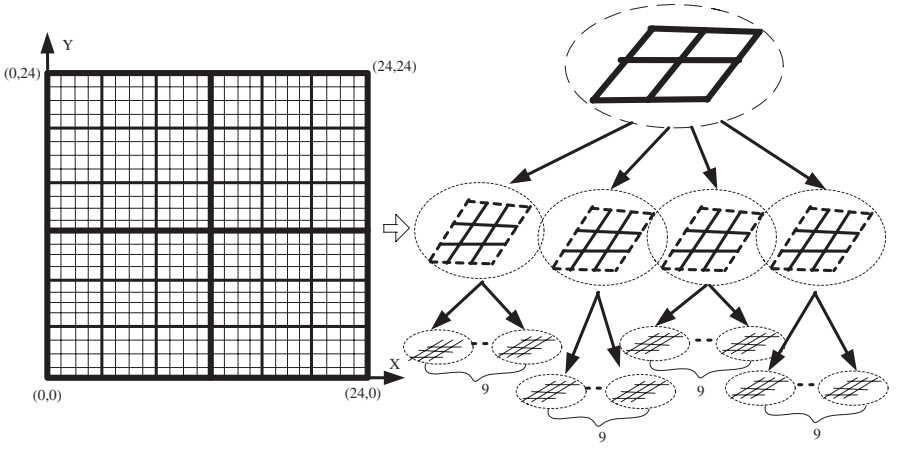

Fig. 1. The example of network-tree construction based on lattice structures

4.2 Experimental Results

We will create a large number of origin and destination nodes (OD pairs) randomly, and the computing efficiency will be measured by the average calculation time, that is $\overline{T} = \frac{\sum_{i=1}^{n} T_i}{n}$, with n being the number of OD pairs.

To better observe the algorithms' computing performance when calculating OD pairs of different distance, we define the test distance of OD pair: Let the original node be $s(x_s, y_s)$ and the destination node be $t(x_t, y_t)$, We adopt the Manhattan values as the test distance of this OD pair, that is $|x_t - x_s| + |y_t - y_s|$.

Our experiment uses a Pentium IV personal computer running at 1.7 GHz with 256 MB RAM. The operating system is Windows98; the programming language is Visual C++ 6.0.

We present the following computing efficiency comparisons of $NTSP$, hierarchical encoded path query ($HEPQ$), and $Dijkstra$ algorithm.

Experiment 1. Comparison of $NTSP$, $HEPQ$, and $Dijkstra$ algorithm with two-level expanded network-tree under different test distances. The parameters are set as $k = 2$, $l_1 = b_1 = 6$, $l_2 = b_2 = 10$, $w_1 = 2$ and $w_2 = 5$, thus the total node number is 3721, the total arc number is 14876, results are demonstrated in Fig. 2.

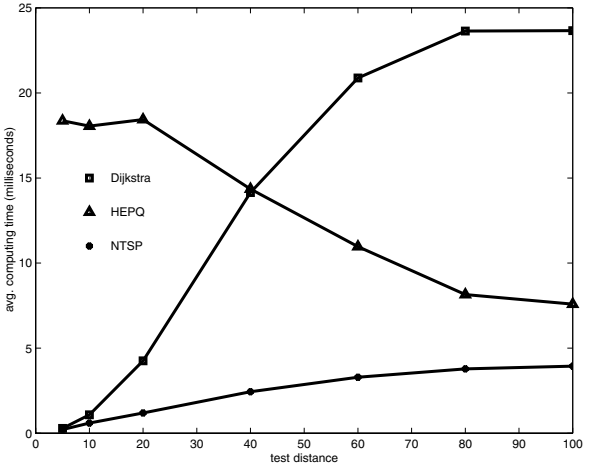

Fig. 2. Comparison of NTSP, HEPQ, and Dijkstra algorithm with two-level expanded network-tree

Experiment 2. Comparison of three *NTSP* algorithms and *Dijkstra* algorithm with three-level expanded network-tree under different test distances. The parameters are set as $k = 3$, $l_h = b_h = 6(1 \leq h \leq 3)$, $w_1 = 2$, $w_2 = 4$ and $w_3 = 7$, thus the total node number is 47089, the total arc number is 188348, results are demonstrated in Fig. 3.

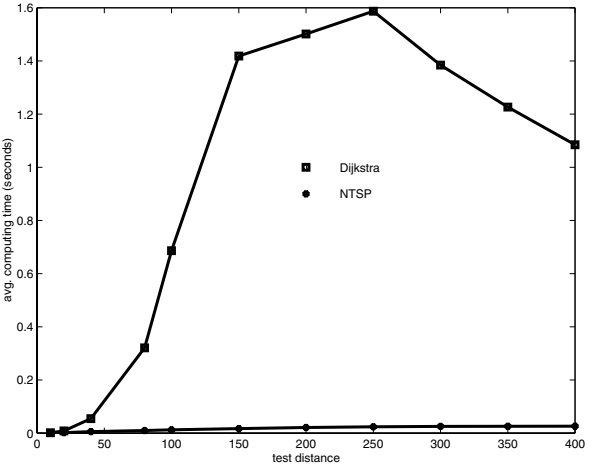

Fig. 3. Comparison of NTSP algorithms and Dijkstra algorithm with three-level expanded network-tree

Experiment 3. Comparison of $NTSP$ algorithms, $HEPQ$ and $Dijkstra$ algorithm under different network sizes. The parameters are set as $k = 2$, $w_1 = 2$ and $w_2 = 5$, and results are demonstrated in Table 1.

Table 1. The comparing result under different network sizes (seconds)

Size	$l_1 = b_1 = 4$ $l_2 = b_2 = 5$ Nodes:441 Arcs:1756	$l_1 = b_1 = 6$ $l_2 = b_2 = 10$ Nodes:3721 Arcs:14876	$l_1 = b_1 = 8$ $l_2 = b_2 = 10$ Nodes:6561 Arcs:26236	$l_1 = b_1 = 10$ $l_2 = b_2 = 11$ Nodes:12321 Arcs:49276	$l_1 = b_1 = 12$ $l_2 = b_2 = 12$ Nodes:21025 Arcs:84092	$l_1 = b_1 = 16$ $l_2 = b_2 = 16$ Nodes:66049 Arcs:264188
$Dijkstra$	0.000913	0.012562	0.028487	0.065932	0.155256	1.371372
$HEPQ$	0.000776	0.004755	0.049369	∞	∞	∞
$NTSP$	0.000413	0.002278	0.003710	0.006154	0.010132	0.026215

4.3 Interpretation of Results

The test results show the different computing performances of $NTSP$, $HEPQ$ and $Dijkstra$ algorithm: Experiment 1 shows that the $NTSP$ algorithm achieves higher computing efficiency than $HEPQ$ and $Dijkstra$ algorithm under all the test distances, especially with the network size increasing, such as Experiment 3, under the best case the $NTSP$ algorithm improved computing time by a factor of 50 over the $Dijkstra$ algorithm. The spatial complexity of the $HEPQ$ algorithm is very high; therefore, the algorithm can accelerate the computing speed to a certain degree when applying to small or general network (see Experiment 1), but if the storage is constrained, the computing performance decreases when the network size increases to a certain size. As Experiment 3 shows, when the network size increases to 12,321 nodes, the $HEPQ$ algorithm required more main memory than our test computer could provide, and the algorithm was unable to find a shortest path.

5 Conclusions

The network-tree model and NTSP algorithm provide a wholly new method to effectively implement the optimization of large-scale network. The advantages of network-tree model express not only at the great efficiency improvement of shortest path computing, but also at the network data partition according to the hierarchical characteristic and regional characteristic, which is very important for many definite application areas.

The network-tree model and NTSP algorithms not only can be apply to the path optimization of static network, but also can be expanded to the dynamic optimal path searching of time-dependent network. The algorithms have an excellent parallel structure, and are well suited to realizing parallel computations of the shortest path.

References

1. Cherkassky B.V., Goldberg A.V., Radzik T.: Shortest paths algorithms: Theory and experimental evaluation. Mathematical Programming, 1996, 73(2): 129–174
2. Ning Jing, Yun-Wu Wang, and Elke A. Rundensteiner: Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. IEEE Transactions on Knowledge and Data Engineering, 1998, 10(3): 409–432
3. A.Car, G.Taylor, C.Brunsdon. An analysis of the performance of a hierarchical wayfinding computational model using synthetic graphs. Computers, Environment and Urban Systems, 2001, 25: 69–88
4. J. Shapiro, J. Waxman, D. Nir: Level Graphs and Approximate Shortest Path Algorithms, Networks, 1992, 22: 691–717
5. Yu-Li Chou, H. Edwin Romeijn and Robert L. Smith: Approximating Shortest Paths in Large-Scale Networks with an Application to Intelligent Transportation Systems, INFORMS journal on Computing, 1998, 10 (2): 163–179
6. Chan-Kyoo Park, Kiseok Sung, Seungyong Doh, Soondal Park: Finding a path in the hierarchical road networks. 2001 IEEE Intelligent Transportation Systems Conference Proceedings-Oakland (CA) USA, August 25–29, 2001, pp. 936–942
7. A. Fetterer, S. Shekhar: A Performance Analysis of Hierarchical Shortest Path Algorithms. IEEE Proceedings of the International Conference on Tools with Artificial Intelligence, Nov 3–8, 1997, pp. 84–92