Dynamic Shortest Path Tree Update for Multiple Link State Decrements

Bin Xiao, Jiannong Cao
Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
csbxiao, csjcao@comp.polyu.edu.hk

Qingfeng Zhuge, Zili Shao, Edwin H.-M. Sha
Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083
qfzhuge, zxs015000, edsha@utdallas.edu

***Abstract*— Previous approaches for the Shortest Path Tree (SPT) dynamic update are mainly focused on the case of one link state change. Little work has been done to the problem of deriving a new SPT based on its old one for multiple link state decrements in a network that applies link-state routing protocols. The complexity of this problem comes from that there is no accurate boundary of nodes to be updated in an updating process and that multiple decrements can be accumulated. In this paper, two dynamic algorithms (*MaxR*, *MinD*) are proposed to reduce the times for node updating. Compared with other algorithms for the SPT update of multiple edge weight decrements, our algorithms yield less number of times for node updated during the dynamic update process. Such achievement is attained by the mechanism of part nodes updating in a branch on the SPT after a particular node selection from a built node list. Simulation results are given to show our improvements.**

I. Introduction

Present-day computer networks depend upon increasingly high speed information exchange. This means higher routing speeds are required in larger routing areas. In widely deployed Internet link-state-based routing protocols such as Open Shortest Path First (OSPF) and Intermediate System-to-Intermediate System (IS-IS), each router computes a Shortest Path Tree (SPT) with itself as the root evolved from the network topology. It is possible to construct on the SPT a routing table displaying routes with lower costs (such as delays) to each destination in a routing area (e.g., an OSPF area). A SPT is outdated whenever a network topology change happens. Such old SPT, and furthermore the routing table, need to be updated inside each router equipped with the OSPF protocol. Thus, efficient SPT update algorithms can greatly reduce the computation time in each router and high routing speed is maintained.

Dynamic SPT update approaches are proposed in recent years on network routing for path determination, with the central problem being to reduce costs, particularly computation times [1], [2], [3], [4]. The case of multiple link state changes deserves more attention for dynamic SPT update. In real networks, it is not worthwhile to update an old SPT whenever a small vibration of a link state occurs. It is possible that the modified weight quickly returns back to its previous value under the situation of unstable network traffic. Frequent SPT update due to one link state change will consume tremendous computation resource and degrade the routing performance. To avoid such resource waste, an adjustable schedule is set to configure SPT during an interval by milliseconds [5]. This throttling feature is successfully applied in the OSPF routing network. Within the millisecond intervals, the event of multiple edge weight changes may occur.

In order to derive a new SPT from an old one in the case of multiple link weight changes within a network topology, dynamic algorithms categorize those weight-changed links into two groups [6], [7], [8], one containing weight-decreased links and the other weight-increased links. Usually, to minimize the computation time the updating process handles the edge weight-decreased group first [9], [10], [7]. Compared to the update procedure on the group of weight-increased links, the SPT update with multiple link state decrements is more complicated for two reasons:

1) In a link state increments scenario, it is possible to accurately recognize the nodes to be updated in an old SPT before the dynamic updating takes place. However, we can not know such group of nodes to handle multiple link state decrements in advance.
2) Given multiple edge weight decrements in a graph, it is hard to calculate a maximum bound of steps to refresh a single node to its final state in the new SPT [10]. In other words, a node could be updated many times during a dynamic update process. Node update redundancy is introduced because the optimal solution is to update each outdated node by once.

In this paper, two dynamic SPT update algorithms (*MinD*, *MaxR*) are proposed to handle multiple edge weight decrements in a network equipped with the link-state routing protocol. The two presented algorithms in this paper are based on different perspectives in the dynamic update process. The node first updated in the *MinD* algorithm is the one that achieves the minimal shortest distance in the new SPT; while in the *MaxR* algorithm is the node that yields a temporary maximal decrement for its new shortest distance. Rather than updating the whole branch in [10], both algorithms update part nodes in a branch rooted at a selected node in the old SPT, which is realized by an introduced node list *M*. Thus, our algorithms generate less updating redundancy during the dynamic update

This work is partially supported by HK Polyu ICRG A-PF86 and CERG Polyu 5196/04E, and also by TI University Program, NSF ETA 0103709, Texas ARP 009741-0028-2001 and NSF CCR-0309461

process. Simulation results show that the redundant factor can be reduced up to 75%.

In the next section, definitions and notations to be used in this paper are given, then the new *MaxR* and *MinD* algorithms are described in detail. In Section III, we present the experimental results. Concluding remarks are given in Section IV.

II. Dynamic Algorithms

Dynamic algorithms, i.e., *MinD* and *MaxR*, are presented in this Section. They aim at reducing the times for a node to be updated to its final shortest distance during a dynamic update procedure. Although these algorithms can not totally remove the update redundancy, they minimize the repeated times as much as possible in the complicated situation of multiple link state decrements. In the meantime, they maintain the efficiency of branch nodes updating by extending such mechanism as far as possible. How far of the update for branch nodes is determined by the following introduced node list M.

A. Definitions and Notations

Definitions and notations used in the rest of the paper are given below. Let $G = (V, E, w)$ denote a directed graph where V is the set of nodes, E is the set of edges and w represents the weight of each edge in E in the graph. G contains no negative-weight edges. Let $S(G) \in V$ denote the source node in the graph G. We use $w(e)$ to denote the weight for each directed edge $e \in E$. Given an edge $e : i \rightarrow j$, i is the source node and j is the end node of e. Let $S(e)$ be the source node of edge e and $E(e)$ be the end node. The length or distance of a directed path is the sum of weights of all edges on the path.

The static method to derive a shortest path tree (SPT) can follow the Dijkstra algorithm. Every node in the SPT can be reached from $S(G)$ through a unique directed path only with edges on the SPT. In the following proposed algorithms, a temporary "SPT" (a tree with no meaning of shortest paths to nodes to be updated) with $S(G)$ as the root is maintained. When the update process is terminated, the temporary tree becomes the final new SPT. For an edge in a given SPT, node i is the parent of node j if i is the source node and j is the end node of this edge. We define the following notations for node i: $P(i)$ is its parent node and $D(i)$ is its shortest distance. The value of $D(i)$ is calculated by the length of the unique path from the source node $S(G)$ to node i in a SPT. For an edge e with a new weight $w(e)$, let $d = D(i) + w(e) - D(j)$, where d represents the increased value to node j if the shortest path to node j via the shortest path from $S(G)$ to node i and the edge e. The value of d can be negative if the weight of edge e becomes smaller. Given a node set $N \subseteq V$ in G, the $Source\_part\{N\}$ is defined as an edge set that contains any edge $e \in E$ with $S(e) \in N$ while $E(e) \notin N$.

The introduced node set M has elements in the form of $\{v, (d_1, p, d_2)\}$, where v denotes the node itself, d_1 is the decreased value for the shortest distance of node v if the shortest path still follows its previous parent node, and d_2 points out the shortest distance decrement on node v for its new shortest path via the new parent node p. Both d_1 and d_2 are non-positive value since we only consider link state decrements of a network topology change. Only a negative value can trigger node v to reach a smaller shortest distance and thus will be explored during below algorithms. The new shortest distance for node v is derived by $D(v) = D(v) + min(d_1, d_2)$ during one update step. d_2 is kept to be 0 and its relevant parent node denoted as -1 in M unless $d_2 < d_1$. The reason is that we maintain the path in the old SPT as much as possible while yield the smallest new shortest distance for each node.

Enqueue$(M, \{v, (d_1', p', d_2')\})$**:**
if $v.floating$ = False **then** $v.floating = True$
  $\{v, (d_1', p', d_2')\}$ is added to M,
else $M.v.d_1 = M.v.d_1 + M.v.d_1'$
  if $M.v.d_2' < M.v.d_1$ and $M.v.d_2' < M.v.d_2$ **then**
    $M.v.d_2 = M.v.d_2'$, $M.v.p = M.v.p'$
  if $M.v.d_1 \leq M.v.d_2$ **then**
    $M.v.d_2 = 0$, $M.v.p = -1$

When an incoming edge can make node v reach a smaller shortest distance, node v is enqueued to the introduced node set M. The program above proceeds. Along with node v, a boolean function $v.floating$ of *True* shows its belonging to M ($v.floating = True$). Otherwise, the value of *False* is assigned to v. At the end of following dynamic update algorithms, each node must be denoted as *False* to demonstrate that none of them is in M and the current maintained tree represents the final new SPT. Whenever a new node v is added to M, $v.floating$ is automatically set to be *True*. If node v is already inside M, the decrements following the old path is accumulated by $M.v.d_1 = M.v.d_1 + M.v.d_1'$. Furthermore, if $M.v.d_2'$ is smaller than both $M.v.d_1$ and $M.v.d_2$, which means the new explored incoming edge can yield even smaller shortest distance to node v than its previous value, this incoming edge is recorded for v in M. Therefore, we know that for a node v kept in M, its $M.v.d_2$ value is either 0 or is smaller than $M.v.d_1$.

The descendants of a node i are all nodes that are reachable from i only through edges in a given SPT. During the proposed dynamic update processes, only part of nodes of a branch in a SPT will be updated. In this paper given a node v, a node set $N_{fixed}(v)$ is introduced as a node set that consists of part of descendants from v (including node v itself). The $N_{floating}(v)$ is defined as a node set that contains all the direct children from nodes in $N_{fixed}(v)$ within the SPT. A tree is consecutively constructed during below dynamic algorithms deployed in the case of multiple edge weights decreased, from its old SPT to a temporary SPT till the final constructed SPT. The updated SPT is achieved when algorithms terminate.

Definition II.1
If $v \in V$ is a node in graph $G = (V, E, w)$, given a SPT from G originated from a source node $S(G)$, an available node list M and suppose that the decreased value of the shortest distance of node v is d, let $N_{fixed}(v)$ be a node set containing node v. Some other nodes (e.g., v') can be recursively added to this node set iff:

1) $P(v') \in N_{fixed}(v)$
2) $v' \notin M$, *or* $v' \in M$ *but* $M.v'.d_1 = 0$ *and* $M.v'.d_2 \geq d$.

Definition II.2
If $v \in V$ is a node in graph $G = (V,E,w)$, given a SPT from G originated from a source node $S(G)$, and a node set $N_{fixed}(v)$, let $N_{floating}(v)$ be a node set where a node is not in $N_{fixed}(v)$ but a direct child of any node in $N_{fixed}(v)$ following the SPT.

B. MinD Algorithm

To dynamically update an old SPT to a new one, *MinD* algorithm improve the computational efficiency by the sequence of node extraction from a node set M, and the subsequent adjustment of part nodes on a branch of a temporarily maintained SPT. The sequence of node extraction from M is defined in the $Find_{MinD}(M)$ function below. Given a node list M, this function returns a node that has the present smallest shortest distance. The present shortest distance for each node $v \in M$ is calculated by $D(v) + min(M.v.d_1, M.v.d_2)$. After the selection of node v from M, the node to be updated is in the node set $N_{fixed}(v)$, which is denoted in Definition II.1.

Node Find$_{MinD}(M)$:
$\{v,(d_1,p,d_2)\} \leftarrow$ the first node in M
$d = D(v) + ((d_2)?d_2 : d_1)$
for $i = 2,\ldots,n$ **do**
  $\{v',(d_1',p',d_2')\} \leftarrow$ the ith node in M
  if $D(v') + ((d_2')?d_2' : d_1') < d$ **then**
    $v = v'$, $d_1 = d_1'$, $p = p'$, $d_2 = d_2'$, $d = D(v') + ((d_2')?d_2' : d_1')$
Return $\{v,(d_1,p,d_2)\}$

The *MinD* algorithm is shown below. It aims at dynamically updating an old SPT to a new one by minimizing the redundant times of node adjustment. Before the running of this dynamic algorithm, we can derive the initial SPT by a static algorithm, such as the Dijkstra SPT algorithm. The node set M is initialized to be empty. Given a node $v \in V$, its $v.floating$ is set to be *False* indicating its non-existence in M. Whenever the update process terminates for a scenario of multiple edge weight decrements, the *MinD* algorithm waits at *Step 1* for another network topology change. Meanwhile, all nodes should be removed from M and thus every node's *floating* boolean property is set to be *False*.

Inside a network topology change, suppose that there are n edges with weight reduced. In *Step 1* after the edge weight updating, we identify those edges that can make their end nodes to reach smaller shortest distances. Given a weight decreased edge e_i, its end node $E(e_i)$ can have a distance $D(S(e_i)) + w(e_i)$ for the path via edge e_i and the route of the shortest path from the root node to node $S(e_i)$. If an edge e_i has the property to let $d < 0$ ($d = D(S(e_i)) + w(e_i) - D(E(e_i))$), its end node $E(e_i)$ should be added to the node list M. Once a node v is input to M, its $v.floating$ is placed to be *True*. If the weight decreased edge is on the old SPT, its relevant end node is enqueued to M only with d_1 changed. Otherwise, the node stored in M must contain the information of this incoming edge e, i.e., the source node of e is depicted as the parent and d_2 is replaced by d.

Step 2 is not ended whenever the node list M is not empty. The first node extracted from M is outlined in the $Find_{MinD}(M)$ function. It is possible for the extracted node v' with two incoming edges. Those two edges are accompanied with decreased value of d_1' and d_2' respectively. The edge with larger decrement will be selected to update all nodes in $N_{fixed}(v')$. If $d_2' < d_1'$, the edge (p',v') becomes a part of the temporary SPT by assigning p' as the new parent of node v'. In the meantime, node v' becomes one of children of node p'. The updated nodes are removed from M. When the shortest distances for nodes in $N_{fixed}(v')$ decrease, their outgoing edges need to be scanned in case any one of them can produce a reduction of the shortest distance on its end node. The edge with $d' < 0$ should be added to the node list M too. In order to reduce update redundancy for nodes in $N_{floating}(v')$ (also in M), they are not changed but with their incoming edges on the current SPT denoted by another further decrement of d.

MinD Algorithm:
Initialization:
  From $G = (V,E,w) \rightarrow SPT$
  $M = \phi$, $\forall v \in V$, $v.floating = False$
  Step 1: wait until n edges e_i, $i = 1,\ldots,n$, with weight decreased by d_i respectively.
  for $i = 1$ to n **do**
    $w(e_i) = w(e_i) + d_i$
    $d = D(S(e_i)) + w(e_i) - D(E(e_i))$
    if $d < 0$ **then**
      if $e_i \in SPT$ **then** $Enqueue(M,\{E(e_i),(d,-1,0)\})$
      else $Enqueue(M,\{E(e_i),(0,S(e_i),d)\})$
  go to **Step 2**
  Step 2:
  while $M \neq \phi$ **do**
    $\{v',(d_1',p',d_2')\} \leftarrow Find_{MinD}(M)$, $v'.floating = False$, remove node v' from M
    if $d_2' = 0$ **then** $d = d_1'$
    else $d = d_2'$, $P(v') = p'$
    $\forall v \in N_{fixed}(v'), D(v) = D(v) + d$
    if $v \in M$ **then**
      $v.floating = False$, v is removed from M
    $\forall e \in Source\_part\{N_{fixed}(v')\}$, select the minimum increased value for each node $E(e)$
    $d' = D(S(e)) + w(e) - D(E(e))$
    if $d' < 0$ **then** $Enqueue(M,\{E(e),(0,S(e),d')\})$
    $\forall v \in N_{floating}(v')$, $Enqueue(M,\{v,(d,-1,0)\})$
  go to **Step 1**

C. MaxR Algorithm

The *MaxR* algorithm is designed to reduce the computation time to re-build a SPT dynamically from an old one in a graph that has multiple edge weight decrements. Its efficiency is achieved by the specialized node extraction sequence from a node list M, while maintaining the same procedure of subsequent part nodes updating as in the *MinD* algorithm. The extraction mechanism to find a node from M is defined below by the function $Find_{MaxR}(M)$. Instead of the first extracted node that has the minimum shortest distance using the *MinD* algorithm, the returned node produces the maximum reduction on its shortest distance. The maximum reduction is the smaller one either from d_1 or from d_2. After a node is removed from the node set M, the following updating procedure in the *MaxR* algorithm is the same as in the *MinD* algorithm. Thus, the pseudocode for the *MaxR* algorithm is the same as the *MinD*

algorithm in above subsection except that the $Find_{MinD}(M)$ function is replaced by the $Find_{MaxR}(M)$ function.

Node Find$_{MaxR}(M)$**:**
$\{v,(d_1,p,d_2)\} \leftarrow$ the first node in M
$d = ((d_2)?d_2 : d_1)$
for $i = 2,\ldots,n$ **do**
  $\{v',(d_1',p',d_2')\} \leftarrow$ the ith node in M
  if $((d_2')?d_2' : d_1') < d$ **then**
    $v = v'$, $d_1 = d_1'$, $p = p'$, $d_2 = d_2'$, $d = ((d_2')?d_2' : d_1')$
Return $\{v,(d_1,p,d_2)\}$

III. Experimental Results

In this section, we compare the computation time for the *MaxR* and *MinD* algorithms and the SPT update algorithm in [10] to solve the problem of dynamic updating the old SPT to a new one when multiple edge weight decrements occur in a graph. All these algorithms concentrate on edges with $d < 0$. Such kind of edges allows their end nodes to attain smaller shortest distance. To contain all those end nodes, each algorithm constructs a node list. The *MaxR* algorithm and the SPT update algorithm described in [10] differ in the ways they update nodes. The *MinD* algorithm has distinct mechanisms for updating and extracting nodes. When the node list is empty and no more nodes need be inserted, these three algorithms terminate, producing a new SPT.

A. Event Generation

A simulation was conducted using a randomly generated network similar to the random topology generator (RTG) software. The topology of a generated network can be totally different in each time. In each simulation when a network graph was created, 5 continuous cases were tested. In one case we randomly chose w edges that had new smaller edge weight. The number (w) of edges with new weight is proportional to the network size N. The simulated network size is increased from 20 to 100.

Whenever there is an update to an old SPT because of edge weight decrements, we keep track of the computation time of the proposed *MaxR*, *MinD* algorithms and the SPT [10] algorithm. For an update process, we compare the observed complexity of algorithms for the main update process, which includes:

- Adding and removing nodes to a constructed node set, which is the node set M for the *MaxR* and *MinD* algorithm, and Q for the SPT algorithm [10].
- Extracting an element from the node list M (or Q).
- Updating a node to its final state.

B. Performance Results

We verify different algorithms by randomly selected edges and generated weight changes. For each network topology modification, a new SPT must be derived. In Table I, the simulation results are listed for the SPT algorithm in [10], *MaxR* and *MinD* algorithms proposed in this paper. The simulation is based on a network with 100 nodes randomly distributed in a 300×300 grid area. To update an old SPT to a new SPT, w ($w = 13$) edges are with weight decreased. After the new SPT is derived by different algorithms, another four times of the same dynamic updating process proceed. The calculation results are shown in Table I according to those 5 continuous network topology changes. In Column "# enqueued", we show the number of nodes enqueued by different algorithms, while the data in Column "# searching" means the computation time to search the appropriate element in the created node list. All algorithms can not completely remove the node update redundancy. In other words, some nodes need to be updated more than once till their final states in the new SPT. The accumulation of shortest distance reduction in the case of multiple edge weight decrements makes it very hard to directly update each node to its destination. The data in the latter three columns illustrates such redundancy. The data in Column "# once" is the simulation result for the number of nodes that are exactly updated once to its final shortest distance by different algorithms. While data in Column "# twice" and "# more" means the number of nodes that must be updated twice, or more than twice respectively to their value in the new SPT.

Table I shows that, compared to the SPT algorithm in [10], the *MaxR* and *MinD* algorithms do not significantly reduce the computation time involved in adding a node to a built node list and then extracting it. The *MaxR* and *MinD* algorithms do however improved is the time required for a node to reach its final result during dynamic updating. Of updated 370 nodes, 36.8% ($\frac{118+18}{234+118+18} = \frac{136}{370}$) had to be updated twice or more using the algorithm in [10]. This percentage has been reduced to 31.4% by the *MaxR* algorithm. Using the *MinD* algorithm, it was necessary to update only 9.2% of the nodes twice, and none of them more than twice. When the *MinD* algorithm was used, the number of nodes requiring more than one update was reduced by up to 75% ($\frac{136-34}{136}$).

The accumulated times of nodes to be updated once, twice and more than twice by different algorithms are shown in Figure 1, 2 and 3 respectively. Because the calculation is accumulated, all curves move up when more nodes scattered in a simulated network area. The *MinD* algorithm demonstrates its efficiency by its curve always above curves of *MaxR* and SPT algorithm in Figure 1, while always lower than others in Figure 2 and 3. Thus, the *MinD* algorithm performs best by increasing the number of nodes that are updated only once. It is obvious that the SPT algorithm in [10] causes an increase in the computation time since it yields more redundant times of node update. Our proposed algorithms improve the efficiency of dynamic SPT updating by greatly reducing the time required to temporarily alter a node and to explore its connecting edges.

IV. Conclusion

In this paper, new efficient dynamic algorithms, i.e., *MaxR*, *MinD*, are presented to solve the problem of updating an outdated SPT for the case of multiple edge weight decrements. This problem is quite complicated compared to the SPT update problem for one edge weight change. It is very hard to completely remove the redundancy of node updating. Compared with the SPT algorithms in [10], the proposed algorithms greatly reduce the repeated times of node update.

Algo.	# enqueued	# searching	# once	# twice	# more
SPT [10]	140	998	234	118	18
MaxR	142	1,013	254	113	3
MinD	133	956	336	34	0

TABLE I

THE PERFORMANCE COMPARISON AMONG DIFFERENT ALGORITHMS IN TERMS OF NODE SIZE 100.

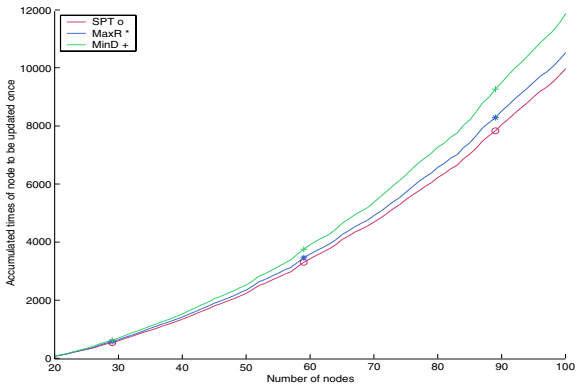

Fig. 1. Accumulated times of nodes to be updated once by different algorithms.

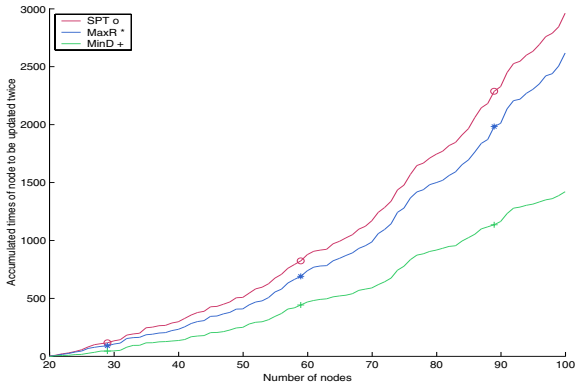

Fig. 2. Accumulated times of nodes to be updated twice by different algorithms.

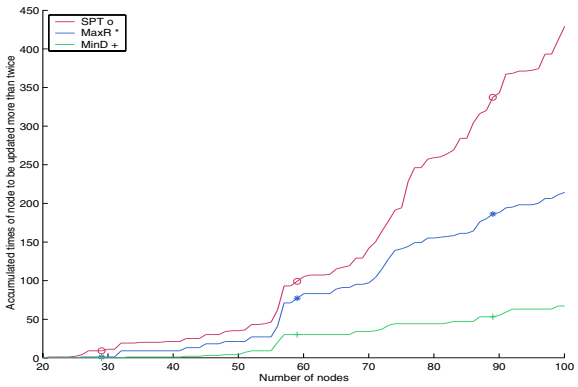

Fig. 3. Accumulated times of nodes to be updated more than twice by different algorithms.

Recent dynamic algorithms to update an old SPT normally adopt a node list to substantiate the updating process. How to construct, destruct a node list containing all relevant information of edges (with $d < 0$) and how to update the old SPT after one node's extraction from the node list play key roles of the algorithm performance. Previous methods to update only one node after one node's extraction is too tedious and inefficient, while the approach to update a branch introduces node update redundancy. To avoid such disadvantages, the proposed algorithms follow the mechanism by updating part nodes on a branch after the extraction of a node from a constructed node list. Thus, as demonstrated by the experimental results, the *MaxR* and *MinD* algorithms improve the update efficiency by tremendously decreasing the number of times that a node need be temporarily altered, and furthermore have less computation cost.

REFERENCES

[1] D. Chakraborty, G. Chakraborty, and N. Shiratori, "A dynamic multicast routing satisfying multiple QoS constraints," *International Journal of Network Management*, vol. 13, pp. 321–335, 2003.

[2] S.-J. Yang, "Design issues and performance improvements in routing strategy on the internet workflow," *International Journal of Network Management*, vol. 13, pp. 359–374, 2003.

[3] S. Zhu and G. M. Huang, "A new parallel and distributed shortest path algorithm for hierarchically clustered data networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, pp. 841–855, Sept. 1998.

[4] B. Zhang and H. Mouftah, "A destination-driven shortest path tree algorithm," in *Proc. IEEE International Conference on Communications*, vol. 4, pp. 2258–2262, April 2002.

[5] "Information About OSPF SPF Throttling." http://www.cisco.com/univercd/cc/td/doc/product/software/ios122s/122snwft/release/122s14/fsspftrl.htm#1027177.

[6] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully dynamic shortest paths in digraphs with arbitrary arc weights," Tech. Rep. ALCOMFT-TR-01-31 of the EC-IST Project ALCOM-FT, March 2001.

[7] B. Xiao, Q. Zhuge, and E. H.-M. Sha, "Efficient algorithms for dynamic update of shortest path tree in networking," *Journal of Computers and Their Applications*, vol. 11, pp. 60–75, March 2004.

[8] B. Xiao, Q. Zhuge, and E. H.-M. Sha, "Minimum dynamic update for shortest path tree construction," in *Proceedings of the GLOBECOM'01*, pp. 126–130, Nov. 2001.

[9] P.Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Trans. Networking*, vol. 8, pp. 734–746, Dec. 2000.

[10] P.Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic SPT algorithm based on a ball-and-string model," *IEEE/ACM Trans. Networking*, vol. 9, pp. 706–718, Dec. 2001.