Dynamic Update of Shortest Path Tree in OSPF

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, USA
{qfzhuge, zxs015000, edsha}@utdallas.edu

Abstract

The Shortest Path Tree (SPT) construction is a critical issue to the high performance routing in an interior network using link state protocols, such as Open Shortest Path First (OSPF) and IS-IS. In this paper, we propose a new efficient algorithm for dynamic SPT update to avoid the disadvantages (e.g. redundant computation) caused by static SPT update algorithms. The new algorithm is based on the understanding of the update procedure to reduce redundancy. Only significantly elements that contribute to the construction of new SPT from the old one will be focused on. The efficiency of our algorithm is improved because it only pay attention to the edges really count for the update process. The running time for the proposed algorithm is maximum reduced, which is shown through experimental results. Furthermore, our algorithm can be easily generalized to solve the SPT updating problem in a graph with negative weight edges and applied to the scenario of multiple edge weight changes.

1 Introduction

In today's computer networks, higher speed information exchange becomes more preferable. Thus it requires the routing speed to be higher than ever while the routing area becomes larger. For link state based routing protocols that are widely deployed in Internet, such as Open Shortest Path First (OSPF) [1, 2] and Intermediate System-to-Intermediate System (IS-IS) [3, 4], each router computes a Shortest Path Tree (SPT) with itself as the root evolved from the network topology. A routing table can be constructed on the SPT and points out a route with less cost (such as delay) to each destination in a routing area (e.g. an OSPF area). Therefore, the efficient SPT building along with the topology change in a network area will directly benefit the routing speed for packet delivery inside it.

The SPT can be constructed by the well-known static algorithms, such as the Dijkstra algorithm [5]. However, the well-studied Dijkstra algorithm becomes very inefficient when only a small part of the SPT needs to be updated for link state changes in a network. This is because several link changes results in re-computation of the whole tree for every router, and involves entries updating in its routing table subsequently. For most cases, the new SPT for a router shows little modification compared with its old one or no difference at all due to smaller number of link state changes in a large size routing area. The method conducted by the static algorithm for the SPT update incurs a lot of unnecessary computation and routing table entry updates. Furthermore, for a particular route there may exist some undesirable traffic load fluctuation because of the complete reconstruction of a new SPT [6]. Thus, it is crucial that algorithms for dynamically updating SPT are introduced to handle link state changes in a network efficiently.

The problem to derive the shortest path is a subject of continual research interest [7, 8, 9]. In order to achieve less computation time, the updating process to separate *weight-increase* operations and *weight-decrease* operations is proposed for the dynamic SPT updates [10]. Previous work [11, 10] on dynamic SPT update reduces the computation time compared to the static methods. However, their algorithms still have some redundant computations. In [12], an algorithm based on a ball and string model where an edge weight change corresponds to the length change of the string is presented. However, there exists some redundant computation because of the update process with unnecessary edge information stored in a queue.

A new improved algorithm is proposed in this paper based on the analysis of the probability of edges contributed to the construction of the new SPT. The number of edges considered in the new algorithm is far less than any other algorithms in the literature. The proposed algorithm not only reduces the computational complexity required to update an old SPT, but also maintains the routing table stability by keeping the topology of an existing SPT as much as possible. Although it is focused on the change of one

Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)
1087-4089/04 $ 20.00 © 2004 IEEE

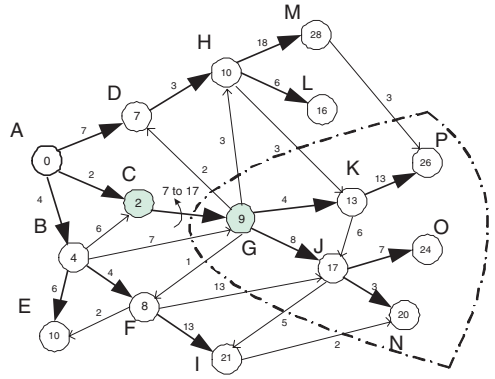

Figure 1. The weight of edge (C, G) increased from 7 to 17.

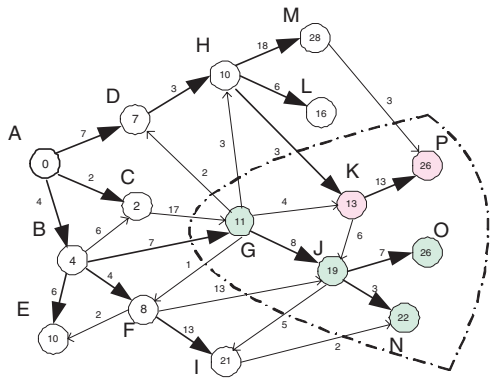

Figure 2. The new SPT for the weight of edge (C, G) increased from 7 to 17.

edge, to apply it to multiple link weight changes can be easily attained. The proposed algorithm is general enough to solve the SPT updating problem in a graph with negative weight edges. The experimental results show that by applying the presented algorithm in this paper, the number of edges put into a queue is reduced up to 31.5%, and the time of searching for the edge with the minimum value from the same queue is reduced up to 27.6% compared against the algorithm in [12].

In Section 2, an example is given to illustrate the redundant computation of previous work and to show some underlying insights. In Section 3, some definitions and notations to be used in this paper are presented, then the new dynamic algorithm is described in detail. The proposed algorithm can be applied to a graph with negative weight edges and is analyzed in Section 4. In Section 5, we present the experimental results. Concluding remarks are given in Section 6.

2 Example

An example is given for the case of one edge weight increased in Figure 1. The whole graph represents a network topology. The node means a router and the weight of one link between two nodes denotes the link state cost (such as network traffic delay) between different routers. The existing SPT for the graph with root on A is made up with all directed bold edges. The shortest distance for each node is the summation of the weight of all edges on the shortest path, which is started at node A and through bold edges till the node itself. The number inside every node represents the shortest distance from the source node A based on the given graph.

When the weight of edge from node C to G increases from 7 to 17, the SPT must be rebuilt. The new SPT is shown in Figure 2. The shortest paths for nodes outside of the area encircled with dotted lines can be the same in both the old and new SPT. This is because the shortest distance for a node can not reach smaller because of the weight of an edge increased. For all nodes encircled by the dotted line, their shortest distances will be either increased by 10 if their shortest paths follow the ways in the old SPT, or increased not much (less than 10) by choosing some alternative paths through nodes outside of the dotted lines. For each node inside the area encircled by the dotted line, i.e. G, J, K, N, P, there may exist several incoming edges. Each incoming edge can make its end node by a new distance from the source node A. Only the edge with the smallest increment to its end node (such as G, J, K, N, P), should be examined during the update process to satisfy the shortest distance property.

In Table 1, nodes that need to be updated are listed in the first row. The edge in the second row indicates the new path to its end node with the smallest increment compared with the old shortest distance. And the third row is the corresponding increased value. For example, if the new shortest distance path to node P goes through edge (M, P), the new distance would be $28 + 3 = 31$. That is an addition of 5 compared to the old distance 26. However, with another route to node P as in Figure 2, the new shortest distance from the source node A can remain 26. That means the edge (M, P) doesn't have any contribution to the construction of new SPT, which is shown in the fourth row of the Table 1. In other words, if an edge is included in the new SPT, that edge is denoted by Yes. No shows no contribution to the new SPT by the corresponding edge. The *Significant Edge* is defined in this paper as the edge only in the new SPT (not in the old SPT). Among all these nodes (G, J, K, N, O, P), node K can have the new smallest increment following the incoming edge (H, K) by 0. For all descendants of node K in the old SPT, they all can reach the new shortest distance by the path through node k. Thus,

Node	G	J	K	N	P
Incoming edge	(B, G)	(F, J)	(H, K)	(I, N)	(M, P)
Increased value	2	4	0	3	5
Contribution	Yes	No	Yes	No	No

Table 1. Edges which may significantly change the structure for the new SPT.

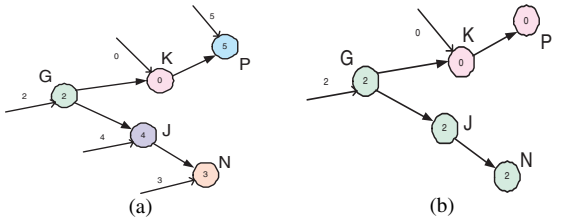

Figure 3. (a) Nodes with one incoming edge to denote the smallest increments; (b) The useful edges to construct the new SPT.

node K and P are updated once with route through the edge (H, K) with no addition to the shortest distance from root A. Node G, J, N, O can be updated through edge (B, G) for new shortest distance increased by 2.

In [12], all edges in Table 1 will be added to a queue Q. Q is an edge list that includes some edges with related information. After that, the update process will select the edge with the minimum increased value (by the same definition as the value for edges shown in the third row in Table 1) from the edge list Q and do some modifications, such as updating the SPT and extracting some edges from Q. The process does not end until the edge set Q is null. The nodes to be updated and their related edges are shown in Figure 3(a). Every incoming edge has a cost to show the increment to the shortest distance of its end node if the shortest path through the edge. The number in the node is the smallest increased value among all its incoming edges. From Table 1 we know that only a small part of edges will contribute to the construction of the new SPT. Some of them have never been used, such as edges $(M, P), (F, J)$ and (I, N). Thus, we do not need to put them into the edge set Q first and remove them later. If we only keep the incoming edge on condition that its increased value is smaller than the ones of all its ancestor nodes, the new graph can be looked like Figure 3(b). The number in the node is the smallest one either from the increments by its incoming edges, or from the value of its ancestors. The graph in Figure 3(b) from the graph in Figure 3(a) can be easily calculated by a Depth-First-Search sequence, which is a comparison of its parent's value and the smallest increased value among its all incoming edges. If the edge list Q only includes incoming edges as in Figure 3(b) (only 2 compared with 5 in Figure 3(a)), the computation time for adding edges to Q, searching the edge with the minimum increased value from Q, and extracting some related edges from Q will be greatly reduced.

3 Dynamic Algorithm

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 cycles. Let $S(G) \in V$ denote the source node in the graph G.

We use $w(e)$ to denote the weight of edge e 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 $E(e)$ be the end node of edge e and $S(e)$ be the source node. $w'(e)$ is used to show the new weight of edge e.

In the proposed algorithm, a temporary SPT with $S(G)$ as the root is maintained. When the update process is terminated, the temporary tree becomes the final new SPT. Node i is the parent node of node j if i is the source node and j is the end node of an edge in the SPT. We define the following notations for node i: $P(i)$ is its parent node and $D(i)$ is its shortest distance in the SPT, which is the length from the source node to node i. For an edge e with a new weight $w'(e)$, let $d = D(i) + w'(e) - D(j)$, where d represents the increment value to node j if the shortest path to node j through the edge e. Sometimes d can be negative if the weight of edge e becomes smaller.

The descendants of a node i are all nodes that are reachable from i only through edges in the SPT. We use $des(i)$ to denote the subset that includes i and all its descendants in the SPT. If we have a node set N, then Source_part$\{N\}$ $= \{ e |\ S(e) \in N,\ E(e) \notin N \}$ and End_part$\{N\}$ $= \{ e |\ E(e) \in N,\ S(e) \notin N \}$.

3.1 Algorithm Specification

In the new algorithm, M denotes a node set for the case of the weight of an edge increased. Every node to be updated is included in M. For a node $v \in M$, there is an increased value with this node, which is $M.v.inc$. Suppose node i is the parent node for node v in the outdated SPT. $M.v.inc$ is the smallest value among the increments of all its incoming edges and $M.i.inc$. The new algorithm also maintains an edge list Q that temporarily stores a subset of edges together with the corresponding attribute min_inc. In Q, each element is depicted by the configuration $\{e, min\_inc\}$, where e denotes an edge including its

source/end node, min_inc denotes the increased value to the shortest distance of node $E(e)$ given that the new shortest distance path goes through the edge e.

Whenever an edge e needs to be put into edge list Q, instruction $Enqueue(Q, \{e, min\_inc\})$ is executed. If the end node of edge e is already in Q, the new element $\{e, min\_inc\}$ will replace the old one when the new min_inc is a smaller value. In our new proposed algorithm, when an edge weight becomes larger, that new element $\{e, min\_inc\}$ will always replace the old one since the maintained list M guarantees that the new min_inc is smaller. The instruction $Extract(Q)$ selects the element with the smallest min_inc and removes it from Q. If there are two or more elements with the same smallest min_inc, we arbitrarily choose one.

The algorithm is described below. Two operations independently proceed to get the new SPT according to an edge weight increased or decreased separately. In the case of weight increment, the node list M is introduced, which will greatly reduce the computation time to add elements, remove elements and search for the minimum min_inc from Q. Whenever the update process is terminated for one iteration, the algorithm waits at Step 2 for another edge weight modification.

Improved Algorithm:

Step 1: From $G = (V, E) \rightarrow SPT$

Step 2: wait until one edge $e : i \rightarrow j$ changed its weight from $w(e)$ to $w'(e)$
1. **if** $w'(e) > w(e)$ and $e \in SPT$ **then**
$d = w'(e) - w(e)$, Go to **Step 3**.
2. **else if** $w'(e) < w(e)$ and $D(i) + w'(e) < D(j)$ **then**
$d = D(i) + w'(e) - D(j)$, Go to **Step 4**.
3. **else** go to **Step 2**

Step 3: $/*$ when one edge weight increased $*/$
1. Initialize
$M \leftarrow j$, $M.j.inc = d$, $Enqueue(Q, \{e, d\})$,
for $\forall v \in des(j)$ by the sequence of DFS from node j in SPT,
from $\forall e$, $E(e) = v$ & $S(e) \notin des(j)$, select the minimum increased value
$min\_inc = D(S(e)) + w(e) - D(E(e))$
if $min\_inc < M.v.inc$ **then**
$Enqueue(Q, \{e, min\_inc\})$, $M.v.inc = min\_inc$
end if
for k is the direct child of v in SPT
$M \leftarrow k$, $M.k.inc = M.v.inc$
end for
end for
2. **while** $Q \neq \varnothing$
$\{e_1,\ min\_inc\} \leftarrow Extract(Q)$, $P(E(e_1)) = S(e_1)$,
$\forall v \in des(E(e_1)), D(v) = D(v) + min\_inc$,
Remove edges from Q, which only with end nodes belong to $des(E(e_1))$
Remove v from M
$\forall e \in Source\_part\{des(E(e_1))\}$ & $e \in End\_part\{M\}$,
$min\_inc = D(S(e)) + w(e) - D(E(e))$
if $min\_inc < M.E(e).inc$ **then**
$M.E(e).inc = min\_inc$, $Enqueue(Q, \{e, min\_inc\})$
end if
end while
3. Go to **Step2**

Step 4: $/*$ for one edge weight decreased $*/$
1. Initialize
$\forall v \in des(j),\ D(v) = D(v) + d,\qquad P(j) = i$
from $\forall e, S(e) \in des(j)$, select the minimum increased value for each node $E(e)$
$min\_inc = D(S(e)) + w(e) - D(E(e))$
if $min\_inc < 0$ **then**
$Enqueue(Q, \{e, min\_inc\})$
end if
2. **while** $Q \neq \varnothing$
$\{e_1,\ min\_inc\} \leftarrow Extract(Q)$, $P(E(e_1)) = S(e_1)$
$\forall v \in des(E(e_1)), D(v) = D(v) + min\_inc$
Remove edges from Q, which only with end nodes belong to $des(E(e_1))$
$\forall e \in Source\_part\{des(E(e_1))\}$, select the minimum increased value for each node $E(e)$
$min\_inc = D(S(e)) + w(e) - D(E(e))$
if $min\_inc < 0$ **then**
$Enqueue(Q, \{e, min\_inc\})$
end if
end while
3. Go to **Step2**

3.2 Multiple Link Weight Changes

The algorithm presented above doesn't discuss the case for multiple link weight changes. However, with just a few modifications to the above algorithm, this problem can be easily solved. It is required to classify edges into two groups in Step 2, group 1 is for case 1 and group 2 for case 2. First, a temporary SPT from an old SPT will be built after the dynamic update process for edge weight decrements in group 2. Then, the update process will continue to reach the final SPT for edges in group 1 based on the temporary SPT.

All edges in group 2 paired with their decreased value d are included in the edge list Q, which has the same definition as the Q in Step 4. Based on the old SPT and the Q, the temporary SPT can be built through the iteration procedure of the update process (the second step in Step 4).

For edges in group 1, if they are still in the temporary SPT, the algorithm will keep them in the edge group 1. Otherwise, they will be removed. The reason is that their end nodes already reach their smaller shortest distances from the source node with alternative paths in the graph G. Among

all remaining edges in group 1, an edge $e_1 : i_1 \rightarrow j_1$ with the smallest shortest distance $D(j_1)$ will be extracted. For edges with the same shortest distances, we arbitrarily choose one. The initialization part in Step 3 is executed for edge e_1. Then we get a node list $M = M_1$ and an edge list $Q = Q_1$. Next, we continue to extract an edge $e_2 : i_2 \rightarrow j_2$ with the increased value d_2 from group 1 that has the smallest distance value $D(j_2)$. The initialization part should be executed again to get M_2. Now $M = M_1 \cup M_2$, which means for $\forall v \in M$, $M.v.inc = M_1.v.inc + M_2.v.inc$. If $v \notin M_1$, let $M_1.v.inc = 0$ and do the same for node v not in M_2. From node set M, Q_2 is built following the initialization step and now $Q = Q_1 \cup Q_2$. When group 1 is empty for all edges are extracted, the final lists, M and Q are obtained. These lists which have the same properties as in the update process for one edge weight increase. From M and Q, the procedure for multiple edge changes is the same as the second step in Step 3 in the new proposed algorithm. Thus, after the update process with edges in group 1 from the temporary SPT, the final new SPT is derived.

4 Graph with Negative Weight Edges

The new algorithms can also be applied to a graph with negative weight edges. For example, given a graph with its old SPT constructed by the bold edge as in Figure 4(a), and the weight of edge (V, U) increased from -2 to 4, we need to update the old SPT because of this edge weight change.

To apply the proposed algorithm to this kind of graph, the Step 1 should be replaced by the static Bellman-Ford algorithm [13] to get an old SPT. The following steps are the same as one edge weight change given that the graph is guaranteed to be without negative weight cycles. Since the edge with weight increment is (V, U) and it is in the old SPT, all descendants of node U, which are U, Y in the old SPT, will be updated to build a new SPT. The incoming edges with the least increased value to node U, Y are edge (Z, U) with increased value of 4 $(6 - 2)$ and edge (X, Y) with increased value of 5 $(7 + (-4) - (-2))$ respectively. According to the update process, U is the parent node of Y and only one edge (Z, U) is added to the edge list Q since edge (X, Y) has bigger increased value than edge (Z, U) does. Then the new shortest distances for all descendants of node U (U, Y) are increased by 4 and node Z becomes the parent of node U in the new SPT. Thus, we update all nodes and the new SPT is in Figure 4(b) (the bold edges construct the new SPT).

5 Simulation Results

In this section, the computation time for the new algorithm and the one in [12] are compared for the case of one

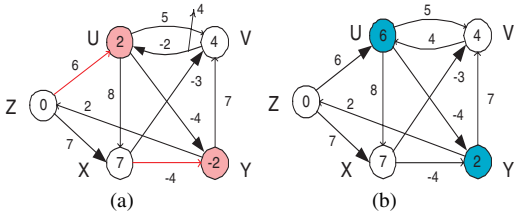

Figure 4. (a) The weight of edge (V, U) increased from -2 to 4; (b) The new SPT denoted with bold edges by the new algorithm

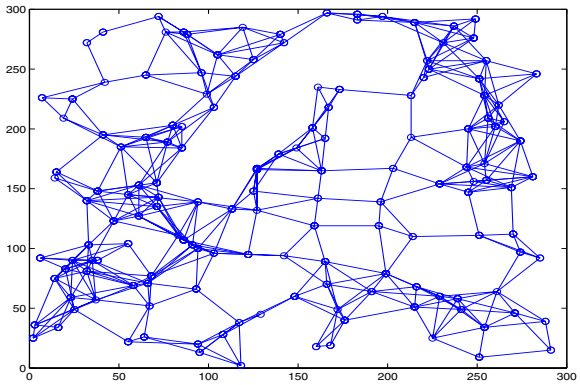

Figure 5. Randomly created network with 150 nodes.

edge weight change. The major difference is that the new algorithm introduces the node list M when the weight of an edge increased and directly updates the node set $des(j)$ when the weight of an decreased. These methods greatly reduce the time to enqueue and dequeue edges from Q and consequently have less time to search for the edge with the minimum value in Q. A randomly generated network topology is shown in Figure 5 with the node size 150 in the simulated area.

In the simulation for a specific network size, 500 continuous weight changes is tested based on one generated graph. We verify different algorithms by randomly generated edge weight changes. Among all 500 continuous edge weight changes, there are around 110 times of edge weight increments for a specific network size. We must dynamically update the SPT for these weight increments. Moreover, there are about 100 times of edge weight decrements for the SPT update. Table 2 shows the comparison results between the new algorithm and the algorithm in [12] for different w. w denotes the maximum value for each edge weight in a generated network topology. We maintain the network node size to be $N = 500$. In Column "cases" under "weight decrement", we show the number of times to update an old SPT to a new SPT for one edge weight decreased among all 500 times of edge weight changes. Under Column "#

w	weight decrement					weight increment				
	cases	algo. [12]		new algorithm		cases	algo. [12]		new algorithm	
		# edge	# search	# edge	# search		# edge	# search	# edge	# search
5	78	356	297	200	219	110	1466	2395	462	661
10	94	530	761	342	667	121	1480	2339	560	819
15	103	578	640	372	537	107	1558	1953	438	568
20	111	726	1400	504	1289	114	1458	2926	530	1000

Table 2. The performance comparison between the new algorithm and algorithm in [12] for different ranges of edge weight.

edge", we show the number of edges added and extracted from list Q. The summation of times to search for the minimum value edge from Q is in Column "# search". The numbers in Column "weight increment" are the simulation results for the update process because of edge weight increased. Compared with the algorithm in [12], the new algorithm always has less computation complexity to dynamically update the SPT. For the weight range $w = 5$, in the case of weight decrement, the new algorithm only generate 56.2% (200/356) # edges and 73.7% (219/297) # searching times related to edge list Q compared with the algorithm in [12]. While in the case of weight increment, the new algorithm improves even more, which is 31.5% (462/1466) # edges and 27.6% (661/2395) # searching times in Q.

6 Conclusion

In this paper, a new efficient algorithm has been presented for dynamically computing a new Shortest Path Tree (SPT) in a network based on the outdated SPT. The new algorithm not only minimizes the computation time, but also makes the minimum number of changes to the SPT structure as well. Thus, it removes the disadvantage caused by static algorithms for SPT update. Compared with all other known dynamic algorithms, the new algorithm achieves the least running time.

During the rebuilding procedure for a new SPT, the new algorithm always update a branch in the graph. An efficient algorithm must select branches efficiently. For the SPT update, a branch in a graph is unique when the related edge is chosen. Our algorithm concentrates on the edges that really contribute to the construction of a new SPT. With less edges in the set Q, the computation time for the algorithm is greatly reduced. This dynamic update process is different from some dynamic algorithms that select one node at a time, or other dynamic algorithms that select an entire branch at once but still with a lot of redundant edge information in the edge list Q. As a result, the new improved algorithm has reduced most redundant computation time. Furthermore, this efficient algorithm can be extended to a graph with negative weight edges and applied to the case of multiple edge weight changes in order to dynamically update an old SPT to a new one.

References

[1] S. Sengupta, D. Saha, and S. Chaudhuri, "Analysis of enhanced OSPF for routing lightpaths in optical mesh networks," in *Proceedings of IEEE International Conference on Communications (ICC 2002)*, pp. 2865–2869, 2002.

[2] R. Rastogi, Y. Breitbart, M. Garofalakis, and A. Kumar, "Optimal configuration of OSPF aggregates," *IEEE/ACM Trans. on Networking*, vol. 11, pp. 181–194, April 2003.

[3] O. Sharon, "Dissemination of routing information in broadcast networks: OSPF versus IS-IS," *IEEE Network*, vol. 15, pp. 56–65, Jan/Feb 2001.

[4] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 756–767, May 2002.

[5] E. Dijkstra, "A note two problems in connection with graphs," *Numerical Math.*, vol. 1, pp. 269–271, 1959.

[6] A. Basu and J. Riecke, "Stability issues in OSPF routing," in *Proc. the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 225–236, 2001.

[7] 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.

[8] 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.

[9] 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.

[10] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully dynamic output bounded single source shortest path problem," in *Proc. 7th Annu. ACM-SIAM Symp. Discrete Algorithms*, (Atlanta, GA), pp. 212–221, 1998.

[11] 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.

[12] 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.

[13] R. Bellman, "On a routing problem," *Quarterly Applied Mathematics*, vol. 16, pp. 87–90, 1958.