Improved Sliding Shortest Path Algorithm: Performance Analysis

Henry Carter[†] and Ramesh Bhandari[‡]
[†]Georgia Institute of Technology
Atlanta, Georgia 30332
carterh@gatech.edu
[‡]Laboratory for Telecommunications Sciences
8080 Greenmead Drive, College Park, Maryland 20740
drbhandari617@gmail.com

Abstract

Given an undirected, weighted graph, and a pair of vertices s and t, connected by the shortest path, and an edge pq not lying on the shortest path, what is the minimal change required in the given graph to cause the shortest path between s and t to pass through edge pq? This is a type of a problem often faced by network administrators in the telecommunication world. Unfortunately, the problem is NP-hard and one resorts to heuristics. Recently, a heuristic called the *Improved Sliding Shortest Path Algorithm* was presented as an improvement of an earlier heuristic. In this paper, we provide a detailed numerical comparison of the two algorithms and demonstrate the superiority of the improved version via applications to real-life networks.

Keywords: shortest path, algorithm, weighted graph, undirected, sliding, constrained, rerouting, network, optimization, performance analysis, minimal edge weight changes.

1 Introduction

Telecommunication network management often requires that traffic flows over a network be adjusted to move congestion from overloaded links to underutilized links [2-5]. Each link in such a network is assumed to have a certain cost of transmission, and data is assumed to flow from a source to a terminus by following the least-cost path through the network. In particular, the task of adjusting the link costs (or weights) to cause a certain network flow to traverse an underutilized link can be modeled in a graph as the following problem: in a

given undirected and weighted graph, find a minimal cardinality set of edges with minimal edge weight changes such that the shortest path between two vertices *s* and *t* will then include an edge *pq*. The algorithm assumes that a simple path exists between *s* and *t* that includes edge *pq*, that this path is unique, and that the modified graph after weight changes has non-zero positive weights. The problem of finding the absolute minimum number of edge changes required for the shortest path to pass through edge *pq* is provably an NP-hard problem [6], requiring every possible set of edge changes to be compared. Thus, practical application requires developing a heuristic to make a minimal selection in a reasonable amount of time. Along with making a minimal cardinality of edge weight changes, the magnitudes of these changes must also be minimized. This is because the more edges that are changed or the larger the changes, the more disturbance there will be within the network. This disturbance can be seen practically in the convergence time of a network using the *Open Shortest Path First (OSPF)* routing protocol [8].

In a recent publication, Bhandari [7] developed an algorithm called the *Sliding Shortest Path* (*SSP*) algorithm to determine a set of positive integer edge weight changes that may be the smallest for any given graph. His initial algorithm used a technique of cutting edges until the shortest path from s to t included edge *pq*. By modifying this algorithm to increment edge weights instead of cutting edges, the *SSP* algorithm accomplishes the task of forcing the shortest path from *s* to *t* across a given link *pq*. In a later publication, he developed an improved algorithm that combines the *SSP* algorithm with negative weight changes along the desired shortest path from *s* to *t* that includes the edge *pq* [6]. This algorithm, called the *Improved Sliding Shortest Path (ISSP)* algorithm, incorporates all possible solutions of the *SSP* and adds the potential for solutions using negative edges weight changes. Because the solution set of the *ISSP* is a superset of the solution set of the *SSP*, the *ISSP* will always output a solution that changes the same number or fewer edge weights than the *SSP*.

In this paper, we attempt to quantify the improvement of solutions output by the *ISSP* over solutions output by the *SSP*. By executing both algorithms for the same choice of *s, t,* and *pq,* our goal is to demonstrate that the *ISSP* algorithm provides better solutions than the *SSP*. We vary the choice of *s, t,* and *pq* to demonstrate the types of graphs and choices of source and destination where the *ISSP* shows the most improvement. The paper is organized as follows: Section 2 considers the formal definition of the problem, Sections 3 and 4 revisit the definitions of the *SSP* and *ISSP* algorithms respectively, Section 5 describes our experimental setup, Section 6 presents our quantified results, and Section 7 offers concluding remarks.

2 Problem Definition

Let a graph $G = (V, E)$, composed of the vertex (node) set V and the edge (link) set E, be an undirected, weighted graph where all edge weights are positive integers (> 0). We assume there are no loops or multiple edges in G. We also assume that G is bi-connected, (i.e. for any two vertices s and t and any edge pq, there exists a path from s to t crossing pq that visits each vertex in the path exactly once). This type of path (i.e. simple path) is referred to in the remainder of the text as a *path*. Our final assumption is that traffic crossing this graph from one vertex to another flows along a single shortest path.

Let $SP(s, t)$ be the shortest path between the given vertices s and t in the graph G. Given any edge pq in G, let Γ_a be a set of edges within the graph G whose weights are changed to alter the given shortest path, forcing $SP(s, t)$ to cross the edge pq. We denote the number of edges in Γ_a as $|\Gamma_a|$. The problem is stated as follows:

Given an undirected, weighted graph $G = (V,E)$, a pair of vertices $s, t \in V$, and an edge $pq \in E$,

$$minimize\ |\Gamma_a| \tag{1}$$

subject to arc pq (or arc qp) $\in SP(s,t)$

Intuitively, the problem is to force the shortest path from s to t to cross a given edge pq by changing the weights of as few edges as possible. This problem is proven to be NP-hard [6]. This NP-hard nature is the motivation for developing and comparing the performance of the heuristics outlined in the next sections.

3 The Sliding Shortest Path Algorithm

For reference purposes, we summarize the *SSP* algorithm from [6] below.

Given an undirected, weighted graph $G = (V, E)$, the *SSP* algorithm will calculate a minimal cardinality set of positive edge weight increments which force the shortest path between two vertices s and t to cross a chosen edge pq. The algorithm assumes that a simple path connecting s and t and including edge pq exists, and that the shortest path connecting s to t and crossing pq is unique.

The algorithm begins by calculating the desired shortest path from s to t which crosses the edge pq by computing the shortest pair of vertex-disjoint paths [9, 10], one connecting s to one end of pq and the other connecting t to the opposing end of pq. The algorithm calculates these paths simultaneously and determines if the shortest path from s to t crosses pq from node p to q or vice versa, and selects the shortest pair. The algorithm states that the cost of the desired shortest path is the sum of the edge costs in the path connecting s to p (or q), the cost of pq, and the sum of the edge costs in the path connecting t to q (or p). Starting from vertex s, the algorithm increments iteratively the weight of the first edge on the current shortest path from s to t that does not lie along the desired shortest path crossing pq until the shortest path crosses pq. The incremented amount in each iteration is always the difference between the cost of the current shortest path and the cost of the desired shortest path plus one. The algorithm then repeats, this time starting from vertex t. Once the algorithm has executed in both directions, the solutions are compared, and the solution requiring fewer edges to change weights on is returned. A full description of the algorithm, proofs of correctness, and an example can be found in [6].

4 The Improved Sliding Shortest Path Algorithm

For reference purposes, we summarize the *SSP* algorithm from [6] below.

Given an undirected, weighted graph $G = (V, E)$, the *ISSP* algorithm will calculate a minimal cardinality set of positive and/or negative edge weight changes which force the shortest path between two vertices s and t to cross a chosen edge pq. The algorithm assumes that a simple path connecting s and t and including edge pq exists, and that the shortest path connecting s to t and crossing pq is unique. One critical assumption to this algorithm is that every edge weight in the graph remains a positive integer. Thus, any edge weight decrement for a given edge e_i with weight w_i *must* be less than or equal to $w_i - 1$.

The *ISSP* functions like the *SSP*, but also explores for decrements on the edges of the desired final path. Prior to incrementing the first edge not on the desired shortest path, the *ISSP* decrements the edges on the desired shortest path that are not on the current shortest path to make the desired shortest path shorter than the current shortest path. Further adjustments (increments on some other edges) may be needed to ensure that the desired shortest path is indeed the shortest path in the graph. These decrements (along with the other possible adjustments) then constitute one potential solution. Solutions obtained in subsequent iterations are always a mixture of increments and decrements. At the start of each iteration (providing a possible solution), the weights that were decremented in the earlier iteration, along with other weights that might have been incremented, are reset to the weights in the original graph. The algorithm recalculates the current

shortest path and increments the weight of the first edge along this newly recalculated shortest path that is not on the desired shortest path (as in the SSP algorithm), and then subjects the non-overlapping part of the desired shortest path to decrements (and adjustments) as described earlier. . This produces another potential solution, which includes one increment and a set of decrements (and adjustments). Each subsequent iteration adds a new increment on an edge of the current shortest path, and the algorithm terminates when the current shortest path passes over the edge *pq*. From the set of solutions obtained (one in each iteration, with the first solution comprising all decrements (and some possible increments), the last comprising all increments, and the intermediate solutions mixtures of increments and decrements), one selects the solutions with the minimal cardinality. In a practical implementation, one obtains the optimal solutions dynamically, discarding solutions found, if they are worse than the previous, and also discarding previous solutions if they are worse than the current. A complete and full description of the algorithm, proofs of correctness, and an example can be found in [6].

5 Experimental Setup

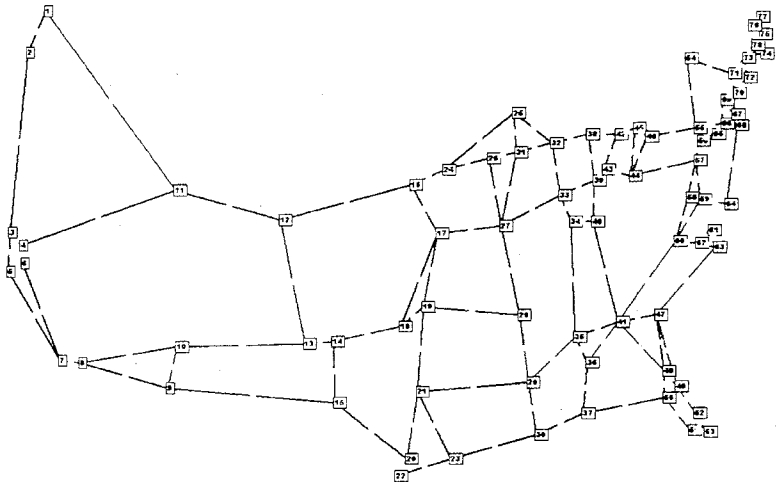

Figure 1. The first test graph.

To compare the performance of the *SSP* and the *ISSP* algorithms, we selected two graphs simulating real-life networks. As in the work by Coleman and Moré [1], we define the density D of a graph $G = (V, E)$ as:

$$D = \frac{2|E|}{|V|(|V|-1)} \tag{2}$$

where $|V|$ is the number of vertices in G and $|E|$ is the number of edges in G. The density of a graph relates the number of edges in a graph to the number of edges possible in a graph. It ranges from a graph with no edges, with density 0, to a complete graph, with density 1. The first graph, pictured in Figure 1, is composed of 78 vertices, and has a density of approximately 0.038. The second graph, pictured in Figure 2, is composed of 51 vertices, and has a density of approximately 0.068. To capture the exact nature of the *ISSP* algorithm's improvement, we performed two separate experiments.

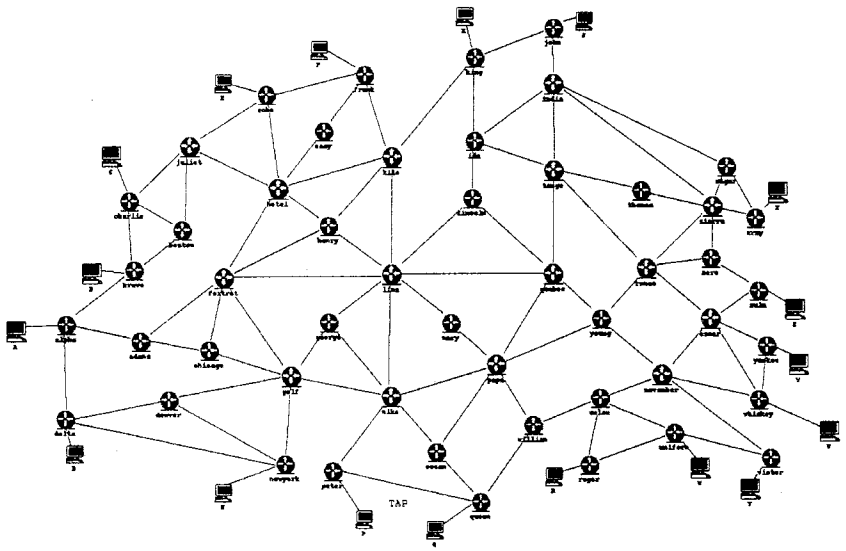

Figure 2. The second test graph

In our first experiment, we selected two nodes s and t from our test graph where each edge is assigned a random weight from 1 to 100. For the given choice of s and t, we ran both the *SSP* and the *ISSP* once for every possible choice of edge pq. For 6-7 different s-t paths, we collected the following statistics:

- Percent Changed (Chng): the percentage of pq edges where the *ISSP* produced a better solution than the *SSP*

- Percent Improved (Imp): For the *pq* edges where *ISSP* produced a better solution, the average percentage decrease in edges changed from the *SSP*
- Global Improved (G_imp): For all *pq* edges tested in the graph, the average decrease in edges changed by the *ISSP*
- Maximum by *ISSP* (MaxNew): The maximum number of edges changed for any single execution of the *ISSP*
- Maximum by *SSP* (MaxOld): The maximum number of edges changed for any single execution of the *SSP*

After completing the first experiment, we repeated it 50 more times, re-randomizing the weights in the graph after each iteration and reporting all statistics as an average over the 50 trials.

The second experiment, we selected a link *pq* from our test graph where each edge is assigned a random weight from 1 to 100. For the given choice of *pq*, we ran both the *SSP* and the *ISSP* once for a selection of 12 possible choices of path *s-t*. For 7-8 different *pq* edges, we collected the following statistics:

- Percent Changed (Chng): the percentage of *s-t* paths where the *ISSP* produced a better solution than the *SSP*
- Percent Improved (Imp): For the *s-t* paths where *ISSP* produced a better solution, the average percentage decrease in edges changed from the *SSP*
- Global Improved (G_imp): The global average decrease in edges changed by the *ISSP*
- Maximum by *ISSP* (MaxNew): The maximum number of edges changed for any single execution of the *ISSP*
- Maximum by *SSP* (MaxOld): The maximum number of edges changed for any single execution of the *SSP*

After completing the second experiment, we repeated it 50 more times, re-randomizing the weights in the graph after each iteration and reporting all statistics as an average over the 50 trials.

6 Performance Analysis

Experiment 1: For the *s-t* paths selected, we classified them in a range of "far" to "near", where the number of edges in the original shortest path connecting *s* to *t* ranged from 17 for the "farthest" pair to 1 for the "nearest" pair. For the *s-t* paths where *s* and *t* were spread farther apart, the new algorithm showed dramatic improvement over the old algorithm. In the first test graph, this trend is apparent both in the number of cases improved (Chng), as well as the decrease

in the number of edges changed (Imp). In the second test graph, the trend is less apparent in the number of cases that improved, but still strongly present in the decrease in the number of edges changed.

Table 1 Experiment 1, test graph 1, single weight results: All averages taken over one graph testing all possible edges pq for the chosen path s-t. The paths are arranged from the farthest pair (1-53) to the nearest pair (60-70)

s	t	Chng(%)	Imp(%)	G_imp(%)	MaxNew	MaxOld
1	53	50	36	24	12	12
9	73	45	31	20	8	10
25	30	47	23	13	10	11
13	55	47	38	20	6	10
29	47	35	19	07	12	12
60	70	12	27	03	11	11

Table 2 Experiment 1, test graph 1, randomized weight results: All averages take over one graph structure, randomizing the weights 50 times and testing all possible edges pq for the chosen path s-t in each random edge weight set. The paths are arranged from the farthest pair (1-53) to the nearest pair (60-70)

s	t	Chng(%)	Imp(%)	G_imp(%)	MaxNew	MaxOld
1	53	46	35	19	11	12
9	73	50	41	27	7	10
25	30	38	29	11	10	11
13	55	41	36	19	8	9
29	47	39	22	09	12	12
60	70	24	22	06	11	12

Table 3 Experiment 1, test graph 2, single weight results: All averages taken over one graph testing all possible edges pq for the chosen path s-t. The paths are arranged from the farthest pair (7-19) to the nearest pair (9-43)

s	t	Chng(%)	Imp(%)	G_imp(%)	MaxNew	MaxOld
7	19	61	45	35	7	9
12	33	67	36	29	8	11
16	34	88	35	32	8	11
38	48	75	33	28	12	15
15	27	90	27	26	12	14
5	46	49	41	27	7	8
9	43	81	53	44	5	8

Table 1.4 Experiment 1, test graph 2 results: All averages take over one graph structure, randomizing the weights 50 times and testing all possible edges pq for the chosen path s-t in each random edge weight set. The paths are arranged from the farthest pair (7-19) to the nearest pair (9-43)

s	t	Chng(%)	Imp(%)	G_imp(%)	MaxNew	MaxOld
7	19	67	41	31	7	9
12	33	63	40	30	7	10
16	34	78	32	27	9	12
38	48	73	30	25	11	14
15	27	80	33	30	11	14
5	46	65	42	34	7	10
9	43	65	45	35	6	10

The reason for the *ISSP* algorithm's improved performance in the case where s and t are far apart is that the difference between the current shortest path and the desired shortest path is small. Since more of the pq links lie generally in between s and t when they are farther apart, the difference between the two paths can be absorbed more easily by decrementing along the desired shortest path. Also, since there are only a limited number of edges along the desired shortest path that can be decremented, the difference between the cost of the current shortest path and the desired shortest path must be small for edge decrements to effectively shift the shortest path. By contrast, when s and t are generally closer together, the pq edges spread around the graph create a large difference between the desired shortest path cost and the current shortest path cost. For these pq edges, simply decrementing along that path is not enough to absorb this large difference. Thus, numerous increments are required to bring the difference down. In practice, the extreme edge increments may cause bottlenecks since routers will avoid sending traffic across the incremented edges.

Density of the graph also factors into the problem in a similar way. The more possible paths there are between s and t, the more edges that have to be changed to force the desired shortest path to be shorter than all other possible paths from s to t. Thus, there is less improvement between the new and old algorithms with more dense graphs than with more sparse graphs. However, since our application is considering telecommunication networks, which tend to be sparse, density plays only a small role in our experimental results.

Experiment 2: For the pq links considered in our experimentation, we classified them in a range of "central" to "peripheral" based on their position within the graph. To quantify this concept, we state that if an edge is central

within the graph, then the shortest path from either endpoint to any other vertex in the graph will contain a small number of edges. If an edge is peripheral within the graph, there will be a vertex on the opposite side of the graph such that the shortest path from an endpoint to that vertex will contain a large number of edges. In our example graphs, the most central edges can be connected to any other vertex in the graph by a path containing only 5 edges while the most peripheral edges have some vertex that is at least 17 edges away from one endpoint. For the *pq* links where *pq* was located "centrally", the new algorithm again showed marked improvement over the old algorithm. However, when the *pq* link was located "peripherally", there was little difference between the two algorithms performances. As in test 1, this difference is always highlighted by the percentage decrease in edges changed, but only partly shown in the number of cases that improved.

Table 1.5 Experiment 2, test graph 1, single weight results: All averages taken over one graph testing twelve possible s-t paths through the chosen edge pq. The paths are arranged from the most central constraint edge (27, 31) to the most peripheral constraint edge (4, 11)

p	q	Chng(%)	Imp(%)	G imp(%)	MaxNew	MaxOld
27	31	70	53	38	5	5
71	73	40	14	07	12	13
41	60	67	43	26	8	8
21	29	38	46	29	3	6
48	49	67	21	18	11	12
34	35	44	43	27	4	6
56	57	11	14	02	9	9
4	11	00	00	00	11	11

Table 1.6 Experiment 2, test graph 1, random weight results: All averages take over one graph structure, randomizing the weights 50 times and testing 12 s-t paths through the chosen edge pq for each random edge weight set. The paths are arranged from the most central constraint edge (27, 31) to the most peripheral constraint edge (4, 11)

p	q	Chng(%)	Imp(%)	G imp(%)	MaxNew	MaxOld
27	31	57	41	29	4	6
71	73	23	10	04	13	13
41	60	43	32	16	7	8
21	29	43	41	23	3	4
48	49	62	32	24	8	9
34	35	60	44	31	4	5

56	57	36	26	10	9	9
4	11	10	16	03	9	10

Table 7 Experiment 2, test graph 2, single weight results: All averages taken over one graph testing twelve possible s-t paths through the chosen edge pq. The paths are arranged from the most central constraint edge (16, 21) to the most peripheral constraint edge (24, 34)

p	q	Chng(%)	Imp(%)	G_imp(%)	MaxNew	MaxOld
16	21	58	38	32	6	9
26	32	64	39	32	4	7
36	41	58	24	14	14	14
17	22	73	43	39	6	8
28	29	33	17	08	10	11
6	11	40	31	20	6	9
24	34	50	41	30	4	6

Table 1.8 Experiment 2, test graph 2, random weight results: All averages take over one graph structure, randomizing the weights 50 times and testing 12 s-t paths through the chosen edge pq for each random edge weight set. The paths are arranged from the most central constraint edge (16, 21) to the most peripheral constraint edge (24, 34)

p	q	Chng(%)	Imp(%)	G_imp(%)	MaxNew	MaxOld
16	21	70	41	35	6	8
26	32	75	44	37	4	7
36	41	46	23	13	11	13
17	22	66	37	29	6	7
28	29	71	32	26	8	12
6	11	75	40	34	7	10
24	34	59	43	32	4	6

When the *pq* link is centrally located, a random *s-t* path will most likely pass through or near the link already. Again, this causes the difference in the desired shortest path and the current shortest path to be generally small, requiring only a few edge decrements to absorb the difference between the two. However, when edge *pq* is on the periphery of the graph, a larger number of *s-t* paths will have to diverge significantly from their current shortest path to cross *pq*, creating a larger difference in the cost of the current shortest path and the desired shortest path. Again, this will require a more significant number of edge increments to absorb the difference, which makes the two algorithms perform similarly.

7 Summary and Discussion

In this paper, we have compared two heuristics designed to shift the shortest path through a graph to cross a chosen edge by making a minimal number of edge weight changes. The first algorithm, the *SSP* algorithm, only considers solutions that make positive edge weight changes. The second algorithm, the *ISSP* algorithm. expands upon this solution set by also considering negative edge weight changes. To quantify the improvement of solutions produced by the *ISSP*, we examine a number of statistics taken over two test graphs designed to simulate telecommunications networks. Our results conclusively show that the *ISSP* produces more efficient solutions in a majority of cases, and in some cases can reduce the number of edge weights changed by 53%. The amount of improvement from the *ISSP* is maximized when the difference between the cost of the desired shortest path and the cost of the current shortest path is small. These results would be of significant interest to the telecommunication network operators, employing algorithms to move the traffic around to achieve optimal throughput.

References

[1] T. Coleman, J. Moré, *Estimation of Sparse Jacobian Matrices and Graph Coloring Problems,* SIAM Journal on Numerical Analysis, 20 (1983), no. 1, 187-209.

[2] B. Fortz and M. Thorup, *Internet Traffic Engineering by Optimizing OSPF Weights*, Proc. of the 19th IEEE INFOCOM, Tel-Aviv, Israel (2000) pp. 519-528.

[3] B. Fortz and M. Thorup, *Optimizing OSPF/IS-IS Weights in a Changing World*, IEEE Journal on selected areas in communication, **20** (2002) pp.756-767.

[4] B. Fortz, J. Rexford, and M. Thorup, *Traffic Engineering with Traditional IP Routing Protocols, IEEE Communications Magazine*, **40** (2002) pp.118-124.

[5] M. Ericsson, M.G.C. Resende, and P. M. Pardalos, *A Genetic Algorithm for the Weight Setting Problem in OSPF Routing,* J. Combinatorial Optimization, **6** (2002) 299-333.

[6] R. Bhandari, *The Improved Sliding Shortest Path Algorithm,* Congressus Numerantium, 203 (2010) 175-192.

[7] R. Bhandari, *The Sliding Shortest Path Algorithms*, Proc. of the 8th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, Paris, France (2009) pp. 95-101.

[8] J. Doyle, *Routing TCP/IP, Volume I*, Cisco Press (2001).

[9] R. Bhandari, *Survivable Networks: Algorithms for Diverse Routing*, Kluwer Academic Publishers, 1998.

[10] J. W. Suurballe, *Disjoint Paths in a Network,* Networks **4** (1974) pp.125-145.