Who's *The Weakest Link*?

Nikhil Devanur , Richard J. Lipton, and Nisheeth K. Vishnoi
{nikhil,rjl,nkv}@cc.gatech.edu

Georgia Institute of Technology, Atlanta, GA 30332, USA.

Abstract. In this paper we consider the following problem: Given a network G, determine if there is an edge in G through which at least c shortest paths pass. This problem arises naturally in various practical situations where there is a massive network (telephone, internet), and routing of data is done via shortest paths and one wants to identify most congested edges in the network.
This problem can be easily solved by one all pair shortest path computation which takes time $O(mn)$, where n is the number of nodes and m the number of edges in the network. But for massive networks - can we do better? It seems hard to improve this bound by a deterministic algorithm and hence we naturally use randomization. The main contribution of this paper is to a give a practical solution (in time significantly less than $O(mn)$) to a problem of great importance in industry.

Keywords. Shortest Path, Massive Networks, Approximation, Random Sampling.

1 Introduction

Recent times have seen the emergence of huge networks, for instance, the internet, telephone networks, peer to peer networks, the World Wide Web, etc. Of particular interest are the internet and the telephone networks ([16]), which present a horde of practical problems.

Routing is a key element in making these huge networks work efficiently. The most common routing scheme used is the shortest-path-routing, i.e. the route between two nodes is the shortest path joining them. The advantage of using shortest paths[1] is that all the routes can be computed in $O(mn)$ time (where m is the number of edges and n the number of nodes) (see [4, 7, 18, 8]).

Maintenance of such huge networks is an immense challenge. The problem that arises in this scenario is that of identifying congested edges/nodes or hot-spots or bottlenecks. This is because these are the places which are most prone to breakdown. Moreover, any problem at these spots causes maximum damage. Another objective is load-balancing in order to reduce delays. This is done by identifying the congested edges/nodes and rerouting some routes. So it is only natural that identifying such edges/nodes is of top priority.

Congestion of an edge (a node) is defined as the number of routes that pass through an edge (a node). Assume that there is a unique route between any two nodes. For the sake of simplicity of presentation, we consider only simple, undirected graphs, with unweighted edges. One can consider many problems related to congestion in a network. This paper is centered around the following problems:

1. Is there an edge with congestion at least c?
2. Find the most congested edge.
3. More generally, find the top t most congested edges/nodes.
4. Given an edge, estimate its congestion.

Note that if we can solve 1, then we can solve 2 by binary search. In fact 1 seems to be the most useful and general problem one might want to solve. This is exactly the problem we address in this paper. This problem arises in industry (in particular is of interest to Telecordia [16]) where one wants to identify most congested edges in the network.

One may construct simple examples where deterministic algorithms don't perform much better than $\Omega(mn)$. So the natural approach is to use randomness. Although these techniques are very simple, they are very powerful and have been used to give important results: [9, 12, 13, 5, 15] and [14]. Similar ideas using randomization were used by Karger *et al.* in [10] and [11] to estimate the value of max-flow.

[1] Even though we only consider shortest-path-routing in this paper, we *do not* use any particular property of the shortest path. Our techniques will work for most routing schemes, but the interesting ones are those in which a route can be computed in $O(m)$ time. For any such routing scheme, all the routes can be computed in $O(mn^2)$ time. Notice that one cannot hope to do better than $O(mn)$ time, since for a tree, at least $O(n^2) = O(mn)$ time is required.

Since we use randomization naturally leads us to consider a slight relaxation of the problem: Given parameters c and ϵ, if there exists an edge with congestion $\geq c(1+\epsilon)$, then output YES with probability $\geq 1-1/\text{poly}(n)$. If for all edges, the congestion is $\leq c(1-\epsilon)$, then output NO with probability $\geq 1-1/\text{poly}(n)$. Note that Problem 2 can also be approximately solved using the above relaxation.

We also consider those graphs where there is a "clear" separation between the set of most congested edges and the rest, and we are needed to output the edges among the most congested.

This and similar problems are omnipresent in industry and to our surprise, have not been systematically studied in the Computer Science literature- to the best of our knowledge. In this paper we provide a formal setting for some of these problems and give extremely practical algorithms using elementary ideas. We make no attempts to optimize the constants in this paper.

1.1 Formal Setting

Let $G = (V, E)$ be an undirected, unweighted, simple graph with $n := |V|, m := |E|$.

Definition 1. *Given an edge $e \in E$, define $\sigma(e)$, the* **congestion** *of an edge, to be the number of shortest paths passing through e.*

To make this well defined, we label the vertices and use the lexicographically least shortest path for each pair of vertices. One can also define the congestion of a node similarly:

Definition 2. *Given a node $v \in V$ define $\sigma(v)$, the* **congestion** *of a node, to be the number of shortest paths containing v as an interior point.*

As before, we label the nodes and consider the lexicographically shortest path. Also note that

$$\sum_{e:e\sim v} \sigma(e) = 2\sigma(v) + n - 1$$

Definition 3. *Let P be any set of shortest paths. Define $\sigma_P(e)$ to be the number of shortest paths in the set P that pass through e.*

Definition 4. *Let $g(G) := \max_{e\in E} \sigma(e)$ be the maximum congestion in the graph G.*

Definition 5. *Suppose that there exist $\delta > 0$ and a function $f(n)$, such that E is partitioned into two classes,*

$$S_1 := \left\{e \in E : \sigma(e) \geq n^{1+\delta}(1 + f(n))\right\} \tag{1}$$

$$S_2 := \left\{e \in E : \sigma(e) \leq \frac{n^{1+\delta}}{1 + f(n)}\right\}. \tag{2}$$

Then we say that there is an **f-separation** *at δ in G.*

That is, we assume that there is a separation between the congestion of edges in S_1 and that of the others, S_2. The problems that we want to solve are[2]:

Problem 1. Given c, is there an edge $e \in E$ with $\sigma(e) \geq c$?

Problem 2. Find an edge $e \in E$ with the maximum congestion, i.e., $\sigma(e) = g(G)$. Or more generally, find t most congested edges.

Problem 3. Given that there is an f-separation at δ in G, find the set S_1, as defined in (1) (and hence S_2).

Problem 4. Given an edge $e \in E$, estimate $\sigma(e)$, its congestion.

1.2 Our results

We outline the main results[3] obtained in this paper.

We give an algorithm for solving Problem 1. The following theorem proves the correctness and establishes its running time:

Theorem 1. *For all $\epsilon > 0$, there exists an algorithm A running in time $O(km)$ where $k = \frac{\log n}{\epsilon^2} \cdot \frac{n^2}{c}$ such that*

- *If $\exists e \in E, \sigma(e) \geq c(1+\epsilon)$ then $\Pr[A \text{ outputs YES}] \geq 1 - n^{-2}$*
- *If $\forall e \in E, \sigma(e) \leq c(1-\epsilon)$ then $\Pr[A \text{ outputs NO}] \geq 1 - n^{-2}$*

An immediate corollary of the above theorem gives a solution to Problem 2.

Corollary 1. *Given a graph G such that $g(G) \gg n \log^3 n$ there exists[4] an algorithm running in time $o(mn)$ which finds an edge e such that*

$$\Pr\left[\sigma(e) < \left(1 - \frac{1}{\log n}\right) g(G)\right] < \frac{1}{n}$$

A variant of the previous algorithm solves Problem 3

Theorem 2. *given a graph G with an f-separation at δ, there is an algorithm such that the probability that it outputs S_1 correctly is at least $1 - n^{-2}$, and runs in time $O(km)$ where $k = \frac{\log n}{\epsilon^2} \cdot n^{1-\delta}$ and $\epsilon = 1/f(n) - 1/f^2(n)$*

Again, we get as a corollary that Problem 4 can be solved.

Corollary 2. *Given $G = (V, E)$, $e \in E$, with $\sigma(e) \gg n \log^3 n$, there exists an algorithm which runs in time $o(mn)$ and finds $\tilde{\sigma}(e)$ such that*

$$\Pr\left[|\tilde{\sigma}(e) - \sigma(e)| \geq \sigma(e)/\log n\right] \leq n^{-4}$$

[2] We define all the problems w.r.t the congestion on edges. However, they can also be defined w.r.t the congestion on vertices.

[3] As with the problems, the results are also stated w.r.t the congestion on edges, and the corresponding results for congestion on vertices follow immediately.

[4] For functions $f(n), g(n), f(n) \gg g(n)$ is the same as $g(n) = o(f(n))$.

1.3 Organization of the paper

In section 2 we give an algorithm to solve Problem 1, and prove Theorem 1. We give a slight variation of the previous algorithm in Section 3 that proves Theorem 2. In the same section we compare our algorithm with the deterministic $O(mn)$ algorithm. In Section 4, using results from the theory of random graphs, we show how to further improve the running time. Section 5 considers a coupon-collection approach to the problem. In section 6 we present related open problems.

2 Main result

2.1 Algorithm

Given an undirected graph $G = (V, E)$ on n vertices and m edges, the number of shortest paths is $\binom{n}{2}$. The main idea of the algorithm is random sampling followed by finding if there exists an edge whose observed congestion is greater than some cut-off.

The algorithm picks k paths at random, uniformly and independently[5]. k will be determined later. Given k, define the **"cut-off"** to be

$$c(k, n, c) := \frac{kc}{\binom{n}{2}}.$$

The algorithm is as follows:

```
Input: The graph $G = (V, E), c$
Choose $k$ paths at random, uniformly and independently. Let the set of paths
chosen be $P$;
Compute all the shortest paths and $\sigma_P(e)$ for each $e \in E$;
if for some edge $e \in E$, $\sigma_P(e) \geq c(k, n, c)$ then
    Output: YES
else
    Output: NO
end
```

Algorithm 1: Algorithm for detecting an edge with high congestion

Each of the shortest paths in P can be computed in $O(m)$ time, so it takes $O(km)$ time to compute all the shortest paths in P. Note that in the same amount of time, $\sigma_P(e)$ can be computed for all $e \in E$. So the total running time of the algorithm is $O(mk)$.

[5] One way to do this is pick two vertices randomly without replacement and consider the path with those as endpoints.

2.2 Analysis

We need to determine k, the number of paths to be picked. Also, we need to analyze the probability of success. In fact, k will depend on the probability of success desired, besides c and f.

Let $X_e := \sigma_P(e), \forall e \in E$ be random variables where P is the set of paths picked by the (randomized) algorithm. Each X_e can be written as a sum of k independent and identical Bernoulli trials, $X_e = X_{e1} + X_{e2} + \ldots + X_{ek}$ where X_{ei} is 1 if the i^{th} path chosen by the algorithm passes through e and 0 otherwise.

$$\Pr[X_{ei} = 1] = \frac{\sigma(e)}{\binom{n}{2}}$$

$$\mu := \mathrm{E}[X_e] = \frac{k\sigma(e)}{\binom{n}{2}}$$

Lemma 1. *For all $\epsilon > 0$, if*

$$k \geq \frac{8 \log n}{\epsilon^2} \cdot \frac{n^2}{c} \text{ and } \exists e \in E, \sigma(e) \geq c(1 + 2\epsilon),$$

Then

$$Pr[e \text{ is not picked by the algorithm}] < n^{-4}$$

Proof. All probabilities are taken over the choice of the set of paths chosen, P.

$$\Pr[e \text{ is not picked}] = \Pr\left[X_e < c(k, n, c)\right]$$

The version of Chernoff bound we use is [1],

$$\Pr[X_e < (1 - \epsilon)\mu] < \exp(-\mu\epsilon^2/2)$$

For e,

$$\begin{aligned} \mu &\geq \frac{kc(1+2\epsilon)}{\binom{n}{2}} \\ \Rightarrow \mu(1-\epsilon) &\geq \frac{kc}{\binom{n}{2}} = c(k, n, c) \\ \Rightarrow \Pr[X_e < c(n, k, c)] &\leq \Pr\left[X_e < (1-\epsilon)\mu\right] \\ &\leq \exp(-\mu\epsilon^2/2) \end{aligned}$$

Moreover,

$$\begin{aligned} k &\geq \frac{8 \log n}{\epsilon^2} \cdot \frac{n^2}{c} \geq 4 \log n \cdot \frac{2\binom{n}{2}}{\epsilon^2 c} \\ \Rightarrow \frac{\epsilon^2 \mu}{2} &\geq \frac{\epsilon^2 kc}{2\binom{n}{2}} \geq 4 \log n \\ \Rightarrow \Pr[X_e < c(n, k, c)] &\leq \exp(-\mu\epsilon^2/2) \leq \exp(-4 \log n) = n^{-4} \end{aligned}$$

Similarly, the probability that some edge with low congestion is picked by the algorithm can also be bounded: (We skip the proof.)

Lemma 2. *For all $\epsilon > 0$, if*

$$k \geq \frac{12 \log n}{\epsilon^2} \cdot \frac{n^2}{c} \text{ and } \forall e \in E, \sigma(e) \leq c(1 - 2\epsilon)$$

Then

$$Pr[e \text{ is picked by the algorithm}] < n^{-4}$$

The proof of Theorem 1 is an easy application of the above lemmas.

Proof (of Theorem 1). Consider Algorithm 2.1 with $k = \frac{12 \log n}{\epsilon^2} \cdot \frac{n^2}{c}$. For each edge e with $\sigma(e) \geq c(1 + \epsilon)$ or $\sigma(e) \leq c(1 - \epsilon)$, the probability that it ends up resulting in the wrong answer is n^{-4}. Since there are at most n^2 edges, the probability that one of them ends up resulting in the wrong answer is at most n^{-2}. So with probability at least $1 - n^{-2}$ the algorithm outputs correctly.

2.3 Comparison

To get a running time better than $O(mn)$, we need c to be asymptotically bigger than $n \log n$. (n is large so any reasonably growing function like $\log n$ will do.) Another important fact about these massive graphs is that they are sparse. Thus we may assume that $m = O(n)$. Hence the maximum congestion is $\Omega(n)$. In particular, if $c \sim n^{1+\delta}$ then the running time would be $O\left(\frac{mn^{1-\delta} \log n}{\epsilon^2}\right)$. For $\delta = 1/2$ and $n = 10^6$, our algorithm is *guaranteed* to be roughly 1000 times faster than the deterministic algorithm. This is a significant saving in the running time. Moreover simulations suggest that the running time is much less than our guarantee. Also if the maximum congestion is $O(n)$, then it means that no edge is congested too much, and the graph is "nice". Hence our algorithm can be used to detect that too!

3 Finding separation

Suppose that we are given that there is an f-separation at δ in G. Using essentially the same technique as in the previous section, one can identify the set S_1 of most congested edges. Define the **"cut-off"** to be

$$c(k, n, \delta) := \frac{kn^{1+\delta}}{\binom{n}{2}}.$$

```
**Input**: The graph $G = (V, E), \delta, f()$
Choose $k$ paths at random, uniformly and independently. Let the set of paths chosen be $P$;
Compute all the shortest paths and $\sigma_P(e)$ for each $e \in E$;
**Output**: The edges with $\sigma_P(e) \geq c(k, n, \delta)$.
```

Algorithm 2: Algorithm for finding separation

We state, without proof, the corresponding parameters that give the required result:

Lemma 3. *If*

$$k \geq \frac{12 \log n}{\epsilon^2} \cdot n^{1-\delta} \text{ and } \epsilon = \frac{1}{f(n)} - \frac{1}{f^2(n)}$$

Then

$$Pr[e \in S_1 \text{ is not picked by the algorithm}] < n^{-4}$$

$$Pr[e \in S_2 \text{ is picked by the algorithm}] < n^{-4}$$

To prove Theorem 2 consider Algorithm 3 with $k = \frac{12 \log n}{\epsilon^2} \cdot n^{1-\delta}$ and $\epsilon = 1/f(n) - 1/f^2(n)$.

We need $f(n)$ to diverge as n goes to infinity. This means that $1 + 1/f(n)$ gets close to 1 as n tends to infinity. In other words, the separation need not be very "strong", but should exist. This is fairly reasonable. However, we need to know where (i.e., the δ) and by how much (i.e., the f) the separation occurs.

4 Further optimization using random graphs

A (further) reduction in running time might be obtained by the fact that the time taken to compute single pair shortest path is the same as that for single source shortest path computation. So one might choose an appropriate set of vertices and run single source shortest path computations only on those vertices. Consider a graph H on the same set of vertices as G, with an edge between u and v if and only if the algorithm chose the shortest path between u and v. Clearly, it is enough to run single-source shortest path algorithms on any vertex cover of H. Since finding a minimum vertex cover is NP-Hard, we find a 2-approximation to the minimum vertex cover by finding a maximal matching in H, [17].

Erdös and Rényi in [6] define the random graph model $\mathcal{G}(n, p)$ which consists of all graphs on n vertices, in which the edges are chosen independently and with probability p. Since we pick the shortest paths uniformly at random, H belongs to $\mathcal{G}(n, p)$ where $p = k/\binom{n}{2}$ and k is the number of samples. For the parameters in our case, where $p = o(1/n)$, the size of the minimum vertex cover is $\Omega(k)$ (see [3]). Unfortunately, it turns out that this does not give an order of magnitude improvement. It does, however, give an improvement in the constants.

5 A heuristic based on Coupon Collection

In this section we give a heuristic for detecting the edge with maximum congestion based on the idea of *coupon collection*. The main idea is explained via the following toy problem: "given a bin with b balls of k different colors, where b_i are the number of balls of color i. What is $\text{argmax}_i\ b_i$?" The coupon collection based approach for this problem is the following: "For a parameter t, which will depend upon b_i's, sample with replacement from the bin, and output the color which is the first color to repeat t times." Two remarks are in order:

1. This technique will in general not give good estimates for the values b_i's.
2. The probability of success and the running time of this algorithm is a function of b_i's.

Of course, the random sampling technique used in the previous sections applies here. But this technique might give faster estimates in cases where b_i's are extremely non-uniform. For a full technical discussion on this problem and its varians refer to the book by Blom, Holst and Sandell [2].

The problem of identifying the edge e such that $\sigma(e) = g := \max_{e' \in E} \sigma(e')$ is a generalization of the coupon collection scheme spelled above. In our setting the balls are the paths and the colors are the edges. Each ball can have multiple colors. Hence for a given parameter t we sample paths until some edge repeats t times, and output that edge.

There are simple graphs like K_n where this heuristic behaves very badly. Another case where this heuristic will fail is when the number of edges with congestion slightly less than g outnumber by far the number of edges with congestion g. But for most graphs like the following, this heuristic is very quick.

This gives a very quick heuristic to detect such cases. We leave as an open problem to fully analyze the running time and success probability of this heuristic.

6 Conclusion and Open Problems

In this paper, we considered the problem of finding if there exists an edge with congestion $\geq c$ for given c. We give fast and practical algorithms which are also simple to implement. One of the main contributions of this paper is to formalize this extremely important problem and put it in a theoretical framework. Any improvement in the running time would be of significance. Another interesting question is to find $\sigma(e)$ given e, in time better than $O(mn)$. We partially answer the question in Section 4. A deterministic solution to this problem would be very interesting.

References

1. N. Alon, J. Spencer, *The Probabilistic Method*, Wiley Interscience, 2000.

Fig. 1. A Good Graph

2. G. Blom, L Holst, D Sandell, *Problems and Snapshots from the World of Probability*, Springer-Verlag, 1994.
3. B. Bollobas, *Random Graphs*, Cambridge University Press, 2001.
4. E.W. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematik*, 1, (1976),83-89.
5. M. Dyer, A. Frieze, R. Kannan, A random polynomial algorithm for approximating the volume of convex bodies, *Journal of the ACM*, (1991), 1-17.
6. P. Erdös, A. Rényi, On the evolution of random graph, *Publ. Math. Inst. Hung. Acad. Sci.*, 5, (1960), 17-61.
7. R.W. Floyd, Algorithm 97: Shortest Path, *Communications of the ACM*, 5, (1962), 345.
8. D.B. Johnson, Efficient algorithms for shortest paths in sparse networks, *Journal of the ACM*, 24, (1977), 1-13.
9. N.L. Johnson, S. Kotz, *Urn Models and Their Applications*, John Wiley, New York, 1977.
10. D.R. Karger, Better Random Sampling Algorithms for Flows in Undirected Graphs. *Proc. SODA* 1998.
11. D.R. Karger, M.S. Levine, Random sampling in residual graphs, *Proc. ACM STOC*, 2002.
12. R.M. Karp, M. Luby, Monte Carlo algorithms for the planar multi-terminal network reliability problem, *Journal of Complexity*, 1, (1985), 45-64.
13. R.M. Karp, M. Luby, N. Madras, Monte Carlo approximation algorithms for enumeration problems, *Journal of Algorithms*, 10, (1989), 429-448.
14. R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

15. K. Mulmuley, *Computational Geometry, An Introduction Through Randomized Algorithms*, Prentice Hall, 1994.
16. Telecordia. Private Communication, 2002.
17. V.V. Vazirani, Approximation Algorithms, Springer Verlag, 2001.
18. S. Warshall, A theorem on Boolean matrices, *Journal of the ACM*, 9, (1962), 11-21.