Suggesting Ghost Edges for a Smaller World

Manos Papagelis
University of Toronto
papaggel@cs.toronto.edu

Francesco Bonchi
Yahoo! Research
bonchi@yahoo-inc.com

Aristides Gionis
Yahoo! Research
gionis@yahoo-inc.com

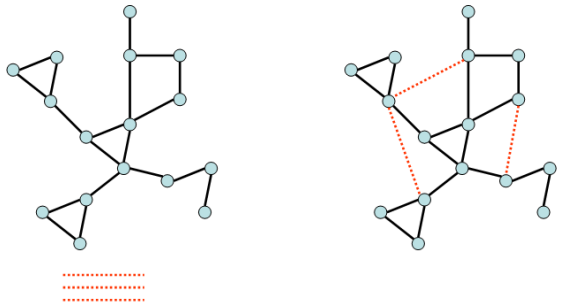
ABSTRACT

Small changes in the network topology can have dramatic effects on its capacity to disseminate information. In this paper, we consider the problem of adding a small number of *ghost edges* in the network in order to minimize the average shortest-path distance between nodes, towards a smaller-world network. We formalize the problem of suggesting ghost edges and we propose a novel method for quickly evaluating the importance of ghost edges in sparse graphs. Through experiments on real and synthetic data sets, we demonstrate that our approach performs very well, for a varying range of conditions, and it outperforms sensible baselines.

Categories and Subject Descriptors [**H.3.4**]: Systems and Software - *Information networks*
General Terms Algorithms, Peformance, Human Factors
Keywords Graph Augmentation, Social Networks, Small World

1. INTRODUCTION

The way that information disseminates in a social network depends, to a great extent, on its topology. As a matter of fact, small changes in the network topology can have dramatic effects on how efficiently information spreads among individuals. In this paper, we consider the problem of adding a small number of *ghost edges*, i.e., edges between existing nodes in the graph that could exist but have not yet been realized, with the objective of improving the network efficiency of information propagation. While having plenty of choices for measuring the network efficiency, we focus on a structural feature, namely the *average all-pairs shortest path distance* $\bar{L}(G)$. Minimizing $\bar{L}(G)$ gives smaller-world networks [4], in which information can travel from one node to another by traversing a smaller number of intermediate edges, and distinguishes an easily negotiable network from one which is inefficient, with a network with shorter $\bar{L}(G)$ being more desirable. Now, let $G(V, E)$ be a connected undirected graph and assume that we are allowed to augment G with a small number of edges. Note that, adding edges in a network results in altering some of the shortest paths for some pairs of nodes in the graph. It is worth noting that while adding long-haul edges may bring a large gain in terms of connectivity for some node pairs (the new shortest path connecting them will be much shorter), long-haul edges are likely to affect the shortest path of only a few pairs. On the other hand, suggesting short-haul, intra-community edges is likely to affect the shortest path length of many node pairs, but the gain in connectivity for each of them might be limited (the new shortest path connecting them will not be much shorter). Therefore in trying to suggest edges to be added to G we are facing an interesting trade-off. We are now in position to formally define the problem of interest in this paper:



(a) $G = (V, E)$, $|R| = k$ (b) $G_{aug} = (V, E \cup R)$

Figure 1: The problem: (a) the input consists of G and an integer k, (b) the augmented graph G_{aug} is a possible output. The goal is to find the set of ghost edges R that minimizes $\bar{L}(G_{aug})$.

PROBLEM 1. *Let $G(V, E)$ be a connected, undirected graph, and let $\bar{L}(G)$ be the average all-pairs shortest path length in G. Let $R \subseteq V \times V \setminus E$ where $|R| \leq k$ be a set of candidate ghost edges for augmentation. If $G_{aug} = (V, E \cup R)$ is the augmented graph of G, then the objective is, given G and an integer k, to determine the subset of edges R that minimizes $\bar{L}(G_{aug})$.*

This is a challenging problem, as traditional graph augmentation problems, where we ask to find a minimum-cost set of edges to add to a graph to satisfy a specified property, such as biconnectivity, have been shown to be NP-complete [2, 7]. In the case that the problem is considered with a hard limit on the number of edges to be added, such as in [1, 5], authors observe a relation between the single-source version of the problem, where we want to minimize the largest distance from a given source vertex, and the well-known k-median problem. In [1], the focus is on minimizing the diameter of the network, while in [5] the suggested edges are all incident to a single node. The problem draws connections, as well, to the *link revival* problem [6]. Our methods can be adopted in a variety of strategies for speeding up communication or increasing the reliability of a network, and as such, can form the core component of many interesting applications. To name a few, they can be used in algorithms that suggest friends in a social network with the global objective of bringing anyone closer enabling efficient information dissemination or in algorithms that suggest new edges in a network with the global objective of destroying its clustering structure.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

2. ALGORITHMS

In this section, we describe algorithms that solve the problem of interest; they take as input a graph G and a constant k and return a set R of the top-k ghost edges for augmentation. Central to the algorithms is the concept of the utility of an edge. Let U_{xy} represent a measure of the utility of adding an edge $(x, y) \in C = V \times V \setminus E$ in G. The measure of the utility U_{xy} represents the gain in decreasing the average shortest path length of the network. To evaluate U_{xy}, we need to recompute the shortest paths between all pairs of vertices $s, t \in V$. This is because a previously shortest path connecting s, t may now need to include the newly added edge (x, y). Let $\ell(x, y)$ and $\ell'(x, y)$ represent the length of the shortest path between x and y in G and G_{aug} respectively. It is easy to see that addition of an edge (x, y) will always decrease the average shortest path length of the network. We can compute the utility U_{xy} of adding an edge (x, y) by summing up the differences in the shortest path length of any pair $s, t \in V$ in G and G_{aug} respectively:

$$U_{xy} = \sum_{(s,t) \in V \times V \setminus E} (\ell(s, t) - \ell'(s, t))$$

2.1 Greedy Algorithm

We first describe a general greedy algorithm where all possible candidate ghost edges are considered and an edge is selected in each iteration that lowers the average shortest path length of the graph as much as possible (locally optimal choice). The algorithm operates in k rounds. In each round i, the algorithm determines the edge with the highest utility and adds it to the set of the top-k edges to be suggested for augmentation. Equivalently, this means that the edge selected in round i is the one that minimizes the average shortest path length in this round and for this graph. To find this edge, the algorithm iterates over all candidate ghost edges $e \in C$ and estimates the associated utility U_e by adding the edge to the graph and recomputing the new average shortest path length in the augmented graph. Computation of the utility of each candidate ghost edge takes $O(n^3)$ time (equivalent to running the Floyd-Warshall algorithm one time), there are $|C| = n(n-1)/2$ candidate ghost edges to be considered in each round i, i.e., $O(n^2)$, and there are k rounds, thus the Greedy algorithm takes $O(kn^2n^3)$ or $O(kn^5)$ time to complete. However, it is possible to omit k if instead of computing the best ghost edge, one at a time, we use the already computed edge utilities from the first iteration and determine the best k ghost edges all at once (batch processing). While this variant is expected to be faster, it will affect the quality of edges suggested, as inclusion of a candidate ghost edge may lower the utility of a subsequent ghost edge.

2.2 Heuristic Method

It's easy to see, that in the general case the addition of an edge (x, y) to G, alters the shortest path lengths for many pairs of vertices (s, t) with $s \in V$ and $t \in V$. However, a large number of the shortest paths are not affected by the addition. Let us now make a simplifying assumption that if $p_{st} =< x_1, x_2, ..., x_n >$ is the shortest path between s and t in G then both x and y belong in p_{st} (i.e., they are nodes in the existing shortest path between s and t) (see Figure 2). The premise of our assumption is that at least as good candidate ghost edges can still be found despite the fact that we constrain the set of (s, t) pairs that depend on edge (x, y) to be pairs that their existing (s, t)-shortest path in G passes through both nodes x and y. Let this smaller set of (s, t) pairs that depend on a candidate edge (x, y) be $D(x, y)$. It's easy to see that this assumption does not always hold (see Figure 2), however, this assumption

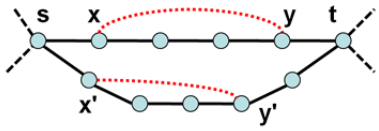


Figure 2: The (s, t) pair would benefit from both the ghost edge (x, y) and (x', y'). But due to our simplifying assumption (s, t) belongs to $D(x, y)$ and not to $D(x', y')$.

is unlikely to harm the final solution set. This is especially true in the case of sparse graphs where the shortest paths between any two nodes are more prominent. We show in the experimental evaluation that our algorithm performs very well in practice. This assumption suggests large savings in the computation of the utility U_{xy}, since now we need to only sum up the differences in the shortest path length of a much smaller set of pairs $(s, t) \in D(x, y)$, as follows:

$$\begin{aligned} U_{xy} &= \sum_{(s,t) \in D(x,y)} (\ell(s, t) - \ell'(s, t)) \\ &= \sum_{(s,t) \in D(x,y)} ((\ell(s, x) + \ell(x, y) + \ell(y, t)) \\ &\qquad - (\ell(s, x) + \ell'(x, y) + \ell(y, t))) \\ &= \sum_{(s,t) \in D(x,y)} (\ell(x, y) - \ell'(x, y)) \\ &= \sum_{(s,t) \in D(x,y)} (\ell(x, y) - 1) \end{aligned}$$

Note that the utility U_{xy} eventually depends on two factors: (a) the size of the set $D(x, y)$ (i.e., the number of (s, t) shortest paths affected by (x, y) in G_{aug}), and (b) the length $\ell(x, y)$ in G (i.e., the shortest path length of x, y in G).

We are now in position to describe our heuristic algorithm (see Algorithm 1) that can speed up the process of finding a good enough solution to the problem by incrementally computing the utility U_{xy} of each candidate edge. The algorithm first uses a modification of the classic Johnson's Algorithm that does not only provide the lengths of the paths between all pairs of vertices, but also reconstructs the actual path between any two endpoint vertices. The algorithm works in $O(n^3)$ time and stores all (s, t) paths of G in P. For each path $p \in P$ the algorithm uses a δ-size window to determine the x and y endpoints of a candidate edge (x, y) and incrementally updates its utility U_{xy}. Each time a ghost edge (x, y) is encountered in a path p an increment equal to $(\delta - 1)$ is contributed to its total utility. Note that δ is a variable that defines the size of the current δ-size window, and that for each path $p \in P$ takes values that range from $\delta = 2$ (i.e., consider ghost edges between not adjacent nodes in G) to $\delta = l$, where l is the length of the current path p. It's easy to see that the maximum δ to be considered in the whole process will be equal to the *diameter* d of G, which is defined as the longest shortest path between any two nodes in G, so the range of values of δ is $2 \leq \delta \leq d$. In the end of the procedure, for each candidate ghost edge (x, y) considered, the algorithm has associated a utility value U_{xy}. Finally, the algorithm determines the top-k candidate edges by sorting the edges on their U_{xy} value and returning the first k.

Algorithm 1 takes $O(n^3) + O(n^2\bar{L}(G)\bar{L}(G))$ or just $O(n^3))$ time to complete. First, it requires $O(n^3)$ time to find all paths in G (equivalent to running the Floyd-Warshall algorithm one time). Then for each path (i.e., $O(n^2)$) needs to vary the δ-size of the window (i.e, $O(\bar{L}(G))$) and slide the window over the path p (i.e., $O(\bar{L}(G))$).

Algorithm 1 Heuristic (G, k)

```
1: procedure HEURISTIC(G, k)
2:     G: Social Graph G(V, E)
3:     C: The set of candidate edges for addition
4:     TopK ← ∅
5:     C ← ∅
6:     P = getAllPairsShortestPaths(G)
7:     for all p ∈ P do
8:         l = length(p)
9:         /* Variable size of window */
10:        for δ = 2 to l do
11:            /* Slide a δ-size window over nodes of p */
12:            for i = 0 to l − δ do
13:                x = p[i]
14:                y = p[i + δ]
15:                if (x, y) ∉ C then
16:                    U_xy = (δ − 1)
17:                    add < (x, y), U_xy > in C
18:                else
19:                    update < (x, y), U_xy + (δ − 1) > in C
20:    Sort C on the U_xy value descending
21:    return Top − k in C
```

2.3 Heuristic Sampling-based Method

Our *Heuristic* algorithm is much faster than the *Greedy* algorithm, but it's still impractical for very large graphs as it requires that all-pairs shortest paths can be efficiently computed ($O(n^3)$). In this section we describe a sampling-based variant of our heuristic, based on the assumption that even a limited number of shortest paths could provide a good approximation of the utility of each edge U_{xy}. Our sampling method works as follows: we first uniformly sample Q sources $Q = \{q_1, ..., q_q\}$; then, starting from a source q_i, $i = 1, ..., q$, we execute the Dijkstra algorithm and compute the single source shortest path tree from all q_i to all nodes x in the graph. Ideally, we would like to sample uniformly at random a few shortest paths from the set of all available ones. However, the way we perform the sampling is likely to introduce bias, as some paths are more likely to exist in the shortest path trees of sampled sources [3]. But, in our setting, we are only interested in suggesting a few ghost edges with very high utility, and these edges are expected to be part of many shortest paths, so it's expected that they will be sufficiently represented in the sampled paths. We experimentally assessed the effect of this bias and found it to be minor. Once we have obtained a set of shortest paths using our sampling method, we feed the Algorithm 1 (line 6). The rest of the algorithm remains the same. The point is that the above computation is relatively efficient, since instead of computing all-pairs shortest paths, computes q times the single-source shortest-path Dijkstra. So the overall running time is $O(q(m + nlogn))$.

3. EXPERIMENTAL EVALUATION

For the needs of our experimental evaluation we consider both real (*karate*, *dolphins*, *football*, *netscience*, *yeast*) and synthetic (*ba-s*, *ba-m*, *ba-l*) *network topologies* that represent connected, undirected, unweighted graphs (Table 1). We assess the performance of the various methods according to *accuracy* (how well a method suggests a set of k ghost edges) and *efficiency* (how fast a method suggests a set of k ghost edges) measures. To assess the accuracy we define a *gain* metric, which expresses the percentage change of $\bar{L}(G)$ before and after augmention. To assess the efficiency we measure the execution time, in *milliseconds*. Given G and k, we assess the performance of the following methods:

Table 1: Networks

Name	Nodes	Edges	Ghost Edges	Density	L(G)
karate	34	78	483	0.139	2.408
dolphins	62	159	1732	0.084	3.357
football	115	616	5939	0.093	2.508
netscience	379	914	70717	0.012	6.042
yeast	2224	7049	2464927	0.003	4.376
ba-s	50	49	1176	0.040	4.640
ba-m	500	499	124251	0.004	7.398
ba-l	1000	999	498501	0.002	8.132

Greedy-S: This is the greedy method described in paragraph 2.1.
Greedy-B: This is the *batch variant* of the *Greedy-S* method.
Heuristic: This is our heuristic method described in Algorithm 1.
Random: This is a baseline method that randomly selects k ghost edges to add in G.
Rand Star: This is a more sensible baseline method that selects k ghost edges to add in G that resemble the topology of a star. The intuition behind this method is that stars are network topologies with very small average shortest path distance [5].

We evaluate the accuracy and efficiency of our heuristic method against the greedy methods, for a varying range of network topologies and number of ghost edges k to be added. More specifically, we experiment with *karate*, *dolphins*, *football* and *ba-s* and we want to suggest 1, 5, 10 and 20 ghost edges. We had to limit our experiments on very small networks; this is because *Greedy-S* is extremely slow as we discuss below.

Accuracy: Figure 3 presents the accuracy results for the various networks. In all instances, *Greedy-S* performs better than any other method; this is because it evaluates any of the k ghost edges one at a time. Even if this method is still suboptimal, it ensures that at each iteration the correct utility for each of the subsequent ghost edges is computed, taking into consideration the previously suggested edges. As such, it avoids suggesting ghost edges that are gainful for overlapping sets of st-shortest paths. On the other hand, *Random* performs poorly in all instances, as expected, and forms a baseline for the performance of the other methods. The accuracy performance of the rest three methods, *Greedy-B*, *Heuristic*, *Rand Star* is comparable. They all perform worse than *Greedy-S*, and outperform *Random*. However, from the three, *Rand Star*, is inconsistent, as its performance largely depends on the random process with which the center node and its incident edges are selected. Overall, in terms of accuracy, the performance of *Heuristic* is always comparable and even better than that of the *Greedy-B* method. On another point it is important to note that the difference between the performance of *Greedy-S* and the other methods is becoming more evident as the number of ghost edges to be added, k, is increasing. This is to be expected; as all the other methods suggest edges on a batch way, the larger the number of suggested edges is, the larger the likelihood that these edges share overlapping st-shortest paths will be, rendering the overall gain to be smaller.

Efficiency: Table 2 presents the efficiency results for the various networks. Note that, for all methods other than the *Greedy-S* it's enough to report only one value for each network, since these methods will evaluate the utility of all edges only once. On the other hand, for the case of *Greedy-S* we report values for all different instantiations of k (i.e., *Greedy-S-1*, *Greedy-S-5*, *Greedy-S-10*, *Greedy-S-20*). It becomes obvious from Table 2 that *Greedy-S* is very inefficient, as it suggests one edge at a time, and in fact, it is (almost) k times slower than *Greedy-B*. More importantly, our method, *Heuristic*, runs multitude times faster than *Greedy-B*. The gain in effi-

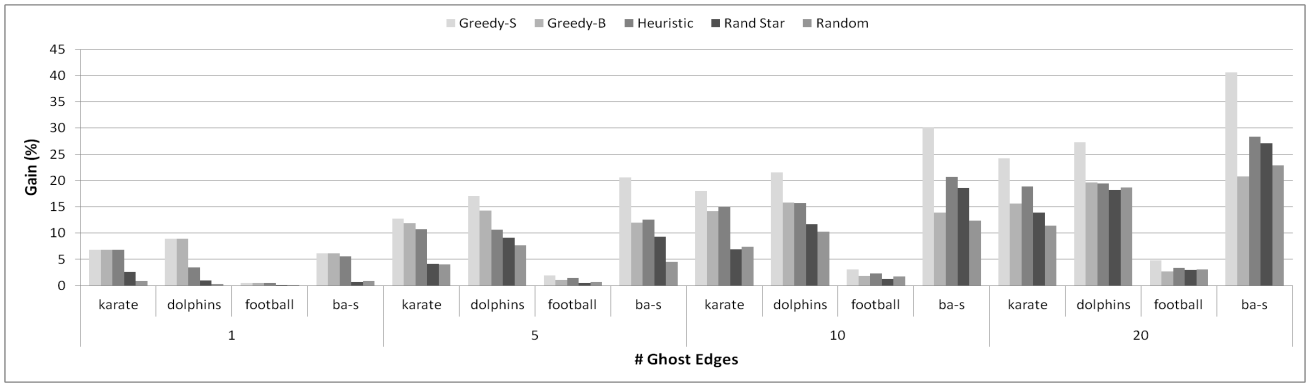


Figure 3: Heuristic Performance: Accuracy

Table 2: Heuristic Performance: Efficiency

Method	karate	dolphins	football	ba-s
Greedy-S-1	3288	39931	809550	12745
Greedy-S-5	16412	205175	4046788	62284
Greedy-S-10	32372	403252	8094511	127333
Greedy-S-20	65359	816703	16191000	257063
Greedy-B	3891	55203	956579	13188
Heuristic	**47**	**110**	**375**	**32**
Random	2	2	2	2
Rand Star	2	2	2	2

Table 3: Efficiency of Heuristic in Figure 5 (msec)

dolphins	netscience	yeast	ba-s	ba-m	ba-l
63	3156	133078	31	4828	25078

ciency is due to the fact that, while *Greedy-B* will need to compute the utility of all possible ghost edges, *Heuristic* will constrain the search space and look for ghost edges that connect nodes of already existing shortest paths. As such, our method suggests huge savings in the computation of the importance of ghost edges.

Heuristic Sampling-based (HSB) Performance: We have demonstrated that *Heuristic* is much faster than *Greedy-S* and *Greedy-B*, while maintaining high levels of accuracy. Now, we assess the performance of the sampling based variant of our heuristic *HSB* against the *Heuristic* method that serves as the ground truth and the *Random* method that serves as the baseline for our sampling based method. We experiment in real and synthetic networks of varying size and for varying number of ghost edges to be added, as well as, varying sample size. In particular, for each network, we experiment with sample sizes of 10%, 20% and 30% of the total nodes in network that define the methods *HSB (10%)*, *HSB (20%)* and *HSB (30%)* respectively. Figure 4 presents the accurracy results for the various instances of real and synthetic networks when adding 5% of the total number of ghost edges. In all cases, *HSB* has a better accuracy than *Random* and it's accuracy is slightly lower than the accuracy of *Heuristic*. Moreover, the accuracy of HSB increases with the sample size; this behavior is expected, as more shortest paths are evaluated in larger samples. Figure 5 presents the efficiency results for the various methods in varying networks. Note that for demonstration issues, we report on the relative time, where the execution time of the *Heuristic* represents the time of the slower method. We also report the actual execution times of *Heuristic* in Table 3 to complement Figure 5. For example, while the *Heuristic* method requires almost 25sec for suggesting ghost edges in the *ba-l* network, *HSB-10* requires less than 5sec (i.e., less than 20% relative time). Overall, *HSB* can be used to boost the performance of *Heuristic*, without duly affecting its accuracy, and as such, it's the method we propose to use for suggesting edges in larger networks.

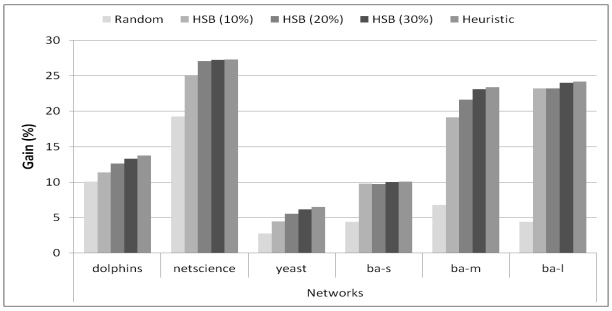


Figure 4: HSB Performance: Accuracy

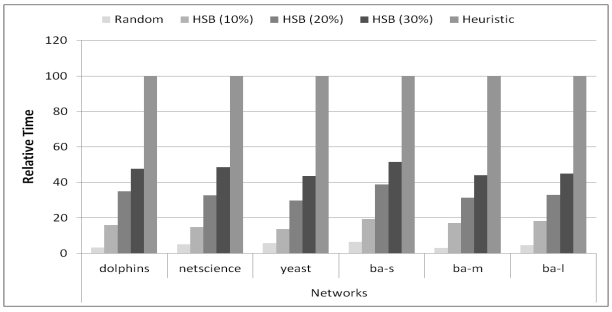


Figure 5: HSB Performance: Efficiency

4. REFERENCES

[1] E. D. Demaine and M. Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *SWAT*, 2010.

[2] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4), 1976.

[3] S. Kandula and R. Mahajan. Sampling biases in network path measurements and what to do about it. In *IMC*, 2009.

[4] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC*, 2000.

[5] A. Meyerson and B. Tagiku. Minimizing average shortest path distances via shortcut edge addition. In *APPROX-RANDOM*, 2009.

[6] Y. Tian, Q. He, Q. Zhao, X. Liu, and W.-c. Lee. Boosting social network connectivity with link revival. In *CIKM*, 2010.

[7] T. Watanabe and A. Nakamura. Edge-connectivity augmentation problems. *J. Comput. Syst. Sci.*, 35, 1987.