Effective and Efficient Graph Augmentation in Large Graphs

Ioanna Filippidou
Athens University of Economics and Business
76 Patission Street, Athens, Greece
filippidoui@aueb.gr

Yannis Kotidis
Athens University of Economics and Business
76 Patission Street, Athens, Greece
kotidis@aueb.gr

***Abstract*—**
The graph augmentation problem seeks to suggest new edges that, when added to an input graph, improve the overall connectivity of the nodes. For social network applications, the latter is typically computed as the average shortest path length in the network. In this work we first introduce one interesting variation of the problem that focuses on improving the connectivity between nodes belonging to a specific sub-graph (for instance nodes of the same class or users that share interests). Key to our method for solving the original graph augmentation problem and its suggested variation is an intuitive algorithm we propose. The algorithm operates by first constructing a summary graph that retains important structural properties of the input graph. Using this summary our algorithm computes an effective list of suggested short-cuts. Unlike existing techniques, the proposed algorithm does not require complex computations over the whole graph (such as the computation of all-pair shortest paths). This makes it applicable for larger graphs where existing proposals fail to operate. Our experimental results demonstrate the efficiency and effectiveness of our techniques on graphs of various sizes and characteristics.

I. Introduction

In this paper, we propose an algorithm that increases the connectivity of an input graph by considering the topology of the network itself. To do so we use a graph augmentation technique, where a small set of non-existing edges (often referred to as *short-cuts*) are selected and added to the graph in order to increase its connectivity and further improve its capacity to carry on various information propagation processes.

As is common in the literature, we evaluate network connectivity by the average shortest path distance of the nodes in the graph. The minimization of the average shortest path of a network is a desired property since it improves dynamic processes that are crucial to today's big and evolving networks. A well connected network is of value to both the users and the owners of the networked data and, thus, it is in their best interest to maintain and enhance this property. Network owners can benefit from small average shortest paths, since this is one of the key properties that helps viral marketing and information diffusion [17].

Beside the graph augmentation problem, in this work we also define one variation of the general problem termed as sub-graph augmentation policy. In this variation we seek to increase the connectivity between nodes of a specific group by adding short-cuts to the whole graph. These short-cuts, can significantly reduce the average shortest path between nodes of the sub-graph while at the same time improve the connectivity of the whole graph. The sub-graph variation of our algorithm can be used in cases where we need to improve the connectivity of specific nodes that have loose connections, or to bring closer nodes that belong to a specific group or have similar properties. For instance, users that belong to the same professional group in a social network want to be close to each other in order to be fast informed about topics of their profession.

For solving the general graph augmentation problem and the suggested sub-graph variation we use the same underlying algorithm proposed in this work, with minor modifications. The algorithm is based on the idea of creating a small summary of the graph, which retains some specific properties that help us make accurate short-cut suggestions. All the necessary calculations for the short-cut addition problem are performed solely on the formed summary. Our algorithm suggests short-cuts that have significant impact in the minimization of the average shortest path distance of the whole graph or the selected sub-graph, depending on the variant of the problem. Moreover, because it operates over a much smaller summary of the input graph, it can be used efficiently to augment graphs of realistic sizes, as our experiments demonstrate.

Although there has been some previous work on selecting edges to add to a graph so that specific criteria are met, such as minimizing the average all-pair shortest path distances, our work is the first to consider one important variation of the problem but also to suggest an algorithm that makes such calculations feasible, even in large graphs. Our contributions can be summarized as follows:

- We revisit the graph augmentation problem and introduce an interesting variation of it that seeks to increase the connectivity between nodes that form a sub-graph. As it is shown in our experiments, improving the connectivity of the sub-graph often has significant impact on the connectivity of the whole graph.
- We propose an intuitive algorithm that is based on building a small summary of the graph and doesn't require any computations of shortest path distances

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

over the whole graph structure. This property of the algorithm enables it to perform efficiently even in large graphs.
- We evaluate the accuracy and efficiency of our method using real and synthetic graphs of various sizes. Our results demonstrate that our techniques outperform existing algorithms by suggesting short-cuts that better improve the connectivity of the network for both variations of the augmentation problem considered. At the same time, our algorithm manages to compute the suggested set of short-cuts within seconds, even on graphs of larger sizes.

II. Problem Definition

In this section we first introduce some basic notation from graph theory that is used throughout the paper and then formally state our problems. A graph $G = (V, E)$ is a connected undirected simple graph where V is the set of vertices and E the set of edges. The length of the shortest path between two nodes u, v is denoted as $d(u, v)$. The sum of all pairs shortest path lengths is $L = \sum_{(u,v)\in V\times V, u\neq v} d(u, v)$. The average shortest path length over all pairs of nodes in the graph is defined as $\overline{L} = L/\binom{n}{2}$. $\overline{L}$ is a basic measurement of how close the nodes of the graph are.

Graph Augmentation Problem: The graph augmentation problem described in this work can be seen as the selection of k edges (short-cuts) to add in a graph G in order to maximize a utility function $g()$. We define as ASC the set of all candidate short-cut edges (i, j) between any non-adjacent nodes $i, j \in V$. Consider $SC \subseteq ASC$ with $|SC|$=k be a selection of k short-cut edges from ASC. We denote the resulted graph that is created by the addition of the short-cut edges in SC to G as the *augmented graph* $G_{aug} = (V, E \cup SC)$. In order to quantify the improved connectivity in the augmented graph G_{aug}, resulted from the addition of the short-cut edges in SC, we use the average shortest path length and we define $g(SC)$ as a function that computes the difference of the average all pairs shortest path length $\overline{L}$ in G and G_{aug}:

$$g(SC) = \overline{L}(G) - \overline{L}(G_{aug}) \geq 0 \tag{1}$$

Sub-graph Augmentation Problem: The sub-graph augmentation problem is defined as the selection of k edges (short-cuts) to add in a graph G in order to minimize the average shortest path length between a subset V_m of the nodes in V. Given V_m, we derive a sub-graph G_S of G that contains these selected nodes as well as all nodes and edges that belong to the shortest paths between all pairs of nodes in V_m, when these shortest paths are computed over the initial graph G. The average shortest path length in the sub-graph is defined as $\overline{LS} = LS/\binom{m}{2}$ where LS is the sum of all pairs shortest path lengths between nodes of V_m.

Intuitively, the sub-graph augmentation problem is a variation of the graph augmentation problem where we only focus on the connectivity between a certain subset of nodes from the graph. We define $gs(SC)$ as a function that maps a given subset of short-cut edges $SC \subseteq ASC$ to a non-negative number that represents the difference of the average all pairs shortest path length $\overline{LS}$ in G and G_{aug}:

$$gs(SC) = \overline{LS}(G) - \overline{LS}(G_{aug}) \geq 0 \tag{2}$$

In this work we consider the general graph augmentation problem where new short-cuts can be added anywhere in the network. An integer k, that is provided as input specifies the total number of short-cuts to be added in the graph. In the same way in the sub-graph augmentation problem k edges can be added between any nodes of the graph and not only between nodes of the sub-graph.

III. Graph Augmentation Algorithm

The proposed algorithm for both variations of the graph augmentation problem discussed in this work is based on building a small summary from the graph, on which we will perform all necessary calculations for the edge addition problem. The algorithm can be divided into two concrete phases that when combined will suggest the requested set of short-cuts to be added to the graph. We will fist describe these phases for the general graph augmentation policy and we will then discuss the necessary modifications for the sub-graph augmentation policy.

The first phase of the algorithm is the coarsening phase, where we iterate through the elements of the graph in order to construct a summary containing a small number of super-nodes formed by grouping together multiple graph nodes. Finding groups of nodes that are closely related but in the same time have loose external connections is the main goal of the coarsening phase. The proposed coarsening algorithm has low complexity as it mainly requires shorting the graph nodes in decreasing order by their degree.

The coarsening phase of the algorithm proceeds until a summary with a predefined number of super-nodes is returned. This number is denoted by an input parameter $numNodes$. At the beginning of the coarsening phase, the nodes of the provided graph G are sorted based on their degree in descending order. Then, the $numNodes$-highest degree nodes are removed from the list of the graph nodes and are added in the summary graph as *leader nodes*. These leader nodes constitute the initial set of super-nodes. Until all remaining nodes of the graph are assigned into one of the selected super-nodes, we repeat the following coarsening step. Beginning from the highest degree node in the summary graph (the list of super-nodes is ordered in decreasing order of the leaders' degrees), we include all its neighbours to the super-node and adjust the weight of the super-node with the number of added nodes. Beside that we also inform the super-node adjacency list, each time a new node from the graph is coarsened by a super-node. The

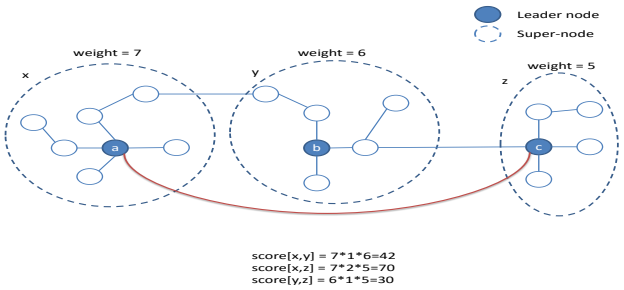

Figure 1. Coarsening of sample graph and selected short-cut

coarsening procedure continues with the remaining super-nodes, until the initial list of nodes is empty and, thus, all nodes have been assigned to super-nodes. At the end a summary of the graph is returned with the predefined number of $numNodes$ super-nodes. In Figure 1 we depict the result of the coarsening algorithm for $numNodes$=3 super-nodes. The resulting super-nodes are depicted with dotted circles and their weights are shown in the figure.

The second phase in our algorithm is the short-cut addition phase. It is executed after the coarsening procedure and seeks to locate which super-nodes we should connect with short-cuts in order to maximize the gain in the connectivity of the whole graph. At the beginning of the short-cut addition phase, we compute the distances of all pairs shortest paths of the summary graph returned by the coarsening phase. Next, the algorithm computes scores for every possible short-cut between the super-nodes of the summary graph. Let x, y be two super-nodes in the summary graph and let a, b be their leader nodes in the input graph, respectively. A short-cut (x, y) in the summary graph implies the addition of edge (a, b) in the input graph. The score of (x, y) is calculated by the following function:

$$score[x, y] = x.weight \times distance[x, y] \times y.weight \quad (3)$$

From the calculated scores we select the highest gain short-cut between two super-nodes that are not connected in the original graph and add the corresponding edges to the summary as well as the input graph. This procedure is repeated from the beginning until the required number of short-cuts have been created.

Continuing with the example of Figure 1, there are three super-nodes resulting in three candidate short-cuts among them in the summary graph (dotted nodes): (x, y), (x, z) and (y, z). The figure depicts the computed scores for these short-cuts. The final top-1 selection is the short-cut that connects super-nodes x and z. This short-cut is instantiated at the input graph as a new edge between leader nodes a and c.

Sub-graph Augmentation Policy: The only modification that is required in the algorithm for this policy is on how the weights of the super-nodes are computed. In particular, weights of nodes that are not part of the sub-graph are set to zero and the weights of the sub-graph nodes are set to 1. This way each formed super-node has a total weight indicating the number of nodes from the sub-graph that it includes.

Name	V	E	Avg. Degree	Av.Shortest Path	Diameter
dolphins	62	159	2.57	3.36	8
netscience	379	914	2.41	6.04	17
facebook	4,039	88,234	21.85	3.69	8
power	4,941	6,594	1.34	18.99	46
wingNodal	10,937	75,488	6.90	11.54	26
4elt	15,606	45,878	2.94	44.77	102
amazon0302	262,111	899,792	3.43	8.83	30
web − Stanford	255,265	1,941,926	7.61	6.78	75
web − NotreDame	325,729	1,090,108	3.35	7.17	46
auto	448,695	3,314,611	7.39	36.42	73
web − BerkStan	654,782	6,581,871	10.05	7.11	97
roadNet − CA	1,957,027	2,760,388	1.41	312.13	796

Table I
SUMMARY OF THE EVALUATION DATASETS

IV. EXPERIMENTS

A. Experimental Setup

For our evaluation we use a diverse collection of synthetic and real-world graphs with various sizes and different edge distributions selected from [7], [8], [12], [13], [19]. Table I summarizes the basic statistics about each graph used in our experiments. In cases where a graph is not connected, we extract its largest connected component and operate on it.

To assess the accuracy of the various methods we define a gain metric, which expresses the percentage change on the average shortest path length $\overline{L}$, in the graph G and in the augmented graph G_{aug} given by:

$$Gain = \frac{\overline{L}(G) - \overline{L}(G_{aug})}{\overline{L}(G)} \times 100$$

For the sub-graph version of our algorithm the gain metric expresses the percentage change on the average shortest path length between the nodes in V_m. Finally, we also report the time required in order to compute the short-cuts.

In the experiments on the smaller graphs in our datasets (the first six graphs of Table I) we use a default maximum value of 500 for the number of super-nodes that will be created in the coarsening phase of our algorithm and the number of added short-cuts used is k=16. For the larger datasets, the number of created super-nodes is 1000 and the number of short-cuts increases to k=100.

All algorithms have been implemented in JAVA, and all experiments were performed in a single machine equipped with an Intel Core i7-4700MQ CPU and 8GB of main memory.

B. Graph Augmentation Algorithm Evaluation

For the evaluation of our graph augmentation method we use two alternative algorithms newly suggested for the short-cut addition problem, the Path Screening Method [14] and the random method. The Path Screening Method works by decomposing the computation of the sum of the shortest path lengths into sub-problems. The method uses a modification

of Johnson's Algorithm [10] that not only computes the lengths of the all-pairs shortest paths but also reconstructs and stores them in a set P. Then, for each shortest path $p \in P$ it determines what are the (x, y) short-cuts that could shorten the current path p by sliding a variable size window of size δ over the paths nodes. Variable δ takes values that range from $\delta = 2$ (denoting short-cuts between non-adjacent nodes), up to $\delta = l$, where l is the length of the current path p. Each time a short-cut (x, y) is considered in a path p, an increment equal to $(\delta - 1)$ is contributed to its utility score. The procedure continues over all paths $p \in P$ and for each candidate short-cut (x, y) its total utility score is computed. At the end from the computed utility scores the algorithm suggests the top k short-cuts to be added to the graph. Even though the algorithm performs computations of all-pairs shortest paths over the whole graph, the gains in accuracy may be negatively affected from the suggestion of short-cuts in the latter step, since the insertion of a single short-cut in the graph may significantly alter the utility scores of all other computed short-cuts. Finally, the random method used as a baseline in the experiments, selects k short-cuts uniformly at random to add in the graph G.

In Figure 2 we present comparative results for the gain that each algorithm achieves on the set of small graphs. We notice that in all cases our coarsening algorithm performs much better than its competitors and manages to reduce the average shortest path length up to 40% with the addition of only 16 short-cuts to the corresponding graph.

Next, in Figure 3 we present comparative results for the gain of using the graph augmentation policy in large graphs. In this experiment we compare our solution only to the random algorithm due to performance limitations, since the alternative path screening method requires all pairs shortest path to be calculated and screened in order to operate. In contrast, our algorithm performs all the required computations in the summary graph produced by its coarsening step and can, thus, operate on graphs of much larger sizes.

In order to be able to evaluate the output of the graph augmentation algorithm in large graphs we perform sampling in the calculations of the all pair shortest paths, as has been suggested in the literature [14]. Since sampling at random a few shortest paths from the set of all available ones is rarely feasible [11], we sample uniformly at random a set $Q = \{q_1, \ldots, q_q\}$ of nodes from the graph. Then, for every node q_i, we compute the single source shortest path tree (using Breadth First Search) from q_i to all other nodes of the graph. Based only on the single source shortest paths of the sample nodes Q, we then calculate an estimate of the average shortest path of the graph.

We notice that in all the selected graphs our coarsening algorithm increases significantly the connectivity of the graph independently of the graph type and size. In most of the cases, it significantly outperforms the random method, which often can not produce measurable gains from the short-cuts it selects.

In Table II we present results for the small graph datasets of the time required (in seconds) for each algorithm to compute the proposed short-cuts. We notice that our algorithm outperforms the path screening method in all graphs and the difference increases significantly as the graph size increases. From this experiment it is evident that the path screening method is impractical for large graphs as it requires that all-pair shortest paths are computed ($O(n(logn + m))$ and then screened ($O(n^2\overline{L}(G)\overline{L}(G))$). Moreover, the space requirements for the path screening method are significant, since it requires that all shortest paths of the graph and all candidate short-cuts are held in main memory during the screening process.

Data set	Coarsening Method	P.Screening Method
dolphins	0.009	0.061
netscience	0.082	0.205
facebook	0.183	14.850
power	0.138	18.403
wingNodal	0.168	88.390
4elt	0.174	781.015

Table II
GRAPH AUGMENTATION METHODS PERFORMANCE (IN SEC) FOR SMALL GRAPHS

In table III, we present results of the time required (in seconds) for our algorithm to calculate the proposed short-cuts, for the larger graphs in our collection. The presented times include both the coarsening and the short-cut addition phase of our algorithm for k=100 short-cuts (instead of 16 used in the smaller graphs). We notice that even in bigger datasets our algorithm manages to suggest short-cuts that significantly increase the connectivity of the whole graph (Figure 3), in just a few seconds.

Data set	Coarsening Method
amazon0302	3.037
web-Stanford	9.675
web-NotreDame	6.809
auto	3.347
web-BerkStan	30.198
roadNet-CA	4.647

Table III
GRAPH AUGMENTATION METHODS PERFORMANCE (IN SEC) FOR LARGE GRAPHS

C. Sub-graph Augmentation Algorithm Evaluation

For the experiment in Figure 4 we have selected uniformly at random 100 nodes and we present results on the connectivity gain on the average shortest paths between the nodes of the sub-graph. At the same time we measure the connectivity gain the sub-graph augmentation algorithm achieves in the whole graph. In this experiment we have altered the random algorithm in order to select short-cuts only between nodes of the sub-graph, and measure the resulting sub-graph and graph connectivity gains. We notice that our algorithm not

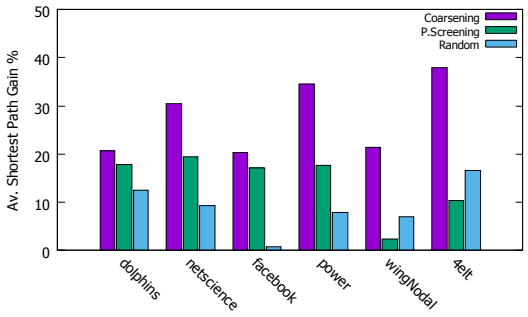

Figure 2. Graph augmentation policy gains in small graphs

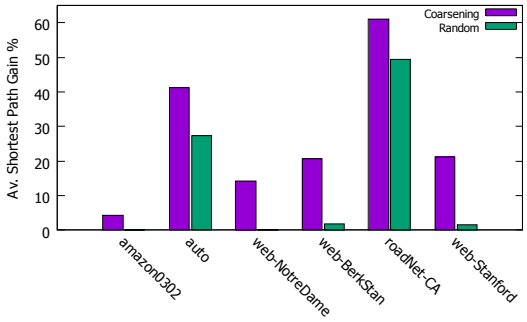

Figure 3. Graph augmentation policy gains in large graphs

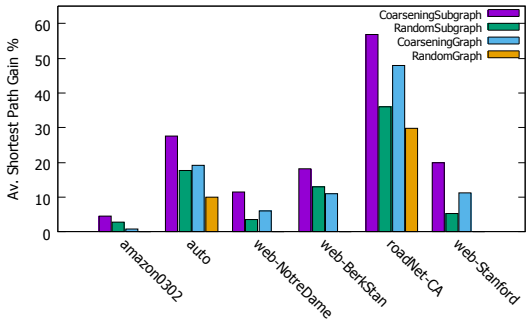

Figure 4. Sub-graph augmentation policy gains in large graphs

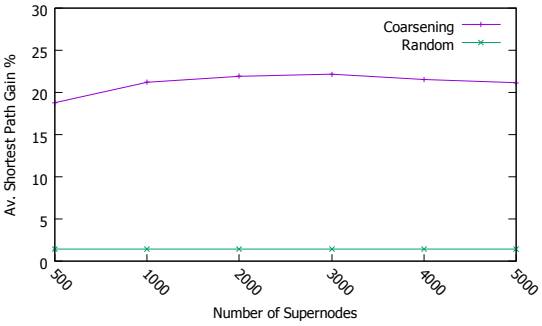

Figure 5. Sensitivity to the number of Super-nodes (web-Stanford graph)

only manages to achieve significant gains to the connectivity of the targeted sub-graph nodes but, at the same time, it helps increase the connectivity of the whole graph.

D. Algorithm Sensitivity Evaluation

In the experiment depicted in Figure 5, we use the web-Stanford graph and measure the gain that our algorithm achieves by altering the number of super-nodes that the summary graph will contain. The number of short-cuts added in all measurements are 100. We observe that the gain returned by the selected short-cuts is about the same, even when a small number of super-nodes are used. Making the summary graph a lot larger has a small impact on the quality of the short-cuts. This is because of the greedy nature of the short-cut selection process when a lot of super-nodes (of smaller size) are created. Keeping the summary small seems to prevent bad decisions during the short-cut addition phase. Furthermore, it has a positive impact on the overall execution time as depicted in Figure 6, where we measure separately the required time for the two phases of our algorithm, as the number of super-nodes increases. We notice that the second phase of our method runs significantly faster using smaller summary graphs.

In Figure 7, we measure the gain of our algorithm for different number of added short-cuts. Again we use the web-Stanford graph as an example, running the coarsening phase for 1000 super-nodes. In this experiment we notice that by just adding a few short-cuts to the graph (in the range 100-200) we manage to increase the connectivity of the 255K nodes in the graph significantly, up to 24%. The small drop in the gains depicted in the figure is due to the sampling process that we use to compute the average shortest path length in the large graphs.

V. Related Work

The graph augmentation problem seeks the minimum cost set of edges to add to a graph in order to improve a specific property, such as bi-connectivity or strong connectivity. These problems have been shown to be NP-complete in the case of connected graphs [6], while a number of approximation algorithms have been suggested. In a recent work [3], the authors studied the problem of minimizing the diameter of a graph by adding k short-cut edges and they developed constant-factor approximation algorithms for different variations of the problem. The problem of interest in this paper, is not the same as those in the aforementioned graph theoretical research, where connectivity of an undirected graph refers to the minimum number of disjoint paths that can be found between any pair of vertices. Shortcut selection is also strudied in the context of RDF data graphs in order to help reduce the cost of a selected set of user queries [4], [5].

The problem of minimizing the average all-pairs shortest path distance of the whole graph via edge addition was

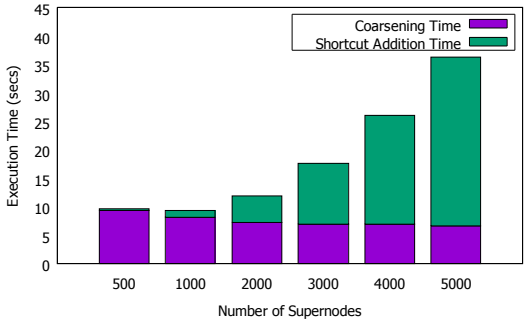

Figure 6. Algorithm execution times varying the number of super-nodes (web-Stanford graph)

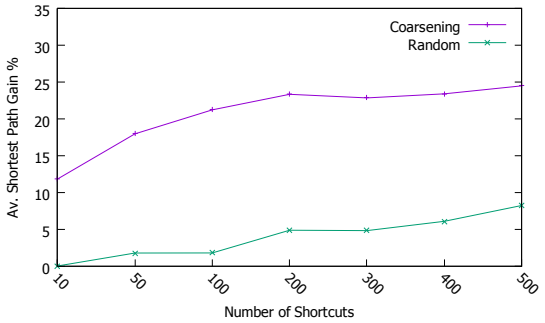

Figure 7. Sensitivity to the number of short-cuts (web-Stanford graph)

previously studied in [14], [15]. In [16], a variation of the problem where the set of candidate edges is given as an input is considered, and the goal is to select a subset of k of them. The problem is also related to the average closeness centrality of nodes in the graph. In [9], the problem of finding the k edges whose addition to the graph maximizes the centrality of a specific node is studied. The same problem was considered in [2], where they show that the greedy algorithm provides a tight $(1 - 1/\epsilon)$ approximation factor.

Our problem is also related link prediction and recommendation, in which the objective is to predict which new interactions among members of a social network are likely to occur in the near future. In [1], the authors address this problem in social networks by performing supervised random walks on the graph. In a similar context, the work of [18] studies the link revival problem, where the objective is to turn already existing edges with a few interactions to be more active so that the resulted connections will improve the social network connectivity.

VI. Conclusions

In this paper we first defined one variation of the general graph augmentation problem: the sub-graph augmentation policy and then introduced an intuitive algorithm that can perform in both variations of the suggested problem. The algorithm at its first step coarsens the whole graph in a way that the created summary preserves information that are important for the short-cut suggestion process. Subsequently, all the necessary calculations are performed using this summary. Our experimental evaluation demonstrated that our algorithm suggests more effective short-cuts than prior techniques to both variations of the problem. At the same time, the proposed algorithm has low time and space complexity and can process graphs with millions of nodes and edges.

References

[1] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

[2] P. Crescenzi, G. D'Angelo, L. Severini, and Y. Velaj. Greedily Improving Our Own Centrality in A Network. In *SEA*, 2015.

[3] E. D. Demaine and M. Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *SWAT 2010, Bergen, Norway*, pages 420–431, 2010.

[4] V. Dritsou, P. Constantopoulos, A. Deligiannakis, and Y. Kotidis. Optimizing Query Shortcuts in RDF Databases. In *ESWC*, pages 77–92, 2011.

[5] V. Dritsou, P. Constantopoulos, A. Deligiannakis, and Y. Kotidis. Shortcut selection in RDF databases. In *DESWeb Workshop of ICDE*, pages 194–199, 2011.

[6] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4):653–665, 1976.

[7] I. Filippidou and Y. Kotidis. Online and On-demand Partitioning of Streaming Graphs. In *IEEE International Conference on Big Data, Santa Clara, CA*, pages 4–13, 2015.

[8] I. Filippidou and Y. Kotidis. Online Partitioning of Multi-Labeled Graphs. In *GRADES 2015, Melbourne, VIC, Australia, May 31 - June 4, 2015*, pages 3:1–3:6, 2015.

[9] V. Ishakian, D. Erdös, E. Terzi, and A. Bestavros. A framework for the evaluation and management of network centrality. In *SIAM, Anaheim, CA*, pages 427–438, 2012.

[10] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, Jan. 1977.

[11] S. Kandula and R. Mahajan. Sampling biases in network path measurements and what to do about it. In *IMC, Chicago, Illinois, USA, November 4-6, 2009*, pages 156–169, 2009.

[12] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[13] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74:036104, Sep 2006.

[14] M. Papagelis. Refining social graph connectivity via shortcut edge addition. *TKDD*, 10(2):12, 2015.

[15] M. Papagelis, F. Bonchi, and A. Gionis. Suggesting ghost edges for a smaller world. In *CIKM*, pages 2305–2308, 2011.

[16] N. Parotsidis, E. Pitoura, and P. Tsaparas. Selecting shortcuts for a smaller world. In *SIAM 2015*, pages 28–36, 2015.

[17] N. Parotsidis, E. Pitoura, and P. Tsaparas. Centrality-aware link recommendations. In *WSDM*, pages 503–512, 2016.

[18] Y. Tian, Q. He, Q. Zhao, X. Liu, and W. Lee. Boosting social network connectivity with link revival. In *CIKM 2010, Toronto, Ontario, Canada*, pages 589–598, 2010.

[19] D. J. Watts and S. H. Strogatz. Collective dynamics of'small-world'networks. *Nature*, 393(6684):409–10, 1998.