On Mining Distances in Large-Scale Dynamic Graphs

Serafino Cicerone, Mattia D'Emidio, Daniele Frigioni

Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Via Vetoio, I–67100, L'Aquila, Italy.
{serafino.cicerone, mattia.demidio, daniele.frigioni}@univaq.it

Abstract. Applications relying on processing *large-scale graphs* have experienced an exponential diffusion in the recent past. In this domain, mining shortest-path relationships is one of the most basic primitives. Classic approaches fail at being practical when graphs are very large in size, hence in the last few years there has been an increasing demand for more and more efficient methods to implement this fundamental feature. Currently, computing a 2-hop cover labeling is considered the best approach in this sense. Its main limit is not being suited for in *dynamic* scenarios, i.e. when the graph can undergo changes over time. Few solutions have been proposed for dynamic 2-hop cover labelings. We overview these solutions and survey some recent developments and open problems.

1 Introduction

Mining shortest-path relationships is considered a fundamental operation on graph data, as it is a building block of some of the most important applications in modern networked systems, e.g. social networks analysis [10], route planning [9], intelligent transport systems [3]. Of particular relevance is the problem of mining *distances* (weights of shortest paths) since this information can be exploited for a variety of prominent purposes, e.g. community detection or web analytics.

Baseline methods for answering to *distance queries* are Breadth First Search (BFS) for unweighted graphs, and Dijkstra's algorithm for weighted ones. These methods yield unsustainable *query times* in the *large-scale graphs* arising from the mentioned applications. For this reason, many smarter approaches have been proposed (e.g., [7,9]), adopting the common strategy of preprocessing the graph to speed-up the query phase. Among them, the 2-hop cover labeling is based on the idea of representing the shortest paths of a graph as the concatenation at an intermediate so-called *hub node* of the two shortest paths emanating from the corresponding endpoints. The label at each node contains the length of a shortest path towards each hub and, to retrieve the distance between two nodes, it suffices to join their labels and find a common hub minimizing the sum of the two distances. The difficult point here is to find a small set of hubs covering a shortest path for each pair of nodes in the graph, since in this way the corresponding labeling is compact. Indeed, finding a minimum-size set of hubs is NP-hard [4].

Nevertheless, the 2-hop cover labeling is considered the best scheme in practice, since: i) it is general (it can be applied to all kinds of graphs [8,9]); ii) even in billion-nodes networks, it combines extremely low query times with affordable space overhead and reasonable preprocessing effort [1]; iii) several practical heuristic/approximation algorithms for computing it are known [4,9]. A further level of complexity is that real-world networks are dynamic and every time the network is subject to an update (e.g., an edge removal), the preprocessed data must be updated to keep queries correct. Preprocessing the graph from scratch after an update is not a reasonable strategy, as it requires large amounts of computational time [8]. In the recent past, researchers have worked to adapt known solutions to function in dynamic scenarios [2,7,8], i.e. to devise *dynamic algorithms* able to update the preprocessed data to reflect graph changes.

In particular, in [2] an *incremental algorithm* for 2-hop cover labelings under *edge additions* and *edge weight decreases* only is introduced. Despite having the same worst-case complexity of the best from scratch counterpart, the algorithm is experimentally shown to behave very well in practice. In [8] a *decremental algorithm* for 2-hop cover labelings under *edge removals* and *edge weight increases* only is given. In terms of worst-case complexity, the algorithm is worse than the best from scratch algorithm, however major experimental evidences of the practical efficiency of the method are provided. In addition, the authors combine and analyze their algorithm and the algorithm of [2], thus obtaining the first *fully dynamic* solution, able to update 2-hop cover labelings under any kind of update.

Notwithstanding said research efforts, practical fully dynamic algorithms for maintaining 2-hop cover labelings on general, large-scale graphs, are still missing. In this paper, we overview existing dynamic solutions and highlight the issues they exhibit that need to be tackled to make them more general and efficient. Alongside, we briefly describe some ideas to attack these open problems.

2 Background

Given an undirected unweighted[1] graph $G = (V, E)$, we denote by $d(u, v)$ the distance between nodes u and v, i.e. the number of edges in a shortest path between u and v in G. If u and v are not connected, then $d(u, v) = \infty$. For each node v of G, the *label* $L(v)$ of v is a set of pairs (*entries*) (u, δ_{uv}), where u is a node in V and $\delta_{uv} = d(u, v)$. The set $\{L(v)\}_{v \in V}$ is referred to as a *labeling* of G. Label entries can be used to answer to a *query* on the distance between two nodes s and t as follows: $\textsc{Query}(s, t, L) = \min_{v \in V}\{\delta_{sv} + \delta_{vt} \mid v \in L(s) \wedge v \in L(t)\}$ if $L(s) \cap L(t) \neq \emptyset$, while $\textsc{Query}(s, t, L) = \infty$ otherwise. A labeling $L(G)$ is a *2-hop cover* of G if, for each pair $s, t \in V$, $L(s) \cap L(t)$ contains at least a node u in a shortest path between s and t, called *hub* node of pair (s, t). In this case: i) the *hub* is said to *cover* pair (s, t); ii) the pair is said to be *covered* by L; iii) the labeling is said to satisfy the *cover property* for G; iv) $\textsc{Query}(s, t, L) = d(s, t)$ [4]. A labeling L of G is *minimal* if and only if, for

[1] All methods discussed in the paper can be extended to directed and weighted graphs.

each $v \in V$ and for each $(u, \delta_{uv}) \in L(v)$, there exist two nodes s, t such that $\textsc{Query}(s, t, L') \neq d(s, t)$, where L' is obtained by removing (u, δ_{uv}) from $L(v)$.

Computing a labeling of minimum size is NP–Hard. However, several heuristics exist for computing practical (minimal) approximations [4], among them those in [1,9] achieve considerably better scalability than others. The common strategy adopted by these methods is as follows. First, an ordering $\{v_1, v_2, \ldots, v_n\}$ of the nodes (according to some centrality measure) is computed. Then, starting from an empty labeling L_0, for each $v_k \in \{v_1, v_2, \ldots, v_n\}$ a BFS rooted at v_k is performed and, whenever a node u is reached with distance δ, the algorithm checks whether $\textsc{Query}(v_k, u, L_{k-1}) \leq \delta$. If this condition holds, then L_{k-1} contains a hub for (v_k, u) and for all pairs (v_k, x) such that there exists a shortest path between v_k and x passing through node u. Therefore, the BFS rooted at v_k is *pruned* at u. Otherwise, the algorithm sets $L_k(u) = L_{k-1}(u) \cup \{(v_k, d(v_k, u))\}$ and continues. It can be proven that L_n is a 2-hop cover labeling [1].

3 Dynamic Algorithms

In this section we summarize the incremental algorithm of [2], denoted as IncPLL, and the decremental algorithm of [8], denoted as DecPLL.

Incremental. IncPLL is able to update a 2-hop cover labeling L of a graph of G to consider a new edge $\{a, b\}$ that was previously absent. If G' is the new graph, i.e. G plus $\{a, b\}$, IncPLL computes a 2-hop cover labeling L' of G' by updating L. The algorithm is based on two insights: (i) if the distance between two nodes v_k and u changes, then all new shortest paths between v_k and u must include $\{a, b\}$; (ii) if the shortest path P between v_k and $u \neq a, b$ changes, then the distance between v_k and w changes, where w is the penultimate node in P. Based on the above insights, and assuming w.l.o.g. that $d(v_k, a) \leq d(v_k, b)$, the idea is that it suffices to *resume* the BFS from b originally rooted at v_k and to stop at unchanged nodes, for every $v_k \in V$. In other words, instead of inserting $(v_k, 0)$ to the initial queue of the BFS, the algorithm inserts pair $(b, d(v_k, a) + 1)$. The search adopts a pruning mechanism similar to that described in Section 2.

It is known that it suffices to resume searches originally rooted at v_k for all $v_k \in L(a) \cup L(b)$ to obtain a 2-hop cover labeling L' of G' from L, using IncPLL [2]. When computing L', to save time, *outdated* label entries of L, i.e. entries that do not to correspond to shortest paths in G', are not removed. However, it can be easily seen that the cover property of L' for G holds even if this *lazy approach* is employed [2], as queries always search for the minimum.

Decremental. When handling a decremental update operation (say the removal of an edge), outdated label entries must be removed from L to obtain a 2-hop cover labeling L' of G', as otherwise the cover property would be broken. In particular, it is easy to see how a query on L, even after a single update operation, might return an arbitrary underestimation of the true value of distance in G'.

For this reason, DecPLL works in three phases. In the first phase, it performs a first BFS–like search of the graph, by exploiting both L and G', in order to detect so-called *affected nodes*, i.e. nodes that contain at least one outdated label

entry. In the second phase, the algorithm scans such nodes and removes outdated label entries. The result is that pairs of affected nodes might or might be not covered by L, depending on the structure of G'. Therefore, a third phase to restore the cover property is performed. In particular, a set of BFS–like searches, rooted at affected nodes, is executed with the aim of computing new label entries to be added to L in order to obtain a labeling L' that covers G'. Such visits essentially discover shortest paths connecting pairs of affected nodes and accordingly add entries to the corresponding labels. This phase also includes a pruning mechanism based on the ordering of nodes which however is more relaxed than that of IncPLL since shortest paths, connecting affected nodes in G', can traverse non-affected nodes (see [8] for more details). IncPLL and DecPLL has been properly combined in [8] to obtain a fully dynamic algorithm, able to deal with both incremental and decremental updates, named FulPLL.

4 Open Problems

In this section, we highlight some open problems toward the efficient dynamic maintainance of 2-hop cover labelings. By combining both the theoretical and experimental analyses provided in [2,8], three main issues can be identified.

Computing minimal labelings in the incremental case. IncPLL adopts the lazy approach of not removing outdated label entries, while at the same time preserving the 2-hop cover property. The latter is clearly a desirable behavior from the computational viewpoint, since the algorithm essentially focuses only on finding new shortest paths induced by the incremental operation and on accordingly adding new labels. However, under this strategy, the minimality of the labeling can be violated after a single update. As a result, the size of the labels can grow arbitrarily, thus affecting negatively both space occupancy and query time. This effect grows with the number of update operations, thus making necessary to periodically execute a from scratch algorithm. Preliminary experiments show such growth to be significant even after few updates, especially in large-scale graphs. Hence, a more refined approach is required. A first direction in this sense could be that of adapting the notion of affected nodes, used by DecPLL, to IncPLL, to remove outdated label entries. This would be beneficial also for FulPLL, whose performance are influenced by this behavior.

More efficient decremental algorithms. Computationally speaking, the main issue currently affecting DecPLL resides in the third phase, where the cover property is restored. Preliminary experiments show that, on most of the inputs, more than 90% of the running time of DecPLL is spent on this stage, which is expected, given the analysis of [8]. In fact, such a phase performs BFS–like visits on the new graph with the aim of "reconnecting" pairs of affected nodes from the cover property viewpoint. Shortest paths in the new graph between such nodes can clearly traverse non affected nodes, therefore the pruning mechanism based on the nodes' ordering is more relaxed, w.r.t. that of both the from scratch methods and IncPLL, and hence much less effective. This is even more evident when dealing with weighted graphs, where the visits incorporate a priority queue.

Hence, further investigation should be dedicated to devising an algorithm with a better pruning strategy. In this sense, an idea is that of exploiting the closeness of affected nodes to hubs that are not affected by the change.

Batch solutions. Another direction that deserves surely further investigation is that of batch algorithms. In particular, it would be interesting to extend DecPLL or IncPLL to make them able to efficiently handle *batches* of updates, i.e. sets of updates occurring on the network simultaneously. This is a quite frequent graph update operation in practice (see e.g. [5,6]). A first way to explore in this sense could be that of studying how batch updates impact on the structure of the labeling when occurring simultaneously, and comparing this effect with operations taken individually. Preliminary experiments show that such interference is noticeable, especially when dealing with sparse graphs. In particular, in the case of decremental operations, sets of affected nodes corresponding to graph changes that are topologically close exhibit a high similarity.

References

1. Akiba, T., Iwata, Y., Yoshida, Y.: Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In: Proceedings International Conference on Management of Data (SIGMOD13). pp. 349–360. ACM (2013)
2. Akiba, T., Iwata, Y., Yoshida, Y.: Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In: Proceedings 23rd Int. World Wide Web Conference, (WWW14). pp. 237–248. ACM (2014)
3. Cionini, A., D'Angelo, G., D'Emidio, M., Frigioni, D., Giannakopoulou, K., Paraskevopoulos, A., Zaroliagis, C.: Engineering graph-based models for dynamic timetable information systems. J. of Discrete Algorithms **46–47**, 40–58 (2017)
4. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM Journal on Computing **32**(5), 1338–1355 (2003)
5. D'Andrea, A., D'Emidio, M., Frigioni, D., Leucci, S., Proietti, G.: Dynamically maintaining shortest path trees under batches of updates. In: 20th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO13). Lecture Notes in Computer Science, vol. 8179, pp. 286–297. Springer (2013)
6. D'Andrea, A., D'Emidio, M., Frigioni, D., Leucci, S., Proietti, G.: Experimental evaluation of dynamic shortest path tree algorithms on homogeneous batches. In: 13th International Symposium on Experimental Algorithms (SEA14). Lecture Notes in Computer Science, vol. 8504, pp. 283–294. Springer (2014)
7. D'Angelo, G., D'Emidio, M., Frigioni, D.: Fully dynamic update of arc-flags. Networks **63**(3), 243–259 (2014)
8. D'Angelo, G., D'Emidio, M., Frigioni, D.: Distance queries in large-scale fully dynamic complex networks. In: 27th Int. Workshop on Combinatorial Algorithms (IWOCA16). Lecture Notes in Comp. Sc., vol. 9843, pp. 109–121. Springer (2016)
9. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Robust distance queries on massive networks. In: Proceedings 22th European Symp. on Algorithms (ESA14). Lecture Notes in Computer Science, vol. 8737, pp. 321–333. Springer (2014)
10. Vieira, M.V., Fonseca, B.M., Damazio, R., Golgher, P.B., Reis, D.C., Ribeiro-Neto, B.A.: Efficient search ranking in social networks. In: Proc. 16th Conf. on Information and Knowledge Management, (CIKM07). pp. 563–572. ACM (2007)