Information Sciences 405 (2017) 123–140

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An adaptive amoeba algorithm for shortest path tree computation in dynamic graphs

Xiaoge Zhang[a,b], Felix T.S. Chan[c], Hai Yang[d], Yong Deng[a,*]

[a] School of Computer and Information Science, Southwest University, Chongqing 400715, China
[b] School of Engineering, Vanderbilt University, Nashville, TN 37235, USA
[c] Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Hung Hum, Kowloon, Hong Kong
[d] Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 3 May 2016
Revised 7 April 2017
Accepted 11 April 2017
Available online 12 April 2017

Keywords:
Shortest path tree
Physarum
Dynamic graphs
Graph algorithms
Routing protocols

ABSTRACT

In today's Internet, the shortest path tree (SPT) construction is an important issue in data exchange. To forward a data packet, each router uses routing protocols and link state information to identify the shortest paths from itself to other routers, which yields the shortest path tree. In reality, the network topology often varies over time. In existing studies, the locally affected nodes are identified and the shortest paths are recomputed so as to update the SPT. However, when the network size becomes large, the process of reconstructing the shortest paths for the affected nodes is very time consuming. Herein, we propose an adaptive amoeba method to build SPT in dynamic graphs. The proposed method is illustrated using a three-step procedure. First, we generalize the original *Physarum* model to enable it to have the ability to find the shortest paths in directed graphs. Secondly, the *Physarum* model is further extended to construct the shortest path tree when there are multiple sink nodes. Finally, we demonstrate that the developed method is capable of reconstructing the SPT by adapting the tube flow when link weight changes occur. Different from previous methods, the proposed algorithm is capable of identifying and recomputing the shortest paths for the affected nodes as well as maintaining the original paths for the unaffected vertices spontaneously. We demonstrate the performance of the proposed algorithm by comparing it with the Label Setting algorithm and Dijkstra algorithm in four randomly generated graphs. The computational results suggest the most appropriate algorithms to be used in different scenarios.

1. Introduction

In today's networks, there is a growing demand for high-speed data exchange. To satisfy end-user demand in a large routing area, it is highly desirable to route data packets in a timely and efficient manner. Typically, a data packet is forwarded by a router through a forwarding table. To build the forwarding table, routing protocols, e.g., IS-IS [35], and Open Shortest Path First (OSPF) [26], are utilized to disseminate the link state information (link costs, link up or down) among the routers, thus enabling the routers to have an intact picture of the network topology. Using the link costs, each router runs

* Corresponding author.
E-mail addresses: zxgcqupt@gmail.com (X. Zhang), prof.deng@hotmail.com (Y. Deng).

http://dx.doi.org/10.1016/j.ins.2017.04.021
0020-0255/

the shortest path finding algorithms, e.g., Dijkstra algorithm [13], to identify the minimum cost paths from itself to other routers in the same area, yielding the shortest path tree (SPT). The computed SPT is then utilized to build the forwarding table, which encompasses important information for forwarding a data packet to the target node along the minimum cost path.

However, in realistic scenarios, the network topology in a routing region usually varies over time [16,38], e.g., a new router is added to the network, a subnetwork fails, link cost increases, or a node fails. In networks, beaconing is used to detect all the network topology changes [39]. Basically, beaconing periodically sends messages across the network and detects the failures after a few missed messages. Afterwards, the identified changes are broadcast to all the routers in the same area. After each router is notified of this change, it updates the network topology in its data structure and recomputes the SPT. A simple way to update the SPT is to recalculate the shortest path from itself to all the other nodes one by one, thus reconstructing the entire forwarding table. Most of today's routers in the market update SPT by removing the existing SPT from each router and constructing a new one from scratch using widely recognized static algorithms, i.e., Dijkstra algorithm [13]. However, such a computation of the entire SPT consumes a large amount of time, which might result in serious downstream delays throughout the network, thus preventing other vital routing functions from being executed. To say the least, even if there are some changes in the link state, the structure of the new SPT does not vary significantly from the old one. Nevertheless, static algorithms that compute SPT from scratch incur unnecessary cost, and they do not utilize the information contained in the outdated SPT. Thus, it is desirable to develop dynamic algorithms that are able to update the SPT efficiently so as to process link state updates.

Mathematically, we formulate the SPT problem in a dynamic environment as follows: In a given network $G(V, E, L)$, where V denotes the set of nodes, E represents the set of links, and L is the nonnegative link weights. With time, some link weight changes occur in G, which turns $G(V, E, L)$ into another network $G'(V, E, L')$. Assume T_s is the SPT rooted at node s in G, and T_s' is the SPT in G'. Then the SPT in the dynamic graphs is to compute T_s' from T_s. In the past years, a large number of papers have focused on this problem. Generally, these approaches can be grouped into two categories: static algorithms and dynamic algorithms. Static algorithms, i.e., Dijkstra algorithm [13], Bellman algorithm [17], solve this problem by reconstructing the SPT whenever there are edge weights changes. However, if only a small portion of the links in the network experience weight changes, then these algorithms will lose efficiency. In addition, since static algorithms always compute the shortest path from scratch, if there are multiple routes of the same distance from one router to another, these algorithms might choose a different routing path, thus resulting in the frequent updates of the routing table, which will increase the risk of routing errors and router failures.

Following the idea of updating SPT by recomputing the shortest paths for the affected vertices [15], researchers have centred on developing dynamic algorithms to reduce the SPT recomputation time. In the past decades, seminal work has been done [11,14,33]. For example, the first fully dynamic algorithm was developed by King [22], which maintained the shortest paths of all pairs in digraphs, however, an imposition was that the weight associated with each link must be nonnegative. To generalize King's method, Demetrescu and Italiano [12] developed an improved dynamic algorithm in digraphs by allowing any arbitrary link weights. By recomputing SPT from the topology of the previously computed SPT, Narvaez et al. [32] presented a new dynamic SPT algorithm to handle a single link weight change. However, the above studies only emphasize single link weight change. If a set of updates occurs, then these algorithms process them as a sequence of link updates. But in many cases, processing a set of updates sequentially is not an efficient solution, particularly when the updated SPT needs to be computed in a short time. To overcome this deficiency, Narvaez et al. [31] developed a semidynamic algorithm using a ball-and-string model, in which the set of link weight updates were processed as a batch. Unfortunately, their method for link weight increase is incorrect in certain cases [5]. A fully dynamic algorithm named DynamicSWSF-FP was proposed in [37]. However, this method spends too much computational resources on finding the shortest path for the influenced nodes. Later, Chan et al. [5] developed a few semidynamic algorithms by optimizing several previously studied SPT algorithms, i.e., DynamicSWSF-FP, BallString, DynDijkstra, but their method implemented different programs to handle various link weight changes. For instance, *DynDijkInc* and *MBallStringInc* were used to find the SPT when the link weight increases, while *DynDijkDec* and *MBallStringDec* were employed to handle link weight decreases. Qu et al. [36] developed a dynamic algorithm based on a modified model of pulse-coupled neural networks (M-PCNNs) to compute SPT for large-scale problems. However, when multiple link changes occur, it becomes increasingly complicated to determine the start time of the neurons.

Additionally, there are some drawbacks common to all the algorithms discussed above. First of all, when the network size becomes large or a large number of mixed edge changes occur in the network, the process of analysing the affected vertices becomes very complicated, and will consume a large amount of CPU time. The rapid development of the Internet has witnessed spectacular growth of the network. In this case, the analysing procedure will take much longer time than before, which might cause a long latency, thus affecting end-users' experience. Secondly, the aforementioned algorithms implement different programs to handle different link weight updates, i.e., link weight increase, link weight decrease, and mixed change of link weights. Suppose additional constraints are required to characterize real-world scenarios, such as, limiting the maximum time for forwarding the packet, we need to look into three different programs and make corresponding changes, which in turn increases the workload.

Consider the above deficiencies, it is important to investigate alternative approaches to address the SPT problem in dynamic graphs. Computer researchers often look into the behaviour and mechanics of natural-world systems in order to