Pruning the Computation of Distributed Shortest Paths in Power-law Networks

Gianlorenzo D'Angelo
Department of Mathematics and Informatics, University of Perugia,
Via Vanvitelli, 1, I-06123, Perugia, Italy
E-mail: gianlorenzo.dangelo@dmi.unipg.it

Mattia D'Emidio and Daniele Frigioni
Department of Computer Science, Information Engineering and Mathematics,
University of L'Aquila, Via G. Gronchi, 18, I-67100, L'Aquila, Italy
E-mail: mattia.demidio@univaq.it, daniele.frigioni@univaq.it

Keywords: communication networks, distributed algorithms, distance-vector algorithms, shortest paths, experimental analysis.

Received: October 24, 2012

We propose a general, simple and practical technique, named Distributed Leafs Pruning *(*DLP*), which can be combined with every distance vector routing algorithm based on shortest paths, allowing to reduce the total number of messages sent by that algorithm. We combine the new technique with three algorithms known in the literature:* DUAL*, which is loop-free and is part of CISCO's widely used EIGRP protocol;* DUST*, which has been shown to be effective on networks with power law node degree distribution, although it suffers of looping;* LFR*, which has been very recently introduced, is loop-free and has been shown to be very effective on real networks. We give experimental evidence that these combinations lead to an important gain in terms of the number of messages sent by* DUAL*,* DUST *and* LFR*, on networks having a power-law node degree distribution. We also notice that, in many cases the use of* DLP *determines a gain in terms of the maximum and the average space occupancy per node.*

Povzetek: Članek predlaga novo metodo DLP, ki jo je moč integrirati v poljuben algoritem za iskanje najkrajših poti v omrežjih.

1 Introduction

Shortest paths is surely one of the most important and studied combinatorial problems in the literature. In particular, the problem of computing and updating efficiently all-pairs shortest paths in a distributed network whose topology dynamically changes over the time is considered crucial in today's communication networks. The solutions found in the literature for this problem can be classified as *distance-vector* and *link-state* algorithms.

In a distance-vector algorithm, a node needs to know and possibly to store the distances from each of its neighbors to every other node in the network. This information is used to compute the distance and the next node in the shortest path to each destination, which are stored in a data structure usually called routing table. Most of the distance-vector solutions for the distributed shortest paths problem proposed in the literature (e.g., see [5, 8, 13, 14, 16, 21, 22, 24]) rely on the classical Distributed Bellman-Ford method (DBF from now on), originally introduced in the Arpanet [17], which is still used in real networks and implemented in the RIP protocol. DBF has been shown to converge to the correct distances if the link weights stabilize and all cycles have positive lengths [3]. However, the convergence can be very slow (possibly infinity) due to the well-known *looping* and *count-to-infinity* phenomena. A loop is a path induced by the routing table entries, such that the path visits the same node more than once before reaching the destination. A node "counts to infinity" when it increments its distance to a destination until it reaches a predefined maximum distance value.

In a link-state algorithm, as for example the *OSPF* protocol widely used in the Internet (e.g., see [18]), a node must know the entire network topology to compute its distance to any network destination (usually running the centralized Dijkstra's algorithm for shortest paths). Link-state algorithms are free of the looping and count-to-infinity problems. However, if a network change occurs, each node needs to receive up-to-date information on the entire network topology. This is achieved by broadcasting each change of the network topology to all nodes [18] and by using a centralized dynamic algorithm for shortest paths, as for example that in [12].

In the last years, there has been a renewed interest in devising new light-weight distributed shortest paths solutions for large-scale Ethernet networks (see, e.g., [9, 11, 19, 23,

25, 26]), where distance-vector algorithms seem to be an attractive alternative to link-state solutions when scalability and reliability are key issues or when the memory power of the nodes of the network is limited.

1.1 Related work

Notwithstanding this increasing interest, the most important distance vector algorithm is still DUAL (Diffuse Update ALgorithm) [13], which is free of the looping and count-to-infinity phenomena, thus resulting an effective practical solution (it is in fact part of CISCO's widely used EIGRP protocol). Another distance vector algorithm is DUST (Distributed Update of Shortest paThs), which has been first introduced in [6] and successively developed in [7]. Compared with DUAL, DUST suffers of the looping and count-to-infinity phenomena, even though it has been designed to heuristically reduce the cases where these phenomena occur. However, DUST uses an amount of data structures per node which is much smaller than those of both DBF and DUAL. In [6, 7] the practical performance of DBF, DUAL, and DUST have been measured in terms of both number of messages, and space occupancy per node on both realistic and artificial instances of the problem. Another distance vector algorithm, named LFR (Loop Free Routing), has been recently proposed in [10]. Compared with DUAL, LFR has the same theoretical message complexity but it uses an amount of data structures per node which is much smaller than that of DUAL, and slightly higher than that of DUST. Moreover, in [10] LFR has been experimentally shown to be a good compromise between the number of messages sent and the memory requirements per node with respect to DUAL on both realistic and artificial instances of the problem.

1.2 Results of the paper

In this paper, we provide a new general, simple, and practical technique, named *Distributed Leafs Pruning* (DLP), which can be combined with every distance-vector algorithm for shortest paths with the aim of overcoming some of their main limitations in large scale networks (high number of messages sent, high space occupancy per node, low scalability, poor convergence). This technique has been devised to be effective mainly in networks having a power-law node degree distribution, which are able to model many real-world networks such as the Internet, the World Wide Web, citation graphs, and some social networks [1]. The main idea of DLP relies on the following observations: a network with power-law node degree distribution with n nodes typically has a very small average node degree and a high number of nodes with unitary degree; any shortest path from a node with unitary degree v to any other node of the network has necessarily to pass through the unique neighbor of v in the network, and hence v does not provide any useful information for the distributed computation of shortest paths.

In order to check the effectiveness of DLP, we combined it with DUAL, DUST, and LFR, by obtaining three new algorithms named DUAL-DLP, DUST-DLP, and LFR-DLP, respectively. Then, we implemented the six algorithms in the OMNeT++ simulation environment [20], a network simulator which is widely used in the literature. As input to the algorithms, we considered the same instances of networks with a power-law node degree distribution used in [6, 7, 10], that is the Internet topologies of the *CAIDA IPv4 topology dataset* [15] (CAIDA - Cooperative Association for Internet Data Analysis is an association which provides data and tools for the analysis of the Internet infrastructure), and the random topologies generated by the *Barabási-Albert* algorithm [2]. The results of our experimental study can be summarized as follows: the application of DLP to DUAL, DUST and LFR provides a significant improvement in the global number of sent messages with respect to the original algorithms, and in many cases an improvement also in the maximum and in the average space occupancy per node. In particular, the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.12 and 0.48; the ratio between the number of messages sent by DUST-DLP and DUST is within 0.04 and 0.81; the ratio between the number of messages sent by LFR-DLP and LFR is within 0.36 and 0.51. Concerning the space occupancy, we have observed a gain in the maximum case for DUAL and LFR, and in the average case for DUAL.

1.3 Structure of the paper

The paper is organized as follows. In Section 2 we introduce some useful notation and definitions used in the paper. In Section 3 we describe the Distributed Leafs Pruning technique. In Section 4 we describe the combination of DLP with DUAL, DUST, and LFR. In Section 5 we give experimental evidence of the effectiveness of DLP. Finally, in Section 6 we give some concluding remarks and outline future research directions.

2 Background

We consider a network made of processors linked through communication channels that exchange data using a message passing model, in which: each processor can send messages only to its neighbors; messages are delivered to their destination within a finite delay but they might be delivered out of order; there is no shared memory among the nodes of the network; the system is asynchronous, that is, a sender of a message does not wait for the receiver to be ready to receive the message.

2.1 Graph notation

We represent the network by an undirected weighted graph $G = (V, E, w)$, where V is a finite set of nodes, one for each processor, E is a finite set of edges, one for

each communication channel, and w is a weight function $w : E \rightarrow \mathbb{R}^+ \cup \{\infty\}$ that assigns to each edge a real value representing the optimization parameter associated to the corresponding channel. We assume that the graph is connected. An edge in E that links nodes $u, v \in V$ is denoted as $\{u, v\}$. Given $v \in V$, $N(v)$ denotes the set of neighbors of v. The maximum degree of the nodes in G is denoted by *maxdeg*. A path P in G between nodes u and v is denoted as $P = (u, ..., v)$. The *weight* of P is the sum of the weights of the edges in P. A *shortest path* between nodes u and v is a path from u to v with the minimum weight. The *distance* $d(u, v)$ from u to v is the weight of a shortest path from u to v. Given two nodes $u, v \in V$, the *via* from u to v is the set of neighbors of u that belong to a shortest path from u to v. Formally: $via(u, v) \equiv \{z \in N(u) \mid d(u, v) = w(u, z) + d(z, v)\}$. Given a time t, we denote as $w^t()$, $d^t()$, and $via^t()$ the edge weight, the distance, and the via at time t, respectively. We denote a sequence of k update operations on the edges of G by $\mathcal{C} = (c_1, c_2, ..., c_k)$. Assuming $G_0 \equiv G$, we denote as G_i, $0 \leq i \leq k$, the graph obtained by applying the operation c_i to G_{i-1}. The operation c_i either inserts a new edge in G_i, or deletes an edge of G_i, or modifies (either increases or decreases) the weight of an existing edge in G_i. We consider the case in which $\mathcal{C}$ is a sequence of *weight increase* and *weight decrease* operations, that is operation c_i either increases or decreases the weight of edge $\{x_i, y_i\}$ by a quantity $\epsilon_i > 0$. The extension to *delete* and *insert* operations is straightforward, in fact deleting an edge $\{x, y\}$ is equivalent to increase $w(x, y)$ to $+\infty$, and inserting an edge $\{x, y\}$ with weight α is equivalent to decrease $w(x, y)$ from $+\infty$ to α.

2.2 Distance-vector algorithms

We consider the generic routing problem between all the nodes of a network, in which each node needs to find a shortest path to each other node. This problem can be tackled in different ways. The most reliable, robust and used approach is that based on distributed all-pairs shortest paths. We are interested in the practical case of a dynamic network in which an edge weight change (increase/decrease) can occur while one or more other edge weight changes are under processing. A processor v of the network might be affected by a subset of these changes. As a consequence, v could be involved in the concurrent executions related to such changes.

Distance-vector routing algorithms based on shortest-paths usually share a set of common features. In detail, given a graph $G = (V, E, w)$, a generic node v of G:

- knows the identity of every other node of G, the identity of all its neighbors and the weights of the edges incident to it;
- maintains and updates its own routing table that has one entry for each $s \in V$, which consists of at least two fields: $\mathtt{D}^t[v, s]$, the estimated distance between v and s at time t, and $\mathtt{VIA}^t[v, s]$, the neighbor used to forward data from v to s at time t;
- handles edge weight increases and decreases either by a single procedure (see, e.g., [13]), which we denote as WEIGHTCHANGE, or separately (see, e.g., [7]) by two procedures, which we denote as WEIGHTINCREASE and WEIGHTDECREASE;
- requests information to its neighbors and receives replies by them through a specific exchange of messages (see, e.g., message $query$ in [13] or message *get.feasible.dist* in [10]) and propagates a variation to the estimated routing information as follows: (i) if v is performing WEIGHTCHANGE, then it sends to its neighbors a message, from now on denoted as $update$; a node that receives this kind of message executes procedure UPDATE. (ii) if v is performing WEIGHTINCREASE or WEIGHTDECREASE, then it sends to its neighbors message $increase$ or $decrease$, respectively; a node that receives $increase$/$decrease$ executes procedure INCREASE/DECREASE, respectively.

2.3 Power-Law networks

Networks having a power-law node degree distribution, from now on referred as "power-law networks", are very important in practice and includes many of the currently implemented communication infrastructures, like the Internet, the World Wide Web, some social networks, and so on [1]. Practical examples of power-law networks are the Internet topologies of the *CAIDA IPv4 topology dataset* [15], and the artificial instances generated by the *Barabási-Albert* algorithm [1].

The CAIDA dataset is collected by a globally distributed set of monitors. The monitors collect data by sending probe messages continuously to destination IP addresses. Destinations are selected randomly from each routed IPv4/24 prefix on the Internet such that a random address in each prefix is probed approximately every 48 hours. The current prefix list includes approximately 7.4 million prefixes. For each destination selected, the path from the source monitor to the destination is collected, in particular, data collected for each path probed includes the set of IP addresses of the hops which form the path and the Round Trip Times (RTT) of both intermediate hops and the destination.

A Barabási–Albert topology is generated by iteratively adding one node at a time, starting from a given connected graph with at least two nodes. A newly added node is connected to any other existing nodes with a probability that is proportional to the degree of the existing nodes. In Figure 1 we show the power-law node degree distribution of a *CAIDA* network (a) and of a *Barabási-Albert* network (b).

3 The new technique

The main goal of Distributed Leafs Pruning (DLP) is to reduce the number of messages sent by a generic distance-

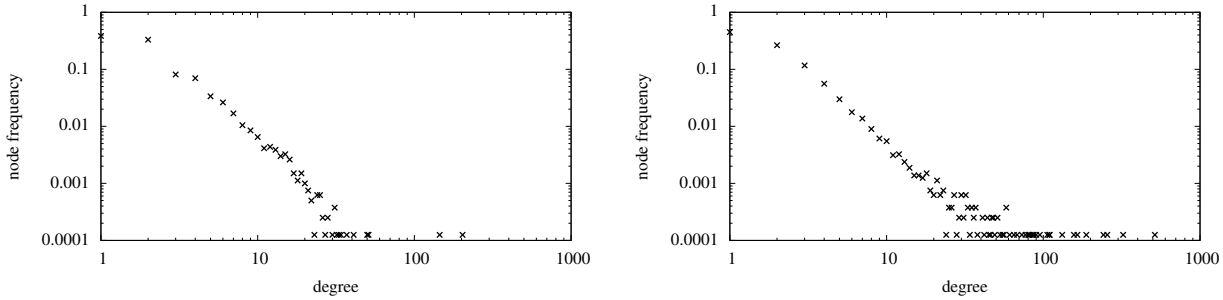

Figure 1: Power-law node degree distribution of: a *CAIDA* graph with 8000 nodes and 11141 edges (a); a *Barabási-Albert* graph with 8000 nodes and 12335 edges (b).

vector algorithm. DLP has been designed to be efficient mainly in power-law networks. The idea underlying DLP is very simple and it is based on the following observations:

- a power-law network with n nodes typically has average node degree much smaller than n and a number of nodes with unitary degree which is generally high. For example, the graphs of the *CAIDA IPv4 topology dataset* have average node degree approximately equal to $n/2000$, and a number of nodes with unitary degree approximately equal to $n/2$;
- nodes with unitary degree do not provide any useful information for the distributed computation of shortest paths. In fact, any shortest path from a node with degree one v to any other node of the network has necessarily to pass through the unique neighbor of v in the network.

To describe the technique we need to introduce some preliminary definitions. Given an undirected weighted graph $G = (V, E, w)$, the *core* of G is the graph $G_c = (V_c, E_c, w_c)$ which represents the maximal connected subgraph of G having all nodes of degree greater than one. A node $v \in V$ is a *central node* if $v \in V_c$, otherwise v is a *peripheral node*. An edge of G that links two central nodes is a *central edge*, an edge that links a central node with a peripheral node is a *peripheral edge*. For each peripheral node u, the unique central node v adjacent to u is called the *owner* of u.

3.1 Data structures

Given a generic distance-vector algorithm **A**, DLP requires that a generic node of G stores some additional information with respect to those required by **A**. In particular, a node v needs to store and update information about central and peripheral nodes and edges of G. To this aim, v maintains a data structure called *Classification Table*, denoted as CT_v, which is an array containing one entry $CT_v[s]$, for each $s \in V$, representing the list of the peripheral neighbors of s. A central node is not present in any list of CT_v. A peripheral node is present in $CT_v[s]$, for exactly one $s \in V$, and s is its owner. Each list contains at most $maxdeg$ entries and the sum of the sizes of all the lists is always smaller than n. Hence the space overhead per node due to CT_v is $O(n)$.

3.2 Properties

The main purpose of DLP is to force distributed computation to be carried out only by the central nodes. The peripheral nodes receive updates about routing information passively from the respective owners, without starting any kind of distributed computation. Then, the larger is the set of the peripheral nodes of the network, the bigger is the improvement in the global number of messages sent by the algorithm. The following lemma introduces some basic relationships between the paths that link central and peripheral nodes.

Lemma 3.1. *Given an undirected weighted graph $G = (V, E, w)$, and its core $G_c = (V_c, E_c, w_c)$, let $\{p, c\}$ be a peripheral edge such that $c \in V_c$ at time t. The following relations hold:*

- $d^t(x, p) = d^t(x, c) + w^t(c, p),\ \forall\ x \in V \setminus \{p\}$;
- $via^t(x, p) = via^t(x, c),\ \forall\ x \in V \setminus \{p\}$.

Proof. By contradiction, let us assume that $d^t(x, p) \neq d^t(x, c) + w^t(c, p)$ for a certain $x \in V \setminus \{p\}$. Then, two cases can occur: if $d^t(x, p) < d^t(x, c) + w^t(c, p)$, it follows that there exists, at time t, another path from node x to node p that does not include node c, which is a contradiction, as p is a peripheral node and has a unique adjacent node at time t; on the other hand, if $d^t(x, p) > d^t(x, c) + w^t(c, p)$ it follows that $d^t(x, p)$ is not the weight of a shortest path, which is again a contradiction. □

Some useful additional relationships can be derived from Lemma 3.1. In particular, if between the time instants t_i and t_{i+1} the weight of the edge $\{p, c\}$ between a peripheral node p and his corresponding owner c changes, that is $w^{t_i}(p, c) \neq w^{t_{i+1}}(p, c)$, then p can update its own routing

table towards each node of the network $x \in V$ simply by computing:

$$d^{t_{i+1}}(p,x) = d^{t_i}(p,x) + w^{t_{i+1}}(p,c) - w^{t_i}(p,c), \quad (1)$$

$$via^{t_{i+1}}(p,x) = \{c\}. \quad (2)$$

In a similar way, if a generic node of the network $x \in V$, between the time instants t_i and t_{i+1}, receives an update about a weight change in the path towards a generic central node c (that is, $d^{t_{i+1}}(x,c) \neq d^{t_i}(x,c)$), then the nodes involved in the change are x, c and the peripheral neighbors of c, if they exists. By Lemma 3.1, these nodes can update their estimated routing tables by computing, for all peripheral nodes p with owner c:

$$d^{t_{i+1}}(x,p) = d^{t_i}(x,p) + d^{t_{i+1}}(x,c) - d^{t_i}(x,c), \quad (3)$$

$$via^{t_{i+1}}(x,p) = via^{t_{i+1}}(x,c). \quad (4)$$

3.3 Distributed Leafs Pruning

The application of DLP to a distance vector algorithm **A** induces a new algorithm denoted as **A**-DLP. The global behavior of **A**-DLP can be summarized as follows. While in a classic routing algorithm every node performs the same code thus having the same behavior, in **A**-DLP central and peripheral nodes have different behaviors. In particular, central nodes detect changes concerning both central and peripheral edges while peripheral nodes detect changes concerning only peripheral edges. If the weight of a central edge $\{u,v\}$ changes, then node u (v, respectively) performs the procedure provided by **A** for that change only with respect to central nodes for the distributed computation of the shortest paths between all the pairs of central nodes. During this computation, if u (v, respectively) needs information by its neighbors, it asks only to neighbors in the core (see Figure 3(a)). Once u (v, respectively) has updated its own routing information, it propagates the variation to all its neighbors through the $update$, $increase$ or $decrease$ messages of **A** (Figure 3(b)). When a generic node x receives an $update$, $increase$ or $decrease$ message concerning node s, it stores the current value of $\mathtt{D}[x,s]$ in a temporary variable $\mathtt{D}_{old}[x,s]$. Now, if x is a central node, then it handles the change and updates its routing information by using the proper procedure of **A** (UPDATE, INCREASE, or DECREASE) and propagates the new information to its neighbors (see Figure 3(c)). Otherwise, x handles the change and updates its routing information towards s by using Lemma 3.1 and the data received from its owner. At the end, x calls the procedure UPDATEPERIPHERALS reported in Figure 2 using s and $\mathtt{D}_{old}[x,s]$ as parameters. If the routing table entry of s is changed (line 1), then the information about the peripheral neighbors of s, if they exist, is updated by using Equations 3 and 4 (lines 3–4).

If a weight change occurs in a peripheral edge $\{u,p\}$, then the central node u sends message $p\_change(p, w(u,p), u)$ to each of its neighbors (Figure 4(a)), while p sends a $p\_change(p, w(u,p), u)$ message to its owner u. When a generic node x receives message p_change from a node y, it simply performs procedure PERIPHERALCHANGE of Figure 5, which is a modified flooding algorithm to forward the message over the network (Figure 4(b)). Procedure PERIPHERALCHANGE first verifies at line 1 whether the message was already received by applying Lemma 3.1. If the message does not provide updated information, it is discarded. Otherwise, the procedure needs to update the data structures at x. We distinguish two cases: if x coincides with p, then the procedure updates the routing table for all the nodes $s \in V$, by using Equations 1 and 2 (see Lines 2-6). Otherwise ($x \neq p$), the procedure simply updates the routing table entry concerning p by using Lemma 3.1 (Lines 8-10). At this point, the procedure propagates the information about the change, forwarding message p_change to all the neighbors, except to nodes u and possibly p (Lines 11–12).

In what follows we provide the correctness proof of **A**-DLP which clearly depends on the correctness of **A** that is assumed. In particular, we proof the correctness of **A**-DLP for a single weight change operation. The extension to the case of multiple weight changes is straightforward.

Theorem 3.2. *Given a graph $G = (V, E, w)$, let c be a weight change operation on G, occurring at a certain time t_c. For each pair of nodes $v, s \in V$ that change their*

Event: node v invokes procedure
UPDATEPERIPHERALS$(s, \mathtt{D}_{old}[v,s])$
Procedure: UPDATEPERIPHERALS$(s, \mathtt{D}_{old}[v,s])$
1 **if** $\mathtt{D}[v,s] \neq \mathtt{D}_{old}[v,s]$ **then**
2 **foreach** $k \in CT_v[s]$ **do**
3 $\mathtt{D}[v,k] := \mathtt{D}[v,k] + \mathtt{D}[v,s] - \mathtt{D}_{old}[v,s]$
4 $\mathtt{VIA}[v,k] := \mathtt{VIA}[v,s]$
5 update any auxiliary data structures of **A**

Figure 2: Procedure UPDATEPERIPHERALS.

Event: node x receives the message
p_change$(p, w(u,p), u)$ from y
Procedure: PERIPHERALCHANGE$(p, w(u,p), u)$
1 **if** $w(u,p) \neq (\mathtt{D}[x,p] - \mathtt{D}[x,u])$ **then**
2 **if** $x \equiv p$ **then**
3 **foreach** $s \in V \setminus \{x\}$ **do**
4 $\mathtt{D}[x,s] := \mathtt{D}[x,s] - \mathtt{D}[x,u] + w(u,p)$
5 $\mathtt{VIA}[x,p] := u$
6 update any auxiliary data structures of **A**
7 **else**
8 $\mathtt{D}[x,p] := \mathtt{D}[x,u] + w(u,p)$
9 $\mathtt{VIA}[x,p] := \mathtt{VIA}[x,u]$
10 update any auxiliary data structures of **A**
11 **foreach** $k \in N(x) \setminus \{y,p\}$ **do**
12 send p_change$(p, w(u,p), u)$ to k

Figure 5: Procedure PERIPHERALCHANGE.

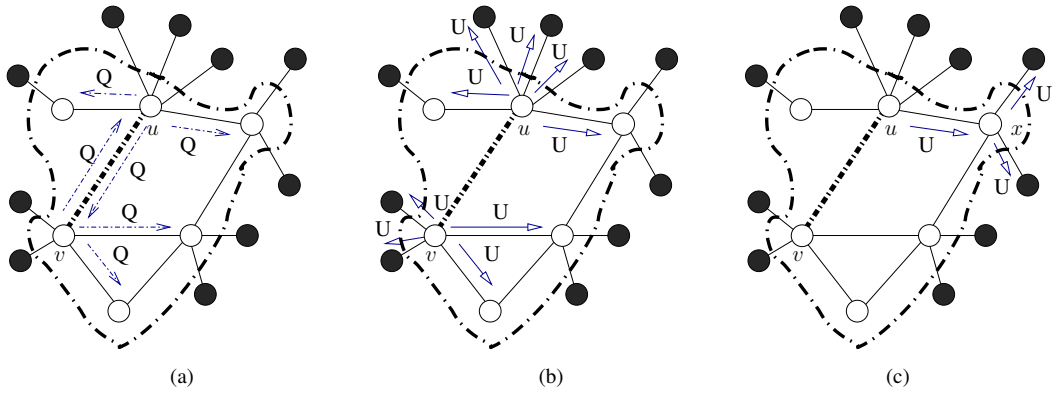

Figure 3: (a) Nodes u and v ask for information to their central neighbors by sending them message Q (query); (b) Nodes u and v propagate updated routing information to all their neighbors through message U (update); (c) A central node x receiving an update message U, propagates it to its neighbors.

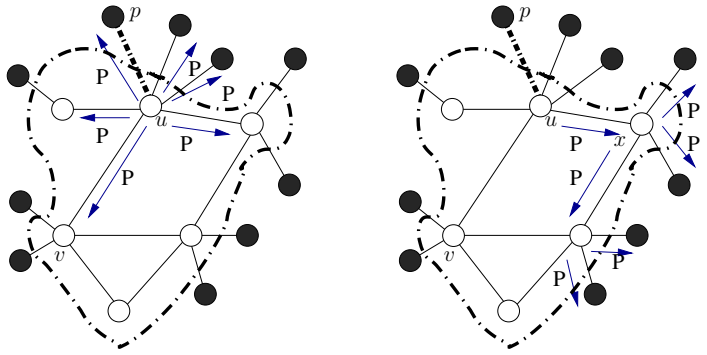

Figure 4: (a) Node u, as a consequence of a weight change on edge $\{u, p\}$, sends message p_change (P) to all its neighbors; (b) Node x receiving a message P, propagates it to the whole network.

distance as a consequence of c, there exists a time instant $t_f \geq t_c$ *such that for each* $t \geq t_f$, $\mathtt{D}[v, s](t) = d^{t_c}(v, s)$.

Proof. The proof is by case analysis on the two possible types of weight change operations on the edges: (i) central; (ii) peripheral. In what follows, we assume that the algorithm is correct at every time $t_0 < t_c$, and given any data structure $\mathtt{X}$ of **A**-DLP, we denote by $\mathtt{X}(t)$ the content of $\mathtt{X}$ at time instant t.

Case (i). If the weight of a central edge $\{x, y\}$ changes at time t_c, then node x (y, respectively) performs the procedure provided by **A** for the distributed computation of the shortest paths, only with respect to central nodes. In this step, the correctness of **A**-DLP follows from the correctness of **A**. In fact, the only difference between the behavior of **A**-DLP and that of **A** is that x (y, respectively) does not ask to peripheral nodes information concerning s, which are, for topological reasons, un-useful. Hence, the correctness of **A** guarantees that, for each central node $s \in V$, there exists a time $t_f \geq t_c$ in which **A**-DLP sets $\mathtt{D}[x, s](t_f) = d^{t_c}(x, s)$ ($\mathtt{D}[y, s](t_f) = d^{t_c}(y, s)$) and this value does not change anymore.

Once x (y, respectively) has updated its own routing information toward a central node s, it propagates the variation to all its neighbors through the $update$, $increase$ or $decrease$ messages of **A**, which carries $\mathtt{D}[x, s](t_f)$, that is the correct value $d^{t_c}(x, s)$.

When a generic node v receives an $update$, $increase$ or $decrease$ message concerning a central node s, it stores the current value of $\mathtt{D}[v, s](t_0) = d^{t_0}[v, s]$ in a temporary variable $\mathtt{D}_{old}[v, s]$. Now, if v is a central node, then it handles the change and updates its routing information by using the proper procedure of **A** (UPDATE, INCREASE, or DECREASE) and propagates the new information to its neighbors. Also in this case, the correctness of **A**-DLP follows by the correctness of **A**.

Otherwise, if v is a peripheral node whose owner is a central node c, then v handles the change and updates its routing information towards s by setting, at a certain time t_f greater than $t_c \geq t_0$, $\mathtt{D}[v, s](t_f) = \mathtt{D}[c, s](t_f) + w^{t_c}(v, c)$, where $\mathtt{D}[c, s](t_f) = d^{t_c}(c, s)$ is the received correct value. Hence, by Lemma 3.1 **A**-DLP properly assigns $\mathtt{D}[v, s](t_f) = d^{t_c}(v, s)$, and the statement of the theorem is true also in this case.

After updating the routing information toward the central node s, v calls procedure UPDATEPERIPHERALS, re-

ported in Figure 2, using s and $\mathtt{D}_{old}[v,s]$ as parameters, whose aim is to update, if needed, the routing information about the non-central nodes whose owner is s, if they exist. If the routing table entry of s is changed (line 1), then v sets, at a certain time $t_F \geq t_f$, for each peripheral node y whose owner is s, $\mathtt{D}[v,y](t_F) = \mathtt{D}[v,y](t_f) + \mathtt{D}[v,s](t_f) - \mathtt{D}_{old}[v,s](t_f)$. This assignment statement, by Lemma 3.1, is clearly correct and guarantees that $\mathtt{D}[v,y](t_F) = d^{t_c}(v,y)$ since: (i) $\mathtt{D}[v,y](t_f) = d^{t_0}(v,y)$; (ii) $\mathtt{D}[v,s](t_f)$ is the received value equal to $d^{t_c}(v,s)$; (iii) $\mathtt{D}_{old}[v,s](t_f) = d^{t_0}[v,s]$. Hence, also in this case **A**-DLP is correct.

Case (ii). If a weight change occurs in a peripheral edge $\{u,p\}$, then the central node u sends message $p\_change(p, w^{t_c}(u,p), u)$ to each of its neighbors (Figure 4(a)), while p sends a $p\_change(p, w^{t_c}(u,p), u)$ message to its owner u.

When a generic node x receives message p_change from a node y, at a certain time t, it simply performs procedure PERIPHERALCHANGE of Figure 5, which is a modified flooding algorithm to forward the message over the network (Figure 4(b)). Procedure PERIPHERALCHANGE first verifies at line 1 whether the message was already received by applying Lemma 3.1. If the message does not provide updated information, it is discarded.

Otherwise, the procedure needs to update the data structures at x. We distinguish two cases: if x coincides with p, then the procedure updates the routing table for all the nodes $s \in V$, by setting, at a certain time t_f greater than t, for all the nodes $s \in V$, $\mathtt{D}[x,s](t_f) = \mathtt{D}[x,s](t) - \mathtt{D}[x,u](t) + w^{t_c}(u,x)$, which is correct, again by Lemma 3.1, and guarantees that $\mathtt{D}[x,s](t_f) = d^{t_c}(x,s)$ as $\mathtt{D}[x,s](t) = d^{t_0}(x,s)$ and $\mathtt{D}[x,u](t) = d^{t_0}(x,u) = w^{t_0}(x,u)$. Therefore, **A**-DLP is correct in this case.

If $x \neq p$, then the procedure simply updates the routing table entry concerning p by setting, at a certain time t_f greater than t, $\mathtt{D}[x,p](t_f) = \mathtt{D}[x,u](t) + w^{t_c}(u,p)$, where $\mathtt{D}[x,u](t) = d^{t_c}(x,u)$ and $w^{t_c}(u,p)$ is the correct received value of the weight of edge (u,p). It follows that x again by Lemma 3.1 correctly assigns $\mathtt{D}[x,p](t_f) = d^{t_c}(x,p)$. At the end, the change is forwarded through the network by sending a message p_change to all the neighbors, except to nodes u and possibly p. Therefore, **A**-DLP is correct in in all cases. □

4 Combinations

In this section we briefly describe algorithms DUAL, DUST, and LFR, and how they can be combined with DLP.

4.1 Combination of DLP with DUAL

DUAL (Diffuse Update ALgorithm) [13] stores, for each node v and for each destination s, the routing table where the two fields are the distance $\mathtt{D}[v,s]$ and the *feasible successor* $\mathtt{S}[v,s]$, respectively. In order to compute $\mathtt{S}[v,s]$, DUAL requires that each node v is able to determine, for each destination s, a set of neighbors called the Feasible Successor Set, denoted as $\mathtt{FSS}[v,s]$. To this aim, each node v stores, for each $u \in N(v)$, the distance $\mathtt{D}[u,s]$ (the topology table) from u to s and computes $\mathtt{FSS}[v,s]$ by choosing neighbors which satisfy SNC, a condition, introduced in [13], that guarantees the algorithm to be loop-free. In detail, node $u \in N(v)$ satisfies SNC if the estimated distance $\mathtt{D}[u,s]$ from u to s is smaller than the estimated distance $\mathtt{D}[v,s]$ from v to s. If a neighbor $u \in N(v)$, through which the distance from v to s is minimum, is in $\mathtt{FSS}[v,s]$, then u is chosen as feasible successor. When a node v experiences a weight change operation in one of the adjacent edges, it executes procedure WEIGHTCHANGE, in order to update $\mathtt{FSS}[v,s]$.

DUAL includes an important sub-routine, named Diffuse-Computation, which is performed by a generic node v, when $\mathtt{FSS}[v,s]$ does not include the node $u \in N(v)$ through which the distance from v to s is minimum. The Diffuse-Computation works as follows: node v sends queries to all its neighbors with its distance through $\mathtt{S}[v,s]$ by using message *query*. From this point onwards v does not change its feasible successor to s until the Diffuse-Computation terminates. When a neighbor $u \in N(v)$ receives a *query*, it tries to determine if a feasible successor to s, after such update, exists. If so, it replies to the *query* by sending message *reply* containing its own distance to s. Otherwise, u continues the Diffuse-Computation: it sends out queries and waits for the replies from its neighbors before replying to v's original *query*. In [13] it has been proved that the Diffuse-Computation always terminates. When a node receives messages *reply* by all its neighbors it updates its distance and feasible successor, with the minimal value obtained by its neighbors and the neighbor that provides such distance, and finishes the Diffuse-Computation. At the end of a Diffuse-Computation, a node sends message *update* containing the new computed distance to its neighbors. A generic node that receive an *update* message handled it by performing procedure UPDATE. In order to guarantee mutual exclusion in case of multiple weight change operations, DUAL uses a finite state machine to process these multiple updates sequentially.

DUAL can be combined with DLP as described in Section 2. In addition, the generic procedures reported in Figure 2 and 5, are modified, by using the data structures of DUAL, to generate two specific procedures, called DUAL-PERIPHERALCHANGE and DUAL-UPDATEPERIPHERALS. The main changes can be summarized as follows: (i) in Procedure PERIPHERALCHANGE (at Lines 5 and 9) and in Procedure UPDATEPERIPHERALS (at Line 4) the data structure VIA is replaced by the data structure S; (ii) in Procedure UPDATEPERIPHERALS, Line 5 is removed, as the auxiliary data structures of DUAL, like e.g. the FSM data structures or the topology table, are used only in the single distributed computa-

tion phase and hence it is not necessary to store them with respect to peripheral nodes; (iii) in Procedure PERIPHERALCHANGE, Line 6 is removed for the same reason while at Line 10 the auxiliary data structures of DUAL are updated, as s is a central node, according to the algorithm.

4.2 Combination of DLP with DUST

DUST maintains only the routing table described in Section 2 and, for each node v and for each source s, $\texttt{VIA}[v,s]$ contains the set $\texttt{VIA}[v,s] \equiv \{v_i \in N(v) \mid \texttt{D}[v,s] = w(v,v_i) + \texttt{D}[v_i,s]\}$. Algorithm DUST starts every time an operation c_i on edge (x_i, y_i) occurs. Operation c_i is detected only by nodes x_i and y_i. If c_i is a *weight increase* (*weight decrease*) operation, x_i performs procedure WEIGHTINCREASE (WEIGHTDECREASE) that sends message *increase*(x_i, s) (*decrease*$(x_i, s, \texttt{D}[x_i, s])$) to y_i for each $s \in V$. Node y_i has the same behavior of x_i. If a node v receives message *decrease*$(u, s, \texttt{D}[u, s])$, then it performs procedure DECREASE, that relaxes edge (u, v). In particular, if $w(v,u) + \texttt{D}[u,s] < \texttt{D}[v,s]$, then v updates $\texttt{D}[v,s]$ and $\texttt{VIA}[v,s]$, and propagates the updated values to nodes in $N(v)$. If $w(v,u) + \texttt{D}[u,s] = \texttt{D}[v,s]$, then u is a new estimated via for v with respect to s, and hence v adds u to $\texttt{VIA}[v,s]$. If a node v receives *increase*(u, s), then it performs procedure INCREASE which checks whether the message comes from a node in $\texttt{VIA}[v,s]$. In the affirmative case, v removes u from $\texttt{VIA}[v,s]$. As a consequence, $\texttt{VIA}[v,s]$ may become empty. In this case, v computes the new estimated distance and via of v to s. To do this, v asks to each node $v_i \in N(v)$ for its current distance, by sending message *get-dist*(v, s) to v_i. When v_i receives *get-dist*(v, s) by v, it performs procedure SENDDIST which sends $\texttt{D}[v_i, s]$ to v, unless one of the following two conditions holds: (i) $\texttt{VIA}[v_i, s] \equiv \{v\}$; (ii) v_i is updating its routing table with respect to destination s. In this case v_i sends ∞ to v. When v receives the answers to the *get-dist* messages by all its neighbors, it computes the new estimated distance and via to s. If the estimated distance is increased, v sends an *increase* message to its neighbors. In any case, v sends to its neighbors *decrease*, to communicate them $\texttt{D}[v,s]$. In fact, at some point, v could have sent ∞ to a neighbor v_j. Then, v_j receives the message sent by v, and it performs procedure DECREASE to check whether $\texttt{D}[v,s]$ can determine an improvement to the value of $\texttt{D}[v_j, s]$.

DUST can be combined with DLP, by modifying its behavior as described Section 2. In addition, the generic procedures reported in Figures 2 and 5, are modified, by using the data structures of DUST, to generate two specific procedures, called DUST-PERIPHERALCHANGE and DUST-UPDATEPERIPHERALS. The main changes can be summarized as follows: (i) in Procedure PERIPHERALCHANGE (at Lines 5 and 9) and in Procedure UPDATEPERIPHERALS (at Line 4) the data structure VIA is modified to be a set instead of a single value variable; (ii) since DUST does not use any additional data structures, in Procedure PERIPHERALCHANGE Lines 6 and 10 are removed and in Procedure UPDATEPERIPHERALS Line 5 is removed.

4.3 Combination of DLP with LFR

LFR stores, for each node v, the estimated distance $\texttt{D}[v,s]$ and the *feasible via*, $\texttt{FVIA}[v,s]$, that is the node through which the distance to s is minimum and which satisfies SNC. In addition, node v maintains for each $s \in V$, the following data structures: $\texttt{ACTIVE}_v[s]$, which represents the state of node v with respect to a certain source s, in detail, v is in *active* state and $\texttt{ACTIVE}_v[s] = true$, if and only if it is trying to update $\texttt{FVIA}[v,s]$ after a generic weight change operation occurred on edge $\{v, \texttt{FVIA}[v,s]\}$; the *upper bound distance* $\texttt{UD}[v,s]$ which represents the distance from v to s through $\texttt{FVIA}_v[s]$, in particular, if v is active $\texttt{UD}[v,s]$ is always greater than or equal to $\texttt{D}[v,s]$, otherwise they coincide. In addition, in order to compute loop-free values of $\texttt{FVIA}[v,s]$, node v stores a temporary data structure tempD which is allocated only when needed, that is when v is active with respect to s, and it is deallocated when v turns back in passive state with respect to s. The entry $\texttt{tempD}[u,s]$ contains $\texttt{UD}[u][s]$, for each $u \in N(v)$.

The algorithm starts when the weight of an edge $\{x_i, y_i\}$ changes. As a consequence, x_i (y_i, respectively) performs procedure WEIGHTCHANGE, that sends to y_i (x_i, respectively) an *update* message carrying the value $\texttt{D}[x_i, s]$ ($\texttt{D}[x_i, s]$, respectively). Messages received at a node with respect to a source s are stored in a queue and processed in a FIFO order to guarantee mutual exclusion. If an arbitrary node v receives an *update* message from $u \in N(v)$, then it performs procedure UPDATE in which, basically, v compares the received value $\texttt{D}[u,s] + w(u,v)$ with $\texttt{D}[v,s]$ in order to determine whether v needs to update its estimated distance and its estimated $\texttt{FVIA}[v,s]$. The update procedure works as follows. If node v is active, the processing of the message is postponed by enqueueing it into the FIFO queue associated to s. Otherwise, if $\texttt{D}[v,s] > \texttt{D}[u,s] + w(u,v)$, then v performs a relaxing phase, updating both $\texttt{D}[v,s]$ and $\texttt{FVIA}[v,s]$, while if $\texttt{D}[v,s] > \texttt{D}[u,s] + w(u,v)$, node v performs a phase called Local-Computation in which it sends a *get.dist* to all its neighbor in order to know the corresponding estimated distances towards s. Each neighbor $u \in N(v)$ immediately replies to the *get.dist* message with its $\texttt{UD}[u,s]$. When v receives these values, it tries to determine whether a new $\texttt{FVIA}[v,s]$ exists, by comparing the received distances with $\texttt{D}[v,s]$. If this phase succeeds, node v updates its routing information and propagates the change, Otherwise, node v initiates a distributed phase, named Global-Computation. It sets $\texttt{UD}[v,s] = \texttt{UD}[\texttt{FVIA}[v,s], s] + w(v, \texttt{FVIA}[v,s])$ and sends to all its neighbors a *get.feasible.dist* message, carrying $\texttt{UD}[v,s]$. A node $k \in N(v)$ that receives such a message first verifies whether $\texttt{FVIA}[k,s] = v$ or not. In the first case, it replies immediately to v and terminates. In the second case, it performs the Local-Computation and possibly the Global-Computation, in order to update its routing

information and to reply to v. Note that this distributed procedure can involve all the nodes of the network. Finally, if $\mathtt{D}[v,s] = \mathtt{D}[u,s] + w(u,v)$, that is there exists more than one shortest path from v to s, the message is discarded and the procedure ends.

LFR can be combined with DLP, by modifying its behavior as described Section 2. In addition, the generic procedures reported in Figures 2 and 5, are modified, by using the data structures of LFR, to generate two specific procedures, called LFR-PERIPHERALCHANGE and DUST-UPDATEPERIPHERALS. The main changes can be summarized as follows: (i) in Procedure PERIPHERALCHANGE (at Lines 5 and 9) and in Procedure UPDATEPERIPHERALS (at Line 4) the data structure `VIA` is replaced by the data structure `FVIA`; (ii) in Procedure UPDATEPERIPHERALS, Line 5 is removed, as the auxiliary data structures of LFR, like e.g. `tempD` or the state `ACTIVE`, are used only in the distributed computation phases and hence it is not necessary to store them with respect to peripheral nodes; (iii) in Procedure PERIPHERALCHANGE, Line 6 is removed for the same reason while at Line 10 the auxiliary data structures are updated, as s is a central node, according to the algorithm.

5 Experimental analysis

In this section, we present the experimental evaluation we performed in order to check the practical effectiveness of DLP. In particular, we combined DLP with DUAL, DUST and LFR and then we implemented, in C++, and tested both the the original algorithms and the new combinations, namely DUAL-DLP, DUST-DLP and LFR-DLP.

In what follows, we report the results of such experimental study. Our experiments have been performed on a workstation equipped with a Quad-core 3.60 GHz Intel Xeon X5687 processor, with 12MB of internal cache and 24 GB of main memory and consist of simulations within the well-known OMNeT++ 4.0p1 environment [20]. The software has been compiled with GNU g++ compiler 4.4.3 under Linux (Kernel 2.6.32).

OMNeT++ is an object-oriented modular discrete event network simulator, useful to model protocols, telecommunication networks, and other distributed systems. An OMNeT++ model consists of hierarchically nested modules, that communicate through message passing. In our model, we defined a basic module *node* to represent a node in the network. A node v has a communication *gate* with each node in $N(v)$. Each node can send messages to a destination node through a *channel* which is a module that connects gates of different nodes (both gate and channel are OMNeT++ predefined modules). In our model, a channel connects exactly two gates and represents an edge between two nodes. We associate two parameters per channel: a *weight* and a *delay*. The former represents the cost of the edge in the graph, and the latter simulates a finite but not null transmission time.

5.1 Executed tests

As input to the algorithms we used both real-world and artificial instances of the problem. In detail, we used real-world networks of the *CAIDA IPv4 topology dataset* [15] and random networks generated by the *Barabási-Albert* algorithm [2], subject to randomly generated sequences of edge update operations.

Concerning CAIDA instances, we parsed the files provided by CAIDA to obtain a weighted undirected graph G_{IP} where a node represents an IP address contained in the dataset (both source/destination hosts and intermediate hops), edges represent links among hops and weights are given by Round Trip Times. As the graph G_{IP} consists of almost 35000 nodes, we cannot use it for the experiments, as the amount of memory required to store the routing tables of all the nodes is $O(n^2 \cdot \mathit{maxdeg})$ for any implemented algorithm. Hence, we performed our tests on connected subgraphs of G_{IP}, with a variable number of nodes and edges, induced by the settled nodes of a breadth first search starting from a node taken at random. We generated a set of different tests, each test consists of a dynamic graph characterized by a subgraph of G_{IP} (we denoted each n nodes subgraph of G_{IP} with G_{IP-n}) and a set of k random edge updates, where k assumes values in $\{5, 10, \ldots, 200\}$. An edge update consists of multiplying the weight of a random selected edge by a percentage value randomly chosen in $[50\%, 150\%]$. For each test configuration (a dynamic graph with a fixed value of k) we performed 5 different experiments (for a total amount of 200 runs) and we report average values.

Concerning Barabási–Albert instances, we randomly generated a set of different tests, where a test consists of a dynamic graph characterized by a n nodes Barabási–Albert random graph, denoted as G_{BA-n} and a set of k random edge updates, where k assumes values in $\{5, 10, \ldots, 200\}$. Edge weights are non-negative real numbers randomly chosen in $[1, 10000]$. Edge updates are randomly chosen as in the CAIDA tests. For each test configuration (a dynamic graph with a fixed value of k) we performed 5 different experiments (for a total amount of 200 runs) and we report average values.

5.2 Analysis

We performed experiments on subgraphs of G_{IP} and on Barabási-Albert graphs having 1200, 5000 and 8000 nodes. The results of our experiments are quite similar on these different instances and then we report only the results on graphs with 8000 nodes. These results are shown in Figures 6, 7, and 8, concerning the number of messages sent, and in Table 1 concerning the space occupancy per node.

In detail, in Figure 6, 7, and 8 we report the number of messages sent by DUAL and DUAL-DLP, DUST and DUST-DLP, and LFR and LFR-DLP, respectively, on a

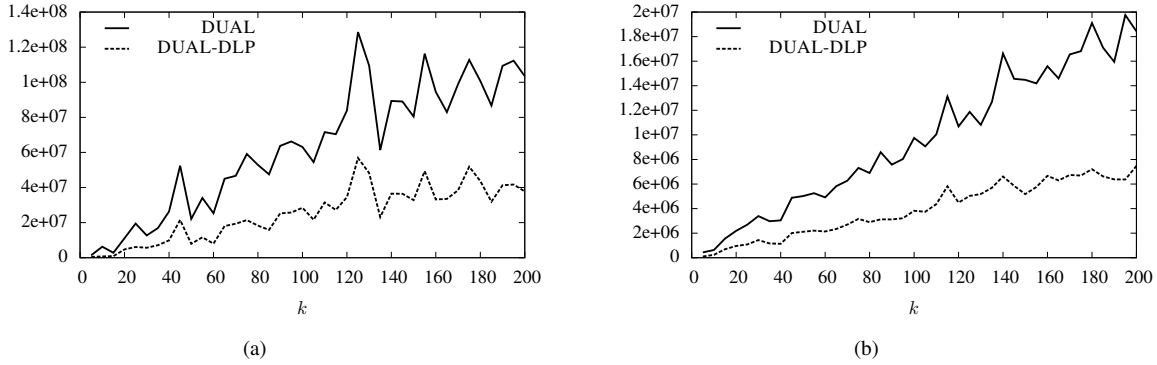

Figure 6: Number of messages sent by DUAL and DUAL-DLP on $G_{IP-8000}$ (a) and $G_{BA-8000}$ (b).

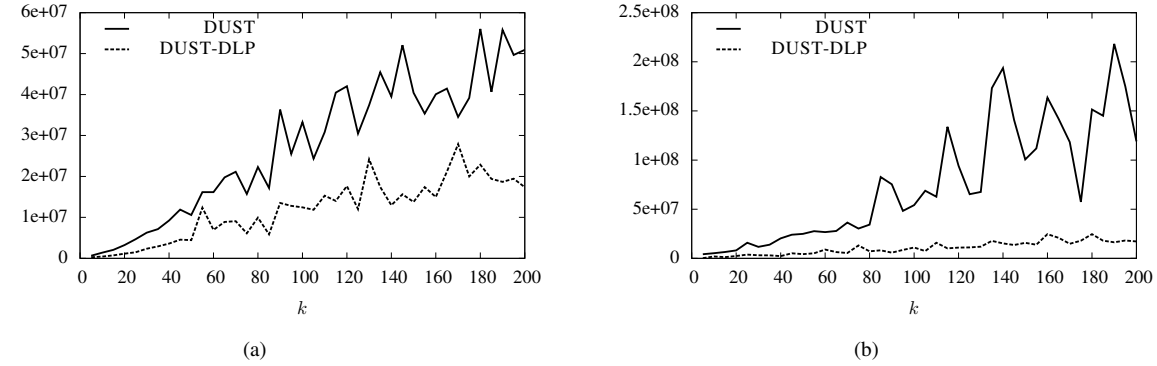

Figure 7: Number of messages sent by DUST and DUST-DLP on $G_{IP-8000}$ (a) and $G_{BA-8000}$ (b).

CAIDA graph having 8000 nodes and 11141 edges (a) and on a Barabási-Albert graphs having 8000 nodes and 12335 edges (b), when the number k of edge updates ranges from 5 to 200. These figures show in general that the combination of DLP with DUAL, DUST and LFR provides a significant improvement in the global number of messages sent by these algorithms.

Regarding $G_{IP-8000}$, we can observe what follows. In the tests of Figure 6(a), the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.12 and 0.46 which means that the number of messages sent by DUAL-DLP is between 12% and 46% that of DUAL. In the tests of Figure 7(a), the ratio between the number of messages sent by DUST-DLP and DUST is within 0.30 and 0.81. In the tests of Figure 8(a), the ratio between the number of messages sent by LFR-DLP and LFR is within 0.36 and 0.51.

Regarding $G_{BA-8000}$, we can observe what follows. In the tests of Figure 6(b), the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.23 and 0.48. In the tests of Figure 7(b), the ratio between the number of messages sent by DUST-DLP and DUST is within 0.04 and 0.43. In the tests of Figure 8(a), the ratio between the number of messages sent by LFR-DLP and LFR is within 0.38 and 0.47.

In summary, from our data follows that the algorithms integrating DLP send, on average, 0.42 (0.34, respectively) times the number of messages sent by the original algorithms on $G_{IP-8000}$ ($G_{BA-8000}$, respectively), which represents a substantial improvement in the practical applications where these algorithms are used.

To conclude our analysis, we have considered the space occupancy per node, which is summarized in Table 1, where we report the maximum and the average space occupancy per node of each algorithm, and the memory overhead required by the combination of that algorithm with DLP.

Note that DUAL requires a node v to store, for each destination, the estimated distance given by each of its neighbors and a set of variables used to guarantee loop-freedom, DUST only needs the estimated distance of v and the set `VIA`, for each destination and LFR requires to store, for each node v, the estimated distance of v and the feasible via to each source s, that is the node through which the distance to s is minimum, plus other variables needed to guarantee loop-freedom. Since in the sparse graphs we considered it is not common to have more than one via to a destination, the memory requirement of DUST is much smaller than that of DUAL, and smaller than that of LFR. Note also that the space occupancy of the data structure needed to implement DLP is not a function of the degree of the graph and is always bounded, in the worst case, by n. However, our experiments show that the use of DLP induces, in most of the cases, a clear improvement also in

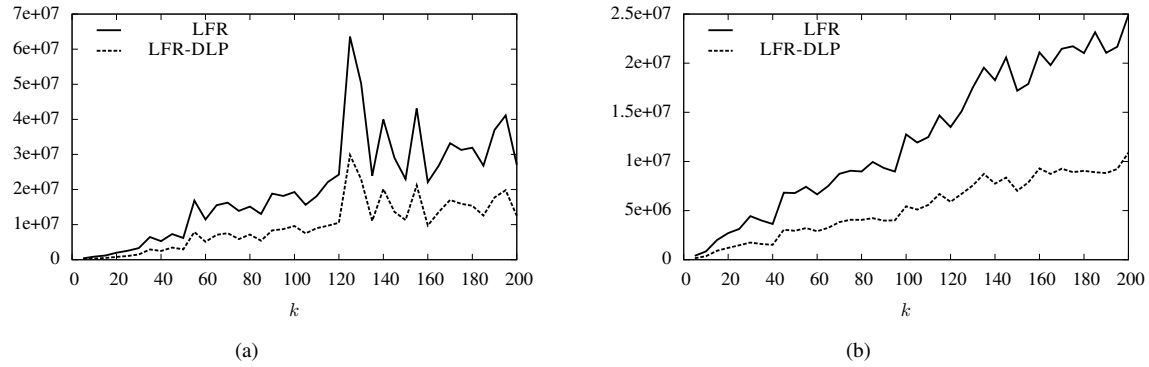

Figure 8: Number of messages sent by LFR and LFR-DLP on $G_{IP-8000}$ (a) and $G_{BA-8000}$ (b).

the maximum and in the average space requirements per node (see Table 1). This is due to the fact, that the overhead induced by the data structures needed to implement DLP, in most of the cases is overcome by the fact that DLP allows nodes to avoid to store some data concerning peripheral nodes, and the number of peripheral nodes in the networks used for the experiments is quite high. In particular, we can observe what follows. Concerning DUST, the combination with DLP determines an overhead which is always around 20% of the space occupancy of DUST, both in the maximum and in the average cases. This is due to the fact that DUST needs a really small amount of space and hence the data structures of DLP have a certain impact in the space occupancy of DUST. Concerning DUAL (LFR), we can observe that the use of DLP induces a gain in the maximum space occupancy per node which is around 38% (27%) on $G_{IP-8000}$ and around 45% (31%) on $G_{BA-8000}$. Notice that, the improvement is more evident in the case of DUAL, as its maximum space occupancy per node is by far higher than that of LFR. Concerning DUAL, this behavior is confirmed also in the average case, where the use of DLP induces a gain around 23% on $G_{IP-8000}$ and around 27% on $G_{BA-8000}$. On the contrary, our data show that the average space occupancy per node of LFR-DLP is slightly greater than that of LFR and that the use of DLP hence induces an overhead in the average space occupancy per node which is around 6% in $G_{IP-8000}$ and around 7% in $G_{BA-8000}$. This is due to the fact that the average space occupancy of LFR is quite low by itself and that, in this case, the space occupancy overhead needed to store the CT data structure is greater than the space occupancy reduction induced by the use of DLP.

6 Conclusions and future work

We have proposed a simple and practical technique, which can be combined with every distance vector routing algorithm based on shortest paths, allowing to reduce the total number of messages sent by that algorithm and the average space occupancy per node. We have combined the new technique with DUAL, DUST, and the recent LFR. We have given experimental evidence that these combinations lead to an important gain in terms of both the number of messages sent and the space occupancy per node.

Some research directions deserve further investigation: (i) to extend DLP in some way, for example considering nodes of small degree (greater than one) as peripherals; (ii) to know how DLP is scalable to bigger networks than those considered in this paper; (iii) to perform simulations on different power-law models, as for example the Generalized Linear Preference model (GLP) [4], which have particular properties that could impact on the performances of DLP.

Acknowledgments

Support for the IPv4 Routed/24 Topology Dataset is provided by National Science Foundation, US Department of Homeland Security, WIDE Project, Cisco Systems, and CAIDA.

References

[1] R. Albert and A.-L. Barabási. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[2] R. Albert and A.-L. Barabási. Statistical mechanics of complex network. *Reviews of Modern Physics*, 74:47–97, 2002.

[3] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall International, 1992.

[4] T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *Proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 37–48. IEEE, 2002.

[5] S. Cicerone, G. D'Angelo, G. Di Stefano, and D. Frigioni. Partially dynamic efficient algorithms for distributed shortest paths. *Theoretical Comp. Science*, 411:1013–1037, 2010.

Graph	Algorithm	Max		Avg	
		Bytes	DLP **Overhead**	**Bytes**	DLP **Overhead**
$G_{IP-8000}$	DUAL	8 320 000	—	311 410	—
	DUAL-DLP	5 161 984	-37.96%	240 754	-22.69%
	LFR	757 268	—	192 984	—
	LFR-DLP	554 987	-26.71%	204 724	6.08%
	DUST	64 088	—	64 000	—
	DUST-DLP	76 376	19.17%	76 288	19.20%
$G_{BA-8000}$	DUAL	20 800 000	—	323 350	—
	DUAL-DLP	11 483 200	-44.80%	240 550	-25.61%
	LFR	685 498	—	193 267	—
	LFR-DLP	475 466	-30.64%	207 076	7.15%
	DUST	64 056	—	64 000	—
	DUST-DLP	76 344	19.18%	76 288	19.20%

Table 1: Space complexity - Results of a dynamic execution with $k = 200$ over $G_{IP-8000}$ and $G_{BA-8000}$.

[6] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and V. Maurizio. A new fully dynamic algorithm for distributed shortest paths and its experimental evaluation. In *Proc. 9th Symposium on Experimental Algorithms (SEA)*, volume 6049 of *Lecture Notes in Computer Science*, pages 59–70, 2010.

[7] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and V. Maurizio. Engineering a new algorithm for distributed shortest paths on dynamic networks. *Algorithmica*, 66(1):51–86, 2013.

[8] S. Cicerone, G. Di Stefano, D. Frigioni, and U. Nanni. A fully dynamic algorithm for distributed shortest paths. *Theoretical Computer Science*, 297(1-3):83–102, 2003.

[9] G. D'Angelo, M. D'Emidio, D. Frigioni, and V. Maurizio. A speed-up technique for distributed shortest paths computation. In *Proc. 11th International Conference on Computational Science and Its Applications (ICCSA)*, volume 6783 of *Lecture Notes in Computer Science*, pages 578–593, 2011.

[10] G. D'Angelo, M. D'Emidio, D. Frigioni, and V. Maurizio. Engineering a new loop-free shortest paths routing algorithm. In *Proc. 11th Symposium on Experimental Algorithms (SEA)*, volume 7276 of *Lecture Notes in Computer Science*, pages 123–134, 2012.

[11] K. Elmeleegy, A. L. Cox, and T. S. E. Ng. On count-to-infinity induced forwarding loops in ethernet networks. In *Proc. of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–13, 2006.

[12] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251–281, 2000.

[13] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Trans. on Networking*, 1(1):130–141, 1993.

[14] P. A. Humblet. Another adaptive distributed shortest path algorithm. *IEEE Trans. on Communications*, 39(6):995–1002, April 1991.

[15] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, and KC Claffy. The CAIDA IPv4 routed/24 topology dataset. `http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml`.

[16] G. F. Italiano. Distributed algorithms for updating shortest paths. In *Proc. of the International Workshop on Distributed Algorithms*, volume 579 of *Lecture Notes in Computer Science*, pages 200–211, 1991.

[17] J. McQuillan. Adaptive routing algorithms for distributed computer networks. Technical Report BBN Rep. 2831, Cambridge, MA, 1974.

[18] J. T. Moy. *OSPF: Anatomy of an Internet routing protocol*. Addison-Wesley, 1998.

[19] A. Myers, E. Ng, and H. Zhang. Rethinking the service model: Scaling ethernet to a million nodes. In *ACM SIGCOMM HotNets*. ACM Press, 2004.

[20] OMNeT++. Discrete event simulation environment. `http://www.omnetpp.org`.

[21] A. Orda and R. Rom. Distributed shortest-path and minimum-delay protocols in networks with time-dependent edge-length. *Distr. Computing*, 10:49–62, 1996.

[22] K. V. S. Ramarao and S. Venkatesan. On finding and updating shortest paths distributively. *J. of Algorithms*, 13:235–257, 1992.

[23] S. Ray, R. Guérin, K. W. Kwong, and R. Sofia. Always acyclic distributed path computation. *IEEE/ACM Trans. on Networking*, 18(1):307–319, 2010.

[24] S. R. Soloway and P. A. Humblet. Distributed network protocols for changing topologies: A counterexample. *IEEE Trans. on Communications*, 39(3):360–361, March 1991.

[25] N. Yao, E. Gao, Y. Qin, and H. Zhang. Rd: Reducing message overhead in DUAL. In *Proc. of the 1st International Conference on Network Infrastructure and Digital Content (IC-NIDC09)*, pages 270–274. IEEE Press, 2009.

[26] C. Zhao, Y. Liu, and K. Liu. A more efficient diffusing update algorithm for loop-free routing. In *Proc. of the 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom09)*, pages 1–4. IEEE Press, 2009.