Stable Routing under the Spanning Tree Protocol

Fabrizio Grandoni[a], Gaia Nicosia[b], Gianpaolo Oriolo[*,c], Laura Sanità[d]

[a]*Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, 00133 Roma, Italy.*
[b]*Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, Via della Vasca Navale 79, I 00146 Roma, Italy.*
[c]*Dipartimento di Ingegneria dell'Impresa, Università di Roma Tor Vergata, via del Politecnico 1, 00133 Roma, Italy.*
[d]*Institute of Mathematics, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland.*

Abstract

The Spanning Tree Protocol routes traffic on shortest path trees. If some edges fail, the traffic has to be rerouted consequently, setting up alternative trees. In this paper we design efficient algorithms to compute polynomial-size integer weights so as to enforce the following stability property: if $q = O(1)$ edges fail, traffic demands that are not affected by the failures are not redirected. Stability is a goal pursued by network operators in order to minimize transmission delays due to the restoration process.

Key words: Stable routing, Spanning Tree Protocol, Randomized algorithms.

1. Introduction

Traffic in real-world networks is ruled by *routing protocols*, which establish along which path the traffic between a given pair of vertices has to be routed. One well known such protocol is the Spanning Tree Protocol (STP) [14], used for instance in Ethernet based networks [7]. We are given an undirected connected graph G, with n vertices $V(G)$, m edges $E(G)$, and edge weights $w : E(G) \to \mathbb{Z}_{\geq 1}$. (We assume that weights are positive integers due to technological reasons, though our results can be easily extended to more general settings). The STP chooses a root vertex r of G, and computes a shortest path tree T (namely, the *routing tree*) rooted at r which spans G. Traffic between each pair of vertices u and v is then routed along the unique u-v path in T. In this context, the weights w do not reflect network properties (such as delays etc.), but are rather *artificial* weights set up by traffic engineers in order to achieve the desired goals (e.g., minimizing edge congestion).

In this paper we focus on the behavior of the STP when a set F of at most q edges fail (i.e., F is removed from the edge set). We can think of q as a small constant, which is derived by a statistical analysis of the network and of its components.

If $G \setminus F$ is still connected, the STP will restore the traffic by computing a new routing tree T' rooted at r. Similarly, if $G \setminus F$ consists of a set of connected components $C_1, C_2, \dots, C_h$, the STP will choose a root r_i for each component C_i, and compute a shortest path tree T_i of C_i rooted at r_i. The T_i's form a *shortest path forest* T' of G. (Of course, in the latter case it is not possible to restore the traffic between vertices in different components).

Since rerouting is often expensive and may cause large delays [3, 8], in order to meet QoS constraints a key property of the rerouting strategy is requiring that *traffic which is not affected by failures is not redirected.* This property has been called *minimum service disruption* [9] or *strong resilience* [1]. The main goal of this paper is studying under which conditions this property can be enforced by properly choosing weights.

1.1. Our Contribution

Observe that requiring that a path of the routing tree T, that is not affected by the failures F, is also a path of the routing tree T' in $G \setminus F$, is the same as requiring that any edge e of T which does not belong to F, is also an edge of T'. Therefore, we consider the following alternative definition of *stability* which deals with the edges of T rather than the paths of T. (See also Figure 1).

Definition 1. *Let G be an undirected connected graph and q a non-negative integer. A weight function $w : E(G) \to \mathbb{Z}_{\geq 1}$ is* q-stable *if G admits a spanning tree T such that, for each $e \in E(T)$ and every $F \subseteq E(G)$, $|F| \leq q$, if $e \notin F$ then e belongs to every shortest path forest of $G \setminus F$. In this case, we say that w* defines *T. Also, if $q = |E(G)|$, we simply say that w is* stable.

Note that our definition implies (for $F = \emptyset$) that T is the *unique* shortest path tree of G, independently from the choice of the root r. This is convenient since STP has little control of the root (each node of the network has a given integer label, and STP chooses the root as the node with

[*]Corresponding author.
Email addresses: `grandoni@disp.uniroma2.it` (Fabrizio Grandoni), `nicosia@dia.uniroma3.it` (Gaia Nicosia), `oriolo@disp.uniroma2.it` (Gianpaolo Oriolo), `laura.sanita@epfl.ch` (Laura Sanità)

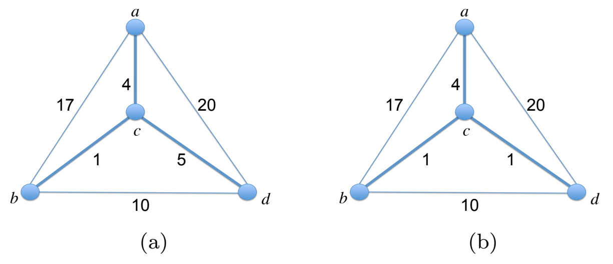

Figure 1: Two weighted graphs, with the routing trees determined by the STP for the no-failure state, with $r = a$, in bold. The weight function in (a) is not 1-stable: when edge ac fails, the new routing tree determined by the STP does not include the edge cd. The weight function in (b) instead is 1-stable: if any edge $e \in \{ac, bc, cd\}$ fails, the edges in $\{ab, bc, ad\} \setminus \{e\}$ belong to the new routing tree determined by the STP. The same holds if we move the root from a to some other vertex.

the minimum label) and network operators like to choose the routing tree T that is used in the no-failure state. For the latter reason, we will always assume that T is given, and search for q-stable weight functions which define T.

We point out that Definition 1 ensures stability of the routing paths determined by the STP *only with respect to the no-failure state* of the network. In other words, we do not put any constraint on the stability of the routing paths when passing from some failure state $F \neq \emptyset$ to some other failure state $F' \neq F$, $F' \neq \emptyset$. However, this is not a severe restriction, since the average time between two (sets of simultaneous) failures is usually greater than the time needed to repair the network (see e.g. [6]).

A first natural question is whether a stable weight function always exists (for any given T). As it is shown in Section 2, the answer is positive, and a rather simple weight function works: assign weight 1 to the edges of T, and weights $2n, 4n, 8n, \dots, 2^{m-n+1}n$ to the other edges of G.

While this is good news, this solution has a serious drawback: the weights are exponentially large in the size of the graph. This is a problem because the value of edge weights is upper bounded for technological reasons; for example, only a limited number of bits might be reserved to encode those weights (see for instance the ranges in [17]). We provide therefore in Section 3 deterministic and randomized algorithms that compute q-stable weight functions, with weights that are polynomially bounded in the size of G (though they are exponentially large in q). The exponential dependence on q is not a severe drawback since, as we said, typically q can be assumed to be small.

In order to obtain our results, we go as follows. We first show that dealing with q-stable weight functions is challenging, from a mathematical point of view. We therefore consider a more restricted class of q-stable weight functions, that we call normal and q-unbalanced. We then show that a normal and q-unbalanced weight function is enforced by a simple randomized assignment: this leads to a randomized (Monte-Carlo) algorithm. We are able to de-randomize this algorithm with the method of conditioned expectations, at the cost of a higher running time, which is still polynomial in the size of G.

Our algorithms are easy to implement and are fast for small values of q: hence they might be of practical interest.

1.2. Related Work

In the literature there is a number of papers that address the problem of setting the weights on the edges of a network so that, under some routing protocol, a certain objective is attained.

OSPF [13] and IS-IS [12] are well known routing protocols, which route each u-v demand on the shortest path from u to v according to the edge weights defined by the network operator. Many papers addressed the problem of setting edge weights in order to minimize the load network balance under those protocols [4, 5, 10, 11].

Other popular protocols, such as STP [14], RSTP [15], and MSTP [16], route traffic on a shortest path tree rooted at some root vertex. Some papers, see e.g. [2, 3, 8], deal with the problem of reconnecting the shortest path tree after the failure of *one* edge. However, in those papers edge weights are actually not chosen but given as input, and the back-up edge is selected so as to minimize a given objective function. To the best of our knowledge, the unique exception is [9], that addresses the problem of setting weights in order to achieve stability in networks using MSTP/STP protocols in the case of one edge failure. However, the assumption of having no more than one single failure at a time is not fully realistic. In contrast, we are able to deal with any number of failures.

1.3. Assumptions and Notation

For an undirected graph G, we let $V(G)$ and $E(G)$ denote the vertex and edge set of G, respectively. Given an edge weight function w, by $w_{max} := \max_{e \in E}\{w(e)\}$ we denote the largest weight. Furthermore, for any $E' \subseteq E(G)$, $w(E') := \sum_{e \in E'} w(e)$. By $H \subseteq G$ we mean that H is a subgraph of G. We sometimes use $w(H)$ as a shortcut for $w(E(H))$. A *q-subgraph* of G is a graph $H \subseteq G$ such that $V(H) = V(G)$, $E(H) \subseteq E(G)$ and $|E(H)| \geq |E(G)| - q$. In other words, a q-subgraph of G is a spanning subgraph of G that arises from the deletion of at most q edges of G. For a subset of edges $F \subseteq E(G)$, $G \setminus F$ is the graph with vertices $V(G)$ and edges $E(G) \setminus F$. If T is a tree connecting two vertices u and v, we denote by $P_{uv}(T)$ the *unique* path connecting u to v on T. A graph G is *k-edge-connected* if, for every subset $F \subseteq E(G)$ with $|F| < k$, the graph $G \setminus F$ is connected.

Since weights may assume exponential values (in the size of G), we take into account the number of bits of edge weights in the running time analysis. We assume that generating one random bit takes constant time.

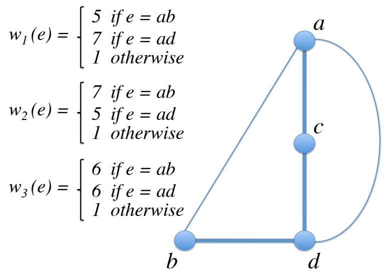
$w_1(e) = \begin{cases} 5 & \text{if } e = ab \\ 7 & \text{if } e = ad \\ 1 & \text{otherwise} \end{cases}$
$w_2(e) = \begin{cases} 7 & \text{if } e = ab \\ 5 & \text{if } e = ad \\ 1 & \text{otherwise} \end{cases}$
$w_3(e) = \begin{cases} 6 & \text{if } e = ab \\ 6 & \text{if } e = ad \\ 1 & \text{otherwise} \end{cases}$
a
c
b
d

Figure 2: The tree T consists of the edges of weight 1. It is easy to check that weight functions w_1 and w_2 are 1-stable. However, function w_3, which is a convex combination of w_1 and w_2, is not 1-stable: whenever edge ac fails, edge bd is not anymore in the shortest path tree rooted at a.

2. Unbalanced Weight Functions

Consider any weight function $w : E(G) \to \mathbb{R}_{\geq 1}$ (where we relaxed the integrality constraint). As it is shown in Figure 2, the set $\mathcal{W}$ of q-stable weight functions is in general even not convex! This suggests that dealing with stable weight functions is challenging, from a mathematical point of view. Indeed, we do not even know whether *recognizing* if a given (fractional) weight function is q-stable can be done in polynomial time, if q is not a constant. (Enumeration of the failure scenarios trivially works, if q is constant).

In this paper, we therefore concentrate on some *sufficient* conditions for a weight function to be stable. In particular, we restrict our attention to the following class of *normal* weight functions:

$$w(e) = \begin{cases} 1 & \text{if } e \in E(T); \\ c_e \cdot n & \text{otherwise,} \end{cases} \qquad (1)$$

where the c_e's are positive integer *edge coefficients*. It is straightforward to check that, if w is normal, then T is the unique shortest path tree of G rooted at r, for any vertex $r \in V(G)$. In the following, we refer to the edges of T as *tree edges*, and to the edges of $E \backslash T$ as *non-tree edges*. Also if we are given an integer q, and a normal weight function w, we define:

- $B(w)$, the set of *back-up edges*, i.e. non-tree edges that belong to some shortest path forest of a q-subgraph of G

The following definition and theorem are crucial in our analysis.

Definition 2. *For a given integer q, a normal weight function w is* q-unbalanced *if each pair of non-empty, disjoint subsets of back-up edges K_1, K_2, with $|K_1| + |K_2| \leq \min\{q+1, n-1\}$, is such that $\sum_{e \in K_1} w(e) \neq \sum_{e \in K_2} w(e)$.*

Observe that Definition 2 requires the knowledge of $B(w)$. Clearly, $B(w)$ can be computed by enumerating all the possible sets F of at most q failing edges, and, for each such scenario, the set of edges belonging to some minimum spanning forest. However, this is unnecessary, since we later deal with weight functions (see the one defined in 2) such that a superset of $B(w)$ is clear from the context, and that is enough for our purposes.

Theorem 3. *If a normal weight function w is q-unbalanced, then it is q-stable.*

Proof. Suppose by contradiction that the weight function w is not q-stable. We already observed that T is the unique shortest path tree of G rooted at any vertex in $V(G)$. We may therefore assume that there exists a q-subgraph $H \subset G$, a shortest path tree T' rooted at some node r', for some component C of H, and vertices u and v such that $P_{uv}(T) \subseteq H$ while $P_{uv}(T')$ contains some back-up edge. That is, if we let $Z_{uv} := E(P_{uv}(T')) \setminus E(T)$ be the set of back-up edges that belong to $P_{uv}(T')$, then $Z_{uv} \neq \emptyset$.

Without loss of generality, we assume that $E(P_{r'u}(T')) \cap E(P_{r'v}(T')) = \emptyset$, and therefore r' belongs to $E(P_{uv}(T'))$. Otherwise, there exists a node $r'' \neq r'$ such that r'' belongs to $E(P_{uv}(T'))$, and $P_{r'u}(T')$ and $P_{r'v}(T')$ share the subpath $P_{r'r''}(T')$. In this case, in our analysis, we can replace r' with r'' and T' with any shortest path tree for C that is rooted at r'' and includes the path $P_{uv}(T')$.

We will show that, without loss of generality, in the above hypotheses, the subgraph of G with edges $E(P_{r'u}(T')) \cup E(P_{uv}(T))$, that we will refer to as J, contains a path between r' and v that is shorter than $P_{r'v}(T')$. Since J is also a subgraph of H, that is a contradiction.

We claim that Z_{uv} has at least two back-up edges. Partition it into two disjoint subsets $Z_u := Z_{uv} \cap E(P_{r'u}(T'))$ and $Z_v := Z_{uv} \cap E(P_{r'v}(T'))$. Both Z_u and Z_v are non-empty: if e.g. $Z_u = \emptyset$, then J contains a path between r' and v composed by tree edges, whose cost is therefore at most $n-1$. This path would then be shorter than $P_{r'v}(T')$, whose cost is $\geq n$ since it contains at least one back-up edge, a contradiction. With similar arguments, it is possible to show that, for each edge $xy \in Z_{uv}$, the path $P_{xy}(T)$ does not belong to H; therefore, if we let T_H be the spanning forest with vertices $V(G)$ and edges $E(T) \cap E(H)$, x and y belong to different components of T_H.

Also observe that, since $|E(H)| \geq |E(G)| - q$, it follows that T_H has at most $q+1$ connected components $C_1, \ldots, C_{q+1}$. Clearly u and v belongs to the same component, say C_1, since $P_{uv}(T) \subseteq T_H$. Note that each edge in $E(T) \cap E(H)$ belongs to some component C_i and has weight 1, while each edge in Z_{uv} lies between two components C_i and C_j, $1 \leq i < j \leq q+1$ and has weight at least n. Therefore the path $P_{uv}(T')$ starts from C_1 and visit each component of T_H at most once, but for C_1, that is visited exactly twice, since the path starts and ends there. It follows that $|Z_u| + |Z_v| \leq q+1$. Also, trivially, $|Z_u| + |Z_v| \leq n-1$.

Therefore Z_u and Z_v are two non-empty subsets of back-up edges, with $Z_u \cap Z_v = \emptyset$ and $|Z_u| + |Z_v| \leq \min\{q+

$1, n-1\}$. Since w is q-unbalanced, we know that $|w(Z_v) - w(Z_u)| \geq n$. Assuming without loss of generality that $w(Z_u) + n - 1 < w(Z_v)$, it follows that in J there exists a path Π between r' and v such that:

$$w(\Pi) \leq w(Z_u) + n - 1 < w(Z_v) \leq w(P_{r'v}(T'))$$

that is, $P_{r'v}(T')$ is not the shortest path from r to v in C, a contradiction. $\square$

Building upon the previous theorem, we can show that a very simple (deterministic) polynomial-time algorithm, that we call `EXP-stable`, computes a stable weight function: The algorithm sets $w(e) = 1$ for all $e \in E(T)$, and lets $w(e_i) = 2^i n$ for the remaining edges $e_1, e_2, \ldots, e_{m-n+1}$.

Theorem 4. *Algorithm* `EXP-stable` *computes a stable weight function. The running time of the algorithm is $O(m^2)$, and the largest weight is $O(2^m n)$.*

Proof. Consider *any* two non-empty, disjoint subsets of *non-tree* edges K_1, K_2. The largest weight $w(e)$, $e \in K_1 \cup K_2$, is larger than $w(K_1 \cup K_2 - \{e\})$, from which $w(K_1) \neq w(K_2)$. We conclude by Theorem 3 that w is stable. The largest weight is $O(2^m n)$. Finally, the algorithm takes $O(m \log(2^m n)) = O(m^2)$ time to assign the weights. $\square$

We conclude the section with a remark. Using these results, one could give an alternative characterization of k-edge-connected graphs.

Definition 5. *A spanning tree T of a graph G has the q-exchange property, if, for each $F \subseteq E(G)$ with $|F| \leq q$, there exists $H \subseteq E(G) \setminus F$ such that $(T \setminus F) \cup H$ is a spanning tree of $G \setminus F$.*

Lemma 6. *Let G be an undirected connected graph. G has a spanning tree T with the q-exchange property if and only if G is $(q+1)$-edge-connected. Moreover, in this case, each spanning tree of G has the q-exchange property.*

Proof. Necessity is trivial. For the if part, let T be any spanning tree of a $(q+1)$-edge-connected graph G. Let F be a subset of at most q edges of $E(G)$: the graph $G \setminus F$ is still connected. It follows from Theorem 4 that there exists $w : E(G) \to \mathbb{Z}_{\geq 1}$ that is stable and defines T. Let T' be any shortest path tree of $G \setminus F$. By stability, $T' = (T \setminus F) \cup H$, for some suitable $H \subseteq E(G) \setminus F$. $\square$

We point out that a direct proof of Lemma 6, i.e. a proof that does not require Theorem 4, can also be given. The details are left to the interested reader.

3. Stability with Few Failures

As already mentioned, the solution produced by Algorithm `EXP-stable` has a serious drawback: namely, the largest weight is $O(2^m n)$, and therefore exponentially large in the size of the graph. In this section, we will therefore focus on the design of algorithms that return q-stable weight functions, with weights that are polynomially bounded in the size of the graph (though they are exponentially large in q). In the following, we let $\tilde{q} = \min\{q, n-2\}$.

The following observation is at the heart of all our algorithms. Let $G_0 = G$ and $S_0 = T$. For $i = 0, 1, \ldots, \tilde{q} - 1$, define $G_{i+1} = G_i \setminus E(S_i)$, and let S_{i+1} be a maximal spanning forest of G_{i+1}. We define $L = \cup_{i=1}^{\tilde{q}} E(S_i)$ and $M = E(G) \setminus (E(T) \cup L)$. It is not hard to see that, in case of no more than q failures F, the edges of L are sufficient to restore the traffic between any pair of vertices which are still connected in $G \setminus F$. This suggests the following strategy: we assign a sufficiently large weight to the edges of M, and focus on the weights of the edges of L. In more detail, we restrict our attention to the following subset of normal weight functions. For a proper integer $R \geq 1$,

$$w(e) = \begin{cases} 1 & \text{if } e \in E(T); \\ c_e n & \text{if } e \in L; \\ R n^2 & \text{if } e \in M, \end{cases} \qquad (2)$$

where $c_e \in \{1, 2, \ldots, R\}$.

Recall that $B(w)$ is the set of back-up edges, i.e. the non-tree edges that belong to some shortest path forest of a q-subgraph of G.

Lemma 7. *With the weight function above, no edge in M is a back-up edge.*

Proof. We observe that M is empty for $q > n-2$, so assume $\tilde{q} = q$. Suppose that there is a q-subgraph $H \subseteq G$, an edge $uv \in M$ and a shortest path forest T' of H such that $uv \in T'$. By construction, in the original graph G there exist $q+1$ edge-disjoint paths from u to v that do not use uv (one path in T, and q paths in $S_1, \ldots, S_q$). Since H is a q-subgraph, at least one of these paths P is still in H. Then, since $w(uv) = Rn^2$ and $w(P) \leq Rn(n-1)$, T' is not a shortest path forest of H. $\square$

All our algorithms will set the coefficients c_e such that w is q-unbalanced (and hence q-stable). They differ in the choice of the coefficients.

3.1. A Monte-Carlo Algorithm

We now present a Monte-Carlo algorithm `MC-stable` to compute q-stable weight functions. We recall that a Monte-Carlo algorithm is a randomized algorithm which fails to provide a correct answer with positive probability.

Algorithm `MC-stable` simply chooses the coefficients c_e uniformly and independently at random in the range $\{1, 2, \ldots, R\}$, for a proper choice of R. The following lemma shows that, for R large enough, the resulting weight function is q-unbalanced with large probability.

Lemma 8. *The weight function w computed by* `MC-stable` *is q-unbalanced with probability at least $1 - (2|B(w)|)^{\tilde{q}+1}/R$, where $\tilde{q} = \min\{q, n-2\}$.*

Proof. Call *feasible* a pair $\{K_1, K_2\}$ of non-empty, disjoint subsets of $B(w)$, with $|K_1| + |K_2| \le \tilde{q} + 1$. If $\{K_1, K_2\}$ is a feasible pair, we let $BAD(K_1, K_2)$ denote the event that $w(K_1) = w(K_2)$, i.e. that $\sum_{e \in K_1} c_e = \sum_{e \in K_2} c_e$.

Take a feasible pair $\{K_1, K_2\}$ and let $e' \in K_1$. Define $X := \sum_{e \in K_2} c_e - \sum_{e \in K_1 \setminus \{e'\}} c_e$. Clearly, $Pr[BAD(K_1, K_2)] = Pr[c_{e'} = X]$. Recall that, for any $x \in \mathbb{Z}, Pr[c_{e'} = x] \le 1/R$. Therefore, if we condition on the value of X, we have:

$$Pr[BAD(K_1, K_2)] = Pr[c_{e'} = X] = \sum_{x \in \mathbb{Z}} Pr[X = x] \cdot Pr[c_{e'} = x]$$
$$\le \sum_{x \in \mathbb{Z}} Pr[X = x] \frac{1}{R} = \frac{1}{R}.$$

Let N denote the number of feasible pairs, and let $h^* \in \{2, 2, \ldots, \tilde{q} + 1\}$ be such that $\binom{|B(w)|}{h^*} = \max_{h=2,\ldots,\tilde{q}+1} \binom{|B(w)|}{h}$. Then:

$$N \le \sum_{h=2}^{\tilde{q}+1} \binom{|B(w)|}{h} 2^{h-1} \le \binom{|B(w)|}{h^*} \sum_{h=2}^{\tilde{q}+1} 2^{h-1} \le (2|B(w)|)^{\tilde{q}+1}.$$

Finally, if we denote as BAD the event that $BAD(K_1, K_2)$ is true, for some pair of feasible subsets K_1 and K_2, from the union bound we have:

$$Pr[BAD] \le \sum_{K_1, K_2} Pr[BAD(K_1, K_2)] \le \frac{N}{R} \le \frac{(2|B(w)|)^{\tilde{q}+1}}{R}.$$

□

In the following, we assume $R = 2(2\tilde{q}n)^{\tilde{q}+1}$.

Theorem 9. *Algorithm* `MC-stable` *computes a q-stable weight function with probability at least $1/2$. The running time of the algorithm is $O(mn \log n)$ and the largest weight is $O((2\tilde{q}n)^{\tilde{q}+1} n^2)$, where $\tilde{q} = \min\{q, n-2\}$.*

Proof. The weight function in (2) is normal. Also it follows from Lemma 7 that each back-up edge belongs to L. Since $|L| \le \tilde{q}n$, it follows from Lemma 8 that w is q-unbalanced, and therefore q-stable by Theorem 3, with probability at least $1 - (2\tilde{q}n)^{\tilde{q}+1}/R = 1/2$. The largest weight is $O((2\tilde{q}n)^{\tilde{q}+1} n^2)$. The spanning forests can be computed in $O(m\tilde{q}) = O(mn)$ time. All the weights can be assigned in $O(m \log R) = O(mn \log n)$ time. □

Of course, we can reduce the failure probability of `MC-stable` to any $\varepsilon > 0$, by choosing $R = (2\tilde{q}n)^{\tilde{q}+1}/\varepsilon$.

3.2. A Las-Vegas Algorithm

Algorithm `MC-stable` can be turned into a Las-Vegas algorithm `LV-stable`, at the cost of a higher running time. We recall that a Las-Vegas algorithm is a randomized algorithm which answers correctly deterministically, but whose running time is a random variable. The idea is simply running `MC-stable` (with $R = 2(2\tilde{q}n)^{\tilde{q}+1}$), and checking whether the weight function w produced is q-unbalanced. If not, the process is iterated.

Theorem 10. *Algorithm* `LV-stable` *computes a q-stable weight function. The expected running time is $O((2\tilde{q}n)^{\tilde{q}+1} \cdot \tilde{q}^2 \log n)$, and the largest weight is $O((2\tilde{q}n)^{\tilde{q}+1} n^2)$, where $\tilde{q} = \min\{q, n-2\}$.*

Proof. The largest weight is $O((2\tilde{q}n)^{\tilde{q}+1} n^2)$ since the same holds deterministically for each execution of `MC-stable`. By Theorem 9, each execution of `MC-stable` succeeds with probability at least $1/2$, and hence the expected number of iterations is 2. By Theorem 9, each execution of `MC-stable` takes $O(mn \log n)$ time. Checking whether the weight function w produced at some iteration is q-unbalanced (and therefore q-stable by Theorem 3) can be done by checking, for each pair $\{K_1, K_2\}$ of non-empty, disjoint subsets of L, with $|K_1| + |K_2| \le \tilde{q} + 1$, whether $w(K_1) \ne w(K_2)$. For each given pair $\{K_1, K_2\}$, this can be done in time $O(\tilde{q} \log R) = O(\tilde{q}^2 \log n)$. As we have shown in the proof of Lemma 8, the number of candidate pairs is $O((2|L|)^{\tilde{q}+1}) = O((2\tilde{q}n)^{\tilde{q}+1})$. The claim follows. □

3.3. A Deterministic Algorithm

We now describe how to achieve the same task as `LV-stable` with a deterministic algorithm `DET-stable`, by further increasing the running time. The idea is derandomizing `MC-stable` via the method of conditioned expectation. In more details, the coefficients c_e are fixed through the following deterministic procedure. We arbitrarily order the edges $e_1, e_2, \ldots, e_l$ in L, and arbitrarily fix the coefficient of e_1 (say, $c_{e_1} = 1$). Then there is a sequence of $l - 1$ iterations. At the beginning of iteration i, for $i = 2, \ldots, l$, the coefficients of edges $L_{i-1} := \{e_1, \ldots, e_{i-1}\}$ are fixed, and at the end of the iteration c_{e_i} is fixed as well. The value of c_{e_i} is simply set to any value in $\{1, 2, \ldots, R\}$ which satisfies the following:

$$\begin{aligned} &\forall K_1, K_2 \subseteq L_i, K_1, K_2 \ne \emptyset, K_1 \cap K_2 = \emptyset, \\ &|K_1| + |K_2| \le \tilde{q} + 1 : \quad \sum_{e \in K_1} c_e \ne \sum_{e \in K_2} c_e. \end{aligned} \tag{3}$$

Note that the q-unbalance property just follows from (3) when $i = l$.

Theorem 11. *Algorithm* `DET-stable` *computes a q-stable weight function. The running time is $O((2\tilde{q}n)^{2(\tilde{q}+1)} \tilde{q}^3 n \log n)$, and the largest weight is $O((2\tilde{q}n)^{\tilde{q}+1} n^2)$, where $\tilde{q} = \min\{q, n-2\}$.*

Proof. The claim on the largest weight follows from Theorem 10. Checking whether one value c_{e_i} satisfies (3) takes $O((2\tilde{q}n)^{\tilde{q}+1} \tilde{q} \log R)$ time, since the number of pairs $\{K_1, K_2\}$ is $O((2\tilde{q}n)^{\tilde{q}+1})$, and checking each pair takes $O(\tilde{q} \log R)$ time. The number of values checked is $O(R\tilde{q}n)$, since we have $O(\tilde{q}n)$ edges and $O(R)$ possible values for each edge. Hence the running time is $O(R\tilde{q}n(2\tilde{q}n)^{\tilde{q}+1} \tilde{q} \log R) = O((2\tilde{q}n)^{2(\tilde{q}+1)} \tilde{q}^3 n \log n)$. As already observed, if the algorithm halts correctly by setting all the coefficients c_{e_i}, $i = 2, \ldots, l$, then the resulting weight function is q-unbalanced (and therefore q-stable).

It remains to show that at each iteration i a value of c_{e_i} exists such that $c_{e_1}, c_{e_2}, \ldots, c_{e_i}$ satisfies (3). The claim holds trivially for $i = 2$, by choosing $c_{e_2} \neq c_{e_1}$. Now assume by induction that the claim holds up to some iteration $i \geq 2$, and consider iteration $i+1$. We show that a proper coefficient $c_{e_{i+1}}$ exists via a probabilistic argument. In more detail, we show that, if we choose $c_{e_{i+1}}$ uniformly at random in $\{1, 2, \ldots, R\}$, then $c_{e_{i+1}}$ has the desired property with *positive* probability; this implies that at least one value in $\{1, 2, \ldots, R\}$ satisfies the claim (deterministically).

Consider two subsets $K_1, K_2 \subseteq L_{i+1}$ such that $K_1, K_2 \neq \emptyset, K_1 \cap K_2 = \emptyset$, and $|K_1| + |K_2| \leq \tilde{q} + 1$. If $K_1, K_2 \subseteq L_i$ then, by the inductive hypothesis, $\sum_{e \in K_1} c_e \neq \sum_{e \in K_2} c_e$ holds deterministically. So assume that e.g. $e_{i+1} \in K_1$. By the same argument as in the proof of Lemma 8, the event $\sum_{e \in K_1} c_e = \sum_{e \in K_2} c_e$ happens with probability at most $1/R$. Furthermore, the number of pairs K_1, K_2 is at most $R/2$ given our choice of R. Hence, $c_{e_1}, \ldots, c_{e_i}, c_{e_{i+1}}$ satisfy (3) with probability at least $1/2$. □

One may notice that the running time of `DET-stable` is exponential in q. This is not the case for the other deterministic algorithm `EXP-stable` described in Section 2. However, we remark that the weights computed by `EXP-stable` can be as large as $\Theta(2^{\Theta(n^2)} n)$, while the value of the weights computed by `DET-stable` are polynomially bounded in the size of G for $q = O(1)$, and in any case the maximum weight is no larger than $O((2\tilde{q}n)^{\tilde{q}+1} n^2) = O((2n^2)^n n^2) = O(2^{O(n \log n)})$.

Our results can be easily extended to the case where zero weights are allowed, i.e. $w : E(G) \rightarrow \mathbb{Z}_{\geq 0}$. It is enough to change (2) as follows: set to zero the weight of edges in T, and reduce by a factor n the weights of the remaining edges. Similarly, rational weights can be handled by scaling weights by a proper factor.

3.4. A Lower Bound on the Largest Weight

As observed before, the weights computed by our algorithms are exponentially large in q. The next theorem shows that this must be the case if we insist on q-unbalanced weights.

Theorem 12. *For any $q \geq 1$ with $\sqrt{q} \in \mathbb{Z}$, there is a graph G such that any normal, q-unbalanced weight function w satisfies: $w_{max} \geq (\sqrt{q}-2)!/(\sqrt{q}-1)$.*

Proof. Let $n \geq 2\sqrt{q}$. Consider a graph G consisting of a clique Q of $\sqrt{q}$ vertices, plus $n - \sqrt{q}$ vertices of degree one that are adjacent only to some $x \in Q$. Assume by contradiction that, for some normal, q-unbalanced weight function w, $w_{max} := \max_{e \in E}\{w(e)\} < (\sqrt{q}-2)!/(\sqrt{q}-1)$.

Choose two vertices u and v in Q. Consider the collection $\mathcal{P}$ of simple u-v paths. Note that each path in $\mathcal{P}$ takes vertices only from Q. The number of paths in $\mathcal{P}$ is at least $(\sqrt{q}-2)! > w_{max} \cdot (\sqrt{q}-1)$, and the weight of any of these paths is an integer number in $\{1, 2, \ldots, w_{max} \cdot (\sqrt{q}-1)\}$. It follows that two paths P' and $P'' \in \mathcal{P}$ have the same weight. Now observe that each edge of Q, and in particular each edge of $E(P') \cup E(P'')$, is either a back-up edge, or a tree edge. In fact, let $y, z \in Q$ and consider the subgraph $H \subseteq G$ that is obtained by removing from G all the edges of Q, but the edge yz. Since there are less than q edges of G between vertices of Q, H is a q-subgraph. Trivially yz belongs to every shortest path forest of H.

Let $H \subseteq G$ be such that $V(H) = V(G)$ and $E(H) = (E(P') \setminus E(P'')) \cup (E(P'') \setminus E(P'))$. Notice that H has a cycle, and therefore an edge $e \in E(H)$ does not belong to T (i.e., e is a back-up edge). Without loss of generality assume that $e \in E(P_1)$. Let K_1 and K_2 be the back-up edges in $E(P') \setminus E(P'')$, and $E(P'') \setminus E(P')$, respectively. By definition, $K_1 \cap K_2 = \emptyset$ and our previous assumption shows that $K_1 \neq \emptyset$. Now recall that w is normal, so the tree edges have weight 1, while each back-up edge f has weight $c_f n$, with c_f a positive integer. It follows that, if $\sum_{e \in E(P')} w(e) = \sum_{e \in E(P'')} w(e)$, then $\sum_{e \in K_1} w(e) = \sum_{e \in K_2} w(e)$, and, in particular, that $K_2 \neq \emptyset$. But then, K_1, K_2 is a pair of non-empty, disjoint subsets of back-up edges, with $|K_1| + |K_2| \leq \tilde{q} + 1$, such that $\sum_{e \in K_1} w(e) = \sum_{e \in K_2} w(e)$, a contradiction. □

Open Questions. There are many questions left open. Which is, for a given graph G, the smallest value $w_{max}(G)$ such that a q-stable weight function with weights not larger than $w_{max}(G)$ exists? Which is the value of $w_{max}(n) := \max_{G:|V(G)|=n}\{w_{max}(G)\}$? Is $w_{max}(n) = O(n^c)$, for some constant c independent from q? We know by Theorem 12 that this is not the case if w is normal and q-unbalanced.

Another question is whether a q-stable weight function with $w_{max} = w_{max}(G)$ can be computed efficiently for a given G. We remark that we do not even know how to check that a given weight function is q-stable in polynomial time unless q is constant.

The running time of `DET-stable` grows exponentially in q. Is there an algorithm producing weights asymptotically not larger than `DET-stable`, and with running time $O(n^c)$ for some constant c independent of q? Note that this holds for `MC-stable`, but in that case the algorithm fails with positive probability.

Acknowledgments. The author Laura Sanità was supported by Swiss National Science Foundation within the project "Robust Network Design". The authors wish to thank the industrial partners Paola Iovanna, Giorgio Lembo and Fabio Palozza, for introducing them to the problem. They also thank Carlo Mannino, Nicolai Hähnle and Ezio Sperduto for useful discussions. Finally a special thank is due to Martin Skutella, for sparking the idea of a randomized approach.

References

[1] G. Brightwell, G. Oriolo, F.B. Sheperd, *Reserving Resilient Capacity in a Network.* Siam Journal on Discrete Mathematics, 14(4), pp. 524-539, 2001.

[2] A. Di Salvo, G. Proietti, *Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor*. Theoretical Computer Science, 383 (1), pp. 23-33, 2007.

[3] P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, P. Widmayer, *Computing all the best swap edges distributively.* Journal of Parallel and Distributed Computing, 68(7), 2008.

[4] B. Fortz, M. Thorup, *Internet traffic engineering by optimizing OSPF weights*, in Proc. of IEEE INFOCOM, pp. 519-528, 2000.

[5] B. Fortz, M. Thorup, *Optimizing OSPF/IS-IS weights in a changing world.* IEEE Journal on Selected Areas in Communications, 20(4), pp.756–767, 2002.

[6] J-P. Vasseur, M. Pickavet, P. Demeester, *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[7] R.M. Metcalfe, D.R. Boggs, *ETHERNET: Distributed Packet Switching for Local Computer Networks.* Communications ACM, 19(7), pp. 395-404, 1976.

[8] E. Nardelli, G. Proietti, P. Widmayer, *Swapping a Failing Edge of a Single Source Shortest Paths Tree Is Good and Fast.* Algorithmica, 35, pp. 56-74, 2003.

[9] A. F. de Sousa, G. Soares, *Improving Load Balance and Minimizing Service Disruption on Ethernet Networks using IEEE 802.1S MSTP.* in Proc. of Workshop on IP QoS and Traffic Control, Lisbon, Portugal, (1), pp. 25-35, 2007.

[10] N. Taft, A. Nucci, S. Bhattacharyya, C. Diot, *IGP edge Weight Assignment for Operational Tier-1 Backbones.* IEEE/ACM Transactions on Networking, Vol. 15 (4), pp. 789-802, 2007.

[11] Y. Wang, Z. Wang, L. Zhang, *Internet Traffic Engineering without Full Mesh Overlaying.* in Proc. IEEE INFOCOM, 2001.

[12] ISO/IEC 10589:2002 - IS-IS protocol.

[13] RFC 1131 - OSPF protocol.

[14] Standard IEEE 802.1D - Spanning Tree protocol.

[15] Standard IEEE 802.1w - Rapid Spanning Tree protocol.

[16] Standard IEEE 802.1s - Multiple Spanning Tree protocol.

[17] http://www.hojmark.net/stp-port-cost.html