Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

How Stable is Stable in Interdomain Routing: Efficiently Detectable Oscillation-Free Configurations

Luca Cittadini[1], Giuseppe Di Battista[1], and Massimo Rimondini[1]

RT-DIA-132-2008 **July 2008**

(1) Dipartimento di Informatica e Automazione,
Università di Roma Tre,
Rome, Italy.
{ratm,gdb,rimondin}@dia.uniroma3.it

This work is partially supported by the Italian Ministry of Research, Grant number RBIP06BZW8, FIRB project "Advanced tracking system in intermodal freight transportation".

ABSTRACT

Internet Service Providers have at their disposal a powerful policy-based protocol for enforcing a fine grained control of Interdomain Routing: the Border Gateway Protocol. However, the price to pay for the flexibility of BGP is the lack of convergence guarantees. In this paper, we study the stability of BGP configurations. Namely, we tackle the problem of deciding if, given the policy configurations, the routing will converge to a stable state or if there is the potential chance of persistent oscillations. First, we extend the currently largest known class of stable configurations that can be efficiently detected. This is done with a simple algorithm that relies on new properties of policy-based protocols that are shown to be independent on any specific message timing. Second, we provide a sufficient condition to guarantee the absence of potentially persistent oscillations in the routing. We show that this condition is less constraining than currently known ones. Finally, we assess the ability of existing models of policy-based protocols to capture routing oscillations, showing that different models put in evidence different types of oscillations. We prove that all our results are valid even in the most general model.

1 Introduction

Internet Service Providers have at their disposal a powerful policy-based protocol for enforcing a fine grained control of Interdomain Routing: the Border Gateway Protocol (BGP). However, the price to pay for the flexibility of BGP is the lack of convergence guarantees. In fact, a BGP configuration may oscillate forever, never reaching a stable state. This can happen either because a stable state for that configuration does not exist at all or because, even if a stable state exists, the system gets trapped into bad event timings.

Changes in interdomain routing are likely to cause performance degradation or even packet loss [26]. Even worse, continuous changes can severely affect the availability of services. For this reason, the stability of interdomain routing has been the subject of several studies.

The existence of persistent routing oscillations caused by an autonomous route selection process was first studied in [25], where Varadhan et al. discuss how to achieve stability by applying constraints on routing policies.

A number of results, which most of subsequent research efforts regarded as fundamental, have been proposed in a group of papers by Griffin et al. [11, 8, 13, 9]. The main achievements of those papers are: (i) A framework, called the Simple Path Vector Protocol (Spvp), to model the dynamics of path vector protocols. (ii) A static formalism, called the Stable Path Problem, to study BGP routing stability. (iii) A proof of NP-completeness for the Stable Path Problem. (iv) An efficient algorithm to compute a stable state in a greedy manner. (v) A sufficient condition for guaranteed routing convergence to a stable state based on the absence of cyclic structures, called dispute wheels, in the policy configurations.

Thanks to its generality, that sufficient condition has been widely exploited in the literature. Gao et al. [7, 6] defined guidelines on routing policies that, if obeyed, guarantee the convergence. The guidelines impose that preferences on routing paths are based on the commercial customer-provider relationships between Autonomous Systems. In [6] the authors show that enforcing those constraints implies the absence of dispute wheels.

Jaggard et al. focused in [15] on the more general case in which route preferences are assigned to classes of neighbors. They provide a centralized and a distributed algorithm that efficiently verify convergence by checking the possible presence of dispute wheels.

In [5] Feamster et al. provide conditions for convergence for the case in which ISP's can autonomously choose to filter out paths. They propose a sufficient condition that is again based on the absence of dispute wheels and a necessary condition that is based on the absence of a special subclass of dispute wheels called dispute rings.

The stability of path vector protocols has also been studied using an approach based on algebraic models in [24, 23, 10, 18]. These works describe convergence conditions that are based on properties of path rankings, and confirm that the no dispute wheel condition finds a counterpart also in the algebraic model. In particular, in [10] the authors propose a relaxation of the guidelines of [7].

In [17] Chau takes into account the general case in which routing policies allow for non-strict path rankings. Even in this case, the absence of dispute wheels is fundamental to guarantee convergence.

Several works [12, 14, 1, 20, 16, 21] focused on oscillations induced by IBGP configurations. They show how instabilities can arise from improper setup of route reflectors [2] or wrong configuration of MED [22] values, and that checking for the existence of stable states in IBGP is NP-hard [12, 1]. It has also been shown in [14, 16, 21] how the Stable Path Problem formalism and the no dispute wheel condition can be used to study the convergence of IBGP.

Other approaches focused on modifications to BGP that prevent route oscillations from happening. For this purpose, in [4] a global precedence metric is introduced to track the behavior of dispute wheels. The global precedence is used to neutralize those dispute wheels that are actually causing permanent oscillations. Cobb et al. [3] prevent instabilities by imposing consistency constraints on the routing paths that can be selected.

Our contributions can be summarized as follows.

First, we discuss the models to be adopted for studying oscillations. Many of the papers discussed above adopt the Spvp model from [8, 13], either in its original version or with some variations. We argue that the differences among those models impact their ability to capture routing oscillations. In particular, we point out that none of the features of the original version of Spvp can be omitted without missing some classes of oscillations. Hence, we adopt that model for our study.

Second, we propose a deterministic greedy algorithm that efficiently checks whether a BGP system is guaranteed to converge to a stable state. We prove that our algorithm is able to verify the guaranteed convergence of a set of instances that is strictly larger than those that are verified by the greedy algorithm in [9].

Finally, we prove that a sufficient condition to prevent routing oscillations is the absence of a special subclass of dispute wheels which we call Steady Spoke Dispute Wheels. This sufficient condition is less constraining than the widely used no dispute wheel condition presented in [8].

The rest of the paper is organized as follows. Section 2 introduces the notation and discusses the variants of Spvp. In Section 3 we formally define the concept of routing oscillation and choose the model. Our algorithm is described in Section 4 and analyzed in Section 5. Conclusions are drawn in Section 6, where directions for future work are also discussed.

2 Models for Policy-Based Path Vector Routing Protocols

Several models have been proposed in the literature [8, 7, 13, 25, 6, 9, 1, 3, 5] to capture the dynamic behavior of path vector protocols, with special interest to BGP. In this section we propose a taxonomy of existing approaches and introduce the model that we use throughout the paper.

Let $G = (V, E)$ be a simple undirected graph with vertex set $V = \{0, 1, \ldots, n\}$ and edge set E. A *path* P in G is either the empty path, denoted by ϵ, or a sequence of $k+1$ vertices $P = (v_k\ v_{k-1}\ \ldots\ v_0)$, $v_i \in V$ such that $(v_i, v_{i-1}) \in E$ for $0 < i \leq k$. Vertex v_{k-1} is the *next hop* of vertex v_k in P. If $k = 0$ then (v_0) is a path consisting of a single vertex v_0, which we call a *trivial path*. The *concatenation* of two nonempty paths $P = (v_k\ v_{k-1}\ \ldots\ v_i)$, $k \geq i$, and $Q = (v_i\ v_{i-1}\ \ldots\ v_0)$, $i \geq 0$, denoted as PQ, is the path $(v_k\ v_{k-1}\ \ldots\ v_i\ v_{i-1}\ \ldots\ v_0)$. We assume that $P\epsilon = \epsilon P = \epsilon$.

Each vertex $v \in V$ is assigned a set of *permitted paths* $\mathcal{P}^v$. All these paths start from v and end in 0 and represent the paths that v can use to reach 0. Let $\mathcal{P}^0 = \{(0)\}$ and let $\mathcal{P} = \bigcup_{v \in V} \mathcal{P}^v$.

For each vertex $v \in V$, a *ranking function* $\lambda^v : \mathcal{P}^v \to \mathbb{N}$ determines the relative level of preference $\lambda^v(P)$ assigned by v to path P. If $P_1, P_2 \in \mathcal{P}^v$ and $\lambda^v(P_2) < \lambda^v(P_1)$, then P_2 is said to be *preferred* over P_1. Let $\Lambda = \{\lambda^v | v \in V\}$.

We also assume that, for each vertex $v \in V - \{0\}$, the following conditions hold on the paths:

i. $\epsilon \in \mathcal{P}^v$;

ii. $\forall P \in \mathcal{P}^v$ with $P \neq \epsilon$: $\lambda^v(\epsilon) > \lambda^v(P)$;

iii. $\forall P_1, P_2 \in \mathcal{P}^v, P_1 \neq P_2 : \lambda^v(P_1) = \lambda^v(P_2) \Rightarrow P_1 = (v\ u)P_1', P_2 = (v\ u)P_2'$, i.e., v has the same next hop in P_1 and in P_2; and

iv. $\forall P \in \mathcal{P}^v$: P is a simple path (i.e., has no repeated vertices).

Assuming that the empty path represents unreachability of 0, Condition *i* states that all vertices but 0 may be unable to reach the destination. Unreachability is considered as the last chance for a vertex (Condition *ii*). Condition *iii* states that function λ^v induces a total order on all the paths of $\mathcal{P}^v$, with the exception of those paths that begin with the same pair of vertices. Such paths reach 0 via the same neighbor of v, and can therefore be considered equivalent. Condition *iv* accounts for the inexistence of cycles in the routing.

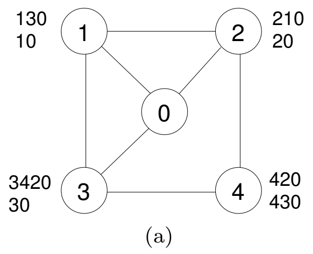
We now define a distributed asynchronous algorithm which is intended to model the dynamic behavior of path vector protocols, with particular reference to BGP [22]. The algorithm was first introduced in [8, 13] under the name of *Simple Path Vector Protocol* (Spvp).

An instance S of Spvp is defined by a t-uple $S = (G, \mathcal{P}, \Lambda)$, where $G = (V, E)$ is a graph, $\mathcal{P}$ is the set of permitted paths, and Λ is the set of ranking functions. Figures 1a and 1b show two instances of Spvp. The graphical convention we use in these figures will be adopted throughout the paper. Namely, each vertex v is equipped with a list of paths representing $\mathcal{P}^v$ sorted by increasing values of λ^v. The empty path and $\mathcal{P}^0$ are omitted for brevity.

The *size* of an instance of Spvp is the sum of the sizes of the sets of paths of $\mathcal{P}$. In fact, we can remove from G all vertices and edges that are not in at least one of these paths.

Observe that an instance of Spvp provides a good abstraction of an interdomain routing system. In fact, vertices can be mapped to Autonomous Systems, edges can be interpreted as peering sessions, and permitted paths, together with the rankings, can represent routing policies.

In Spvp, every vertex attempts to establish a path to 0 relying on the paths used by its neighbors. In order to achieve this, vertices exchange messages containing permitted paths to 0. We assume that



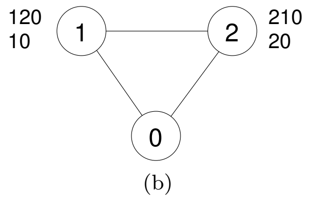


Figure 1: Two interesting instances of Spvp. (a) Bad Gadget: This instance does not admit any stable state. (b) Disagree: In this instance oscillations are only possible in a model that allows simultaneous activations.

```
process spvp($v$)
1: while receive $P$ from $u$ do
2:   rib-in$_t(v \Leftarrow u) := P$
3:   rib$_t(v)$ := best$_t(v)$
4:   if rib$_t(v) \neq$ best$_{t-1}(v)$ then
5:     for all $v \in$ peers($v$) do
6:       send rib$_t(v)$ to $v$
7:     end for
8:   end if
9: end while
```

Figure 2: Algorithm Spvp.

message exchanges are reliable and edges introduce an arbitrary finite delay. Communication between vertices takes place in a totally asynchronous way.

To introduce Spvp we need a few more definitions. We denote the set of neighbors of v by peers(v). Two data structures are used at each vertex v to represent the information v is aware of at time t: the path rib$_t(v)$ that is used to reach 0 and a table rib-in$_t(v \Leftarrow u)$ that stores the latest path received from neighbor $u \in$ peers(v). Thus, vertex v can select a path to 0 among the choices available in

$$\text{choices}_t(v) = \{(v\ u)P \in \mathcal{P}^v \mid P = \text{rib-in}_t(v \Leftarrow u)\}$$

Let W be a subset of the permitted paths $\mathcal{P}^v$ at vertex v, such that each path in W has a distinct next hop. Then the *best* path at v in W is

$$\text{best}(W, v) = \begin{cases} P \in W | P = \arg\min\ \lambda^v(P) & (W \neq \oslash) \\ \epsilon & (W = \oslash) \end{cases}$$

and the overall best path v is aware of at time t is best$_t(v)$ = best(choices$_t(v), v$).

Algorithm Spvp is in Fig. 2. Each vertex $v \in V$ executes an instance of Spvp. When a vertex v receives a path P from one of its neighbors u, it stores P in the local data structure rib-in$_t(v \Leftarrow u)$ and recomputes its best path. If the computed best path P differs from the previous one, u sends a message containing P to all of its neighbors.

A *path assignment* is a function π that maps each vertex $v \in V$ to a path $\pi(v) \in \mathcal{P}^v$. We have that $\pi(0) = (0)$ and, if $\pi(v) = \epsilon$, then v cannot reach vertex 0. In the following, we will refer to π as the *state* of the Spvp instance. Observe that, at any time t, the algorithm in Fig. 2 defines a path assignment π_t where $\pi_t(v)$ = rib$_t(v)$ and each vertex always selects the best available path.

We now discuss different possible variants of Spvp that can affect its dynamic evolution.

2.1 Edge and Vertex Activations and Activation Sequences

An *activation* is informally defined as the action of triggering a new computation of the best path on some vertices of an Spvp instance. Depending on the mechanism used to trigger the computation, we distinguish between edge and vertex activations.

We say that an *edge* (u, v) is *activated* at time t from u to v if v executes the algorithm in Fig. 2 to process the latest message received from u. Edge activations are considered in [13, 6, 9, 25, 3].

The semantic of a vertex activation may be twofold. We say that a *vertex* v is *activated to process* at time t if v executes the algorithm in Fig. 2, assuming that a message is simultaneously received from every vertex u in peers(v). Vertices are activated to process in [1, 8, 7, 5].

On the other hand, to complete the taxonomy of the models, it is quite natural to consider vertex activation from the opposite perspective. We say that a *vertex* v is *activated to send* at time t if v runs a slightly modified variant of the algorithm in Fig. 2. In this variant v first sends its current best path $\text{rib}_t(v)$ to all its neighbors (Steps 5-7 of SPVP), which are supposed to receive $\text{rib}_t(v)$ simultaneously. Then, for every vertex $u \in \text{peers}(v)$, a recomputation of the best path is triggered (Steps 2, 3 of SPVP).

An *activation sequence* is used to represent the order in which messages are exchanged in an SPVP instance. This order needs not to be total, i.e., at a given instant more than one edge can be traversed by messages. Similarly to the case of activations, we distinguish between edge activation sequences and vertex activation sequences. An *edge activation sequence* σ_e is a (possibly infinite) sequence of sets $\sigma_e = (A_0\ A_1\ \dots\ A_i \dots)$ in which each set A_t contains an ordered pair $(u, v) \in E$ for each edge (u, v) that is activated at time t. A *vertex activation sequence* is a sequence $\sigma_v = (A_0\ A_1\ \dots\ A_i \dots)$ in which each set A_t contains a vertex $v \in V$ for each vertex v that is activated at time t.

Observe that vertex activation sequences are special classes of edge activation sequences in which constraints are applied on the sequence of activated edges. An activation sequence where vertices are activated to send can be mapped to an edge activation sequence in which each vertex activation $A_i = v$ corresponds to a sequence of activations $A_{i_k} = (v, u_k)$ for each $u_k \in \text{peers}(v)$. A similar argument applies to an activation sequence where vertices are activated to process. In the latter case, pairs $A_{i_k} = (u_k, v)$ are activated for each vertex activation $A_i = v$.

2.2 Modeling Memory at Vertices

Another possible variant of the basic SPVP model is the one in which there is no rib-in_t [25, 3, 5]. In this case, each vertex v only stores its current best path and computes its new best paths directly referring to the best choices of its neighbors. Set $\text{choices}_t(v)$ would then be redefined in the following way: $\text{choices}_t(v) = \{(v, u)P \in \mathcal{P}^v | P = \text{best}_{t-1}(u)\}$.

Consider that, if vertices are activated to process, there is no need to consider a rib-in_t. In fact, every time a vertex $v \in V$ is activated, it immediately refreshes $\text{choices}_t(v)$, thus replacing any previously known path.

On the other hand, the absence of rib-in_t forces a vertex to query all its neighbors for each computation of a new best path. This corresponds to activating vertices to process. As an alternative, the absence of a rib-in_t can be compensated by forcing vertices to continuously send update messages, for example exploiting a timeout [3].

2.3 Simultaneousness

As a further degree of freedom, we distinguish between models that admit the simultaneous activation of multiple entities (i.e., $|A_i| \geq 1$) [1, 8, 7, 25, 3] and models that only allow a single entity to be activated at a time [5] (i.e., $|A_i| = 1$). Some authors, e.g. [13, 6, 9], enforce simultaneousness by equipping the model with message queues.

2.4 Choosing a model

Throughout the paper we will consider the original version of SPVP in which edges are activated, a local rib-in_t is maintained by each vertex, and simultaneous activations are allowed. Tab. 1 shows a classification of previously adopted models along the dimensions we analyzed.

3 Stable States and Infinite Oscillations

In this section we formally define the concept of routing oscillation and we show that the variant of SPVP that we choose to consider cannot be simplified along any dimension without impacting the ability of the model to capture oscillations.

	Activations	RIB	Simult.
[13, 6, 9]	Edges	Yes	Yes
[5]	Vertices, to process	No	No
[25, 3]	Edges	No	Yes
[1, 8, 7]	Vertices, to process	Yes	Yes

Table 1: A taxonomy of existing models for path vector protocols.

t	A_t	1	2
1	$\{(0,1),(0,2)\}$	*(1 0)*	*(2 0)*
2	$\{(1,2),(2,1)\}$	*(1 2 0)* (1 0)	*(2 1 0)* (2 0)
3	$\{(1,2),(2,1)\}$	*(1 0)*	*(2 0)*

Table 2: An oscillating fair edge activation sequence for Disagree (Fig. 1b). The columns of the table are the time instants, the set of activated edges, and the rib-in$_t$ of each vertex, with the currently selected best path highlighted in italic face.

For the sake of clarity, in the following we will specify activation sequences using a tabular notation as in Tab. 2, where each row corresponds to an activation, the first column specifies activated vertices or edges, and the remaining columns represent the current rib-in$_t$ at each vertex, with the currently selected best path highlighted using italic face. The initial state is assumed to be $\pi_0(v) = \epsilon\ \forall v \in V - \{0\}$.

We say that an activation sequence is *fair* [9] if, whenever vertex u sends a message at time t (Step 6 of Spvp), there exists a time $t' > t$ at which the message is delivered and processed by its recipient. In a model with edge activations, this corresponds to saying that edge (u, v) is eventually activated when u sends a message to v. In a model where vertices are activated to process (to send), this corresponds to saying that vertex v (u) is eventually activated when u sends a message to $v \in \text{peers}(u)$.

A specific execution of Spvp is called a *run*, and it induces a sequence of path assignments $(\pi_0\ \pi_1\ \dots\ \pi_t\ \dots)$ which we call its *trace*. While the Spvp algorithm is inherently non-deterministic, a single run can be studied in a deterministic way by looking at the activation sequence associated with the run. A run is *fair* if the associated activation sequence is fair.

A state $\pi_{t'}$ of an Spvp instance is a *stable state* if $\forall v \in V$: $\pi_t(v) = \pi_{t'}(v)$ for any $t > t'$. For example, two stable states for Fig. 1b are described in Tab. 3.

On the contrary, there are renowned Spvp instances that do not have any stable state (see Fig. 1a [11, 9]). Therefore it is interesting to study the following problem, known as the *Stable Path Problem* [9]:

Problem 3.1. *Given an instance $S = (G, \mathcal{P}, \Lambda)$, does it admit a stable state?*

A stable state in S, if any, is a *solution* to Problem 3.1.

Theorem 3.1. *Problem 3.1 is NP-complete [11, 9].*

Observe that, as also remarked by [11, 9], searching for the existence of a stable state hides another issue, i.e., even if a stable state exists, routing protocols can get trapped: consider, for instance, Fig. 1b. As Tab. 2 highlights, a routing protocol can oscillate forever on that instance. We believe that network operators are much more concerned with the existence of a "potential" divergence condition that, despite the existence of a stable state, could prevent a network from operating correctly. In order to address this problem, we formally define the concept of a routing oscillation.

We say that an instance of Spvp exhibits an *oscillation* if there exists an activation sequence inducing a trace in which $\pi_{t'}(v) = \pi_{t''}(v)\ \forall v \in V$, where $t'' > t'$. In fact, since the two states at times t' and t'' are equal, the portion of activation sequence $A_{t'} \dots A_{t''}$ could be repeated indefinitely, generating an

vertex	0	1	2
stable state 1	(0)	(1 0)	(2 1 0)
stable state 2	(0)	(1 2 0)	(2 0)

Table 3: Two distinct stable states for Disagree (Fig. 1b).

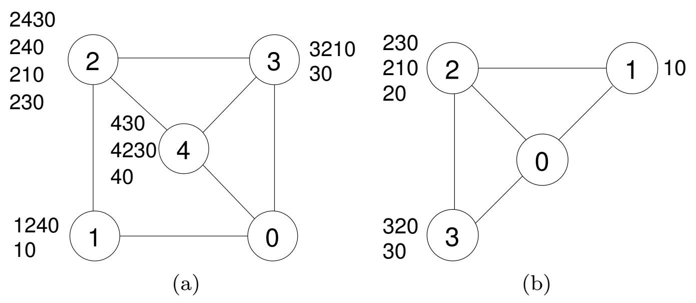


Figure 3: (a) Bleedin-Edge: An instance of Spvp for which a fair oscillation exists only in the edge activation model. (b) Di-safe-gree: A modified Disagree that has a unique guaranteed solution.

t	A_t	1	2	3	4
1	$\{(0,1),(0,3),(0,4)\}$	*(1 0)*	ϵ	*(3 0)*	*(4 0)*
2	$\{(3,2)\}$	*(1 0)*	*(2 3 0)*	*(3 0)*	(4 0)
3	$\{(2,4),(4,2)\}$	*(1 0)*	*(2 4 0)* (2 3 0)	*(3 0)*	*(4 2 3 0)* (4 0)
4	$\{(1,2),(2,1)\}$	*(1 2 4 0)* (1 0)	*(2 4 0)* (2 1 0) (2 3 0)	*(3 0)*	*(4 2 3 0)* (4 0)
5	$\{(4,2)\}$	*(1 2 4 0)* (1 0)	*(2 1 0)* (2 3 0)	*(3 0)*	*(4 2 3 0)* (4 0)
6	$\{(2,3)\}$	*(1 2 4 0)* (1 0)	*(2 1 0)* (2 3 0)	*(3 2 1 0)* (3 0)	*(4 2 3 0)* (4 0)
7	$\{(3,4),(4,3)\}$	*(1 2 4 0)* (1 0)	*(2 1 0)* (2 3 0)	*(3 2 1 0)* (3 0)	*(4 2 3 0)* (4 0)
8	$\{(1,2),(3,2),(4,2)\}$	*(1 2 4 0)* (1 0)	ϵ	*(3 2 1 0)* (3 0)	*(4 2 3 0)* (4 0)
9	$\{(2,1),(2,3),(2,4)\}$	*(1 0)*	ϵ	*(3 0)*	*(4 0)*

Table 4: An oscillating fair edge activation sequence for Bleedin-Edge (Fig. 3a).

infinite activation sequence σ'. If σ' is fair, we say that the instance admits a *fair oscillation*. We argue that, in real-world protocols, it is unlikely for some messages to be never delivered and processed. Hence, throughout the paper we will focus only on fair oscillations. An Spvp instance is *safe* if it does not admit any fair oscillations.

Let Spvp-ns be a variation of Spvp that does not allow simultaneous activations. The following theorem shows that relaxing Spvp by not considering simultaneous activations impacts the capability of the model to capture oscillations.

Property 3.1. Spvp *captures any oscillation captured by* Spvp-ns. *The converse does not hold.*

Proof. Trivially, non-simultaneous activation sequences can always be mapped to simultaneous edge activation sequences. On the other hand, Disagree (Fig. 1b) provides an example in which simultaneuosness is needed to trigger an oscillation. Let $e \in \{(1,2),(2,1)\}$ be the first edge that is activated, at time t, between vertices 1 and 2. Note that the activation of edge $e = (u, v)$ can only be triggered by a previous activation of edge $(0, u)$. This, in turn, implies that path $(u\ 0) \in \text{choices}_t(u)$. Hence, after activating e, $(v\ u\ 0)$ enters $\text{choices}_t(v)$, that forces $\text{best}_t(v) = (v\ u\ 0)$. Since this leads to any one of the two stable states described in Tab. 3, any further activation has no effect. By contrast, Tab. 2 shows that Disagree admits a fair oscillation if simultaneous activations are allowed. □

We already observed in Section 2 that edge activation sequences are more general than vertex activation sequences, regardless of the semantic of the activation of a vertex. Theorem 3.2 shows that relaxing Spvp by not considering the activation of single edges impacts the capability of the model to capture oscillations. To prove the theorem we need the following preliminary lemmas that exploit the instance Bleedin-Edge in Fig. 3a.

Let Spvp-vp (Spvp-vs) be a variation of Spvp in which vertices are activated to process (to send).

Lemma 3.1. *Consider the* Spvp *instance* Bleedin-Edge. *Independently on the activation sequence, if path* $(1\ 0)$ *enters* $\text{choices}_{t'}(1)$ *at time* t', *then* $(1\ 0) \in \text{choices}_t(1)\ \forall t \geq t'$. *The same also holds for paths* $(4\ 0)$ *and* $(3\ 0)$.

vertex	0	1	2	3	4
stable state	(0)	(1 0)	(2 4 3 0)	(3 0)	(4 3 0)

Table 5: A stable state for BLEEDIN-EDGE (Fig. 3a).

Proof. The statement follows from $\mathcal{P}^0 = \{(0)\}$. □

Lemma 3.2. *Consider the* SPVP *instance* BLEEDIN-EDGE. *If vertices are activated to send, no vertex activation sequence and no time t exist such that $\pi_t(4) = (4\ 2\ 3\ 0)$.*

Proof. For vertex 4 to select (4 2 3 0) it is required that 2 selects (2 3 0) first, which in turn requires vertex 3 to be activated at least once. Now, once 3 is activated, vertex 4 can immediately select its best path (4 3 0), and will be unable to select (4 2 3 0) in further steps. Even if we assume that there exists a time instant t such that the rib-in$_t$ at vertex 4 already contains (4 2 3 0), we need 3 to withdraw (3 0), which is preferred at 4. Since, by Lemma 3.1, vertex 3 cannot withdraw (3 0) by announcing ϵ, then 3 must announce (3 2 1 0), which can only happen if 2 picks (2 1 0) as its best and is activated beforehand, thus actually removing (4 2 3 0) from the rib-in$_t$ of vertex 4. □

Lemma 3.3. *Consider the* SPVP *instance* BLEEDIN-EDGE. *If vertices are activated to process, no vertex activation sequence allows vertex 4 to select* (4 2 3 0) *at any time t.*

Proof. We prove the assertion by contradiction. Let t be the first time at which vertex 4 selects (4 2 3 0). This implies $\pi_{t-1}(2) = (2\,3\,0)$ and $\pi_{t-1}(3) = (3\,2\,1\,0)$. In fact, by Lemma 3.1, vertex 3 can never announce ϵ after its first activation, and $\pi_{t-1}(2) = (2\,3\,0)$ implies that 3 was already activated before time $t-1$. Let $t' < t$ and $t'' < t$ be the instants of the last activation, before t, of vertices 2 and 3, respectively. If $t' < t''$, vertex 3 would be unable to select path (3 2 1 0), contradicting $\pi_{t-1}(3) = (3\,2\,1\,0)$. On the other hand, if $t'' < t'$, vertex 2 would be unable to select path (2 3 0). Then it must be $t' = t''$, i.e., vertices 2 and 3 were activated simultaneously. This, in turn, implies $\pi_{t'-1}(2) = (2\ 1\ 0)$, $\pi_{t'-1}(1) = (1\ 2\ 4\ 0)$ and $\pi_{t'-1}(3) = (3\ 0)$. Note that, by Lemma 3.1, it cannot be $\pi_{t'-1}(1) = \epsilon$, as $\pi_{t'-1}(2) = (2\ 1\ 0)$. Moreover, since activating vertex 2 at t' will result in $\pi_{t'}(2) = (2\ 3\ 0)$, we must have $\pi_{t'-1}(4) \neq (4\ 3\ 0)$ and $\pi_{t'-1}(4) \neq (4\ 0)$. This means that $\pi_{t'-1}(4)$ can be either ϵ or (4 2 3 0). The former case contradicts Lemma 3.1, since $\pi_{t'-1}(1) = (1\ 2\ 4\ 0)$ implies that 4 was activated before $t'-1$. The latter one contradicts the hypothesis of t being the first time at which 4 selects (4 2 3 0). □

Theorem 3.2. SPVP *captures any oscillation captured by* SPVP-VS *and by* SPVP-VP. *The converse does not hold.*

Proof. We already noted that edge activation sequences are at least as powerful as vertex activation sequences. BLEEDIN-EDGE proves the strictness. In fact, Tab. 4 shows an edge activation sequence that triggers a fair oscillation on that instance. Observe that the states at instants $t = 1$ and $t = 9$ coincide and each edge is activated at least once in that time interval (fairness). We now prove that vertex activation sequences always converge on BLEEDIN-EDGE. Lemma 3.2 and Lemma 3.3 ensure that $\forall t\ \pi_t(4) \neq (4\ 2\ 3\ 0)$, regardless of the semantic of vertex activation. Moreover, Lemma 3.1 implies that there exists a time t_0 such that $\forall t > t_0\ \pi_t(4) \neq \epsilon$. Hence, $\forall t > t_0$, $\pi_t(4)$ must be either (4 0) or (4 3 0). Observe that both these paths are extended by vertex 2. Hence, there exists a time instant t_1 such that, for any $t > t_1$: $\pi_t(2) = (2\,4\,3\,0)$ or $\pi_t(2) = (2\,4\,0)$. In particular, we have $\pi_t(2) \neq (2\,1\,0)$ which, in turn, implies the existence of a $t_2 > t_1$ such that $\pi_t(3) = (3\ 0)$ for any $t > t_2$. As a consequence, there will be a time $t_3 > t_2$ such that $\pi_t(4) = (4\ 3\ 0)$ for any $t > t_3$. This prevents 2 from selecting path (2 4 0) and therefore ultimately stabilizes 1 on $\pi_t(1) = (1\ 0)$ for any $t > t_4 > t_3$. □

We complete our discussion on the ability of the variations of SPVP to capture oscillations with the following theorem, that puts in evidence the importance of considering rib-in$_t$. Let SPVP-NR be a variation of SPVP that does not equip vertices with a rib-in$_t$.

Theorem 3.3. SPVP *captures any oscillation captured by* SPVP-NR. *The converse does not hold.*

Proof. As we already remarked in Section 2, the absence of rib-in$_t$ forces a vertex to query all its neighbors for each computation of a new best path. This corresponds to activating vertices to process, hence the statement can be proved in a way similar to Theorem 3.2. □

4 A New Greedy Algorithm

In this section we first briefly recall a greedy algorithm (we call it Greedy) that has been proposed in [9] to solve an Spvp instance. Second, we propose a new greedy algorithm, called Greedy$^+$. Finally, we compare Greedy and Greedy$^+$.

Algorithm Greedy attempts to grow a solution by iteratively building a stable path assignment. If the algorithm terminates successfully, the path assignment defines a spanning tree that is a solution for the given instance. Otherwise, the greedy algorithm is only able to identify a stable path assignment for a subset of the vertices.

The algorithm maintains a *stable set* of vertices for which convergence is guaranteed. The stable set at iteration i of the algorithm is denoted by V_i. Vertex 0 is always in the stable set, therefore we set $V_0 = \{0\}$. As the stable set grows, a path assignment π defined on the vertices in V_i is iteratively built.

We say that a path P is *compatible with a path assignment* π if $P = P'(u\ v)\pi(v)$, where P' does not contain vertices in V_i, $(u, v) \in E$, and $v \in V_i$.

Algorithm Greedy is as follows. At iteration i, let P_v be the path with minimum $\lambda^v(P)$ among the paths at v compatible with π. If such a path does not exist, let $P_v = \epsilon$. If there exists a vertex $v \notin V_{i-1}$ such that P_v has a next hop in V_{i-1}, then construct V_i by adding v to V_{i-1} and set $\pi(v) = P_v$. If such a vertex v does not exist, then stop.

Intuitively, at each iteration, vertex v is stabilized because its best compatible path directly reaches an already stabilized vertex. Observe that the algorithm terminates after at most $|V|$ iterations. A solution to the Spvp instance exists if, after k iterations, Greedy ends with $V_k = V$. The *solution* is given by the stable path assignment π.

Note that the description of Greedy we propose here slightly differs from the one in [9], in that we require that only a single vertex enters the stable set at each iteration. We will explain in the following that this modified version is indeed equivalent to the original algorithm. We choose to describe Greedy with this slight modification in order to better introduce the improvements that allow us to overcome some shortcomings of the original algorithm.

Greedy can fail to find a solution even if Spvp is guaranteed to converge. Consider, for example, the instance Di-safe-gree in Fig. 3b. It can be easily verified that any fair activation sequence of Spvp on this instance is finite. In fact, any fair activation sequence is such that vertices 1, 2, and 3 learn about the direct path to 0. After that, pair $(1, 2)$ is eventually activated, and 2 learns about (2 1 0). Henceforth, vertex 2 will permanently be unable to select (2 0), in turn preventing vertex 3 from choosing (3 2 0). Finally, after pair $(3, 2)$ is activated, 2 switches to its best path (2 3 0) and Spvp terminates, as no other message is further generated. Therefore any fair activation sequence is forcedly finite, and this implies that Spvp cannot oscillate on this instance.

We will now walk through the execution of Greedy on Di-safe-gree. At the first iteration, vertex 1 enters the stable set V_1, and $\pi(1) = (1\ 0)$. At the second iteration, the algorithm forcedly stops. In fact, path (2 3 0) is compatible with π because $2, 3 \notin V_1$, $0 \in V_1$, and $(3\ 0) \in E$. However, even if (2 3 0) is the best compatible path at vertex 2, its next hop is not in V_1. A similar argument applies to path (3 2 0). Therefore, no new vertex can be added to the stable set and the algorithm stops without finding a solution, since $V_1 \neq V$.

We now describe a variant of this algorithm, which we call Greedy$^+$. This variant is able to solve Di-safe-gree.

We say that a path P belonging to a set $\mathcal{S}$ of paths is *consistent with* $\mathcal{S}$ if either $P = \epsilon$, $P = (0)$, or $P = (v\ u)P'$ where $(v, u) \in E$ and P' is consistent with $\mathcal{S}$. For example, let $\mathcal{S} = \{(0), (1\ 0), (2\ 1\ 3\ 0)\}$: it is easy to check that (0) and (1 0) are consistent with $\mathcal{S}$, while (2 1 3 0) is not. Further, for each vertex v we define a set $\bar{\mathcal{P}}^v$ of paths called *useful set*. The useful set $\bar{\mathcal{P}}^v$ is initialized with the paths in $\mathcal{P}^v$ that are consistent with $\mathcal{P}$. Let $\bar{\mathcal{P}} = \bigcup_{v \in V} \bar{\mathcal{P}}^v$.

Greedy$^+$ differs from Greedy in that it exploits the useful set in order to prune paths that, starting from a certain iteration, become permanently unavailable. Hence, Greedy$^+$ needs to keep the useful set up to date at each iteration.

What follows is a description of Greedy$^+$. Let $V_0 = \{0\}$. At iteration i, Greedy$^+$ performs the following steps:

i) Exploit the current stable set in order to prune all those paths that cannot be selected because of the presence of a better ranked path offered by a neighbor in the stable set. For each vertex $v \in V - V_{i-1}$ such that v has a neighbor $u \in V_{i-1}$ and there exists a path $P = (v\ u)P'$ such that

i	V_i	C_i	$\bar{\mathcal{P}}^1$	$\bar{\mathcal{P}}^2$	$\bar{\mathcal{P}}^3$
0	$\{0\}$	$\{1\}$	(1 0)	(2 3 0) (2 1 0) (2 0)	(3 2 0) (3 0)
1	$\{0,1\}$	$\{3\}$	(1 0)	(2 3 0) (2 1 0)	(3 0)
2	$\{0,1,3\}$	$\{2\}$	(1 0)	(2 3 0)	(3 0)
3	V	$\oslash$	(1 0)	(2 3 0)	(3 0)

Table 6: A successful execution of Greedy$^+$ on Di-safe-gree (Fig. 3b). The table shows sets V_i, C_i, $\bar{\mathcal{P}}^v$ at iteration i of Greedy$^+$.

$\{P'\} = \bar{\mathcal{P}}^u$, remove from $\bar{\mathcal{P}}^v$ all the paths Q such that $\lambda^v(Q) > \lambda^v(P)$. Intuitively, this step is performed because P will be always available at v.

ii) Enforce consistency on all the paths. For each vertex $v \notin V_{i-1}$, remove from $\bar{\mathcal{P}}^v$ all the paths that are not consistent with $\bar{\mathcal{P}}$.

iii) Grow the stable set, or stop. Let $C_i \subset V - V_{i-1}$ be the set of *candidate* vertices v such that the path $P \in \bar{\mathcal{P}}^v$ with minimum $\lambda^v(P)$ either has a next hop in V_{i-1}, or $P = \epsilon$. If $C_i = \oslash$, then set $V_i = V_{i-1}$ and stop. Otherwise, if $C_i \neq \oslash$, then pick a vertex $u \in C_i$, construct V_i by adding u to V_{i-1}, and set $\bar{\mathcal{P}}^u = \{P\}$.

If Greedy$^+$ stops after k iterations, its *output* consists of a stable set V_k and sets $\bar{\mathcal{P}}^v$ $\forall v \in V$, with $|\bar{\mathcal{P}}^v| = 1$ $\forall v \in V_k$. If $V_k = V$, Greedy$^+$ computes a stable path assignment π for the input instance such that $\bar{\mathcal{P}}^v = \{\pi(v)\}$ $\forall v \in V$.

An example of a successful execution of Greedy$^+$ on Di-safe-gree is shown in Tab. 6. Note that at iteration 1 path (2 0) is evicted from $\bar{\mathcal{P}}^2$ because (2 1 0) is preferred and permanently available (Step *i*)). This action puts in evidence the difference between Greedy$^+$ and Greedy: in fact, it is easy to check that Greedy would have stopped at iteration 1. Step *ii*) then removes (3 2 0) from $\bar{\mathcal{P}}^3$ since it is inconsistent with $\bar{\mathcal{P}}$. This allows vertex 3 to enter the stable set.

Theorem 4.1. *Let n be the size of an* Spvp *instance S.* Greedy$^+$ *can be implemented to terminate on S in time that is polynomial in n.*

Proof. A trivial bound follows.

Step *i*) of Greedy$^+$ applies to those vertices v which extend a path P offered by some neighbor u in the stable set. This step can be implemented by evaluating λ^v for all the paths in each $\bar{\mathcal{P}}^v$ and comparing its value with $\lambda^v((v\ u)P)$. This takes $O(n^3)$ time, since the length of a path is $O(n)$.

Step *ii*) of Greedy$^+$ enforces consistency. This can be accomplished by comparing each path in $\bar{\mathcal{P}}$ with all the others, which takes $O(n^3)$.

Finally, at Step *iii*) of Greedy$^+$ candidate vertices can be found in $O(n^3)$ time.

Since Greedy$^+$ executes at most $|V|$ iterations and an instance of Spvp can have $O(n)$ vertices, Greedy$^+$ can be implemented to run in $O(n^4)$ time. □

The following properties and Lemma 4.1 show that Greedy$^+$ is deterministic in the sense that, at any time where multiple choices are possible, performing any of them does not alter the output.

Property 4.1. *If* Greedy$^+$ *terminates after k iterations, its output is completely defined by sets V_k and $\bar{\mathcal{P}}^v$ $\forall v \in V_k$.*

Proof. The missing portion of the output, $\bar{\mathcal{P}}^v$ $\forall v \in V - V_k$, can be uniquely constructed starting from V_k and $\bar{\mathcal{P}}^v$ $\forall v \in V_k$. Consider a new instance $S' = (G', \mathcal{P}', \Lambda')$ of Spvp with $G' = G$, $\Lambda' = \Lambda$, and, for any $v \in V$:

$$\mathcal{P}'^v = \begin{cases} \bar{\mathcal{P}}^v & \text{if } v \in V_k \\ \mathcal{P}^v & \text{if } v \notin V_k \end{cases}.$$

Now, initialize the stable set V_0 to V_k and execute Steps *i*) and *ii*) of Greedy$^+$ on S'. We now show that, after doing so, $\bar{\mathcal{P}}'^v = \bar{\mathcal{P}}^v$, $\forall v \in V$. This is trivially true for vertices $u \in V_k$, as no path is ever removed from $\bar{\mathcal{P}}'^u$. Observe that the outcome of Step *i*) of GREEDY$^+$ only depends on the topology of the graph G', the ranking functions Λ', and the sets of useful paths $\bar{\mathcal{P}}'^v$, with $v \in V_k$. Because of the way S' has been defined, we know that, at Step *i*), a path is removed from $\bar{\mathcal{P}}^v$ iff it is removed from $\bar{\mathcal{P}}'^v$. Hence, any possible difference must be due to Step *ii*).

We prove by contradiction that the output coincides also for vertices in $V - V_k$. Suppose that this is not the case, i.e., there exists some vertex $v \in V - V_k$ such that $\bar{\mathcal{P}}'^v \neq \bar{\mathcal{P}}^v$. Then, there exists a path P such that either $P \notin \bar{\mathcal{P}}'^v \wedge P \in \bar{\mathcal{P}}^v$ or $P \in \bar{\mathcal{P}}'^v \wedge P \notin \bar{\mathcal{P}}^v$. In the first case, the execution of Step *ii*) on S' has removed from $\bar{\mathcal{P}}'^v$ a path that the execution of GREEDY$^+$ on S regarded as consistent. But this is impossible, since $\forall v \in V$, $\bar{\mathcal{P}}^v \subseteq \mathcal{P}'^v$, so there can be no path that is consistent with $\bar{\mathcal{P}}$ and is not consistent with $\mathcal{P}'$. In the second case, the execution on S has removed from $\bar{\mathcal{P}}^v$ a path P that the execution on S' considered as consistent. Since it cannot be $P \notin \mathcal{P}^v$, then for P to be inconsistent with $\bar{\mathcal{P}}$, it may only be the case that $P = (v\ \ldots\ u)P_u$, where $P_u \notin \bar{\mathcal{P}}^u$ and $P_u \in \bar{\mathcal{P}}'^u$. In turn, this is only possible if there exists a path P_w such that $P_u = (u\ \ldots\ w)P_w$, with $P_w \notin \bar{\mathcal{P}}^w$ and $P_w \in \bar{\mathcal{P}}'^w$. By proceeding this way, we must eventually end up on a vertex x in V_k, possibly 0. By recalling that $\bar{\mathcal{P}}'^v = \bar{\mathcal{P}}^v\ \forall v \in V_k$ by construction, we have a contradiction in that it should be $P_x \notin \bar{\mathcal{P}}^x$ and $P_x \in \bar{\mathcal{P}}'^x$. □

Property 4.2. *Consider a path P that is inconsistent with $\bar{\mathcal{P}}$ at iteration i of* GREEDY$^+$*. Then, P is inconsistent at any iteration $j > i$.*

Proof. The property follows by observing that GREEDY$^+$ never adds new paths to $\bar{\mathcal{P}}$. □

Property 4.3. *At any iteration i of* GREEDY$^+$*, $C_i \cap V_i = V_i - V_{i-1}$.*

Proof. By construction, $C_i \cap V_{i-1} = \oslash$. Now, at iteration i a vertex is picked from C_i and added to V_{i-1} to construct V_i. Therefore, the property follows. □

The following property states the fact that, once a vertex enters the candidate set, it stays there until it is eventually moved to the stable set.

Property 4.4. *Consider an arbitrary iteration i of* GREEDY$^+$ *and a vertex $v \in C_i$. Then there exists an iteration $j > i$ such that $v \in C_h$ for all $i \leq h \leq j$ and $v \in V_k$ for all $k \geq j$.*

Proof. Let $v \in C_i$ be a vertex such that the path $P \in \bar{\mathcal{P}}^v$ with minimum $\lambda^v(P)$ at iteration i either has a next hop in V_{i-1}, or $P = \epsilon$. Since no better path can enter $\bar{\mathcal{P}}^v$ during the execution of GREEDY$^+$ (Property 4.2) and P has the minimum value of λ^v among the paths in $\bar{\mathcal{P}}^v$ that are consistent with $\bar{\mathcal{P}}$, P can never be removed from $\bar{\mathcal{P}}^v$ at Step *i*) of GREEDY$^+$. Moreover, if $P = \epsilon$, by definition P is a consistent path. Otherwise, if $P = (v\ u)Q$, $u \in V_{i-1}$, $\{Q\} = \bar{\mathcal{P}}^u$, then P will remain consistent with $\bar{\mathcal{P}}$ because its next hop is $u \in V_{i-1}$, so $\bar{\mathcal{P}}^u$ will not be updated after iteration i. Thus, P cannot be removed from $\bar{\mathcal{P}}^v$ at Step *ii*). Overall, starting from iteration i, path P will always be available in $\bar{\mathcal{P}}^v$ and will always have the minimum value of λ^v. In other words, v satisfies the conditions of Step *iii*) at any iteration $k \geq i$, i.e., $v \in C_k \cup V_k$.

Since $\forall k \geq i\ v \in C_k \cup V_k$, and GREEDY$^+$ only terminates when the candidate set is empty, by Property 4.3 there must be an iteration j at which v is picked from C_j and added to V_{j-1} to construct V_j. The statement follows by recalling that vertices are never removed from the stable set. □

We now show that, if multiple candidates exist at Step *iii*), the output of GREEDY$^+$ is not affected by the vertex that actually enters the stable set.

Lemma 4.1. *Consider an arbitrary iteration j of* GREEDY$^+$ *and a set C_j of vertices satisfying the criteria of Step* iii*) at iteration j. The output of* GREEDY$^+$ *does not change, regardless of the choice of vertex $v \in C_j$ performed at iteration j.*

Proof. Assume that GREEDY$^+$ terminates at iteration k. First of all consider that, by Property 4.1, it is sufficient to prove the assertion for sets V_k and $\bar{\mathcal{P}}^v$ with $v \in V_k$. Consider an arbitrary vertex $u \in C_j$. By Property 4.4, we know that $u \in C_h$ for any iteration $h \geq j$, until u eventually enters the stable set. Also, as shown in the proof of Property 4.4, the best path $(u\ v)P$, $v \in V_h$ is always in $\bar{\mathcal{P}}^u$. Therefore, regardless of the iteration at which u is actually selected, the set $\bar{\mathcal{P}}^u$ is always updated with path $(u\ v)P$. Moreover, the set of paths that become inconsistent with $\bar{\mathcal{P}}$ after setting $\bar{\mathcal{P}}^u = \{(u\ v)P\}$ does not depend on the iteration either.

Thus, a vertex $u \in C_h$ can be picked by Step *iii*) at any iteration h of GREEDY$^+$ without affecting neither V_k nor $\bar{\mathcal{P}}^v\ \forall v \in V_k$. Since this is true for any vertex $u \in C_h$, GREEDY$^+$ can select an arbitrary candidate vertex at each iteration h without affecting the output. □

Note that algorithm Greedy$^+$ essentially differs from Greedy because of the presence of Step *i*). In fact, if we skip Step *i*), at each iteration i both the algorithms select the best path among the consistent ones having a next hop in V_i. This can be easily verified by observing that, when Step *i*) is removed, the set $\bar{\mathcal{P}}$ is only used to filter out inconsistent paths.

Therefore, it is easy to check that the validity of Lemma 4.1 can be extended to Greedy by considering the path assignment π as its output and skipping any consideration about Step *i*) in the proof of the lemma. This further confirms that the description of Greedy given in this section and the original description given in [9] are indeed equivalent.

We now show that Greedy$^+$ is more powerful than Greedy in that it is able to compute a guaranteed stable state for a strictly larger set of Spvp instances.

Lemma 4.2. *Let S be an instance of* Spvp*. If* Greedy *terminates on S finding a path assignment π^*, then* Greedy$^+$ *also terminates on S finding π^*.*

Proof. By Lemma 4.1 we know that, when multiple vertices can enter the stable set at a given iteration, the solution computed by Greedy$^+$ is independent on the order in which these vertices are considered. Therefore, we prove the assertion by showing that Greedy$^+$ can find π^* by selecting vertices to put in the stable set in the very same order as Greedy does. We show it by mapping each iteration of Greedy to one iteration of Greedy$^+$. In the following, we will refer to Greedy's stable set as V_j, and to Greedy$^+$'s stable set as V_j^+, and we will indicate with π the path assignment defined by Greedy at a given iteration. The proof proceeds by induction on the iteration j. It is trivially true that, at $j = 0$, $V_j = V_j^+ = \{0\}$. Assume that $V_{j-1} = V_{j-1}^+$ and, without loss of generality, that the stable sets have been constructed by adding vertices in the very same order by the two algorithms. Consider vertex u that Greedy selects at iteration j. This implies that $(u\ v)\pi(v)$ is the path with minimum λ^u among those compatible with π, for some $v \in V_{j-1}$. By the induction hypothesis, $\bar{\mathcal{P}}^v = \{\pi(v)\}$, therefore path $(u\ v)\pi(v)$ is consistent with $\bar{\mathcal{P}}$. We show that path $(u\ v)\pi(v)$ must still be in $\bar{\mathcal{P}}^u$ at iteration j. Property 4.2 ensures that Step *ii*) didn't remove path $(u\ v)\pi(v)$ from $\bar{\mathcal{P}}^u$. That is, since path $(u\ v)\pi(v)$ is consistent with $\bar{\mathcal{P}}$ at iteration j, it was always consistent during the previous iterations. By the induction hypothesis, $\forall w \in V_{j-1}\ \bar{\mathcal{P}}^w = \{\pi(w)\}$, therefore all the paths that are regarded as consistent by Greedy$^+$ are necessarily compatible with π. Hence, since $(u\ v)\pi(v)$ is a consistent with $\bar{\mathcal{P}}$ and has minimum λ^u among the paths compatible with π, it must also have minimum λ^u among the paths consistent with $\bar{\mathcal{P}}$. Therefore path $(u\ v)\pi(v)$ cannot be deleted at Step *i*) of Greedy$^+$. Thus, vertex u is a candidate to be inserted in the stable set by Greedy$^+$.

Since, by Lemma 4.1, the output of Greedy$^+$ is unaffected by the order in which vertices enter the stable set, we can assume without loss of generality that Greedy$^+$ too selects vertex u at iteration j. This in turn implies that Greedy$^+$ finds the same path assignment π^*. □

Theorem 4.2. *The set of instances that* Greedy$^+$ *can successfully solve is strictly larger than the set of instances that* Greedy *is able to solve.*

Proof. Lemma 4.2 proves the inclusion. The strictness is supported by Di-safe-gree, which is not solved by Greedy, as we discussed above, while it is solved by Greedy$^+$ as shown in Tab. 6. □

5 Correctness of Greedy$^+$ and a New Condition for Stability

In order to study an instance of Spvp it is useful to look for the presence of gadgets (i.e., portions of the graph) such that the ranking of permitted paths establishes a circular set of dependencies which cannot be simultaneously satisfied. In [8] it is shown that the absence of a particular structure, called dispute wheel, implies the absence of routing oscillations. Further studies stressed the fact that dispute wheels are a powerful tool to study oscillations in policy-based path vector protocols.

A *dispute wheel* (see Fig. 4) $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ is a sequence of vertices $\vec{\mathcal{U}} = u_0, u_1, \ldots, u_{k-1}$ and sequences of nonempty paths $\vec{\mathcal{Q}} = Q_0, Q_1, \ldots, Q_{k-1}$ and $\vec{\mathcal{R}} = R_0, R_1, \ldots, R_{k-1}$ such that: *(i)* R_i is a path from u_i to u_{i+1}, *(ii)* $Q_i \in \mathcal{P}^{u_i}$, *(iii)* $R_iQ_{i+1} \in \mathcal{P}^{u_i}$, and *(iv)* $\lambda^{u_i}(Q_i) > \lambda^{u_i}(R_iQ_{i+1})$.

Subscripts for dispute wheels are to be intended modulo k. We call paths in $\vec{\mathcal{Q}}$ *spoke* paths [15].

As far as we know, all the contributions in the literature that aim at guaranteeing that an instance is safe, either explicitly require the absence of dispute wheels, or assume constraints that prevent dispute wheels from being established. In this section we show the correctness of Greedy$^+$, and we exploit the

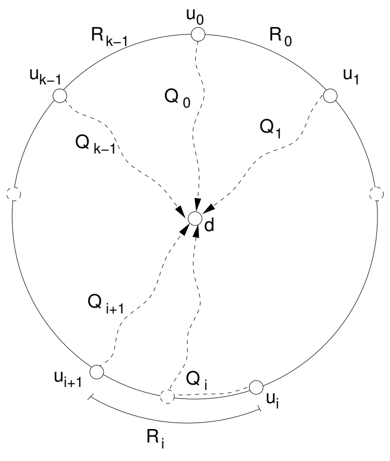


Figure 4: A dispute wheel.

strong relationships between GREEDY$^+$ and SPVP to derive a sufficient condition for safety that is strictly less constraining than the absence of dispute wheels.

Theorem 5.1. *Consider an instance S of* SPVP *and run* GREEDY$^+$ *on S. Let $P \in \mathcal{P}^u$ be any path that* GREEDY$^+$ *deletes at iteration j. Then, for any fair run r of* SPVP *on S, there exists a time t' such that $\forall t > t'$, $\pi_t(u) \neq P$.*

Proof. The statement asserts that GREEDY$^+$ deletes only those paths that will be discarded by any fair run of SPVP. The proof is by induction on the iteration j of GREEDY$^+$. At iteration $j = 1$, since $\bar{\mathcal{P}}^v = \mathcal{P}^v$ for all $v \in V$, GREEDY$^+$ deletes a path P from $\bar{\mathcal{P}}^u$ at either Step *i*) or Step *ii*) according to the following conditions.

Deletion at Step i*)*: Since $V_0 = \{0\}$, it takes place if $\lambda^u((u\ 0)) < \lambda^u(P)$. By the fairness of r, there must exist a time t' such that $\forall t > t'$: $(u\ 0) \in \text{choices}_t(u)$, thus preventing u from selecting P after t'.

Deletion at Step ii*)*: It takes place if P is inconsistent with $\mathcal{P}$, i.e., $P = Q(w)R$ and $R \notin \mathcal{P}^w$. In this case, the statement trivially follows since $\pi_t(w) \neq R\ \forall t$.

Assume, by induction, that the assertion holds for a given iteration $j-1$ of GREEDY$^+$. We now prove that the same property is true for the paths that are deleted during iteration j. Again, during iteration j, GREEDY$^+$ deletes a path P from $\bar{\mathcal{P}}^u$ at either Step *i*) or Step *ii*).

Deletion at Step i*)*: It takes place if there exists $v \in V_{j-1}$ such that $u \in \text{peers}(v)$ and $\lambda^u((u\ v)P') < \lambda^u(P)$, where $\{P'\} = \bar{\mathcal{P}}^v$. Observe that the induction hypothesis assures that previously deleted paths are eventually discarded by r after time t'. Then, by the fairness of r, there must exist a time $t'' > t'$ such that $\forall t > t''\ (u\ v)P' \in \text{choices}_t(u)$. This prevents u from selecting path P, i.e., $\pi_t(u) \neq P\ \forall t > t''$.

Deletion at Step ii*)*: It takes place if P is inconsistent, i.e., $P = Q(w)R$ and $R \notin \bar{\mathcal{P}}^w$. By the induction hypothesis, there exists t' such that $\forall t > t'\ \pi_t(w) \neq R$. Then, by the fairness of r, u must receive a message that withdraws the availability of R at a time $t'' > t'$. Therefore, $\pi_t(u) \neq P\ \forall t > t''$. □

Corollary 5.1. *If* GREEDY$^+$ *terminates successfully on an instance S of* SPVP*, then S is safe and has a unique solution.*

Observe that Corollary 5.1 actually states that GREEDY$^+$ can be used as a centralized, deterministic algorithm to efficiently emulate the behavior of SPVP in the long term, thus dealing with the non-determinism that SPVP features. In our opinion, this property could be effectively exploited, e.g., by a network administrator that wants to analyze how BGP will behave in his/her own network.

Given an instance S of SPVP, we say that a vertex v *predictably selects* path P if, for any fair run r of SPVP on S, there exists a time t' such that $\forall t > t'\ \pi_t(v) = P$. We say that v is *predictable.* Note that 0 is predictable by definition. The following lemma holds.

Lemma 5.1. *If vertex v predictably selects $P = (v\ u)Q$, with Q possibly trivial, then u is predictable.*

Proof. Suppose by contradiction that u is not predictable. Then, there exists at least one fair activation sequence $\sigma = (A_0\ A_1\ \ldots)$ such that for any t' there exists a $t'' > t'$ such that $\pi_{t''}(u) \neq Q$. Consider another activation sequence $\sigma' = (A'_0\ A'_1\ \dots)$ such that $A'_t = A_t\ \forall t \neq t'' + 1$ and $A'_{t''+1} = A_{t''+1} \cup (u, v)$. Then the trace π' induced by σ' cannot be such that $\pi'_{t''+1}(v) = P$, yielding a contradiction. □

Observe that, by simply iterating Lemma 5.1, we can state that all vertices in a predictably selected path are predictable.

Greedy$^+$ and predictability are also related by the following property, which derives from Theorem 5.1.

Property 5.1. *Every vertex that* Greedy$^+$ *puts in the stable set is predictable.*

In the following we show that routing oscillations can only be triggered by a "special" class of dispute wheels. The definition of this class is inspired by the deletion that is carried out at Step i) of Greedy$^+$.

We say that a dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ is a *steady spoke dispute wheel* (shortly, SSDW) if, for every $Q_i = (u_i\ v_i)P_i$, Π satisfies the following condition: $\forall w \in \text{peers}(u_i)$ such that w predictably selects a path P', we have $\lambda^{u_i}((u_i\ w)P')) > \lambda^{u_i}(Q_i)$. This condition means that the spoke path Q_i is better than any other path offered by predictable neighbors of u_i.

We now provide a sufficient condition for safety which is strictly less constraining than the no dispute wheel condition so many papers rely on (see Section 1). Note that this result is interesting given that, as we showed in Section 3, the model we adopt is capable of capturing at least as many oscillations as the other variants proposed in previous works (see Section 2).

Theorem 5.2. *Let S be an instance of* Spvp. *If S contains no SSDW, then S is safe.*

Proof. If S contains no SSDW then, either S does not contain any dispute wheels at all or S only contains dispute wheels which are not SSDW. In the first case, the proof trivially stems from [9]. In the other case, consider any dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$. For Π not to be a SSDW, it must be the case that, for some vertex $u_i \in \vec{\mathcal{U}}$ with $Q_i = (u_i\ v_i)P$, $\exists w \in \text{peers}(u_i)$ such that w predictably selects P' and $\lambda^{u_i}((u_i\ w)P')) < \lambda^{u_i}(Q_i)$. By definition of predictable vertex, for any fair activation sequence σ there exists a t' such that, for any $t > t'$: $\pi_t(w) = P'$. Hence, by the fairness of σ, there exists a $t'' > t'$ such that $(u_i\ w)P' \in \text{choices}_t(u_i)$ for any $t > t''$ which, in turn, prevents u_i from selecting Q_i after t''. Since path Q_i is no longer selected after t'', the dispute wheel Π is prevented from oscillating. □

The following lemma puts in better evidence the relevance of Theorem 5.2, by showing that our sufficient condition for safety is able to capture a strictly larger set of safe instances.

Lemma 5.2. *A SSDW is a dispute wheel. The converse does not hold.*

Proof. The first part of the statement is trivial. The second part is supported by Di-safe-gree: in fact, it is easy to see that there exists a dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ where $\vec{\mathcal{U}} = (2, 3)$, $\vec{\mathcal{Q}} = ((2\ 0), (3\ 0))$, and $\vec{\mathcal{R}} = ((2\ 3), (3\ 2))$. However, Π is not a SSDW since 0 and 1 are predictable neighbors of 2 and $\lambda^2(2\ 1\ 0) < \lambda^2(2\ 0)$. □

6 Conclusions and Open Problems

Several studies on BGP stability rely on a sufficient condition for guaranteed convergence that is based on the absence of dispute wheels. We relax this sufficient condition by showing that only a particular subclass of dispute wheels can actually trigger routing oscillations. The generality of our result is supported by a thorough analysis on the ability of different models for path vector protocols to capture routing oscillations. In our opinion, this result can contribute in making less constraining the previously proposed sufficient conditions that rely on the absence of dispute wheels.

We also provide a deterministic polynomial time algorithm that can verify if an input BGP instance is guaranteed to converge. Our algorithm can successfully check the guaranteed convergence of more instances than the greedy algorithm in [9] does. We believe that this result can be used as a basis to develop tools for automatically checking routing policies against oscillations. This is even more relevant since the model we adopt can also be applied to IBGP configurations [14, 16, 21].

One obvious problem that remains open is filling the gap between sufficient and necessary conditions for stability. While the necessary boundary is currently established in [5], this work narrows that gap by moving the sufficient boundary one step further. Note that, as a side effect, our result introduces another gap to be filled: the one between sufficient conditions for stability and those for robustness (see, e.g., [6, 9, 16]). In fact, Di-safe-gree provides an example of instance in which the no SSDW condition does not imply robustness. At present, the best known condition for robustness remains the no dispute wheel one. Another open problem is how to relate our results to game theoretical models of BGP, e.g., [19]. In particular, it would be interesting to apply our technique to enlarge the set of instances for which a Nash equilibrium can be efficiently computed.

References

[1] Anindya Basu, Chih-Hao Luke Ong, April Rasala, F. Bruce Shepherd, and Gordon Wilfong. Route oscillations in i-bgp with route reflection. In *Proc. SIGCOMM*, 2002.

[2] T. Bates, R. Chandra, and E. Chen. BGP Route Reflection - An Alternative to Full Mesh IBGP. RFC 2796, 2000.

[3] Jorge Arturo Cobb, Mohamed G. Gouda, and Ravi Musunuri. A stabilizing solution to the stable path problem. In *Proc. Self-Stabilizing Systems*, 2003.

[4] Cheng Tien Ee, Vijay Ramachandran, Byung-Gon Chun, Kaushik Lakshminarayanan, and Scott Shenker. Resolving inter-domain policy disputes. In *Proc. SIGCOMM*, 2007.

[5] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Implications of autonomy for the expressiveness of policy routing. *IEEE/ACM Trans. on Networking*, 2007.

[6] Lixin Gao, Timothy Griffin, and Jennifer Rexford. Inherently safe backup routing with BGP. In *Proc. INFOCOM*, 2001.

[7] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. In *Proc. SIGMETRICS*, 2000.

[8] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. Policy disputes in path-vector protocols. In *Proc. ICNP*, 1999.

[9] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. on Networking*, 2002.

[10] Timothy G. Griffin and Joao Luís Sobrinho. Metarouting. In *Proc. SIGCOMM*, 2005.

[11] Timothy G. Griffin and Gordon Wilfong. An analysis of bgp convergence properties. In *Proc. SIGCOMM*, 1999.

[12] Timothy G. Griffin and Gordon Wilfong. On the correctness of ibgp configuration. In *Proc. SIGCOMM*, 2002.

[13] Timothy G. Griffin and Gordon T. Wilfong. A safe path vector protocol. In *Proc. INFOCOM*, 2000.

[14] Timothy G. Griffin and Gordon T. Wilfong. Analysis of the med oscillation problem in bgp. In *Proc. ICNP*, 2002.

[15] Aaron D. Jaggard and Vijay Ramachandran. Robustness of class-based path-vector systems. In *Proc. ICNP*, 2004.

[16] Aaron D. Jaggard and Vijay Ramachandran. Robust path-vector routing despite inconsistent route preferences. In *Proc. ICNP*, 2006.

[17] Chi kin Chau. Policy-based routing with non-strict preferences. In *Proc. SIGCOMM*, 2006.

[18] Chi kin Chau, Richard Gibbens, and Timothy G. Griffin. Towards a unified theory of policy-based routing. In *Proc. INFOCOM*, 2006.

[19] Hagay Levin, Michael Schapira, and Aviv Zohar. Interdomain routing and games. In *Proc. STOC*, 2008.

[20] Ravi Musunuri and Jorge Arturo Cobb. A complete solution for ibgp stability. In *Proc. ICC*, 2004.

[21] Anuj Rawat and Mark A. Shayman. Preventing persistent oscillations and loops in ibgp configuration with route reflection. *Computer Networks*, 2006.

[22] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, 2006.

[23] Joao Luís Sobrinho. Network routing with path vector protocols: Theory and applications. In *Proc. SIGCOMM*, 2003.

[24] Joao Luís Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Trans. on Networking*, 2005.

[25] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 2000.

[26] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet performance. In *Proc. SIGCOMM*, 2006.