Cuts and disjoint paths in the valley-free path model

Author(s):
Erlebach, Thomas; Hall, A.; Panconesi, A.; Vukadinović, D.

Publication Date:
2003

Permanent Link:
https://doi.org/10.3929/ethz-a-004604570 →

T. Erlebach, A. Hall, A. Panconesi, D. Vukadinović

Cuts and Disjoint Paths in the Valley-Free Path Model

TIK-Report
Nr. 180, September 2003

T. Erlebach, A. Hall, A. Panconesi, D. Vukadinović
Cuts and Disjoint Paths in the Valley-Free Path Model
September 2003
Version 1
TIK-Report Nr. 180

Computer Engineering and Networks Laboratory,
Swiss Federal Institute of Technology (ETH) Zurich

Institut für Technische Informatik und Kommunikationsnetze,
Eidgenössische Technische Hochschule Zürich

Gloriastrasse 35, ETH Zentrum, CH-8092 Zürich, Switzerland

Cuts and Disjoint Paths in the Valley-Free Path Model*

T. Erlebach[†], A. Hall[†], A. Panconesi[‡] and D. Vukadinović[†]

September 2003

Abstract

In the valley-free path model, a path in a given directed graph is valid if it consists of a sequence of forward edges followed by a sequence of backward edges. This model is motivated by routing policies of autonomous systems in the Internet. We give a 2-approximation algorithm for the problem of computing a maximum number of edge- or vertex-disjoint valid paths between two given vertices s and t, and we show that no better approximation ratio is possible unless $P = NP$. Furthermore, we give a 2-approximation for the problem of computing a minimum vertex cut that separates s and t with respect to all valid paths and prove that the problem is APX-hard. The corresponding problem for edge cuts is shown to be polynomial-time solvable. We present additional results for acyclic graphs.

1 Introduction

Let $G = (V, E)$ be a directed, simple graph without anti-parallel edges (i.e., if $(u, v) \in E$, then $(v, u) \notin E$). For $s, t \in V$, a path from s to t is *valid* if it consists of a (possibly empty) sequence of forward edges followed by a (possibly empty) sequence of backward edges. Note that a valid path from s to t gives also a valid path from t to s and vice versa. We refer to this model of valid paths as the *valley-free path model.* The reason for this terminology is that if we view directed edges as "pointing upward" towards their heads, a path is valid if and only if it does not contain a "downward" edge followed by an "upward" edge, a valley.

The motivation for studying the valley-free path model comes from BGP routing policies in the Internet on the level of autonomous systems, as explained in more detail in Section 1.1. Robustness considerations of the Internet topology then lead naturally to interesting questions concerning the computation of large sets of disjoint valid paths between two given vertices, and of small vertex or edge cuts separating two given vertices with respect to all valid paths. The corresponding

*Research partially supported by EU Thematic Network APPOL II, IST-2001-32007, with funding for the Swiss partners provided by the Swiss Federal Office for Education and Science.

[†]Computer Engineering and Networks Laboratory (TIK), Department of Information Technology and Electrical Engineering, ETH Zurich, Switzerland. E-mail: {erlebach|hall|vukadin}@tik.ee.ethz.ch. A. Hall is supported by the joint Berlin/Zurich graduate program Combinatorics, Geometry, and Computation (CGC), financed by ETH Zurich and the German Science Foundation (DFG). D. Vukadinović is supported in DICS-Project No. 1838 by the Hasler Foundation.

[‡]DSI – Università La Sapienza, Rome, Italy. E-mail: ale@dsi.uniroma1.it

optimization problems for standard directed paths can be solved efficiently using network flow techniques (e.g., see [1]). In this paper, we initiate the investigation of these problems in the valley-free path model. It turns out that several of these problems are *NP*-hard in this model. Our main results are:

- We give 2-approximation algorithms for the problems of computing a maximum number of vertex- or edge-disjoint valid paths between two given vertices s and t, and we show that it is *NP*-hard to approximate these problems within ratio $2 - \varepsilon$ for any fixed $\varepsilon > 0$.
- We prove *APX*-hardness for the problem of computing a min valid s-t-vertex-cut, i.e., a minimum size set of vertices whose removal from G disconnects all valid paths between s and t. Furthermore, we give a 2-approximation algorithm for this problem.
- For the edge version of the problem, i.e., computing a min valid s-t-edge-cut, we give a polynomial algorithm that computes an optimal solution.
- We prove that the size of a min valid s-t-cut is at most twice the maximum number of disjoint valid s-t-paths, both for the edge version and the vertex version of the problems, and we show that this bound is tight.
- For the special case that the given graph G is acyclic (where "acyclic" is to be understood in the standard sense, i.e., the directed graph G is acyclic if it does not contain a directed cycle), we give a polynomial algorithm for finding k edge- or vertex-disjoint valid paths between s and t if they exist, where k is an arbitrary constant. The algorithm is based on a generalization of ideas due to Fortune, Hopcroft and Wyllie [8]. We also prove *NP*-hardness for the general problem of computing a maximum number of vertex- or edge-disjoint valid s-t-paths in acyclic graphs.

Our results give interesting insights for natural variations of the classical problems of computing disjoint s-t-paths and minimum s-t-cuts. Furthermore, the algorithms we provide may be useful for investigating issues related to the robustness of the Internet topology while taking into account the effects of routing policies.

1.1 Motivation: Autonomous Systems on the Internet

In this section, we provide some information about the issues in Internet routing on the autonomous system level that have motivated our study. An autonomous system (AS) on the Internet is a subnetwork under separate administrative control. ASs are connected by physical links and exchange routing information using the Border Gateway Protocol (BGP). An AS can consist of tens to thousands of routers and hosts. On the level of ASs, the Internet can be represented as an undirected graph by creating a vertex for each AS and adding an edge between two ASs if they have at least one physical link between them. However, such an undirected graph is not sufficient to model the effects of routing policies enforced by individual ASs.

Each AS announces the routes to a certain set of destination ASs (more precisely, address prefixes) to some of its neighbors. The decisions which routes will be announced to which neighbor are determined by BGP routing policies. Policies depend mostly on economic relationships between ASs and represent an important aspect of the Internet structure.

The nature of commercial agreements between ASs has attracted a lot of attention in the Internet economics research community [11, 12, 3]. The main trends in the diversity of these agreements were described in [11, 12]. The impact of economic relationships on the engineering level, more precisely on BGP routing, has not been immediately recognized despite the direct implication that an existing link between two ASs will not be used to transfer traffic that collides with their mutual agreement. Then several papers showing the impact of BGP policies on features such as path inflation and routing convergence have appeared [16, 17, 13].

As a consequence, the previously developed undirected model for the AS topology is not satisfactory because it allows some prohibited paths between ASs and thus might produce a distorted picture of BGP routing. On the other hand, involving all of the peculiarities of the contracts between autonomous systems in a new model would add too much complexity. Thus, a rough classification into a few categories was proposed for the BGP policies adopted by a pair of ASs: customer-provider, peer-to-peer, and siblings (see [9]). Later on, a simplified model with only two categories (customer-provider and peer-to-peer) was proposed [15]. A customer-provider relationship between A and B can be represented as a directed edge from A to B, and a peer-to-peer relationship as an undirected edge. If ASs A and B are in a customer-provider relationship, B announces all its routes to A, but A announces to B only its own routes and routes of its own customers. If they are peers, they exchange their own routes and routes of their customers, but not routes that they learn from their providers or other peers. This leads to the model proposed in [15] that a path is valid if and only if it consists of a sequence of customer-provider edges ($\bullet\!\rightarrow\!\bullet$), followed by at most one peer-to-peer edge ($\bullet\!-\!\bullet$), followed by a sequence of provider-customer edges ($\bullet\!\leftarrow\!\bullet$). Furthermore, it is easy to see that a peer-to-peer edge (undirected edge) between A and B can be replaced by two customer-provider edges from A to X and from B to X, where X is a new node, without affecting the solutions to any of the optimization problems (minimum cut problems and maximum disjoint paths problems) we study in this paper. Therefore, without losing generality, we can consider a model with only customer-provider edges. In other words, this model consists of a directed graph, with ASs as nodes and where the edge directions represent economic relationships. Here the valley-free paths are exactly the paths permitted by the BGP routing policies.

Information about the economic relationships between autonomous systems is not publicly available. Therefore, several approaches to inferring these relationships from available topology data or AS path information have been proposed in the literature [9, 15, 7, 5].

If a communication network is represented as an undirected or directed graph in a model without routing policies, it is natural to measure the connectivity provided to an s-t-pair as the maximum number of disjoint s-t-paths or the minimum size of an s-t-cut (by Menger's theorem, these two quantities are the same). This motivates us to study the corresponding notions for the valley-free path model in this paper.

It seems natural to expect that the directed graph of customer-provider edges will be acyclic (i.e., does not contain a directed cycle), because providers should always be higher in the Internet hierarchy than their customers. However, it turns out that the graphs obtained with the above-mentioned algorithms do in fact contain directed cycles. Therefore, we are interested in cuts and disjoint paths both in general directed graphs (without anti-parallel edges) and in acyclic graphs.

Outline. The remainder of the paper is structured as follows. In Section 2, we give the necessary definitions and discuss some preliminaries. Section 3 contains our complexity results and algorithms for disjoint paths and minimum cuts in general directed graphs. In Section 4, we consider acyclic graphs. We give our conclusions and point to some open problems in Section 5.

2 Preliminaries

Following the terminology from [15, 5, 7], where the problem of classifying the relationships between ASs is called the Type-of-Relationship (ToR) problem, we will call a simple directed graph $G = (V, E)$ a *ToR graph* if G has no loops and no anti-parallel edges (i.e., $(u, v) \in E$ implies $(v, u) \notin E$). In terms of the underlying motivation, a directed edge from u to v, where $u, v \in V$, means that u is a customer of v.

A path $p = v_1, v_2, \cdots, v_r$ in a ToR graph is *valid* (and called a valid v_1-v_r-path), if it satisfies the following condition:

> There exists some j, $1 \leq j \leq r$, such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq j-1$ and $(v_i, v_{i-1}) \in E$ for $j+1 \leq i \leq r$.

The part of the path from v_1 to v_j is called its *forward part*, the part from v_j to v_r its *backward part*. Note that valid paths are symmetric, i.e., the reverse of a valid s-t-path is a valid t-s-path. The existence of a valid s-t-path can be checked in linear time by performing a standard directed depth-first-search from s and from t and testing if any vertex is reachable from both s and t along a directed path.

Let $G = (V, E)$ be a ToR graph and let $s, t \in V$ be two distinct vertices. A set $C \subseteq V \setminus \{s, t\}$ is a *valid s-t-vertex-cut* if there is no valid path from s to t in $G - C$. A smallest such set C is called a *min valid s-t-vertex-cut*. (Note that there is no valid s-t-vertex-cut if there is a direct edge (s, t) or (t, s).) The *min valid s-t-edge-cut* is defined analogously. Two valid s-t-paths are called vertex-disjoint if the only vertices that they have in common are s and t. Similarly, they are called edge-disjoint if they have no edges in common.

The optimization problems that we are interested in are those of computing minimum size cuts and maximum size sets of disjoint paths, both in the vertex version and in the edge version: the min valid s-t-vertex-cut problem, the min valid s-t-edge-cut problem, the max vertex-disjoint valid s-t-paths problem, and the max edge-disjoint valid s-t-paths problem.

An approximation algorithm A for an optimization problem P is a polynomial algorithm that always outputs a feasible solution. We say that A is a ρ-approximation algorithm, or that its approximation ratio is ρ, if for all inputs I, $OPT(I)/A(I) \leq \rho$, if P is a maximization problem, or $A(I)/OPT(I) \leq \rho$, if P is a minimization problem. Here $OPT(I)$ is the objective value of an optimal solution, and $A(I)$ is the objective value of the solution computed by algorithm A, for a given input I.

APX is the class of all optimization problems (with some natural restrictions, see [2]) that can be approximated within a constant factor. A problem is *APX*-hard if every problem in *APX* can be reduced to it via an approximation preserving reduction. A consequence of *APX*-hardness is that there exists a constant $\rho > 1$ such that it is not possible to find a ρ-approximation algorithm for the problem unless $P = NP$. See [2] for further information about approximability classes and approximation preserving reductions.

3 Complexity Results and Algorithms for General Graphs

Before going into the complexity issues and algorithms, we introduce a very helpful *two-layer model* which leads to a relaxation of flows and cuts in ToR graphs.

3.1 The Two-Layer Model

From a ToR graph $G = (V, E)$ and $s, t \in V$ we construct a *two-layer model* H, which is a directed graph, in the following way. Two copies of the graph G are made, called the *lower* and the *upper layer*. In the upper layer all edge-directions are reversed. Every node v in the lower layer is connected with an edge to the corresponding copy of v, denoted v', in the upper layer. The edge is directed from v to v'. Finally we obtain the two-layer model H by identifying the two s-nodes (of lower and upper layer) and also the two t-nodes, and by removing the incoming edges of s and the outgoing edges of t.

A valid path $p = v_1, \dots, v_r$ in G with $v_1 = s$ and $v_r = t$ is equivalent to a directed path in H in the following way. The forward part of p, i.e., all edges $(v_i, v_{i+1}) \in p$ that are directed from v_i to v_{i+1}, is routed in the lower layer. Then there is a possible switch to the upper layer with a (v, v') type edge (there can be at most one such switch). The backward part of p is routed in the upper layer. If there is only a forward respectively a backward part of p, then the corresponding path in H is only in the lower respectively upper layer. See Figure 1 for an example.

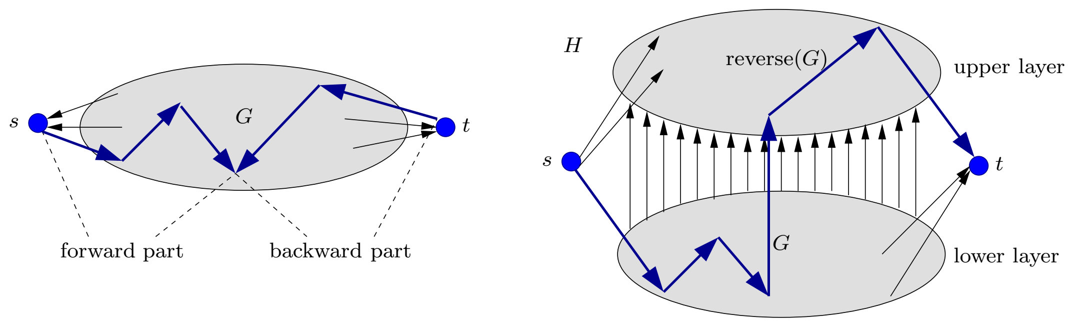

Figure 1: A path in the ToR graph G and the corresponding path in the two-layer model H.

We now detail in which sense the two-layer model yields relaxations of vertex- respectively edge-disjoint paths and vertex- respectively edge-cuts in ToR graphs.

Note that two vertex-disjoint valid paths in G directly give two vertex-disjoint paths in H. But two vertex-disjoint paths p_1, p_2 in H do not necessarily correspond to vertex-disjoint valid paths in G. The path p_1 might use the node v and the path p_2 its counterpart v' in the other layer, yielding two valid paths that are not vertex-disjoint in G. The analogous statements apply to edge-disjoint paths.

A valid s-t-vertex-cut in G directly gives an s-t-vertex-cut in H of twice the cardinality, simply take for each cut node in G the corresponding nodes from both layers in H. But, of course, there might be an s-t-vertex-cut in H without the property that for each node v in the cut, also its counterpart v' is in the cut. Analogous statements apply to edge-cuts.

3.2 Min Valid s-t-Vertex-Cut

3.2.1 *NP*- and *APX*-Hardness

Theorem 1 *For a given ToR graph $G = (V, E)$ and $s, t \in V$, finding the min valid s-t-vertex-cut is NP-hard and even APX-hard.*

Proof: We use a similar technique as in [10], reducing the undirected 3-way edge cut problem to the min valid s-t-vertex-cut problem in ToR graphs. In the undirected 3-way edge cut problem, we are given an undirected graph G, and three terminals v_1, v_2, v_3. The goal is to find a minimum set of edges in G such that after removing this set, all pairs of vertices in $\{v_1, v_2, v_3\}$ are disconnected. This problem is proven to be *NP*-hard in [4].

Let $G = (V, E)$ be such an undirected graph with 3 distinct terminals v_1, v_2 and v_3. We create a ToR graph G' in the following way: each node v of G is replaced with $\deg(v)$ copies of the same node. For each edge $\{u, w\}$ in G, a gadget consisting of 2 new nodes, $e_1^{u,w}$ and $e_2^{u,w}$, is added. The gadget includes an edge from $e_1^{u,w}$ to $e_2^{u,w}$, edges from all copies of u and w to $e_1^{u,w}$ and from $e_2^{u,w}$ to all copies of u and w. We also add two nodes s and t and the edges from s to all copies of v_1, from all copies of v_2 to s and t, and from t to all copies of v_3. See Figure 2 for a simple example.

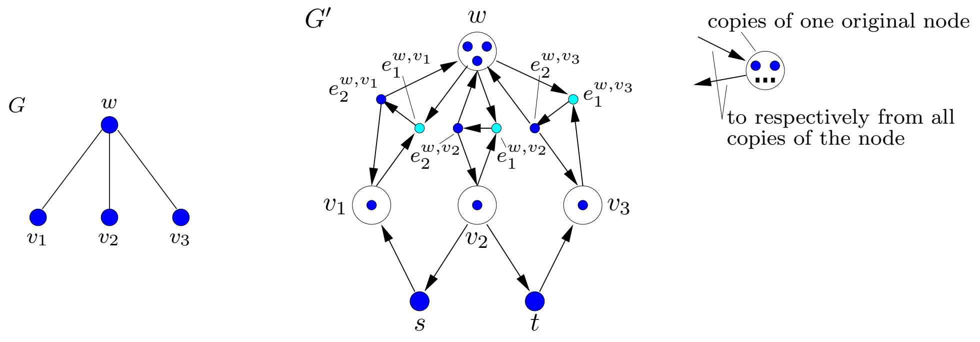

Figure 2: A simple example for the transformation of the original undirected graph G to the ToR graph G' .

Note that every valid path between any copy of u and any copy of w via the gadget added for the edge $\{u, w\}$ contains $e_1^{u,w}$. This holds because no path "copy of w, $e_2^{u,w}$, copy of u" or "copy of u, $e_2^{u,w}$, copy of w" is valid.

In the following we will first show that any valid s-t-vertex-cut in G' can be transformed into a cut of at most the same cardinality that only contains e_1 type nodes. Then we prove that there is a direct correspondence between 3-way-cuts in G and valid s-t-vertex-cuts in G' that contain only such e_1 type nodes. This yields in particular that an approximation algorithm for the min valid s-t-vertex-cut problem gives an approximation algorithm of the same ratio for the 3-way edge cut problem.

Assume we are given a valid s-t-vertex-cut C in G' that contains nodes that are not of type e_1. We can assume that C is a minimal cut (inclusion-wise). If the cut contains a copy u' of u, where u is a node in the original graph G, it must also contain all other copies of u. Otherwise there is always an equivalent "detour" path via one of these other copies, rendering the addition of u' to C superfluous. This contradicts the minimality of C. If C contains all $\deg(u)$ copies of a node u, we replace these nodes in C by the e_1 type nodes of the neighboring gadgets. There are exactly $\deg(u)$

such nodes. Every valid s-t path containing a copy of u traverses at least one of these neighboring gadgets (and therefore its e_1 node, see above). To see this, note that the paths "s, copy of v_2, t" and "t, copy of v_2, s" are not valid. Thus by this replacement we did not reintroduce previously cut paths. The cardinality of C did not increase. If C contains an e_2 type node, we replace it by the corresponding e_1 type node. Once more the cardinality of C does not increase and no valid paths are reintroduced. Now C contains only e_1 type nodes.

Next, we show that for a set of edges Q in the graph G, the corresponding set of nodes $C = \{e_1^{u,w} | \{u, w\} \in Q\}$ in G' is a valid s-t-vertex-cut iff Q is a 3-way cut for terminals v_1, v_2 and v_3 in G. (As previously mentioned, a 3-way cut disconnects all possible pairs in $\{v_1, v_2, v_3\}$.)

Note that the gadgets ensure that for any two nodes u and w in G' corresponding to the endpoints of an undirected edge $\{u, w\}$ in G, there exists a directed path from u to w and from w to u in the gadget added for $\{u, w\}$. To disconnect all such u-w paths, it suffices to cut the $e_1^{u,w}$ node.

First, if C is a valid s-t-vertex-cut, Q is a 3-way cut, because otherwise, if any pair of terminals v_1, v_2, v_3 are connected in G, then there will be a valid path between s and t which will give a contradiction. On the other hand, if Q is a 3-way cut, there is no valid path between s and t in $G' - C$, because at least one gadget is disconnected on every valid path corresponding to an undirected path between v_i and v_j for $i \neq j$ in G and, as noted before, the paths "s, copy of v_2, t" and "t, copy of v_2, s" are not valid.

Thus, we have shown that min valid s-t-vertex-cut is *NP*-hard. *APX*-hardness also follows directly from the *APX*-hardness of 3-way edge cut [4]. □

Note that the proof does not carry over to min valid s-t-edge-cut, because no gadget can be found where the role of the $e_1^{u,w}$ node is taken over by exactly one edge (such that the copies of u and the copies of w are disconnected if this edge is deleted). In fact, in Section 3.3 it is shown that a polynomial time algorithm exists for the min valid s-t-edge-cut problem.

3.2.2 A Simple 2-Approximation

Given a ToR graph $G = (V, E)$ and $s, t \in V$ (where we assume that there is no direct edge in G between s and t, because otherwise a valid s-t-vertex-cut does not exist), the min valid s-t-vertex-cut approximation algorithm is as follows:

```
ALGORITHM VERTEXCUT

1. From $G$ construct the two-layer model $H$, see Section 3.1.

2. Compute a min $s$-$t$-vertex-cut $C_H$ in $H$.

3. Output the set $C_G = \{v \in V | \text{ at least one copy of } v \text{ is in } C_H\}$ as valid $s$-$t$-vertex-cut.
```

Clearly $|C_G| \leq |C_H|$ holds and C_G is a valid s-t-vertex-cut in G. Let C_{opt} be a min valid s-t-vertex-cut in G. As mentioned in Section 3.1, by duplicating C_{opt} for both layers of H one obtains an s-t-vertex-cut in H. Thus $|C_H|$ is at most twice $|C_{opt}|$, which gives Theorem 2.

Theorem 2 *There is a 2-approximation for the min valid s-t-vertex-cut problem in ToR graphs.*

3.3 Min Valid s-t-Edge-Cut

Quite surprisingly there is a polynomial time algorithm for the min valid s-t-edge-cut problem in ToR graphs. This is in contrast to many flow and cut problems in directed and undirected graphs, where the node and the edge variant of the respective problem are of the same complexity.

Let EdgeCut be the reformulation of algorithm VertexCut from Section 3.2.2 which considers edges instead of vertices. It is clear that the same simple argumentation as given in that section yields that EdgeCut is a 2-approximation for the min valid s-t-edge-cut problem in ToR graphs. The proof of the following theorem shows that this algorithm in fact computes an optimal solution.

Theorem 3 *There is a polynomial time algorithm for the min valid s-t-edge-cut problem in ToR graphs.*

Proof: We begin by proving a lemma which states a crucial property of (optimal) valid s-t-edge-cuts in ToR graphs. For this we need to introduce the concept of cutting an edge e only *in one direction.* If e is cut in the forward direction, this means that valid paths that contain e in their forward part are indeed cut, but valid paths that contain e in their backward part are not affected. (Forward and backward parts of valid paths are defined in Section 2.) Cutting e in the backward direction has the analogous implications. In the standard notion of valid edge-cuts, an edge e is always cut *in both directions.* A *generalized valid s-t-edge-cut* is a valid s-t-edge cut that can contain edges that are cut only in one direction as well as edges that are cut in both directions. This notion relates to the two-layer model as follows: edges that are cut in the forward (backward) direction correspond to edges that are cut only in the lower (upper) layer of the two-layer model, and edges that are cut in both directions correspond to edges that are cut in both layers.

Now let G be a ToR graph and e an edge of a valid s-t-edge-cut C_G in G. We assume that C_G is minimal, in the sense that no edge can be removed from C_G while maintaining a valid edge-cut. Assume that we cut e only in the forward direction. Two things can happen: Either the cut is still valid in the sense that all valid s-t-paths are cut by it, or a valid s-t-path that uses e on its backward part and does not contain any other edge of C_G appears. In the first case, we call e a *forward-cut edge* of C_G. Now assume that we cut e only in the backward direction. If the cut is still valid, we call e a *backward-cut edge.* Note that e cannot be a forward-cut edge and a backward-cut edge, because otherwise $C_G \setminus \{e\}$ would be a valid edge-cut. If e is neither a forward-cut edge nor a backward-cut edge, i.e., if there is always a valid s-t-path if e is cut only in one direction, we call e a *bothway-cut edge.* See Figure 3 for an example.

Lemma 1 *Let $G = (V, E)$ be a ToR graph, $s, t \in V$ and C_G a valid s-t-edge-cut in G. C_G can be transformed into a generalized valid s-t-edge-cut of the same cardinality in which every edge is cut only in one direction. In particular, C_G contains no bothway-cut edges.*

Proof: For the given s-t-edge-cut C_G we describe an iterative process that transforms the cut edges into edges that are cut only in one direction while maintaining validity of the cut. In the beginning each edge is cut in both directions. Then for each edge e in C_G we check whether it is contained only in forward respectively backward parts of paths, i.e. whether it is a forward-cut edge or a backward-cut edge. Then we modify the cut so that e cuts paths only in that direction. We show that after each such transformation of an edge, C_G remains a valid cut.

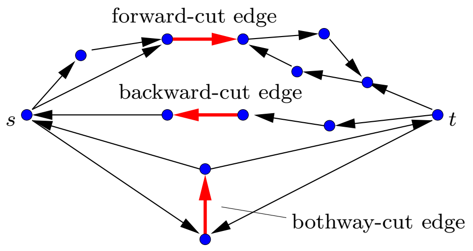

Figure 3: Different types of cut edges. Note that this figure is only for illustration purposes, the marked edges do not form a valid s-t-edge-cut of the graph. There is still a valid path from s to t: simply take the two bottommost edges. In fact, Lemma 1 implies that no valid s-t-edge-cut contains bothway-cut edges.

Consider the transformation of an edge $e = (u, v)$ in C_G, assuming that the already partially transformed C_G is a generalized valid cut. We will prove that by taking e (and no other edge) out of the cut, e only appears in either the forward or the backward parts of valid s-t-paths. Then if e is transformed correspondingly (to an edge that cuts only in the forward respectively backward direction), C_G obviously remains a generalized valid cut.

Let e be directed from u to v. Aiming for a contradiction, we assume that when taking e out of the cut, it appears in the forward part of some path p_1 and the backward part of another path p_2. Clearly the edges in p_1 from s to u are directed towards u (e is in p_1's forward part) and do not contain any cut edge (more precisely, if they pass a cut edge that is cut only in one direction, then they pass it in the opposite direction). Similarly, the edges in p_2 from t to u are also directed towards u and do not contain any cut edge. Thus the first part of p_1 from s to u and the last part of p_2 from u to t form a valid s-t path, contradicting the assumption that C_G is a generalized valid s-t-edge-cut and concluding the proof of the lemma. □

Lemma 1 implies that any s-t-edge-cut C_G in a ToR graph G contains only forward-cut edges and backward-cut edges. Considering the relationship of forward-cut edges and backward-cut edges to the two-layer model, this shows that any s-t-edge-cut C_G in a ToR graph G directly corresponds to an edge-cut C_H of the same cardinality in the two-layer model H: for each forward-cut edge in C_G add the corresponding edge in the lower layer of H to C_H, for each backward-cut edge in C_G add the corresponding edge from the upper layer. Conversely, every cut in H gives a cut in G of at most the same cardinality, cf. Section 3.1. This implies that an optimal s-t-edge-cut in the two-layer model H yields an optimal valid s-t-edge-cut in the ToR graph G. The former can be found in polynomial time by network flow techniques [1]. □

3.4 Max Vertex-/Edge-Disjoint Valid s-t-Paths

3.4.1 NP-Hardness and Inapproximability

Theorem 4 *For a given ToR graph $G = (V, E)$ and $s, t \in V$, finding the maximum number of vertex- respectively edge-disjoint valid s-t paths is NP-hard. Moreover the number of paths is even inapproximable within a factor $2 - \varepsilon$ for any $\varepsilon > 0$, unless P equals NP.*

Proof: We will reduce the problem of finding two disjoint paths between two pairs of terminals in a directed graph to this problem. Let G be directed graph and s_1, t_1 and s_2, t_2 four distinct vertices of G. Form a ToR graph G' from G by adding two vertices s and t, and edges from s to s_1, from t_1 to t, from t to s_2 and from t_2 to s. Note that no path $s, t_2, \cdots, t_1, t$ is valid. Thus, revealing the maximum number of vertex- respectively edge-disjoint valid s-t paths would give a solution to the problem of finding two vertex- respectively edge-disjoint paths between s_1, t_1 and s_2, t_2. The latter two problems are known to be NP-complete in general directed graphs [8].

This also directly gives the inapproximability gap. For an arbitrary $k \in \mathbb{N}$, we simply make k copies of the graph G'. Next we identify all copies of s to one node and all copies of t to one node. We then so to speak have k "parallel" copies of G. Depending on G there are either k or $2k$ vertex- respectively edge-disjoint valid paths between s and t. Let $\varepsilon > 0$ be some constant, independent of k. Clearly if a $(2-\varepsilon)$-approximation existed for max vertex- respectively edge-disjoint valid s-t paths, we could again solve the problem of finding two vertex- respectively edge-disjoint paths between s_1, t_1 and s_2, t_2 in G in polynomial time. □

3.4.2 A Tight Approximation Algorithm

For simplification of presentation we focus on the max vertex-disjoint valid s-t paths problem and comment at the end of the section how the result can be transfered to the max edge-disjoint case.

In order to state the approximation algorithm we need some definitions. If a forward part of a valid s-t path p_1 intersects with the backward part of a path p_2 at a node v, we speak of a *crossing* at v. The two paths can be *recombined* at the crossing to form a new path, consisting of the first part of p_1: $s, \cdots, v$ and the last part of p_2: $v, \cdots, t$. If p_1 and p_2 are recombined at v, the potential crossings on p_1 after node v and on p_2 before node v can be discarded. Note that p_1 and p_2 can be the same, in particular, if a path p contains a forward and a backward part, which meet at node u, we also say that the two parts cross at u.

Algorithm VertexDisjointPaths

1. From G construct the two-layer model H, compute max vertex-disjoint s-t paths $\mathcal{P}_H$ in H.
2. Interpret $\mathcal{P}_H$ as set $\mathcal{P}_G$ of valid s-t paths in G. Note: $\mathcal{P}_G$ is not necessarily vertex-disjoint! Let $\mathcal{F}$ denote the forward parts of paths in $\mathcal{P}_G$ and $\mathcal{B}$ the backward parts. Recombine the parts as follows:
   (a) Select any not yet recombined forward part p_f in $\mathcal{F}$ that has at least one *remaining* (i.e. not discarded, see below) crossing.
   (b) Choose the first *remaining* crossing on p_f, let p_b in $\mathcal{B}$ be the corresponding backward part.
   (c) Recombine p_f and p_b, discard all previous crossings on p_b. In particular if p_b was already recombined with p'_f, mark p'_f as not yet combined.
   (d) Repeat until each forward part is either recombined or has no remaining crossings.

Theorem 5 *The algorithm* VertexDisjointPaths *is a 2-approximation for the max vertex-disjoint valid s-t paths problem.*

Proof: We first prove that the algorithm actually outputs a set of vertex-disjoint paths, then mention why the running time is polynomial and finally show that the approximation ratio of 2 is achieved.

Note that since the paths $\mathcal{P}_G$ are derived in the two-layer model H, all forward parts $\mathcal{F}$ are disjoint and also all backward parts $\mathcal{B}$. Let $\mathcal{R}_i$ denote the set of recombined paths after the ith recombination. $\mathcal{R}_0$ is the empty set and thus vertex-disjoint. We now argue that if $\mathcal{R}_i$ is vertex-disjoint, then also $\mathcal{R}_{i+1}$ will be. In the $i+1$st recombination the selected forward part p_f does not intersect with any backward part in $\mathcal{R}_i$ up to the chosen crossing, say at node v. All potential crossings before v on p_f were discarded previously, see step 2.(b). We argue that also the rest of the backward part p_b: $v, \cdots, t$ does not intersect with any other path q in $\mathcal{R}_i$. Assume the contrary, then there is a path q in $\mathcal{R}_i$ whose forward part q_f intersects with p_b, say at node u. Let q_b be the backward part of q. Since the paths were derived from the two-layer model, at most two paths cross in each node. Thus q_f and q_b were recombined at a node $w \neq u$ and clearly w is after u on the forward part q_f (otherwise p_b would not intersect q_f). This gives the contradiction, since the algorithm would then have recombined q_f and p_b at node u instead of q_f and q_b at node w.

We have shown that the first part of p_f from s to v and the last part of p_b from v to t do not intersect any other path in $\mathcal{R}_i$. In case p_b was already recombined in $\mathcal{R}_i$ with some other forward part, this previous recombination is removed from $\mathcal{R}_i$, see step 2.(c). We conclude that $\mathcal{R}_{i+1}$ is vertex-disjoint. Since each crossing is considered at most once for recombination, the number of recombinations is in $O(|V|)$ and thus the running time is polynomial.

It remains to prove the approximation ratio. Assume that k is the optimal number of paths for a given instance. Clearly $|\mathcal{P}_G|$ is at least k. For each path p in $\mathcal{P}_G$ either its forward part p_f, its backward part p_b or both are recombined by the algorithm. If neither p_f nor p_b are recombined, p_f has at least one remaining crossing: namely the crossing with p_b. This crossing could not have been discarded, since p_b was never recombined with any forward part. Hence, either forward or backward part of each path are recombined, and thus at least $|\mathcal{P}_G|/2 \geq k/2$ disjoint valid s-t paths are found. □

The algorithm can be easily adapted to the edge-disjoint paths setting. Here the crossings are at edges instead of nodes. The recombination of two paths that cross at an edge $e = (u, v)$ is done at node u, where e is directed from u to v. An analogous argumentation as in the proof of Theorem 5 yields:

Theorem 6 *There is a 2-approximation algorithm for the max edge-disjoint valid s-t paths problem.*

Note that Theorem 4 implies that the approximation ratios of Theorems 5 and 6 are best possible unless $P = NP$.

3.5 On the Gap between Disjoint Paths and Minimum Cuts

In the standard model of paths in directed or undirected graphs, it is well known by Menger's theorem that the maximum number of edge-disjoint s-t-paths is equal to the size of a minimum s-t-edge-cut, and the analogous result holds for vertex-disjoint paths and vertex-cuts (provided that there is no direct edge from s to t). Therefore, it is interesting to consider whether similar properties hold for the valley-free path model.

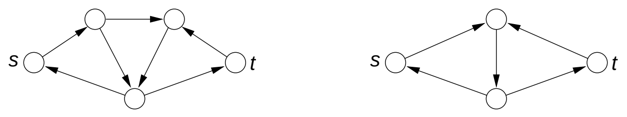

Figure 4: ToR graphs demonstrating a gap of 2 between disjoint paths and cuts.

Our approximation algorithms for disjoint valid paths and valid cuts are based on the two-layer graph model introduced in Section 3.1. If we consider standard directed paths in the two-layer graph H obtained from the ToR graph G, Menger's theorem applies and shows that the maximum number of vertex-disjoint paths from s to t in H is equal to the size of a minimum s-t-vertex-cut in H (assuming that there is no direct edge (s,t) in H). Denote this number by k. Our algorithm of Theorem 2 outputs a valid s-t-vertex-cut of size at most k in G and our algorithm of Theorem 5 produces a set of at least $k/2$ vertex-disjoint valid s-t-paths in G. This shows that if s and t are not connected by a direct edge, the size of a min valid s-t-vertex-cut is at most twice the maximum number of vertex-disjoint valid s-t-paths. To see that the bound of 2 is tight, consider the ToR graph shown in Figure 4 (left). Similar argumentation shows that the bound of 2 applies also to the edge-version of disjoint valid paths and valid cuts, and again the bound is tight as witnessed by the example shown in Figure 4 (right). In both examples, the maximum number of disjoint valid paths is 1 and the size of a minimum valid cut is 2. The examples can be generalized so that the maximum number of disjoint paths is k and the size of a minimum cut is $2k$, simply by introducing k copies of the subgraph between the vertices s and t.

4 Max Vertex-/Edge-Disjoint Valid s-t-Paths in DAGs

In this section, we consider the problem of computing vertex- or edge-disjoint paths in directed acyclic graphs. This is motivated by the consideration that in a strictly hierarchical network where customer-provider edges (cf. Section 1.1) always go from a lower to a higher level of the hierarchy, one would obtain ToR graphs that are acyclic.

4.1 NP-Hardness for Arbitrary Number of Paths

First, we are able to prove that the problems remain NP-hard even for acyclic graphs.

Theorem 7 *For a given acyclic ToR graph $G = (V, E)$ and $s, t \in V$, finding the maximum number of vertex- respectively edge-disjoint valid s-t-paths is NP-hard.*

Proof: In the following we will reduce the well known *3-Partition* problem. In this problem a set of $3n$ items $A = \{1, \ldots, 3n\}$ with associated sizes $a_1, \ldots, a_{3n} \in \mathbb{N}$, and a bound $B \in \mathbb{N}$ are given, with $B/4 < a_i < B/2$, for each i, and $\sum_{i=1}^{3n} a_i = nB$. It is then strongly NP-hard to decide whether A can be partitioned into n disjoint sets $I_0, \ldots, I_{n-1}$ such that $\sum_{i \in I_j} a_i = B$, for $j = 0, \ldots, n-1$. Note that due to the bounds for the item sizes a_i, all sets I_j must have cardinality 3. Since the problem is strongly NP-hard, it is already NP-hard if all a_i and consequently B are polynomially bounded in the input size. For our proof, we assume that this is the case.

For simplicity of notation we again focus on the vertex-disjoint case. It is easy to see that all arguments, with slight modification, transfer to the edge-disjoint case.

We start by describing how to construct an acyclic ToR graph G from the given 3-Partition instance. In optimal solutions for G there will generally be two different types of paths: The *set-paths* where each path corresponds to one of the sets $I_0, \ldots, I_{n-1}$ and the *blocker-paths* which make sure that each item in A is only in one of the sets I_j.

The structure in G which is dedicated to contain the set-paths consists of n rows, where the jth row corresponds to set I_j, and at most one set-path can be routed along this row. Figure 5 depicts the subgraph added for the jth row.

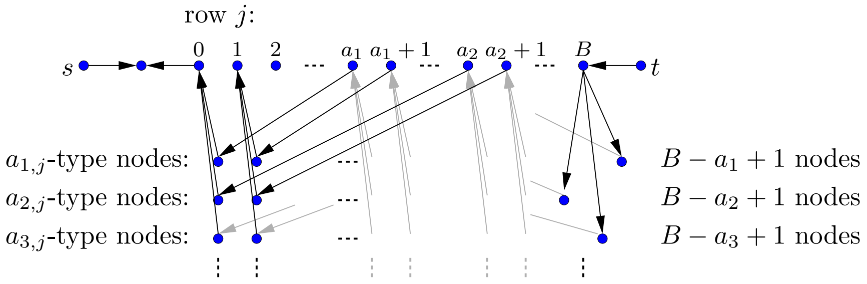

Figure 5: The structure added for the jth set-path. For each item i, a set of $B - a_i + 1$ nodes is added, these are referred to as $a_{i,j}$-type nodes. Each of these nodes serves as a "bridge" to skip a_i nodes in the jth row.

Clearly the graph so far is acyclic: all edges except the ones leaving s are directed from "right to left". An s-t-path traversing the jth row must contain three $a_{i,j}$-type nodes, say an $a_{i_1,j}$-, an $a_{i_2,j}$- and an $a_{i_3,j}$-type node, such that $a_{i_1} + a_{i_2} + a_{i_3} = B$. Note that if we set I_j to be $\{i_1, i_2, i_3\}$, and repeat this analogously for all j, the resulting sets $I_0, \ldots, I_{n-1}$ need not be disjoint. In the previous example it could even hold that e.g. $i_1 = i_2$.

The goal of the blocker-paths is to force that the sets $I_0, \ldots, I_{n-1}$ corresponding to the set-paths are actually disjoint. For item $i \in A$ we add the subgraph depicted in Figure 6. One can check that no cycles are created by the addition of these subgraphs to G. Note that the size of G is polynomial in the size of the 3-Partition instance.

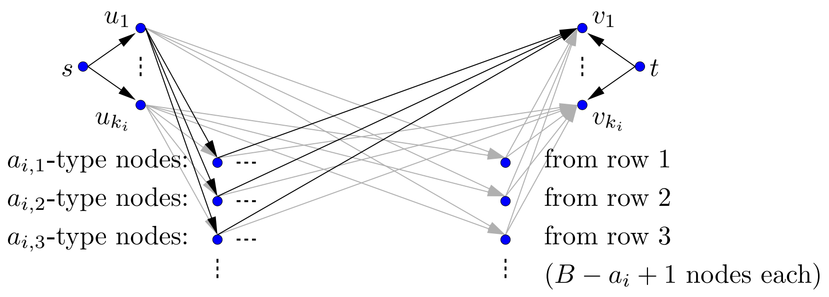

Figure 6: The structure added for the blocker-paths for item i. Note that each of the nodes $u_1, \ldots, u_{k_i}$ is connected to each $a_{i,j}$-type node (for fixed i). Similarly each $a_{i,j}$-type node (for fixed i) is connected to each of the $v_1, \ldots, v_{k_i}$ nodes. The number of potential blocker-paths is bounded by k_i, which is set to be the total number of $a_{i,j}$-type nodes (for fixed i and all $j = 0, \ldots, n-1$) minus one.

The idea of the construction is that in an optimal solution each blocker-path will come from some u node, pass through exactly one $a_{i,j}$-type node and then go directly to t via some v node.

Then for each item i there remains only one $a_{i,j}$-type node which is free and can be traversed by one of the set-paths.

Let $K = \sum_{i=1}^{3n} k_i$, where k_i is defined as in the caption of Fig. 6, be the total number of possible blocker-paths. Our goal is to prove that the given 3-Partition instance can be partitioned in the desired way if and only if there are $K + n$ vertex-disjoint valid s-t-paths in G.

We start with the easier direction: given a partition $I_0, \ldots, I_{n-1}$ we describe how to route $K+n$ vertex-disjoint paths. For the set $I_j = \{i_1, i_2, i_3\}$ we add a set-path p_j via the jth row of G, such that it passes through an $a_{i_1,j}$-, an $a_{i_2,j}$- and an $a_{i_3,j}$-type node. This can be done in an arbitrary order, the choice of the order determines which three $a_{i,j}$-type nodes are actually traversed. Clearly p_j is a valid s-t-path. We repeat this for all $j = 0, \ldots, n-1$. Since $I_0, \ldots, I_{n-1}$ are disjoint, the set-paths will touch exactly one $a_{i,j}$-type node for each item i. Thus the K possible blocker-paths can be routed in the canonical way.

Now we come to the harder direction: given $K + n$ vertex-disjoint paths we show how to derive a partition $I_0, \ldots, I_{n-1}$. First of all note that a valid s-t-path entering row j of G cannot "leave" this row. In other words when it passes through an $a_{i,j}$-type node, say w, it has to continue back to row j: It cannot continue directly to t via a v node (cf. Fig. 6), since this would produce a valley. It also cannot continue via a u node, since such a path cannot be completed to be a valid s-t-path because the only incoming edge of a u node is incident to s.

Thus our set-paths have the desired form and in particular each such path contains exactly three $a_{i,j}$-type nodes, whose corresponding item-sizes sum up to B. We are given $K + n$ vertex-disjoint paths, hence there must be exactly n set-paths and exactly K paths leaving s via the subgraphs added for the blocker-paths. Clearly each of the latter paths contains at least one $a_{i,j}$-type node. This yields that for each item i at most one $a_{i,j}$-type node is free and can be used by a set-path. This concludes the proof, since the sets $I_0, \ldots, I_{n-1}$ derived from the set-paths are disjoint and $\sum_{i \in I_j} a_i = B$ holds for all $j = 0, \ldots, n-1$. □

4.2 Efficient Algorithm for Constant Number of Paths

In general ToR graphs, it turned out to be *NP*-hard to decide whether there are two edge- or vertex-disjoint valid paths from s to t (Theorem 4). For acyclic graphs, on the contrary, we are able to show that this decision problem can be solved in polynomial time for any constant number of paths. Our proof is an extension of a pebbling game introduced by Fortune et al. [8] in order to solve the subgraph homeomorphism problem for subgraphs of fixed size in directed acyclic graphs (for example, their algorithm solves the problem of computing edge- or vertex-disjoint paths for a constant number of terminal pairs in a directed acyclic graph).

Theorem 8 *For a given acyclic ToR graph $G = (V, E)$, $s, t \in V$, and a constant k, one can decide in polynomial time if there exist k vertex-disjoint (edge-disjoint) valid paths between s and t in G (and compute such paths if the answer is yes).*

Proof: We present a polynomial-time algorithm which solves this problem. The algorithm uses a pebbling game played on the vertices of G.

First, consider the vertex-disjoint case. We show below that the winning of the pebbling game corresponds to finding k vertex-disjoint paths in G, and if there is no winning strategy, there are no k vertex-disjoint paths in the graph.

First, for each node we define its level as the length of the longest directed path starting at the node. At the beginning of the game, there are k red pebbles on the vertex s, and k blue pebbles on the vertex t. The game is won when all pebbles are removed. The rules how to move pebbles through the graph and how to remove them are as follows:

1. Pebble P_i can be moved along directed edge (v, w) from vertex v to vertex w if

   a) v has the highest level of any vertex containing a pebble.

   b) There is no pebble with the opposite color at v.

   c) w is equal to s or t; or w does not contain any pebble; or, if P_i is red, w contains exactly one blue pebble and no red pebble, and if P_i is blue, w contains exactly one red pebble and no blue pebble.

2. If v is a vertex of highest level among all vertices containing at least one pebble and if v contains a red pebble and a blue pebble, then these two pebbles can be removed from the graph.

If the pebble game is won, we have k vertex-disjoint valid paths, each one given by the trails of moving pebbles P_i^r from s and P_i^b from t to the vertex where they meet and are removed.

We have to show that these paths are indeed vertex-disjoint. Suppose for a contradiction that the trails of pebbles P_i^x and P_j^y (which do not arrive at the same node and are removed together) cross at a node $w \neq s, t$. The pebble that came first had to be moved away from w before the other pebble arrived, because of rule 1c); otherwise, the two pebbles would have different colors and would be removed together from w, contrary to our assumption. But the first pebble cannot be moved away from w before the second pebble arrives, because rule 1a) ensures that only pebbles at nodes with highest level can be moved (note that the vertex from which the second pebble arrives at w must have higher level than w). Thus, we arrive at a contradiction. This shows that the trails of pebbles belonging to different paths cannot cross at any nodes except s and t, implying that the k paths corresponding to the trails of the pebbles are indeed vertex-disjoint.

On the other hand, if there are k vertex-disjoint valid paths between s and t in G, it is easy to see that the pebbling game can be won. For each of the k paths from s to t, let a pebble start at s and trace the forward part of the path, let a second pebble start at t and trace the (reverse of) the backward part of the path, and remove the pebbles when they meet. During this process, the pebbles can obviously be moved and removed according to the rules above.

A configuration of the pebbling game is given by the at most $2k$ positions of the red pebbles and the blue pebbles. Thus there are at most $(|V| + 1)^{2k}$ configurations. One can build a graph on these configurations, with a directed edge from one configuration to another if it can be reached with one move or removal operation satisfying the rules above. The graph has polynomial size, and it suffices to check whether the node corresponding to the configuration without any pebbles can be reached from the node corresponding to the initial configuration. This can obviously be done in polynomial time, and the path from the initial to the final configuration yields also the trails of the pebbles and thus the vertex-disjoint valid s-t-paths that we are looking for.

The adaptation of this algorithm to the case of edge-disjoint paths is straightforward. Essentially it suffices to place pebbles on edges instead of vertices of G. □

5 Conclusions

In this paper, we initiated the study of disjoint valid s-t-paths and valid s-t-cuts in the valley-free path model. These problems arise in the analysis of the autonomous systems topology of the Internet if commonly used routing policies are taken into account. For example, the size of a minimum valid s-t-vertex-cut can be viewed as a reasonable measure of the robustness of the Internet connection between autonomous systems s and t. If the minimum cut has size k, this means that k autonomous systems must fail in order to completely disconnect s and t. Therefore, our algorithms could be useful for network administrators who want to assess the quality of their network's connection to the Internet.

We have proved that the problem of maximizing the number of vertex- or edge-disjoint valid paths can be approximated within a factor of 2, but no better unless $P = NP$. For the min valid s-t-cut problem, we showed that the vertex version is APX-hard and can be approximated within a factor of 2 while, somewhat surprisingly, the edge version can be solved optimally in polynomial time. For acyclic graphs, we have shown that a constant number of disjoint valid s-t-paths can be found in polynomial time (if they exist) while the max disjoint valid s-t-paths problem remains NP-hard.

There are several potentially interesting problems for future research. First, we do not have any inapproximability results for computing disjoint valid paths in acyclic graphs. It would be useful to study whether the maximum edge-disjoint or vertex-disjoint valid s-t-paths problem can be approximated better for acyclic graphs. Also, one could study the question whether there is a fixed-parameter tractable (FPT) algorithm [6] for the problem of finding k disjoint valid s-t-paths in acyclic graphs, i.e., an algorithm whose running-time is a polynomial of the input size multiplied by an arbitrary function of k. Recently, it was shown that the disjoint-paths problem (in the standard model of directed paths) for k terminal pairs in directed acyclic graphs is W[1]-hard, implying that the existence of FPT algorithms for that problem is unlikely [14].

In addition, it would be interesting to determine the complexity and approximability of the minimum vertex s-t-cut problem for acyclic graphs. Furthermore, we considered only cut and path problems for a single s-t-pair. It would be meaningful to study also the natural generalizations to multi-cuts and multiway-cuts. From a more practical perspective, it would be interesting to apply the concepts and algorithms to real Internet graphs (with economic relationships classified with algorithms such as those suggested in [9, 15, 7, 5]) in order to gain insights into the effects of policies on Internet routing, and we are currently pursuing this direction.

References

[1] A. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Englewood Cliffs, N. J., 1993.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties.* Springer, Berlin, 1999.

[3] P. Baake and T. Wichmann. On the economics of Internet peering. *Netnomics*, 1(1), 1999.

[4] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis. The complexity of multiway cuts. *SIAM J. Comput.*, 4(23):864–894, 1994.

[5] G. Di Battista, M. Patrignani, and M. Pizzonia. Computing the types of the relationships between autonomous systems. In *Proceedings of INFOCOM'03*, 2003.

[6] R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, New York, 1999.

[7] T. Erlebach, A. Hall, and T. Schank. Classifying customer-provider relationships in the Internet. In *Proceedings of the IASTED International Conference on Communications and Computer Networks*, pages 538–545, 2002.

[8] S. Fortune, J. Hopcroft, and J. Willie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

[9] L. Gao. On inferring Autonomous System relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.

[10] N. Garg, V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. In *Proceedings of ICALP'94*, pages 487–498, 1994.

[11] G. Huston. Interconnection, peering and settlements-part I. *Internet Protocol Journal*, March 1999.

[12] G. Huston. Interconnection, peering and settlements-part II. *Internet Protocol Journal*, June 1999.

[13] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary. The impact of Internet policy and topology on delayed routing convergence. In *Proceedings of INFOCOM'01*, 2001.

[14] A. Slivkins. Parametrized tractability of edge-disjoint paths on directed acyclic graphs. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, LNCS 2832, pages 482–493, 2003.

[15] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. Characterizing the Interenet hierarchy from multiple vantage points. In *Proceedings of INFOCOM'02*, 2002.

[16] H. Tangmunarunkit, R. Govindan, and S. Shenker. Internet path inflation due to policy routing. In *Proceedings of SPIE ITcom'01*, 2001.

[17] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. The impact of routing policy on Internet paths. In *Proceedings of INFOCOM'01*, 2001.