Cuts and Disjoint Paths in the Valley-Free Path Model

Thomas Erlebach, Alexander Hall, Alessandro Panconesi, and Danica Vukadinović

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 algorithm 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. For the multiway variant of the cut problem, we give a 4-approximation algorithm. We present additional results for acyclic graphs.

1. Introduction

Let $G = (V, E)$ be a directed, simple graph. 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. 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, i.e., a valley ($\cdot\diagdown\!\!\diagup\cdot$).

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 the problems of computing large sets of disjoint valid paths between two given vertices and of computing small vertex or edge cuts separating two given vertices with respect to all valid paths. The corresponding optimization problems for standard directed paths can be solved efficiently using network flow techniques (see, e.g., [Ahuja et al. 93]). 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. We also consider the multiway version of the cut problems, where the goal is to separate all pairs among a given set of k terminal nodes. Our main results are the following:

- 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 minimum 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 latter problem, i.e., computing a minimum valid s-t edge cut, we give a polynomial algorithm that computes an optimal solution.

- For the problem of computing a minimum-size valid multiway cut, we present 4-approximation algorithms for the vertex version and the edge version.

- We prove that the size of a minimum 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. 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, minimum *s-t* cuts, and minimum multiway 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 in 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) in 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.

ASs exchange BGP update messages that announce or withdraw *routes*. Here, a route is specified by a *prefix*, representing a contiguous block of IP addresses, and an *AS path*, specifying the sequence of ASs that traffic can traverse to reach a destination within that prefix. If AS A announces a route for prefix α with AS path p to AS B, this means that A tells B, "you can send traffic destined for prefix α to me, and I will forward it via path p." The AS B can then insert its own AS number at the beginning of the AS path p and announce the extended route to (some of) its other neighbors, so that these also learn a route to α. Generally, each AS originates some routes (those for prefixes that lie within the AS itself) and receives other routes from neighbors. We refer to the routes that an AS originates as *its own routes* and to the set of all routes that an AS stores as *its routes*. The AS can announce routes to all or some of its neighboring ASs. The decisions of which routes will be announced to which neighbor are determined by BGP routing policies. These policies depend mostly on the economic relationships between the ASs.

The nature of the commercial agreements between ASs has attracted a lot of attention in the Internet economics research community [Huston 99a, Huston 99b, Baake and Wichmann 99]. The main trends in the diversity of these agreements were described in a two-part work by Huston [Huston 99a, Huston 99b]. 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 show-
ing the impact of BGP policies on features such as path inflation and routing
convergence appeared [Tangmunarunkit et al. 01, Labovitz et al. 01].

As a consequence, the previously developed undirected model for the AS topol-
ogy 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 small number of categories was proposed for the BGP policies adopted
by a pair of ASs: customer-provider, peer-to-peer, and siblings [Gao 01]. A sim-
plified model with only two categories, customer-provider and peer-to-peer, was
proposed [Subramanian et al. 02]. 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 (i.e., those that it originates) and routes that it has learnt by announce-
ments from its 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 proposed model [Subramanian et al. 02] 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) that we study in this paper.
Therefore, without losing generality, we can consider a model without peer-to-
peer edges. Furthermore, if we incorporate the third type of AS relationship
[Gao 01], two ASs A and B are siblings if they announce all their routes to each
other. A sibling relationship between A and B can be suitably represented as a
pair of directed edges, one from A to B and one from B to A, because a sibling
relationship can act like a customer-provider edge and like a provider-customer
edge. In summary, the model we consider 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 relation-
ships from available topology data or AS path information have been proposed
in the literature [Gao 01, Subramanian et al. 02, Erlebach et al. 02, Di Bat-

tista et al. 03, Xia and Gao 04]. Other recent work investigates the formation of business relationships between ASs in a game-theoretic model [Anshelevich et al. 06].

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 up in the Internet hierarchy than their customers. However, it turns out that the graphs obtained with several of the abovementioned algorithms do in fact contain directed cycles. Therefore, we are interested in cuts and disjoint paths both in general directed graphs and in acyclic graphs.

1.2. 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 papers in which the problem of classifying the relationships between ASs is called the Type-of-Relationship (ToR) problem [Subramanian et al. 02, Di Battista et al. 03, Erlebach et al. 02], we will call a simple directed graph $G = (V, E)$ without loops a *ToR graph*. 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, and a pair of anti-parallel edges (u, v) and (v, u) means that u and v are siblings.

A path $p = (v_1, v_2, \ldots, v_r)$ 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*. If a path does not satisfy the condition, it is called *invalid*. Note that 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: instead of removing vertices, we remove edges. 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.

For a given ToR graph $G = (V, E)$ and a subset $T \subset V$ of k *terminals*, a *valid multiway vertex cut* is a subset $C \subseteq V \setminus T$ such that no two terminals in T are connected by a valid path in $G - C$. A *valid multiway edge cut* is defined analogously.

The optimization problems in which we are interested 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 min valid multiway vertex cut problem, the min valid multiway 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 [Ausiello et al. 99]) 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 $P = NP$. (For further information about approximability classes and approximation preserving reductions, see [Ausiello et al. 99].)

3. Complexity and Algorithms for General Graphs

Before treating the complexity issues and algorithms, we introduce a *two-layer model* that leads to a relaxation of disjoint paths and cuts in ToR graphs.
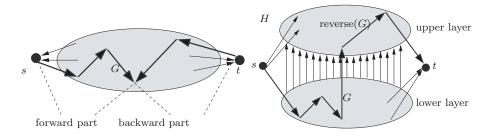
3.1. The Two-Layer Model

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 layer* 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'. For the edge versions of the considered problems, i.e., min valid s-t edge cut and max edge-disjoint valid s-t paths, we actually add $2n = 2|V|$ parallel copies of the edge (v, v'); the reason for this will become clear later. 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 $p = (v_1, \ldots, v_r)$ 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. See Figure 1 for an example. If there is only a forward or a backward part of p, then the corresponding path in H is only in the lower or upper layer, respectively.

We now describe in 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

Figure 1. A path in the ToR graph G (left) and the corresponding path in the two-layer model H (right).

paths. The $2n$ parallel edges of type (v, v') going from each node of the lower layer to its copy in the upper layer have been added to H so as to ensure that an arbitrary number of paths arising from edge-disjoint paths in G can switch from the lower layer to the upper layer at the same node.

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. On the other hand, 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. The $2n$ parallel edges of type (v, v') have been added to H to ensure that no min s-t edge cut in H will ever contain an edge of type (v, v'); note that an s-t edge cut of size at most $2n - 2$ always exists.

3.2.   Min Valid s-t Vertex Cut

3.2.1.   NP- and APX-hardness.   First, we study the complexity and approximation hardness of the min valid s-t vertex cut problem.
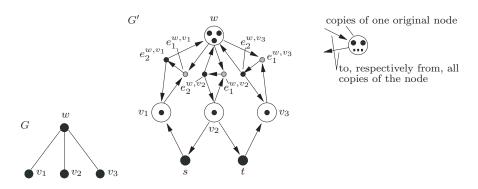
Theorem 3.1. *For a given ToR graph $G = (V, E)$ and $s, t \in V$, finding the min valid s-t vertex cut is NP-hard and even APX-hard.*

Proof. We use a technique similar to Garg et al.'s, reducing the undirected three-way edge cut problem to the min valid s-t vertex cut problem in ToR graphs [Garg et al. 94]. In the undirected three-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 has been proven to be *NP*-hard and *APX*-hard [Dahlhaus et al. 94].

Let $G = (V, E)$ be such an undirected graph with three 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 two 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 edges 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.

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

Figure 2. A simple example for the transformation of the original undirected graph G (left) to the ToR graph G' (right).

nodes. Then we prove that there is a direct correspondence between three-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 three-way edge cut problem.

Assume that 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, which would contradict 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 $(s, \text{copy of } v_2, t)$ and $(t, \text{copy of } v_2, s)$ 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 if and only if Q is a three-way cut for terminals v_1, v_2, and v_3 in G. Recall that a three-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 three-way cut, because otherwise, if any pair in $\{v_1, v_2, v_3\}$ is connected in G, then there is a valid path between s and t, which gives a contradiction. On the other hand, if Q is a three-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 $(s, \text{copy of } v_2, t)$ and $(t, \text{copy of } v_2, s)$ 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 three-way edge cuts [Dahlhaus et al. 94]. $\square$

Note that the proof does not carry over to the 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 we show that a polynomial-time optimal 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 given in Algorithm 1.

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 $|C_{opt}|$. This gives the following theorem.

Theorem 3.2. *There is a 2-approximation algorithm for the min valid s-t vertex cut problem in ToR graphs.*

Algorithm 1. (VertexCut.)

1. From G construct the two-layer model H as described in Section 3.1.

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

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

3.3.   Min Valid s-t Edge Cut

Quite surprisingly, there is a polynomial-time optimal 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 VertexCut (Algorithm 1) that considers edges instead of vertices. It is clear that the same simple argumentation as given in Section 3.2.2 shows that EdgeCut is a 2-approximation algorithm 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 3.3. *There is a polynomial-time optimal algorithm for the min valid s-t edge cut problem in ToR graphs.*
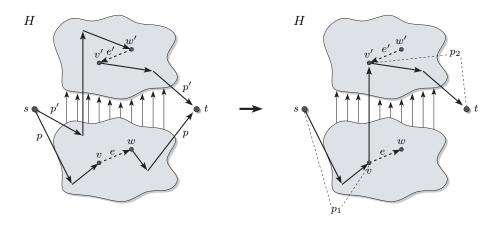
Before proving the theorem, we first prove a lemma that states a crucial property of (optimal) valid s-t edge cuts in ToR graphs.

Lemma 3.4. *Let $G = (V_G, E_G)$ be a ToR graph, $s, t \in V_G$, and C_G any valid s-t edge cut in G. From C_G an s-t edge cut C_H in the corresponding two-layer model $H = (V_H, E_H)$ can be derived with $|C_H| = |C_G|$.*

Proof. We start by adding for each edge $e \in C_G$ the two corresponding edges from the lower and upper layer to C_H. This yields an s-t edge cut C_H in H with $|C_H| = 2 \cdot |C_G|$, as already described in Section 3.1. Then, iteratively for each edge pair $e, e' \in C_H$, either e or e' is removed from C_H, where $e = (v, w) \in E_H$ is in the lower layer and $e' = (w', v') \in E_H$ is its counterpart in the upper layer. We will show that, assuming C_H is a cut before the removal, it will still be a cut afterwards, if the edge is properly chosen. Thus, in the end, after considering all edge pairs in the original cut, C_H is still a cut and $|C_H| = |C_G|$ holds.

Now we consider a single step of the iteration where the pair $e = (v, w), e' = (w', v') \in C_H$ is treated, assuming that the (perhaps already modified) set C_H is still an s-t edge cut in H. Assume that an s-t path p exists that traverses only e and no other edge of the cut, i.e., $e \in p$ and $e_c \notin p$ for all $e_c \in C_H \setminus \{e\}$. We claim that in this case no s-t path p' exists that traverses only e' and no other edge of the cut. It is then safe to remove e' from C_H. Symmetrically, if such a path p' exists, there cannot be a path p and thus e can be removed safely. (If neither p nor p' exists, remove e and continue the iteration.)

Aiming for a contradiction, we assume that both such paths p and p' exist. The edge $e \in p$ is directed from v to w, thus p has the form $(s, \ldots, v, w, \ldots, t)$.

Figure 3. Example showing how p and p' can be recombined.

Let $p_1 = (s, \ldots, v)$ denote the first part of p. Analogously, the edge $e' \in p'$ is directed from w' to v', and thus p' has the form $(s, \ldots, w', v', \ldots, t)$. Let $p_2 = (v', \ldots, t)$ denote the last part of p'. Neither p_1 nor p_2 contain an edge from C_H. Therefore, p_1 and p_2 can be recombined via the edge (v, v') to form an s-t path that does not contain any cut edge. See Figure 3 for an example. This is a contradiction to the assumption that C_H is an s-t edge cut. $\square$

Proof of Theorem 3.3. Lemma 3.4 implies that the optimal s-t edge cut in H has at most as many edges as the optimal valid s-t edge cut in G. Conversely, every cut in H gives a valid cut in G of at most the same cardinality: consider the sets C_G and C_H in the EdgeCut algorithm; clearly $|C_G| \leq |C_H|$ holds. Thus, 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 [Ahuja et al. 93]. Note that a min s-t edge cut in H does not contain any edge of type (v, v'), since there are $2n$ parallel copies of such an edge by construction of the two-layer model. This concludes the proof of Theorem 3.3. $\square$

3.4.   Min Valid Multiway Cut

In this section, we consider the multiway version of the cut problems. We are given a ToR graph $G = (V, E)$ and a subset $T \subset V$ of terminals, and the goal is to separate each terminal from all the other terminals. For the standard model of directed paths in directed graphs, the edge version and the vertex version of the multiway cut problem are polynomially equivalent and have been shown to be NP-hard and APX-hard [Garg et al. 94]. These hardness results hold even

for the case $|T| = 2$, i.e., for the problem of cutting all directed paths from s to t and all directed paths from t to s. The multiway cut problem in directed graphs can be approximated within a factor of 2 using an algorithm due to Naor and Zosin [Naor and Zosin 01].

In order to tackle multiway cut problems in the valley-free path model, we adapt the two-layer model of Section 3.1. Again, we create two copies of the given ToR graph G, the lower layer and the upper layer, and we reverse the edge directions in the upper layer. Furthermore, we join each vertex v in the lower layer to its counterpart v' in the upper layer by a directed edge (or by $4|E| + 1$ parallel directed edges, if we are dealing with the edge version of the valid multiway cut problem). Finally, for each of the terminals in T, we identify its copy in the lower layer with its copy in the upper layer. Let H denote the resulting two-layer model.

To solve the min valid multiway vertex cut problem in G, we compute a standard multiway vertex cut in H using the 2-approximation algorithm from [Naor and Zosin 01]. Let C_H denote the set of vertices in this cut. Then we output the set C_G of vertices in G at least one of whose copies is contained in C_H.

To solve the min valid edge-cut problem, we proceed analogously and apply Naor and Zosin's 2-approximation algorithm [Naor and Zosin 01] to compute a standard multiway edge cut F_H in H; then we output the set of edges of G at least one of whose copies is contained in F_H. Note that taking all edges of the upper layer and of the lower layer gives a multiway edge cut of size $2|E|$ in H, hence the 2-approximation algorithm will output a multiway edge cut of size at most $4|E|$. Thus, as H contains $4|E| + 1$ parallel vertical edges from each vertex in the lower layer to its copy in the upper layer, we can assume without loss of generality that none of the vertical edges are contained in F_H.

Theorem 3.5. *There is a 4-approximation algorithm for the min valid multiway vertex cut problem and for the min valid multiway edge cut problem.*

Proof. Let us first consider the vertex version of the problem. Let C^* be an optimal valid multiway cut in G. By taking both copies of each vertex in C^*, we obtain a multiway cut C_H^* in H satisfying $|C_H^*| \le 2|C^*|$. As the algorithm being used [Naor and Zosin 01] is a 2-approximation algorithm, we get that the computed cut C_H has size at most $4|C^*|$. The set C_G output by the algorithm is a valid multiway cut and satisfies $|C_G| \le |C_H| \le 4|C^*|$.

The analysis for the edge version of the problem is analogous. $\qquad\square$

In Theorem 3.1 we have shown that the min valid s-t vertex cut problem is APX-hard. The min valid multiway vertex cut problem, being a generalization,

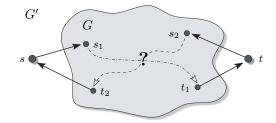is thus also APX-hard. We do not know the complexity of the min valid multiway edge cut problem.

3.5.   Max Disjoint Valid s-t Paths

3.5.1.   NP-hardness and inapproximability.
The following theorem clarifies the complexity of the max vertex- and edge-disjoint valid s-t paths problems and gives an inapproximability bound that turns out to be tight.

Theorem 3.6. *For a given ToR graph $G = (V, E)$ and $s, t \in V$, finding the maximum number of vertex- and 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 a 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, see Figure 4. Note that no path $(s, t_2, \ldots, t_1, t)$ is valid. Thus, revealing the maximum number of vertex- and edge-disjoint valid s-t paths would give a solution to the problem of finding two vertex- and 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 [Fortune et al. 80].

This also directly gives the inapproximability gap, even for instances with arbitrarily large objective value: 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-disjoint valid paths between s and t (same for the edge case). Let $\varepsilon > 0$ be some constant, independent of k. Clearly, if a

Figure 4. From a directed graph G with sources s_1, s_2 and sinks t_1, t_2, a ToR graph G' is constructed. It is NP-hard to decide whether there are two vertex- or edge-disjoint paths between s_1, t_1 and s_2, t_2 in the directed graph G.

$(2 - \varepsilon)$-approximation existed for max vertex- and edge-disjoint valid s-t paths, we could again solve the problem of finding two vertex- and edge-disjoint paths between s_1, t_1 and s_2, t_2 in G in polynomial time. $\square$

3.5.2. 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 transferred to the edge-disjoint case. Furthermore, in the proof of Theorem 3.7, we assume that every valid s-t path in the given graph $G = (V, E)$ contains a nonempty forward and a nonempty backward part. This is not true in general, but we can enforce this property with a simple replacement procedure that produces an instance that is equivalent to the original instance: we replace each edge $e = (v, s) \in E$ by the edges (s, v_e) and (v, v_e), where v_e is a new node, and we proceed similarly for all incoming edges of t. Note that after this replacement s and t have only outgoing edges and thus all valid paths have both a nonempty forward and backward part. Moreover, no new valid paths were introduced, and all previously valid paths remain valid.

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 call this a *crossing* at v. We also refer to v itself as a crossing sometimes. The two paths can be *recombined* at the crossing to form a new path, consisting of the first part of p_1, $(s, \ldots, v)$, and the last part of p_2, $(v, \ldots, t)$. The path that results from recombining p_1 and p_2 at v is denoted by $p_1 \otimes_v p_2$. We also use this notation if p_1 is a forward part of a path and p_2 is the backward part of a path (possibly of the same path). Note that if a path p consists of a forward and a backward part that meet at node u, the two parts cross at u.

The basic idea of the algorithm is to start with a set of paths that is at least as large as the optimal solution but not necessarily vertex-disjoint. However, the forward parts are vertex-disjoint, and so are the backward parts. Then the algorithm repeatedly recombines a forward part p_f at its first crossing x with a backward part p_b. When such a recombination happens, the crossings on p_b that lie before x are discarded. However, if p_b gets recombined with a different forward part p'_f later on, then p_f loses its recombination and can be recombined with another backward part at another crossing. The algorithm maintains the set $\mathcal{X}$ of crossings that have not yet been discarded, and the sets $\mathcal{F}_r$ and $\mathcal{F}$ of forward parts that are and are not currently recombined, respectively. Furthermore, the set $\mathcal{R}_i$ represents the currently recombined paths after i recombination steps. The detailed algorithm is shown in Algorithm 2.

Theorem 3.7. *The VertexDisjointPaths algorithm is a 2-approximation algorithm for the max vertex-disjoint valid s-t paths problem.*
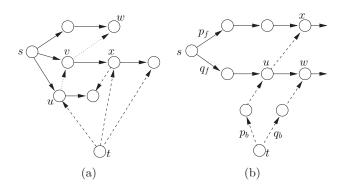

Algorithm 2. (VertexDisjointPaths.)

1. From G construct the two-layer model H, and 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 that $\mathcal{P}_G$ is not necessarily vertex-disjoint! Let $\mathcal{F}$ denote the forward parts of paths in $\mathcal{P}_G$ and $\mathcal{B}$ the backward parts. Let $\mathcal{X}$ be the set of all crossings (i.e., all $v \in V \setminus \{s, t\}$ that are contained in some $p_f \in \mathcal{F}$ and in some $p_b \in \mathcal{B}$). Let $\mathcal{F}_r = \emptyset$, $\mathcal{R}_0 = \emptyset$, and $i = 0$. Repeat the following steps until $\mathcal{F}$ is empty or all paths in $\mathcal{F}$ do not contain a crossing from $\mathcal{X}$:

   (a) Select any $p_f \in \mathcal{F}$ that contains at least one crossing from $\mathcal{X}$.

   (b) Let x be the first crossing from $\mathcal{X}$ on p_f, and let $p_b \in \mathcal{B}$ be the backward part containing x. (There is only one such p_b as all backward parts are vertex-disjoint.)

   (c) Remove from $\mathcal{X}$ all crossings on p_b before x.

   (d) Set $\mathcal{R}_{i+1} = \mathcal{R}_i$. If $\mathcal{R}_{i+1}$ contains a path p that was obtained as a recombination of some forward part p'_f with p_b, remove p from $\mathcal{R}_{i+1}$, remove p'_f from $\mathcal{F}_r$, and insert p'_f into $\mathcal{F}$.

   (e) Remove p_f from $\mathcal{F}$ and insert it into $\mathcal{F}_r$. Insert $p_f \otimes_x p_b$ into $\mathcal{R}_{i+1}$.

   (f) Set $i = i + 1$.

3. Output $\mathcal{R}_i$.

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 $\mathcal{B}$. We observe that once a backward part p_b gets recombined with some forward part p_f, that backward part p_b will remain recombined until the end of the algorithm; it may get recombined with different forward parts over time, however. On the other hand, a forward part may switch back and forth between $\mathcal{F}$ (not being recombined) and $\mathcal{F}_r$ (being recombined) several times. We also observe that if a recombined path $p_f \otimes_x p_b$ is in $\mathcal{R}_i$ at the end of the ith iteration of Step 2, then all the original crossings on p_b from x onward (towards t) are still in $\mathcal{X}$ (only the crossings on p_b before x have been removed from $\mathcal{X}$).

The set $\mathcal{R}_0$ is the empty set and thus vertex-disjoint. We now argue that if $\mathcal{R}_i$ is vertex-disjoint, then $\mathcal{R}_{i+1}$ will be as well. In the $(i + 1)$th recombination

Figure 5. (a) Setting with three forward parts (solid edges) and backward parts (dotted and dashed edges). (b) Impossible situation discussed in the proof of Theorem 3.7.

the selected forward part p_f does not intersect with any backward part of a path in $\mathcal{R}_i$ up to the chosen crossing. Since Step 2(b) chooses the first crossing $x \in \mathcal{X}$ on p_f, all potential crossings before x on p_f have already been removed from $\mathcal{X}$. Crossings that lie on backward parts of paths in $\mathcal{R}_i$ cannot have been discarded, as observed above. See Figure 5(a) for an example. In this figure, the algorithm may first recombine at u, forming the recombined path (s, u, t) and removing crossings v and w from $\mathcal{X}$; if the algorithm now considers the forward part starting (s, v, x), the first crossing from $\mathcal{X}$ on that path is at x, and the algorithm will form the recombined path (s, v, x, t). Indeed, since crossings before x on the forward part have already been discarded, the forward part of the newly formed recombined path does not intersect the backward part of any previously formed recombined path in $\mathcal{R}_i$.

We argue that also the rest of the backward part $p_b = (x, \ldots, t)$, that becomes part of the recombined path does not intersect 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 p_b, say at node u. Refer to Figure 5(b) for an illustration. 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 x and the last part of p_b from x 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 $\mathcal{R}_{i+1}$; see Step 2(d). We conclude that $\mathcal{R}_{i+1}$ is vertex-disjoint. Since each

crossing is considered at most once for recombination, the number of iterations of Step 2 is $O(|V|)$, 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 $\mathcal{P}_G$, we argue that either its forward part p_f, its backward part p_b, or both are in the recombined state when the algorithm terminates (i.e., constitute part of a path in the final set $\mathcal{R}_t$, where t is the number of iterations of Step 2). Assume for a contradiction that neither p_f nor p_b are in the recombined state when the algorithm terminates. (Note that p_f may have been recombined earlier but lost its recombination before the algorithm terminated.) Then, p_f must have at least one crossing in $\mathcal{X}$, 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 the forward or backward part of each path is recombined when the algorithm terminates, and thus at least $|\mathcal{P}_G|/2 \geq k/2$ disjoint valid s-t paths are found. $\square$

The algorithm can be easily adapted to the edge-disjoint paths setting. Here the crossings are at edges instead of nodes (except for the crossings where a forward part meets its corresponding backward part initially). 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 3.7 yields the next assertion.

Theorem 3.8. *There is a 2-approximation algorithm for the max edge-disjoint valid s-t paths problem.*

Note that Theorem 3.6 implies that the approximation ratios of Theorems 3.7 and 3.8 are best possible unless $P = NP$.

3.6.   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.

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
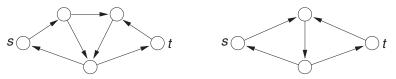
Figure 6. ToR graphs demonstrating a gap of 2 between disjoint paths and cuts.

in H (assuming that there is no direct edge (s, t) in H). Denote this number by k. Algorithm 1 of Theorem 3.2 outputs a valid s-t vertex cut of size at most k in G, and Algorithm 2 of Theorem 3.7 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 6 (left). A 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 6 (right). In both examples, the maximum number of disjoint valid paths is one 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.
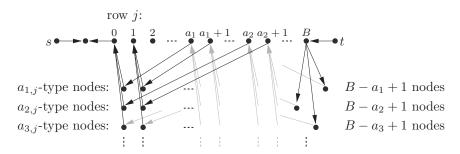
4.   Max Disjoint Valid s-t Paths in DAGs

In this section, we consider the problem of computing vertex- or edge-disjoint valid paths in directed acyclic graphs. This is motivated by the consideration that in a strictly hierarchical network where customer-provider edges (see 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 4.1. *For a given acyclic ToR graph $G = (V, E)$ and $s, t \in V$, finding the maximum number of vertex- or edge-disjoint valid s-t paths is NP-hard.*

Proof. In the following we will reduce the well-known *three-partition* problem [Garey and Johnson 79]. In this problem a set of $3n$ items $A = \{1, \ldots, 3n\}$ with

Figure 7. 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.

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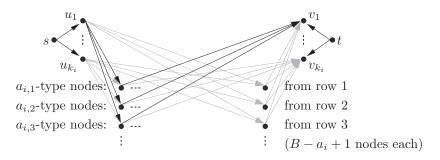
For simplicity of notation we again focus on the vertex-disjoint case. It is easy to see that all arguments, with slight modification, transfer to the edge-disjoint case.

We start by describing how to construct an acyclic ToR graph G from the given three-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* that make sure that each item in A is only in one of the sets I_j.

The structure in G that 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 7 depicts the subgraph added for 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 8. One can check that no cycles are created by the addition

Figure 8. 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.

of these subgraphs to G. Note that the size of G is polynomial in the size of the three-partition instance.

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 that 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 Figure 8, be the total number of possible blocker-paths. Our goal is to prove that the given three-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, it has to continue back to row j: it cannot continue directly to t via a v node (see Figure 8), 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 3.6). 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. 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) [Fortune et al. 80].

Theorem 4.2. *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 that solves this problem. The algorithm uses a pebbling game played on the vertices of G.

First, consider the vertex-disjoint case. We will show that winning the pebbling game corresponds to finding k vertex-disjoint paths in G and that 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 vertex s and k blue pebbles on vertex t. The game is won when all pebbles are removed. The rules for 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 1(c); 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 1(a) 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.

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. 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 for which we are looking.

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.

$\square$

5.   Conclusions

In this paper, we have 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. Note that our approximation algorithm for the min valid s-t vertex cut problem can be easily adapted to the weighted version of the problem, where an AS that is unlikely to fail can be given a large weight.

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. We have given a 4-approximation algorithm for the min valid multiway cut problem, both in the edge version and in the vertex version. 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.

The problems we have studied may be seen as instances of a more general family of problems whose common theme is that the allowed paths in the graph must obey certain restrictions. One example of such a restriction is given by oriented paths, i.e., paths containing at least one directed edge, in mixed graphs, i.e., graphs with undirected and directed edges. Oriented paths were considered by Wanke and Kötter in the context of the analysis of different parcellation schemes of the macaque brain [Wanke and Kötter 04]. Another example is given by paths in graphs with labeled edges where a path is only valid if the sequence of its edge labels forms a word from a given formal language; shortest-path problems for this type of restriction are studied by Barrett et al. in the context of transportation problems [Barrett et al. 00]. It would be interesting to study the max disjoint s-t paths problem and min s-t cut problem in such a setting.

There are also several open problems for the valley-free path model. 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 of whether there is a fixed-parameter tractable (FPT) algorithm [Downey and Fellows 99] 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 [Slivkins 03].

In addition, it would be interesting to determine the complexity and approximability of the min s-t vertex cut problem for acyclic graphs. Furthermore, improved approximation algorithms for the min valid multiway edge and vertex cut problems would be desirable. Moreover, we do not know the complexity of the min valid multiway edge cut problem.

References

[Ahuja et al. 93] A. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[Anshelevich et al. 06] E. Anshelevich, B. Shepherd, and G. Wilfong. "Strategic Network Formation through Peering and Service Agreements." In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pp. 77–86. Los Alamitos, CA: IEEE Computer Society, 2006.

[Ausiello et al. 99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Berlin: Springer, 1999.

[Baake and Wichmann 99] P. Baake and T. Wichmann. "On the Economics of Internet Peering." *Netnomics* 1:1 (1999), 89–105.

[Barrett et al. 00] C. L. Barrett, R. Jacob, and M. Marathe. "Formal Language Constrained Path Problems." *SIAM Journal on Computing* 30:3 (2000), 809–837.

[Dahlhaus et al. 94] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis. "The Complexity of Multiway Cuts." *SIAM Journal on Computing* 4:23 (1994), 864–894.

[Di Battista et al. 03] G. Di Battista, M. Patrignani, and M. Pizzonia. "Computing the Types of the Relationships between Autonomous Systems." In *INFOCOM 2003: Proceedings of the 22nd Annual Joint Converence of the IEEE Computer and Communications Societies*, pp. 156–165. Los Alamitos, CA: IEEE Press, 2003.

[Downey and Fellows 99] R. Downey and M. Fellows. *Parameterized Complexity.* New York: Springer-Verlag, 1999.

[Erlebach et al. 02] 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*, pp. 538–545. Anaheim, CA: ACTA Press, 2002.

[Fortune et al. 80] S. Fortune, J. Hopcroft, and J. Willie. "The Directed Subgraph Homeomorphism Problem." *Theoretical Computer Science* 10:2 (1980), 111–121.

[Gao 01] L. Gao. "On Inferring Autonomous System Relationships in the Internet." *IEEE/ACM Transactions on Networking* 9:6 (2001), 733–745.

[Garey and Johnson 79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W. H. Freeman and Company, 1979.

[Garg et al. 94] N. Garg, V. Vazirani, and M. Yannakakis. "Multiway Cuts in Directed and Node Weighted Graphs." In *Automata, Languages and Programming: 21st International Colloquium, ICALP'94, Jerusalem, Israel, July 11–14, 1994, Proceedings,* Lecture Notes in Computer Science 820, pp. 487–498. Berlin: Springer, 1994.

[Huston 99a] G. Huston. "Interconnection, Peering and Settlements—Part I." *Internet Protocol Journal* 2:1 (1999), 2–16.

[Huston 99b] G. Huston. "Interconnection, Peering And Settlements—Part II." *Internet Protocol Journal* 2:2 (1999), 2–23.

[Labovitz et al. 01] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary. "The Impact of Internet Policy and Topology on Delayed Routing Convergence." In *INFOCOM 2001: Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Commincations Societies,* pp. 537–546. Los Alamitos, CA: IEEE Press, 2001.

[Naor and Zosin 01] J. Naor and L. Zosin. "A 2-Approximation Algorithm for the Directed Multiway Cut Problem." *SIAM Journal on Computing* 31:2 (2001), 477–482.

[Slivkins 03] A. Slivkins. "Parametrized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs." In *Algorithms—ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16–19, 2003, Proceedings*, Lecture Notes in Computer Science 2832, pp. 482–493. Berlin: Springer, 2003.

[Subramanian et al. 02] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. "Characterizing the Internet Hierarchy from Multiple Vantage Points." In *INFOCOM 2002: Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies,* pp. 618–627. Los Alamitos, CA: IEEE Press, 2002.

[Tangmunarunkit et al. 01] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. "The Impact of Routing Policy on Internet Paths." In *INFOCOM 2001: Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Commincations Societies,* pp. 736–742. Los Alamitos, CA: IEEE Press, 2001.

[Wanke and Kötter 04] E. Wanke and R. Kötter. "Oriented Paths in Mixed Graphs." In *Algorithms and Computation: 15th International Symposium, ISAAC 2004, Hong Kong, China, December 20–22, 2004, Proceedings*, Lecture Notes in Computer Science 3341, pp. 629–643. Berlin: Springer, 2004.

[Xia and Gao 04] J. Xia and L. Gao. "On the Evaluation of AS Relationship Inferences." In *Proceedings of IEEE Global Communications Conference*, pp. 1373–1377. Los Alamitos, CA: IEEE Press, 2004.

Thomas Erlebach, Department of Computer Science, University of Leicester, University Rd., Leicester LE1 7RH, UK (t.erlebach@mcs.le.ac.uk)

Alexander Hall, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 (alex.hall@gmail.com)

Alessandro Panconesi, Dipartimento di Informatica, Università La Sapienza, via Salaria 113, 00198 Rome, Italy (ale@dsi.uniroma1.it)

Danica Vukadinović, Computer Engineering and Networks Laboratory (TIK), ETH Zürich, Gloriastrasse 35, 8092 Zürich, Switzerland (vukadin@tik.ee.ethz.ch)