Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs[*]

Aleksandrs Slivkins[†]

June 2003

Abstract

Given a graph and pairs $s_i t_i$ of terminals, the edge-disjoint paths problem is to determine whether there exist $s_i t_i$ paths that do not share any edges. We consider this problem on acyclic digraphs. It is known to be NP-complete and solvable in time $n^{O(k)}$ where k is the number of paths. It has been a long-standing open question whether it is fixed-parameter tractable in k. We resolve this question in the negative: we show that the problem is $W[1]$-hard. In fact it remains $W[1]$-hard even if the demand graph consists of two sets of parallel edges.

On a positive side, we give an $O(m + k!\,n)$ algorithm for the special case when G is acyclic and $G+H$ is Eulerian, where H is the demand graph. We generalize this result (1) to the case when $G+H$ is "nearly" Eulerian, (2) to an analogous special case of the unsplittable flow problem. Finally, we consider a related NP-complete routing problem when only the first edge of each path cannot be shared, and prove that it is fixed-parameter tractable on directed graphs.

1 Introduction

Given a graph G and k pairs $(s_1, t_1)\,,\ \ldots\ ,(s_k, t_k)$ of terminals, the edge-disjoint paths problem is to determine whether there exist $s_i t_i$ paths that do not share any edges. It is one of Karp's original NP-complete problems [10]. Disjoint paths problems have a great theoretical and practical importance; see [9, 13, 22] for a comprehensive survey.

The problem for a bounded number of terminals have been of particular interest. For undirected graphs, Shiloach [19] gave an efficient polynomial-time algorithm for $k = 2$, and Robertson and Seymour [17] proved that the general problem is solvable in time $O(f(k)n^3)$. The directed edge-disjoint paths problem was shown NP-hard even for $k = 2$ by Fortune, Hopcroft and Wyllie [8]. On acyclic digraphs the problem is known to be NP-complete and solvable in time $O(knm^k)$ [8].

Since 1980 it has been an interesting open question whether a better algorithm is possible for acyclic graphs. We should not hope for a polynomial-time algorithm, but can we get rid of k in the exponent and get a running time of $O(f(k)n^c)$ for some constant c, as Robertson and Seymour do for the undirected case? Such algorithms are called *fixed-parameter tractable*.

We resolve this open question in the negative using the theory of fixed-parameter tractability due to Downey and Fellows [5]. Specifically, we show that the directed edge-disjoint paths problem on acyclic graphs is $W[1]$-hard in k.

[*]This paper is available at *http://www.cs.cornell.edu/people/slivkins/research/*. It is a full version of a paper appearing in the 11th Annual European Symposium on Algorithms, 2003. This work has been supported in part by a David and Lucile Packard Foundation Fellowship and NSF ITR/IM Grant IIS-0081334 of Jon Kleinberg.

[†]Department of Computer Science, Cornell University, Ithaca NY 14853. Email: slivkins@cs.cornell.edu.

Fixed-parameter tractability. A problem is *parameterized* by $k \in \mathbb{N}$ if its input is a pair (x, k). Many NP-hard problems can be parameterized in a natural way; e.g. the edge-disjoint paths problem can be parameterized by the number of paths. Efficient solutions for small values of the parameter might be useful. For example, if the best-known solution is $O(2^n)$ and k is the parameter, then an $O(n^k)$ running time might be a big improvement, and $O(n2^k)$ is even better. Call a decision problem P *fixed-parameter tractable* in k if there is an algorithm that for every input (x, k) decides whether $(x, k) \in P$ and runs in time $O(|x|^c f(k))$ for some constant c and some computable function f.

Proving that some NP-complete parameterized problem is *not* fixed-parameter tractable would imply that $\mathrm{P} \neq \mathrm{NP}$. However, Downey and Fellows [5] developed a technique for showing *relativized* fixed-parameter intractability. They use reductions similar to those for NP-completeness. Suppose there is a constant c and computable functions f, g such that there is a reduction that maps every instance (x, k) of problem P to an instance $(y, f(k))$ of problem Q, running in time $O(g(k)|x|^c)$ and mapping "yes" instances to "yes" instances and "no" instances to "no" instances (we call it a *fixed-parameter reduction*). Then if P is fixed-parameter intractable then so is Q.

There are strong reasons to believe that the problem k-CLIQUE of deciding for a given undirected graph G and an integer k whether G contains a clique of size k is not fixed-parameter tractable [5]. Recently Downey et al. [4] gave a simpler (but weaker) alternative justification based on the assumption that there is no algorithm with running time $2^{o(n)}$ that determines, for a Boolean circuit of total description size n, whether there is a satisfying input vector.

Existence of a fixed-parameter reduction from k-CLIQUE to some problem P is considered to be an evidence of fixed-parameter intractability of P. Problems for which such reduction exists are called $W[1]$-hard, for reasons beyond the scope of this paper. For a thorough treatment of fixed-parameter tractability see Downey and Fellows [5].

Our contributions: disjoint paths. All routing problems in this paper are parameterized by the number k of terminal pairs. Given a digraph $G = (V, E)$ and terminal pairs $\{s_i, t_i\}$ the *demand graph* H is a digraph on a vertex set V with k edges $\{t_i s_i\}$. Note that H can contain parallel edges. Letting d^{in} and d^{out} be the in- and out-degree respectively, a digraph is called *Eulerian* if for each vertex $d^{in} = d^{out}$. The *imbalance* of a digraph is $\frac{1}{2}\sum_v |d^{out}(v) - d^{in}(v)|$.

Above we claimed that the directed edge-disjoint paths problem on acyclic graphs is $W[1]$-hard. In fact, we show that it is so even if H consists of two sets of parallel edges; this case was known to be NP-complete [6, 21]. Our proof carries over to the node-disjoint version of the problem.

On the positive side, recall that for a general H the problem is solvable in time $n^{O(k)}$ by [8]. We show a special case which is still NP-complete but fixed-parameter tractable. Specifically, consider the directed edge-disjoint paths problem if G is acyclic and $G + H$ is Eulerian. This problem is NP-complete (Vygen [21]). We give an algorithm with a running time $O(m + k!\, n)$.[1] This extends to the running time of $O(m + (k + b)!\, n)$ on general acyclic digraphs, where b is the imbalance of $G + H$.

Our contributions: unsplittable flows. We consider the unsplittable flow problem [11], a generalized version of disjoint paths that has capacities and demands. The instance is a triple (G, H, w) where w is a function from $E(G \cup H)$ to positive reals and $w(t_i s_i)$ is the demand on the i-th terminal pair. The question is whether there are $s_i t_i$ paths such that for each edge e of G the capacity w_e is greater or equal to the sum of demands of all paths that come through e. The edge-disjoint paths problem is a special case of the unsplittable flow problem with $w \equiv 1$.

[1]An alternative way to see that this problem is fixed-parameter tractable is to show that it is equivalent to the edge-disjoint paths problem on the undirected version of G [21], so the undirected disjoint paths algorithm of Robertson and Seymour [17] applies. However, as they observe in [9], their algorithm is extremely complicated and completely impractical even for $k = 3$.

The unsplittable flow problem can model a variety of problems in virtual-circuit routing, scheduling and load balancing [11, 14]. There has been a number of results on approximation [11, 14, 3, 20, 2]. Most relevant to this paper is the result of Kleinberg [12] that the problem is fixed-parameter tractable on undirected graphs if all capacities are 1 and all demands are at most $\frac{1}{2}$.

We show that the unsplittable flow problem is $W[1]$-hard on acyclic digraphs even if H is a set of parallel edges. If furthermore all capacities are 1 the problem is still NP-hard [11] (since for a two-node input graph with multiple edges it is essentially a bin-packing problem). We show it is fixed-parameter tractable with the running time of $O(e^k)$ plus one max-flow computation. However, the problem becomes $W[1]$-hard again if there are (a) *three* sink nodes, even if all demands are at most $\frac{1}{2}$, (b) two source nodes and two sink nodes, even if all demands are *exactly* $\frac{1}{2}$. This should be contrasted with the result of [12]

Moreover we show that similarly to disjoint paths, the unsplittable flow problem (a) can be solved in time $O(knm^k)$ if G is directed acyclic, (b) becomes fixed-parameter tractable if furthermore $G + H$ is *Eulerian under* w, that is if for each node the total weight of incoming edges is equal to the total weight of outgoing edges. The running time for the latter case is $O(m + k^{4k}n)$.

Our contributions: first-edge-disjoint paths. We looked for s_it_i paths under the constraint of edge-disjointness. To probe the boundary of intractability, we now relax this constraint and show that the associated routing problem is (still) NP-complete on acyclic digraphs but fixed-parameter tractable (even) on general digraphs. This problem turns out to capture a model of starvation in computer networks, see Section 4.2.

Call two paths *first-edge-disjoint* if the first edge of each path is not shared with the other path. The first-edge-disjoint paths problem is to determine whether there exist s_it_i paths that are first-edge-disjoint. We show that on digraphs any instance of this problem can be reduced, in polynomial time, to an instance whose size depends only on k. Then a solution can be found using enumeration or by other means. With some care we get the running time $O(mk + k^5(ek)^k)$. We improve on it for the cases when (a) all sources lie in distinct nodes, (b) the input graph is acyclic.

Further directions. Given our hardness result for edge-disjoint paths, we hope to address the case when G is acyclic and planar. This problem is still NP-complete [21], but becomes polynomial if furthermore $G + H$ is planar (this follows from [16], e.g. see [22]). The directed *node*-disjoint paths problem on planar graphs is solvable in polynomial time for every fixed k due to A. Schrijver [18].

Notation. Terminals s_i, t_i are called sources and sinks, respectively. Each terminal is located at some node; we allow multiple terminals to be located at the same node. We call a node at which one or more sources is located a *source node*. Similarly a node with one or more sinks is a *sink node*. Note that a node can be both a source and a sink node. Both source and sink nodes are called *terminal nodes*. If no confusion arises, we may use a terminal name to refer to the node at which the terminal is located.

We parameterize all routing problems by the number k of terminal pairs. We denote the number of nodes and edges in the input graph G by n and m respectively.

Organization of the paper. In Section 2 we present our hardness results, Section 3 is on the algorithmic results, and Section 4 is on the first-edge-disjoint paths problem.

2 Hardness results

Theorem 2.1 *The edge-disjoint paths problem is* $W[1]$*-hard on acyclic digraphs.*

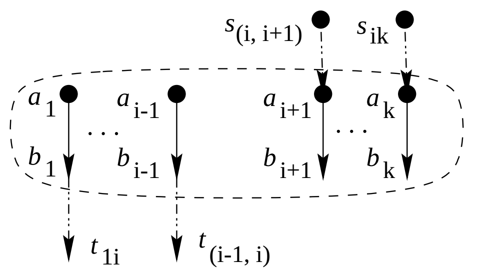
Proof: We define a fixed-parameter reduction from k-CLIQUE to the directed edge-disjoint paths problem on acyclic digraphs. Let (G, k) be the instance of k-CLIQUE, where $k \in \mathbb{N}$ and $G = (V, E)$ is an undirected graph without loops. We construct an equivalent instance (G', k') of the directed edge-disjoint paths problem where G' is a directed acyclic graph and $k' = k(k+1)/2$. Denote $[n] = \{1 \dots n\}$ and assume $V = [n]$.

Idea. We create a $k \times n$ array of identical gadgets. Intuitively we think of each row as a copy of V. For each row there is a path ('selector') that goes through all gadgets, skipping at most one and killing the rest, in the sense that other paths cannot route through them. This corresponds to selecting one gadget (and hence one vertex of G) from each row. The selected vertices form a multi-set of size k. We make sure that we can select a given multi-set if and only if it is a k-clique in G. Specifically, for each pair of rows there is a path ('verifier') that checks that the vertices selected in these rows are connected in G. Note that this way we don't need to check separately that the selected vertices are distinct.

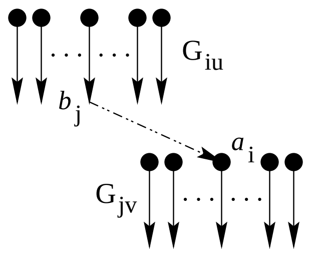
Construction. We'll use k paths P_i ('selectors') and $\binom{k}{2}$ paths P_{ij}, $i < j$ ('verifiers'). Denote the terminal pairs by $s_i t_i$ and $s_{ij} t_{ij}$ respectively. Selector P_i will select one gadget from row i; verifier P_{ij} will verify that there is an edge between the vertices selected in rows i and j.

Denote gadgets by G_{iu}, $i \in [k]$, $u \in V$. The terminals are distinct vertices not contained in any of the gadgets. There are no edges from sinks or to sources. We draw the array of gadgets so that row numbers increase downward, and column numbers increase to the right. Edges between rows go *down*; within the same row edges go *right*.

We start with the part of construction used by verifiers. Each gadget G_{iu} consists of $k-1$ parallel paths (a_r, b_r), $r \in [k] - \{i\}$. For each s_{ij} there are edges $s_{ij} a_j$ to every gadget in row i. For each t_{ij} there are edges $b_i t_{ij}$ from every gadget in row j (Fig. 1a); there will be no more edges from s_{ij}'s or to t_{ij}'s. To express the topology of G, for each edge uv in G and each $i < j$ we create an edge from b_j in G_{iu} to a_i in G_{jv} (Fig. 1b). There will be no more edges or paths between rows. The following claim explains what we have done so far.



(a) Gadget G_{iv}



(b) G_{iu} connected to G_{jv}, $i < j$.

Figure 1: Gadgets and verifiers.

Claim 2.2

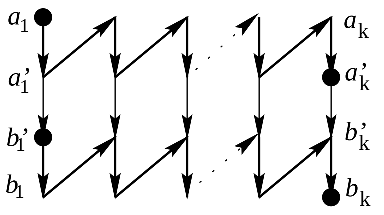
(a) Suppose we erase all gadgets in rows i and j except G_{iu} and G_{jv}. Then an $s_{ij} t_{ij}$ path exists if and only if $uv \in E$.

(b) Suppose we select one gadget in each row and erase all others. Then there exist edge-disjoint paths from s_{ij} to t_{ij} for each $i, j \in [k]$, $i < j$, if and only if the selected gadgets correspond to a k-clique in G.

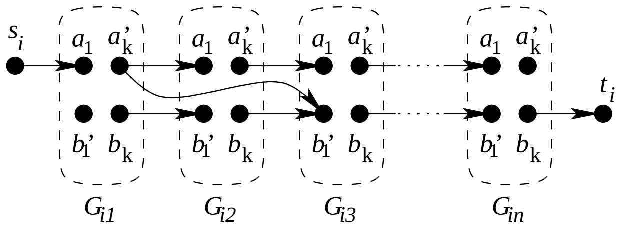
We could prove Claim 2.2 right away, but we will finish the construction first. Recall that each gadget consists of $k-1$ parallel wires (a_r, b_r). Each wire is a simple path of length 3: (a_r, a'_r, b'_r, b_r) (Fig. 2a). Let "level 1" be the set of all a_r and a'_r (in all wires and in all gadgets). Let "level 2" be the set of all b'_r and b_r. Each selector enters its row at level 1. The idea is that the only way it can skip a gadget is by going from level 1 to level 2, so, since within a given row there is no path back to level 1, at most one gadget can be skipped. The remainder of the construction makes this concrete.

First we complete the construction of a single gadget (Fig. 2a). In each gadget G_{iu} there are two edges from each wire r to the next one, one for each level. For $r \neq i-1, i$ these are $(a'_r a_{r+1})$ and $(b_r b'_{r+1})$ (note that there is no wire i). The edges between wires $i-1$ and $i+1$ are $(a'_{i-1} a_{i+1})$ and $(b_{i-1} b'_{i+1})$.

It remains to connect gadgets within a given row i (Fig. 2b). There are edges from s_i to a_1 in G_{i1}, and from b_k in G_{in} to t_i. There are two edges from each gadget to the next one, one for each level: from a'_k to a_1 and from b_k to b'_1. Finally, there are *jumps* over any given gadget in the row: an edge from s_i to b'_1 of G_{i2} jumps over G_{i1}, edges from a'_k of $G_{(i,u-1)}$ to b'_1 of $G_{(i,u+1)}$ jump over G_{iu}, and an edge from a'_k in $G_{(i,n-1)}$ to t_i jumps over G_{in}.



(a) A single gadget
(entry and exit points are circled)



(b) The i-th row
(only one jump edge is shown)

Figure 2: Additional wiring for selectors.

Proof of correctness. First we check that our construction is acyclic. It suffices to provide a topological ordering. For $i \in [k]$ and $j \in [2]$, let Q_{ij} be the ordering of vertices in the level j path in row i, i.e. Q_{i1} is the unique path from a_1 in G_{i1} to a'_k in G_{in} and Q_{i2} is the unique path from b'_1 in G_{i1} to b_k in G_{in}. Then the required ordering is given by

$$(\text{all sources};\ Q_{11}, Q_{12};\ Q_{21}, Q_{22};\ \ldots\ ; Q_{k1}, Q_{k2};\ \text{all terminals}).$$

Now we prove Claim 2.2. We stated part (a) for intuition only. The proof is obvious. For part (b), the 'if' direction is now straightforward since each gadget assigns a separate wire to each verifier than can potentially route through it, and the wires corresponding to a given verifier are connected in the right way. For the 'only if' direction, note that there is at most one edge between any given pair of gadgets in different rows, so the total number of edges between the selected gadgets is at most $\binom{k}{2}$. In fact it is exactly $\binom{k}{2}$ since each verifier has to use at least one of these edges. Therefore any pair of selected gadgets is connected, which happens if and only if the corresponding vertices are connected in G. Claim proved.

Claim 2.3 *For each possible $s_i t_i$ path there is a gadget such that verifiers cannot enter all other gadgets in row i.*

Proof: All edges between rows go "down", so if P_i ever leaves row i, it can never come back up. Thus P_i must stay in row i and visit each gadget in it successively, possibly jumping over one of them. If P_i enters a given gadget at a_1, it can either route through level 1 and exit at a'_k, or switch to level 2 somewhere in the middle and exit at b_k. If P_i enters at b'_1, it must route through level 2 and exit at b_k.

P_i starts out at level 1. If it never leaves level 1 then it uses up every edge $a_r a'_r$ (so verifiers cannot enter any gadget in the row). Else it switches to level 2, either within a gadget or by jumping over a gadget, call it G_{iu}. To the left of G_{iu} all edges $a_r a'_r$ are used by P_i, so verifiers cannot enter. To the right of G_{iu} the selector uses all edges $b'_r b_r$, so verifiers cannot exit the row from any G_{iv}, $v > u$. If a verifier enters such gadget it never leaves the row since within a row inter-gadget edges only go right. Therefore verifiers cannot enter gadgets to the right of G_{iu}, either. □

We need to prove that our construction is a positive instance of the directed edge-disjoint paths problem if and only if (G, k) is a positive instance of k-CLIQUE. For the "if" direction, let $u_i \dots u_k$ be a k-clique in G, let each selector P_i jump over G_{iu_i} and apply Claim 2.2b. For the "only if" direction, suppose our construction has a solution. By Claim 2.3 verifiers use only one gadget in each row (that is, all verifiers use the same gadget). Therefore by Claim 2.2b these gadgets correspond to a k-clique in G. This completes the proof of Thm. 2.1. □

Now we extend our result by restricting the demand graph.

Theorem 2.4

(a) *The edge-disjoint paths problem is* $W[1]$*-hard on acyclic digraphs even if the demand graph consists of two sets of parallel edges.*

(b) *The unsplittable flow problem is* $W[1]$*-hard on acyclic digraphs even if the demand graph is a set of parallel edges.*

Proof: (Sketch) In the construction from the proof of Thm. 2.1, contract all s_i, s_{ij}, t_i and t_{ij} to s, s', t and t', respectively. Clearly each selector has to start in a distinct row; let P_i be the selector that starts in row i. Since there is only one edge to t from the k-th row, P_{k-1} has to stay in row $k-1$. Iterating this argument we see that each P_i has to stay in row i, as in the original construction. So Claim 2.3 carries over. Each $s't'$ path has to route between some pair of rows, and there are at most $\binom{k}{2}$ edges between selected gadgets. This proves Claim 2.2b and completes part (a).

For part (b) all edges incident to s' or t' and all edges between rows are of capacity 1; all other edges are of capacity 2. Each verifier has demand 1, each selector has demand 2. Contract s' to s and t' to t. □

Kleinberg [12] showed that the *undirected* unsplittable flow problem apparently becomes more tractable when the maximal demand is at most a half of the minimal capacity. The next theorem shows that on acyclic digraphs this does not seem to be the case.

Theorem 2.5 *On acyclic digraphs, if all capacities are* 1 *and all demands are at most* $\frac{1}{2}$*, the unsplittable flow problem is* $W[1]$*-hard even if there are only (a) two source nodes and two sink nodes, (b) one source node and three sink nodes. Moreover, the first result holds even if all demands are* exactly $\frac{1}{2}$.

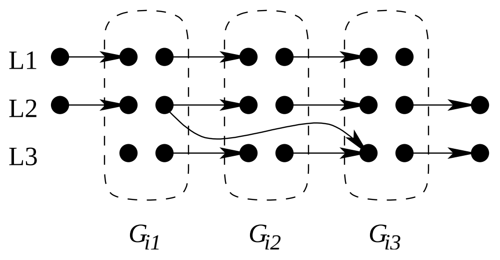
Proof: (Sketch) We will extend the basic construction from Thm. 2.4 and explain why the modifications do what they are supposed to do.

Let's start with part (a): all capacities are 1, all demands are $\frac{1}{2}$, one terminal pair (st) is for selectors, another ($s't'$) is for verifiers. Each row has three levels instead of two, call them L_1, L_2, L_3; two edges from s enter the row (at L_1, L_2) and two edges to t exit it (at L_2, L_3). There are four selectors per row; jump edges are from L_2 to L_3 (Fig. 3a).

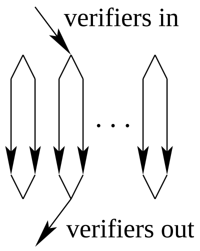
First of all, as in the proof of Thm. 2.4a, all edges to t must be saturated, so each selector must stay in its row. To see that most one gadget can be skipped in a given row, consider some gadget. There are three levels, each of which fits only two of the four selectors. So if neither of the selectors jumps over, each wire in this gadget will have some level occupied by two selectors so that the gadget is blocked (i.e. no verifiers can come through). Therefore if *any* verifiers is to come through, either two selectors jump over, or one jumps over and another switches to L_3 (else there are three selectors on L_1 and L_2, so the gadget is blocked). In either case in the next gadget there are two selectors on L_3. Since selectors cannot leave L_3, the rest of the row is blocked.

For every pair of rows there are now two verifiers instead of one. The wires in each gadget are doubled as shown in Fig. 3b, so that the two verifiers can come through. The inter-row edges are the same as in the basic construction. Claim 2.2b and the rest of the proof follows as in Thm. 2.4a.

For part (b), there is only one source node and three sink nodes: t for selectors, t' for verifiers, and t''. Keep the construction from part (a), merge s and s'. The problem is that now selectors could start on edges designed for verifiers and vice versa. To fix it, we add $k(k-1)/2$ new terminal pairs st'' of demand .2, call them "helpers", and change demand for each verifier to .4. Now if two helpers leave s on the same edge, there will not be enough edges from s to pack all the selectors and verifiers. Therefore the only way a helper can leave s is together with two verifiers. So, for each edge sv designed for verifiers, add edge vt''. Since there are $k(k-1)/2$ such edges and each helper has to use one of them, all verifiers must use them, too. $\square$



(a) The i-th row
(n=3; only one jump edges are shown)



(b) A single gadget (in- and out-edges are shown for only one double wire)

Figure 3: Additional wiring for selectors.

3 Algorithmic results

In this section G is a directed graph on n vertices, and H is the demand graph with k edges. Let S_k be the group of permutations on $[k] = \{1 \dots k\}$. Assuming a fixed numbering $s_1 t_1 \dots s_k t_k$ of terminal pairs, if for some permutation $\pi \in S_k$ there are edge-disjoint $s_i t_{\pi(i)}$ paths, $i \in [k]$, then we say that these paths are *feasible* and *realize* π. Let $\Pi(G, H)$ be the set of all such permutations. By abuse of notation we consider it to be a $k!$-bit vector.

Theorem 3.1 *Suppose G is acyclic and $G + H$ is Eulerian. Then we can compute $\Pi = \Pi(G, H)$ in time $O(m + k!\, n)$. In particular, this solves the directed edge-disjoint paths problem on (G, H).*
Proof: In G, let u be the vertex of zero in-degree, and let $v_1 \dots v_r$ be the vertices adjacent to u. Since $G + H$ is Eulerian, there are exactly r sources sitting in u, say $s_{i_1} \dots s_{i_r}$. Therefore each feasible path allocation induces k edge-disjoint paths on $G' = G - u$ such that there is a (unique) path that starts at each v_i. This observation suggests to consider a smaller problem instance (G', H') where we obtain the new demand graph H' from H by moving each s_{i_j} from u to v_j. The idea is to compute Π from $\Pi' = \Pi(G', H')$ by gluing the uv_i paths with paths in (G', H').

Formally, let H' be the demand graph associated with terminal pairs $s'_1 t_1 \ldots s'_k t_k$ where $s'_i = v_j$ if $i = i_j$ for some j, and $s'_i = s_i$ otherwise. Then, obviously, G' is acyclic and $G' + H'$ is Eulerian. Let $I = \{i_1 \ldots i_r\}$ and $S_I \subset S_k$ be the subgroup of all permutations on I extended to identity on $[k] - I$. We claim that

$$\Pi = \{\pi' \circ \sigma : \pi' \in \Pi' \text{ and } \sigma \in S_I\} \tag{1}$$

Indeed, let $\sigma \in S_I$ and $\pi' \in \Pi'$; let $P'_1 \ldots P'_k$ be paths that realize π' in (G', H'). Then $P_i = s_i s'_{\sigma(i)} \cup P'_{\sigma(i)}$ is a path from s_i to $t_{\pi(\sigma(i))}$ for all i (note that $P_i = P'_i$ for $i \notin I$). Paths $P_1 \ldots P_k$ are edge-disjoint, so $\pi' \circ \sigma \in \Pi$.

Conversely, let $\pi \in \Pi$ and $P_1 \ldots P_k$ be paths that realize it. The same paths restricted to G' realize some $\pi' \in \Pi'$. For each $i \in I$ the path P_i goes through some s'_j, say through $s'_{\sigma(i)}$. Let $\sigma(i) = i$ for $i \notin I$. Then $\sigma \in S_I$ and $\pi = \pi' \circ \sigma$. Claim proved.

We compute Π by iterating (1) n times on smaller and smaller graphs. To choose u we maintain the set of vertices of zero in-degree; recomputing it after each iteration takes time $O(k)$. To compute (1) in time $O(k!)$ we project Π' to $[k] - I$. □

Suppose $G + H$ is 'nearly' Eulerian in the sense that its imbalance $b = \frac{1}{2}\sum_v |d^{out}(v) - d^{in}(v)|$ is small. We can add b new terminal pairs st where s, t are new vertices, along with a few edges from s to G and from G to t, to get a new problem instance (G', H') such that G' is acyclic and $G' + H'$ is Eulerian. It is easy to see that (G, H) and (G', H') are in fact equivalent (Thm. 2 in [21]). This proves:

Theorem 3.2 *The edge-disjoint paths problem on acyclic digraphs is solvable in time* $O(m + (k + b)!\, n)$, *where* b *is the imbalance of* $G + H$.

Now we will extend the argument of Thm. 3.1 to the unsplittable flow problem. Recall that an instance of the unsplittable flow problem is a triple (G, H, w) where w is a function from $E(G \cup H)$ to positive reals. Let $d_i = w(t_i s_i)$ be the demand on the i-th terminal pair.

We will need a more complicated version of $\Pi(G, H)$. Let σ and τ be onto functions from $[k]$ to source and sink nodes respectively. Say (σ, τ) is a *feasible pair* if $\sum_{\sigma_i = s} d_i = \sum_{s_i = s} d_i$ for each source node s and $\sum_{\tau_i = t} d_i = \sum_{t_i = t} d_i$ for each sink node t. In other words, a feasible pair rearranges s_i's on the source nodes and t_i's on the sink nodes without changing the total demand on each source or sink node. Say paths $P_1 \ldots P_k$ *realize* a feasible pair (σ, τ) if these paths form a solution to the unsplittable flow problem on G with terminal pairs $\sigma_i \tau_i$ and demands w_i. Let $\Pi(G, H, w)$ be the set of all such feasible pairs.

Theorem 3.3 *Let* (G, H, w) *be an instance of the unsplittable flow problem such that* G *is acyclic and* $G + H$ *is Eulerian under* w. *Then we can compute* $\Pi = \Pi(G, H, w)$ *in time* $O(m + k^{4k} n)$. *In particular, this solves the unsplittable flow problem on* (G, H, w).

Proof: (Sketch) The proof is similar to that of Thm. 3.1. Again, letting u be a vertex of zero in-degree in G, the idea is to compute Π from a problem instance on a smaller graph $G' = G - u$.

We derive a problem instance (G', H', w') on k terminal pairs $s'_i t_i$ with demands d_i and capacities given by w. Again, letting $v_1 \ldots v_r$ be the nodes adjacent to u in G, the new demand graph H' is obtained from H by moving all sources from u to v_i's, arranging them in any (fixed) way such that $G' + H'$ is Eulerian under w'. It is easy to see that we get such arrangement from any set of paths that realizes some feasible pair. If such arrangement exists we can find it using enumeration; else Π is empty.

Similarly to (1), we compute Π from $\Pi(G', H', w')$ by gluing the uv_i paths with paths in G', except now we only consider uv_i paths that respect the capacity constraints on edges uv_i. □

Returning to the general acyclic digraphs, we extend the $n^{O(k)}$ algorithm of [8] from disjoint paths to unsplittable flows.

Theorem 3.4 *The unsplittable flow problem on acyclic digraphs can be solved in time* $O(knm^k)$.
Proof: We extend the pebbling game from [8]. For each $i \in [k]$ add nodes s_i' and t_i' and infinite-capacity edges $s_i's_i$ and t_it_i'. Define the pebbling game as follows. Pebbles $p_1 \ldots p_k$ can be placed on edges. Each pebble p_i has weight d_i. The *capacity constraint* is that at any moment the total weight of all pebbles on a given edge e is at most w_e. If a pebble p_i sits on edge e, define the *level* of p_i to be the maximal length of a path that starts with e. Pebble p_i can move from edge uv to edge vw iff p_i has the highest level among all pebbles and the capacity constraint on vw will be satisfied after the move. Initially each p_i is on $s_i's_i$. The game is won if and only if each p_i is on t_it_i'.

It is easy to see that the pebbling game has a winning strategy if and only if there is a solution to the unsplittable flow problem (paths in the unsplittable flow problem correspond to trajectories of pebbles). The crucial observation is that if some pebbles visit an edge e then at some moment all these pebbles are on e.

Let G_{state} be the state graph of the pebbling game, with nodes corresponding to possible configurations and edges corresponding to legal moves. The algorithm is to search G_{state} to determine whether the winning configuration is reachable from the starting one. The running time follows since there are m^k possible configurations and at most kn legal moves from each. □

Finally, we consider the case when the demand graph is just a set of parallel edges.

Lemma 3.5 *If the demand graph is a set of parallel edges and all capacities are 1, the unsplittable flow problem on directed or undirected graphs can be solved in time* $O(e^k)$ *plus one max-flow computation.*[2]
Proof: Let s, t be the source and the sink node respectively. Consider some minimal st-edge-cut C of G and suppose its capacity is greater than the total demand (else there is no solution). Any solution to the unsplittable flow problem solves a bin-packing problem where the demands are packed on the edges of C. If such a packing exists, it can be found by enumeration in time $O(\frac{k^k}{k!}) = O(e^k)$. By a well-known Menger's theorem there exist $|C|$ edge-disjoint st-paths. We can route the unsplittable flow on these paths using the packing above. □

4 First-edge-disjoint paths

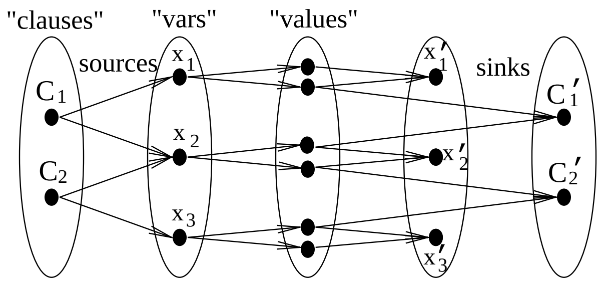
An instance of the first-edge-disjoint paths problem (FEDP) is a directed graph G and k pairs of terminals $s_1t_1 \ldots\ s_kt_k$. A *path allocation* is a k-tuple of paths from each source s_i to its corresponding sink t_i. A path allocation is *first-edge-disjoint* if in each path no first edge is shared with any other path in the path allocation. FEDP is to determine whether such a path allocation exists.

As a start, we claim that FEDP is NP-hard even if the underlying graph is acyclic. Call an edge e *blocked* in a path allocation ρ if it is the first edge of some s_it_i path in ρ.

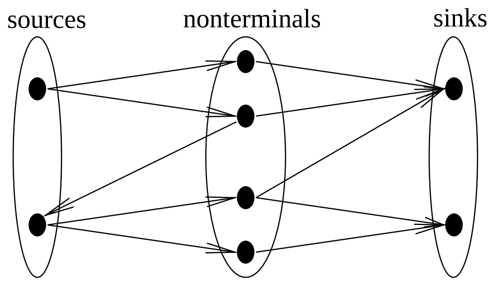
Lemma 4.1 *The first-edge-disjoint paths problem on acyclic graphs is NP-complete.*
Proof: Use a trivial reduction from SAT. Consider a CNF formula ϕ with variables x_i and clauses C_j. Let G be a directed graph with terminal pairs x_ix_i' and C_jC_j', for each x_i and C_j. For each x_i add two (separate) non-terminals u_i, v_i and four edges x_iu_i, u_ix_i', x_iv_i and v_ix_i'. For each $x_i \in C_j$ add edges C_jx_i and u_iC_j'. For each $\bar{x}_i \in C_j$ add edges C_jx_i and v_iC_j'. Then ϕ is satisfiable iff there exists a first-edge-disjoint path allocation on G. For example, the graph in Fig. 4a is equivalent to the formula $(x_1 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_3)$. □

Using similar but more complicated constructions one can show that on undirected and bi-directed graphs FEDP is NP-complete, too.

[2]Recall that without the restriction on capacities the problem is $W[1]$-hard on acyclic digraphs.



(a) reduction from SAT to FEDP



(b) instance of FEDP in normal form.

Figure 4: The first-edge-disjoint paths problem

Call an edge e *blocked* in a path allocation ρ if it is the first edge of some s_it_i path in ρ. Given a set E_b of edges, we can decide in polynomial time if there is a first-edge-disjoint path allocation whose set of blocked edges is E_b. For each edge $s_iv \in E_b$ we can identify the set of sinks reachable from v in $E - E_b$. The rest is a bipartite matching problem: for each source node u, we want to match each source s_i located at u with some edge e from E_b that leaves u, such that t_i is reachable from the tail of e. Thus instead of looking for a complete path allocation, it suffices to identify a suitable set E_b of blocked edges. Note that checking all possible sets of blocked edges is not efficient since source nodes can have large out-degrees. We will show how to prune the search tree.

The first step of our algorithm is to convert the input to a standard form that better captures the structure relevant to the problem.

Definition 4.2 *An instance of* FEDP *is* normal *if the vertex set can be partitioned into three disjoint sets: $l \leq k$ source nodes $u_1 \ldots u_l$, $l' \leq k$ sink nodes $v_1 \ldots v_{l'}$ and a number of nonterminal nodes w_{ij} for each u_i. For each w_{ij}, there is an edge from u_i to w_{ij}. All other edges in the graph lead from nonterminals to terminals (see Fig. 4b). All sources s_i are located at the source nodes and all sinks t_i at sink nodes.*

By Lemma A.1 from now on we will assume that the FEDP instance is normal and has size $O(mk)$; the transformation can be done in time $O(mk)$.

Definition 4.3 *Let k_i be the number of sources located at the node u_i. A terminal node r is i-*accessible *if there are at least $k_i + 1$ nonterminals w_{ij} such that there is an edge $w_{ij}r$. A terminal node r is i-*blockable *if there is an edge $w_{ij}r$ for some j but r is not i-accessible. A non-terminal node w_{ij} is i-*easy *if all nodes reachable from w_{ij} via a single edge are i-accessible. Otherwise, w_{ij} is i-*hard.

Call a path *blocked* if one of its edges is blocked. Note that if a terminal node u is i-accessible, then for any path allocation there is a non-blocked path from u_i to u via some nonterminal w_{ij}.

4.1 Reducing the graph

Given an instance G of FEDP in the normal form, we construct an equivalent smaller instance G_R whose size depends only on k, such that G_R is a yes instance if and only if G is.

Consider first the special case when no two sources are located at the same node. Consider a source s_i located at the node u_i. Let T_i be the set of terminal nodes reachable in one step from some i-easy nonterminal. Let G' be an instance obtained from G by deleting all i-easy nonterminals and adding two new nonterminals w'_{i1} and w'_{i2} with edges from s_i to both w'_{i1} and w'_{i2} and edges from each of w'_{i1}, w'_{i2} to every node in T_i (note that the new nonterminals are i-easy). For any first-edge-disjoint path allocation ρ' in G'

there is a first-edge-disjoint path allocation ρ in G. If in the path allocation ρ' the first edge of the $s_i t_i$ path goes to an i-hard node, the $s_i t_i$ path in ρ may use the same edge. If the $s_i t_i$ path in ρ' goes through one of the new nonterminals (let the next node on the path be some node r), the $s_i t_i$ path in ρ may use any i-easy nonterminal with an edge to r. It is easy to check that this choice preserves the reachability relation between any pair of terminal nodes in G and G'.

Thus we may assume that there are at most two i-easy nodes for each s_i. Notice that for each s_i, there are at most $2k-1$ i-hard nodes. Hence, for each source we have at most $2k$ choices for the first edge. The reduced graph has $O(k^3)$ edges, thus for each set of choices we can determine if it can be extended to a full path allocation in $O(k^4)$ time by depth first search. This reduction can be implemented in time $O(mk)$. Solving the reduced instance by enumeration gives us the following theorem.

Theorem 4.4 *If no two sources are located at the same node, the first-edge-disjoint paths problem can be solved in time* $O(mk + k^4\,(2k)^k)$.

The reduction for the general case is similar. For each i, let T_i be the set of nodes reachable in one step from at least $k_i + 1$ i-easy nonterminals. For each i, we create $k_i + 1$ new nonterminals $w'_{i1}, \ldots, w'_{i(k_i+1)}$ and add edges from u_i to each w'_{ij} and from every w'_{ij} to every node in T_i. Then we delete all edges from old w_{ij} nonterminals to vertices in T_i. Finally, we can delete all nonterminals without outgoing edges. As in the previous case, one can argue that the resulting graph G' is equivalent to the original. Consider source node u_i. There can be at most k_i edges entering each i-blockable terminal node r, there are $l + l' - 1$ terminals distinct from u_i, and hence there are at most $(k+l)k_i$ i-hard nonterminals. For each i-accessible terminal, there can be at most $k_i + 1$ edges entering it from i-easy nonterminals. Hence, there are no more than $2k(k_i + 1)$ i-easy nonterminals. Thus the reduced graph has $O(k^3)$ nodes. This reduction can be implemented in time $O(mk)$. Therefore:

Theorem 4.5 *The first-edge-disjoint paths problem is fixed-parameter tractable.*

A simple way to solve the reduced instance is to try all possible sets of blocked edges. In the rest of this subsection we give a more efficient search algorithm.

For a path allocation ρ let C_ρ be the set of i-hard nonterminals w_{ij} such that the edge $u_i w_{ij}$ is blocked in ρ, for all i. Suppose we are given C_ρ but not ρ itself. Then we can efficiently determine whether C_ρ was derived from a first-edge-disjoint path allocation.

Let E_ρ be the set of edges entering the nonterminals in C_ρ. Then, for each nonterminal w_{ij}, we can compute the set W_{ij} of terminal nodes r such that there is a path from w_{ij} to r in the graph $G - E_\rho$. Now we can formulate the following matching problem: for each source s_a, s_a located at node u_i, assign it an edge $u_i w_{ij}$ so that (a) each edge is assigned to at most one source, (b) $w_{ij} \in C_\rho$ or w_{ij} is i-easy, and (c) the sink t_a lies in W_{ij}.

Each first-edge-disjoint path allocation naturally defines a valid matching. It is easy to see that given a valid matching we can construct a first-edge-disjoint path allocation. Given a set C_ρ, the matching can be computed in $O(k^5)$ time using a standard Ford-Fulkerson max-flow algorithm.

We enumerate all possible sets C_ρ, and for each set check if it can be extended to a first-edge-disjoint path allocation. Recall that for each i, there are at most $x_i = (k + l - 1)k_i$ i-hard nonterminals. Hence, there are at most $\prod_{i=1}^{l} \sum_{j=1}^{k_i} \binom{x_i}{j}$ ways to choose the set C_ρ, which is $O((ek)^k)$ by Lemma A.2. Therefore:

Theorem 4.6 *The first-edge-disjoint paths problem can be solved in time* $O(mk + k^5\,(ek)^k)$.

We can improve this running time if the input graph is acyclic. We give an $O(mk + (k+1)!)$ algorithm for the case when no two sources are located at the same node (Thm. A.3). The general case can be improved similarly. We conclude with a simple fact that if all sinks are located at the same node, FEDP can be solved in polynomial time.

Lemma 4.7 *If all sinks are located at the same node t,* FEDP *can be solved in polynomial time. Moreover, if for each i there are at least $k_i + 1$ non-terminals w_{ij} such that there is a $w_{ij}t$ path, then it is a positive instance of* FEDP.

Proof: If there is a source node u_i for which less than k_i terminals w_{ij} have a path to t, there is no safe path allocation. If there is a source node u_i for which exactly k_i terminals w_{ij} have a path to t, all of them must be used. Since we know that all edges out of u_i are blocked, we may delete the edges from the graph and consider the reduced instance. If every source node u_i has more than k_i nonterminals that can reach t, blocking any k_i edges out of u_i cannot prevent any nonterminal $w_{i'}j$ from reaching t. $\Box$

4.2 Networking motivation

Consider a computer network with links and routers. Suppose each source s_i sends packets to its corresponding sink t_i via a fixed path. If the arrival rate at link e is not greater than the capacity c_e of e, all traffic gets through. Else, the link is said to be *congested*, and only a fraction of the incoming traffic can pass without delay. The rest is put into a buffer, or dropped if the buffer becomes full. Let β_i^e be the rate at which packets from source s_i arrive at link e. Under reasonable assumptions (steady flow, FIFO queuing policy), the rate at which packets from s_i get through e is approximately $c_e\beta_i^e / \sum_i \beta_i^e$ (Le Boudec [15]). If this rate drops below a certain minimum level, the s_it_i connection is said to be *starved*.

In Internet congestion control is implemented in TCP protocol, which has been quite successful at avoiding congestion (see [15] for more background). However, many existing applications use UDP or similar protocols that do not have congestion control at all, and therefore can cause congestion or even starvation [7].

One may wish a network to be *starvation-safe*, in the sense that no single source can, by increasing its sending rate, starve some other connection. The traditional approach to ensure starvation safety and other quality-of-service properties is via (priority) queuing policies in routers (e.g. see discussion in [1]). However, most existing routers use FIFO queuing. What about starvation safety with FIFO queuing? We study how the desired property depends on path allocation.

We model network links as directed edges of some finite capacity. It is easy to see that a connection A can starve a connection B if and only if B contains the first edge of A. Indeed, suppose the first edge e of A is contained in B. Then if A keeps increasing its sending rate, the throughput of B on e drops (arbitrarily close) to zero. For the converse, note that for the edges of A downstream from e the offered rate of A if bounded by c_e, so the throughput of B cannot become arbitrarily low. Claim proved. Therefore a given network is starvation-safe under the FIFO queuing policy if and only if the connections are routed along first-edge-disjoint paths.

In our model we assume full knowledge of the network topology, so it applies only to small ("autonomous") networks. Our FEDP algorithm trivially extends to an incremental path allocation model where some paths are already allocated (in a starvation-safe way), and k new path allocation requests arrive.

Acknowledgments. We thank Jon Kleinberg, Martin Pál and Mark Sandler for valuable discussions and help with the write-up.

References

[1] C. Albuquerque, B.J. Vickers and T. Suda, "Network Border Patrol," *Proc. 19th IEEE Conference on Computer Communications INFOCOM* (1), 2000.

[2] G. Baier, E. Köhler and M. Skutella, “On the k-Splittable Flow Problem,” *Proc. 10th Annual European Symposium on Algorithms*, 2002.

[3] Y. Dinitz, N. Garg and M. Goemans “On the Single-Source Unsplittable Flow Problem,” *Proc. 39th Annual Symposium on Foundations of Computer Science* , 1998.

[4] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto and F. Rosamund, “Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems,” *Computing: The Australasian Theory Symposium*, 2003.

[5] R.G. Downey and M.R. Fellows, *Parameterized Complexity*, Springer-Verlag (1999).

[6] S. Even, A. Itai and A. Shamir, “On the complexity of timetable and multicommodity flow problems,” *SIAM J. Computing*, **5** (1976) 691-703.

[7] S. Floyd and K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet,” *IEEE/ACM Transactions on Networking*, August 1999.

[8] S. Fortune, J. Hopcroft and J. Wyllie, “The directed subgraph homeomorphism problem,” *Theoretical Computer Science*, **10** (1980) 111-121.

[9] B. Korte, L. Lovász, H-J. Prömel, A. Schrijver, eds., *Paths, Flows and VLSI-Layouts*, Springer-Verlag (1990).

[10] R.M. Karp, “Reducibility among combinatorial problems,” *Complexity of Computer Computations*, R.E. Miller, J.W. Thatcher, Eds., Plenum Press, New York (1972) 85-103.

[11] J. Kleinberg, “Single-source unsplittable flow,” *Proc. 37th Annual Symposium on Foundations of Computer Science*, 1996.

[12] —, “Decision algorithms for unsplittable flow and the half-disjoint paths problem,” *Proc. 30th Annual ACM Symposium on the Theory of Computing*, 1998.

[13] —, “Approximation Algorithms for Disjoint Paths Problems,” *Ph.D. Thesis*, M.I.T, 1996.

[14] S.G. Kolliopoulos and C. Stein, “Improved approximation algorithms for unsplittable flow problems,” *Proc. 38th Annual Symposium on Foundations of Computer Science* , 1997.

[15] J.Y. Le Boudec, “Rate adaptation, congestion control and fairness: A tutorial,” *http://ica1www.epfl.ch/people/jy/leboudec_cv.html*.

[16] C.L. Lucchesi and D.H. Younger, “A minimax relation for directed graphs,” *J. London Mathematical Society* **17** (1978) 369-374.

[17] N. Robertson and P.D. Seymour, “Graph minors XIII. The disjoint paths problem,” *J. Combinatorial Theory Ser. B* **63** (1995) 65-110.

[18] A. Schrijver, “Finding k disjoint paths in a directed planar graph,” *SIAM J. Computing* **23** (1994) 780-788.

[19] Y. Shiloach, “A polynomial solution to the undirected two paths problem,” *J. of the ACM* **27** (1980) 445-456.

[20] M. Skutella, “Approximating the single source unsplittable min-cost flow problem,” *Mathematical Programming Ser. B* **91**(3) (2002) 493-514.

[21] J. Vygen, “NP-completeness of some edge-disjoint paths problems,” *Discrete Appl. Math.* **61** (1995) 83-90.

[22] —, “Disjoint paths,” *Rep. #94846*, Research Inst. for Discrete Math., U. of Bonn (1998).

Appendix A: Proofs omitted from Section 4

Lemma A.1 *An instance of the first-edge-disjoint paths problem can be converted, in time $O(mk)$, to an equivalent normal instance of size $O(mk)$.*
Proof: Start with a directed graph G and k terminal pairs $s_1 t_1 \dots s_k t_k$. Let S, T be the sets of source and sink nodes, resp. By replacing a vertex w by an edge uv we mean: create new vertices uv, add an edge uv, replace each edge xw with xu, replace each edge wx with vx, delete w. The reduction is as follows:

1. make sure S and T are disjoint: if $s_i = t_{i'}$ replace s_i by an edge uv and set $s_i = v$ and $t_{i'} = u$.
2. make sure there are no edges $s_i w$ such that w is a terminal node: else, replace w by an edge uv and move to v all terminals that were on w.
3. make sure there are no edges $s_i v$, $s_{i'} v$: else, replace the second edge by $s_{i'} v'$ where v' is a new vertex with the same edges as v.
4. take a transitive closure over non-blockable edges, i.e. for any two non-blockable edges uv and vw add an edge uw.
5. delete all non-terminal nodes to which there is no edge from from some terminal.
6. delete all edges between non-terminal nodes and all edges leaving T.

Clearly each step converts the problem to an equivalent one, and the end result is in the normal form. In steps 1-3 we add at most k, $\binom{k}{2}$ and m vertices, respectively, so the end result G' has $O(m)$ vertices. Since non-terminals in G' are only connected to terminals, G' has $O(mk)$ edges. To get the desired running time, note that steps 1-3 and 6 are trivial, and for steps 4-5 it suffices to compute, for all nodes v, the set R_v of terminals reachable from v via non-blockable edges. This can be done by $2k$ depth-first searches, in total time $O(mk)$. □

Lemma A.2 *Let $k_1 \dots k_l$ be positive integers that sum up to k. Let $x_i = (k+l-1)k_i$. Then*

$$\prod_{i=1}^{l} \sum_{j=0}^{k_i} \binom{x_i}{j} = O((ek)^k) \tag{2}$$

Proof: Let $y_i = \binom{x_i}{k_i}$ and $z = \frac{1}{k+l}$. For each $j < k_i$ we have $\binom{x_i}{j} \le z\binom{x_i}{j+1}$. Iterating this inequality we get $\binom{x_i}{j} \le y_i\, z^{k_i - j}$. Therefore $\sum_{j=0}^{k_i} \binom{x_i}{j} \le y_i \frac{1}{1-z} \le y_i \left(1 + \frac{1}{k}\right)$, so LHS (2) is at most $e \prod_{i=1}^{l} y_i$.

If $k_i = 1$ then $y_i = x_i < 2k$, which only helps us. So wlog $k_i \ge 2$ for all i. Now, since $\binom{a}{b} < \frac{a^b}{b!}$ and $b! > b^b e^{-b} \sqrt{2\pi b}$, we can bound $\prod_{i=1}^{l} y_i$ as

$$\prod_{i=1}^{l} y_i < \prod_{i=1}^{l} \frac{[e(k+l)]^{k_i}}{\sqrt{2\pi k_i}} < \frac{[e(k+l)]^k}{(4\pi)^{-l/2}} = (ek)^k f(\alpha),$$

where $l = \alpha k$, $\gamma = \sqrt{4\pi}$ and $f(\alpha) = (1+\alpha)\,\gamma^{-\alpha}$. Finally, $f(\alpha) \le 1$ on $[0,1]$ since $f(0) = 1$ and $f'(\alpha) < 0$ on $[0,1]$. □

Theorem A.3 *If no two sources are located at the same node, the first-edge-disjoint paths problem on acyclic graphs can be solved in time* $O(mk + (k+1)!)$

Proof: Suppose the input graph G is acyclic. Then the reduced graph G_R is acyclic, too. Without loss of generality we can rearrange s_i's so that if there is an $s_i s_j$ path then $i > j$. Note that if $i' > i$ there are no edges $w_{ij} s_{i'}$; if $i' < i$ edges $w_{ij} t_{i'}$ are meaningless since they cannot be used by the $s_{i'} t_{i'}$ path. Therefore, in G_R there are at most k i-blockable terminals, since at most k i-hard non-terminals w_{ij}, for each i. Therefore, the brute-force algorithm on G_R is $O(k^{k+4})$. We'll show a simple modification of this algorithm that runs in $O(mk + (k+1)!)$.

By *collapsing* a non-terminal w_{ij} we mean contracting the edge $s_i w_{ij}$. In particular, we can do it if the $s_i t_i$ path does not use w_{ij}.

Start with G_R. Consider s_1. Since G_R is acyclic, all possible $s_1 t_1$ paths are of the form $s_1 w_{1j} t_1$. Thus, we can collapse all w_{1j} such that there is no edge $w_{1j} t_1$. Now once we know that the $s_1 t_1$ path is routed through a specific edge $s_1 w$, we can reduce the graph as follows: collapse all $w_{1j} \neq w$, for each path $(w_{ij} s_1 t)$ add an edge $w_{ij} t$, delete s_1 and t_1.

We can detect in time $O(k^2)$ whether there exists a 1-easy non-terminal. If so, then wlog the $s_1 t_1$ path is routed through it, so we just reduce the graph and recurse on s_2. Else, for each remaining 1-hard non-terminal w we try to route the $s_1 t_1$ path through w, reduce the graph accordingly and recurse on s_2. $\square$