On Fixed-Parameter Tractability of Some Routing Problems (extended version*)

Aleksandrs Slivkins † Martin Pál ‡

November 6, 2002

Abstract

Disjoint Paths is the problem of finding paths between given pairs of terminals in a graph such that no vertices are shared between paths. We analyze fixed-parameter tractability of several new Disjoint Paths-like routing problems motivated by congestion control in computer networks.

In one model we are interested in finding paths between k pairs of terminals such that the first edge of each path is not shared with any other path. We prove that this problem is fixed-parameter tractable on directed graphs, in contrast to Disjoint Paths that are known to be NP-hard even for $k = 2$. We improve our algorithm for two special cases: when the graph is acyclic and when all sources lie in distinct nodes. We consider extensions: a second-node-disjoint analog and a slightly generalized version of SAT.

Another model, bottleneck-edge-disjoint paths, is a generalization of Disjoint Paths. For directed acyclic graphs, we show that bottleneck-edge-disjoint paths is $W[1]$-hard and hence unlikely to be fixed-parameter tractable. We give an algorithm that runs in time $n^{O(k)}$. These two results easily extend to Unsplittable Flows.

1 Introduction

Disjoint Paths is the problem of finding paths between given pairs of terminals in a graph such that no vertices are shared between paths. It is one of Karp's original NP-complete problems [10]. Disjoint Paths and related optimization problems have great theoretical and practical importance (e.g. see [7] and [15]). We introduce several new Disjoint Paths-like routing problems motivated by congestion control in computer networks.

New routing problems. In each problem, we are given a directed graph $G = (V, E)$ with k terminal pairs $s_1t_1 \dots s_kt_k$ located at nodes of G. We allow multiple terminals to be placed at a single node. Like in Disjoint Paths, the question is whether there exist s_it_i paths, one for each i, that satisfy a certain restriction. This restriction is different for each problem.

In Disjoint Paths the restriction is that the paths must be disjoint. In *first-edge-disjoint paths* (FEDP) the first edge of each path cannot be shared with any other path. In *second-node-disjoint paths* (SNDP) the second node of each path cannot be shared with any other path. In *bottleneck-edge-disjoint paths* (BEDP) each edge has capacity. The capacity of a path p is the minimum of

*This paper is available online at http://www.cs.cornell.edu/people/slivkins/research/fedp_full.ps

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

‡Department of Computer Science, Cornell University, Ithaca NY 14853. Email: mpal@cs.cornell.edu.

capacities of its edges. Edges of p with this capacity are called *bottleneck edges.* For each terminal pair $s_i t_i$ we are given demand d_i. The restriction is that the bottleneck capacity of each $s_i t_i$ path p_i is at least d_i, and no other path uses any bottleneck edge of p_i.

First-edge disjoint paths. FEDP is NP-complete even if the input graph is acyclic. It can be solved by brute force in time $O(kmn^k)$, where n and m denote the number of nodes and edges, respectively. Is there a better algorithm? For small values of k, it would be useful to have an algorithm whose running time depends polynomially on the size of the graph, although the dependence on k may be arbitrary. This leads us to the notion of fixed-parameter tractability, as defined by Downey and Fellows [3].

We show that for any instance of FEDP with k terminal pairs and n nodes, we can reduce it, in polynomial time, to an instance whose size depends only on k. A solution to the reduced instance can then be found using enumeration or by other means. In fact, we use a more efficient algorithm to solve the reduced problem to get an algorithm for FEDP that runs in time $O(mk+k^5(ek)^k)$. Under the assumption that all sources lie in distinct nodes, the running time improves to $O(mk+k^4(2k)^k)$. If furthermore the input graph is acyclic, a simple algorithm prunes the search tree for a running time of $O(mk+(k+1)!)$. We give heuristics for further pruning the search tree. We use a similar pruning technique to obtain provable bounds on running time for a (more) special case of FEDP that is naturally expressed as a slightly generalized version of SAT.

Fixed-parameter tractability. A decision problem P is (uniformly) fixed-parameter tractable (in parameter k) if there is an algorithm that for every input (x,k) decides if $(x,k) \in P$ and runs in time $O(|x|^c f(k))$ for some constant c and some function f. The function f may be an arbitrary recursive function. For a thorough treatment of the theory of fixed-parameter tractability see the monograph [3].

Assuming that some problem in NP is not fixed-parameter tractable is a stronger assumption than P $\neq$ NP. However, there are strong reasons to believe that the problem k-CLIQUE of deciding for a given graph G and an integer k whether G contains a clique of size k is not fixed-parameter tractable (see chapter 10 of [3] for justification). Downey and Fellows propose a technique for showing *relativized* fixed-parameter intractability. Suppose there is a constant c and recursive 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. Then if P is fixed-parameter intractable, Q is intractable too.

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.

Second-node-disjoint paths. It is easy to see that the first-edge disjoint paths problem can be reduced to SNDP. In fact, SNDP can be shown equivalent to a generalization of FEDP where we allow edges with infinite capacity. The goal there is to find paths so that the first edge of finite capacity on any path is not used by any other path.

We show that SNDP is NP-hard even if all sources are in one node, and all sinks are in another (st-SNDP). If the graph is undirected, st-SNDP can be reduced to the k-leaf spanning tree problem (given a graph G, does it have a spanning tree with at least k vertices?), hence is fixed-parameter tractable [3]. It is an open question whether SNDP is fixed-parameter tractable, or whether it is $W[1]$-hard.

Bottleneck-edge-disjoint paths and unsplittable flow. BEDP is a generalization of the k-edge-disjoint paths problem (EDP) on directed graphs. EDP on directed graphs was shown NP-hard even for $k = 2$ by Fortune, Hopcroft and Wyllie [6]. They also give an algorithm for EDP on directed acyclic graphs (DAGs) that runs in time $n^{O(k)}$. This raises an interesting open question whether EDP on DAGs is fixed-parameter tractable.

For BEDP, however, this question is settled. We show that a very restricted case of BEDP on DAGs is already $W[1]$-hard, thus unlikely to be fixed-parameter tractable. In fact, this special case is also a special case of the k-unsplittable flow problem [13, 14, 23], showing that the latter is $W[1]$-hard on DAGs as well. We provide algorithms for BEDP and k-unsplittable flow problem on DAGs that run in time $n^{O(k)}$.

For undirected graphs, Shiloach [21] gave an efficient polynomial-time algorithm for the 2-disjoint-paths problem, and Robertson and Seymour [20] proved that the k-disjoint path problem is fixed-parameter tractable. It would be interesting to know if such results could possibly be extended to BEDP or k-unsplittable flow.

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 [18]). If this rate drops below a certain minimum level, the $s_i t_i$ connection is said to be *starved*.

In Internet congestion control is implemented in TCP protocol via an algorithm by Van Jacobson [9] (see [18] for more background). TCP has been quite successful at avoiding congestion; see [2, 11, 12] for some theoretical analysis. 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 [4, 5, 19].

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? In the spirit of [16], we study how the desired property depends on path allocation.

We model network links as directed edges of some capacity. The question whether a given path allocation is starvation-safe translates, depending on additional assumptions, into the restrictions of FEDP, SNDP or BEDP. For example, if all capacities are finite, connection A can starve connection B iff B contains the first edge of A (FEDP). In Appendices A-B we explain how to get SNDP and BEDP. In our model we assume full knowledge of the network topology. Therefore it applies only to small ("autonomous") networks.

Since FEDP, SNDP and BEDP address starvation-safe path allocation, results on their complexity might be relevant to routing algorithms that care about starvation-safety. Moreover, 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. This incremental version can be used as a comparison point for "real-world" routing algorithms.

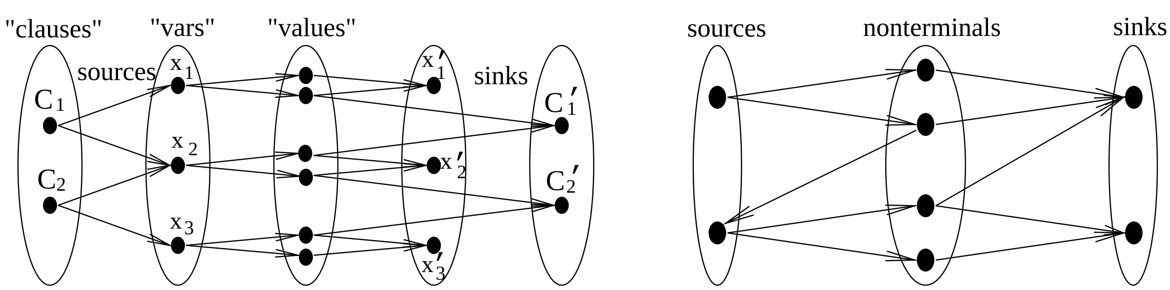
Organization of the paper. In section 2 we prove fixed-parameter tractability of FEDP and give an improved algorithm for the special case when all sources lie in distinct nodes. In Section 3 we consider extensions of FEDP: SNDP and a connection to SAT. Section 4 is about BEDP and Unsplittable Flows. Some proofs are deferred to appendices.

2 First-edge-disjoint paths

The instance of FEDP is a directed graph G and k pairs of terminals $s_1t_1 \ldots\ s_kt_k$. Each terminal is located at some node of G; 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. A *routing* is a k-tuple of paths from each source s_i to its corresponding sink t_i. A routing is *first-edge-disjoint* if in each path no first edge is shared with any other path in the routing. The first-edge-disjoint paths problem (FEDP) is to determine whether such a routing exists. An edge e is *blocked* in a routing ρ, if it is the first edge of some s_it_i path in ρ.

As a start, we claim that first-edge disjoint paths problem is NP-hard even if the underlying graph is acyclic.

Lemma 2.1 FEDP *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 routing on G. For example, the graph in Fig. 1a is equivalent to the formula $(x_1 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_3)$. $\square$



(a) reduction from SAT to FEDP

(b) instance of FEDP in normal form.

Figure 1: First-edge-disjoint paths

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

If we are given a set E_b of edges, we can decide in polynomial time if there is a first-edge-disjoint routing whose set of blocked edges is E_b. For each edge $e \in E_b$ we can identify the set of sinks reachable from the tail of e on paths not using edges from 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 routing, it suffices to identify a suitable set E_b of blocked edges.

In Appendix D we show that in time $O(mk)$ every instance of FEDP can be converted to the following normal form (which can then be stored using $O(mk)$ space).

Definition 2.2 *An instance of* FEDP *is in a* normal form *if the vertex set can be partitioned into three disjoint sets: $l \leq k$ source nodes $u_1 \dots u_l$, $l' \leq k$ sink nodes $v_1 \dots v_{l'}$ 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. 1b). All sources s_i are located at the source nodes and all sinks t_i at sink nodes.*

From now on we assume that we are given a FEDP instance in the normal form.

Definition 2.3 *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 a path $u_i w_{ij} r$. 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*.

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

2.1 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 (DISTINCT-FEDP). 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 routing ρ' in G' there is a first-edge-disjoint routing ρ in G. If in the routing 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 routing in $O(k^4)$ time by depth first search. In Appendix D we show how to implement the above reduction in $O(mk)$ time. Solving the reduced instance by enumeration gives us the following theorem.

Theorem 2.4 DISTINCT-FEDP *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 $w'_{i1}, \dots, w'_{i(k_i+1)}$ 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. Therefore:

Theorem 2.5 FEDP *is fixed-parameter tractable.*

In Appendix D we show how to implement the above reduction in $O(mk)$ time. 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.

Suppose we know, for each i, the set C_i of i-hard nonterminals w_{ij} such that the edge $u_i w_{ij}$ is the first edge of some path in a routing ρ. Let $C = \bigcup_i C_i$. 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 $G - \{e \mid e = (u_{i'} w_{i'j'}), w_{ij} \in C)\}$. Now we can formulate the following matching problem: for each source s_a, s_a located at node u_i, assign it an edge $e = u_i w_{ij}$ so that no edge is assigned to more than one source and

- either $w_{ij} \in C_i$ or w_{ij} is i-easy
- the sink $t_a \in W_{ij}$

Each feasible routing naturally defines a valid assignment. Also, given a valid matching, we can construct a feasible routing. For each a, let the first edge of the $s_a t_a$ path be the edge assigned to s_a. We claim that this can be extended to a feasible routing. Consider a source s_a located at node u_i. For its assigned w_{ij} we know that $t_a \in W_{ij}$. Hence there is a path from w_{ij} to t_a not using edges leading to nonterminals in C. For each i', if an edge $u_{i'} w_{i'j'}$ was selected where $w_{i'j'}$ is an i'-easy node, it must be the case that all terminal nodes reachable from $w_{i'j'}$ are also reachable from $u_{i'}$. Hence without loss of generality we can assume that the path from s_a to t_a doesn't use the edge $u_{i'} w_{i'j'}$. This holds for any $w_{i'j'}$ and any s_a, hence a feasible routing exists.

Given a set C, the matching can be computed in $O(k^5)$ time. 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, which is $O((ek)^k)$, as proved in Appendix F. We enumerate all possible sets C, and for each set check if it can be extended to a feasible routing. Therefore:

Theorem 2.6 FEDP *can be solved in time $O(mk + k^5\,(ek)^k)$.*

In Appendix C we show how to improve the 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. The general case can be improved similarly. We also give two heuristics for pruning the search tree.

We conclude with a simple fact.

Lemma 2.7 *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 routing. 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. □

3 Extensions of first-edge-disjoint paths

3.1 Second-node-disjoint paths

Let's restate the problem. There is a graph G (directed or undirected). There are k pairs of terminals $s_1t_1 \dots\ s_kt_k$ (not necessarily distinct). As before, a *routing* is defined a k-tuple of paths from each source s_i to its sink t_i. Say a routing is *second-node-disjoint* if in each path no first node (after the source) is shared with any other path in the routing. The second-node-disjoint paths problem (SNDP) is to determine whether such a routing exists.

In Appendix G we show that:

Lemma 3.1 SNDP *is equivalent to the first-*finite-*edge-disjoint paths problem, a generalization of* FEDP *that allows infinite capacities on edges.*

Consider a simple special case of SNDP when all sources sit in node s, all sinks sit in another node t (call this problem st-SNDP). We'll show that st-SNDP on undirected graphs is NP-complete and fixed-parameter tractable. We'll reduce to and from the k-leaf spanning tree problem (given an undirected graph G, does it have a spanning tree with at least k vertices?). This problem is known to be NP-complete [8] and fixed-parameter tractable [3] with an algorithm that runs in time $O(n + (2k)^{4k})$.

Note that if a graph has a subgraph T with at least k leaves, then T can be extended to a spanning tree with at least this many leaves: first break cycles in T, then add remaining vertices one-by-one.

Lemma 3.2 *st-SNDP on undirected graphs is NP-complete.*
Proof: Let (G, k) be an instance of the k-leaf spanning tree problem, with nodes $v_1 \dots v_n$. Let G_i be G with added node s and edges sv, $v \neq v_i$. Consider G_i as an instance of st-SNDP, with $t = v_i$.

Suppose G has a spanning tree T with k leaves. Then for any v_i that is not a leaf of T, there are k second-node-disjoint sv_i paths (using the leaves of T as the "second nodes"). Conversely, if for at least one i there are k second-node-disjoint sv_i paths, their union minus s is a subgraph of G with (at least) k leaves, which can be extended to a $(k+1)$-leaf spanning tree. Therefore to solve (G, k) it suffices to solve n instances of st-SNDP. □

Lemma 3.3 *st-SNDP on undirected graphs is fixed-parameter tractable.*
Proof: Let G be an (undirected) instance G of st-SNDP such that there is no edge st (else the problem is trivial). We can assume that for each non-terminal node v there is an edge sv: else, we can wlog take a transitive closure over v (that is, if there are edges uv and vw, add edge uw), and remove v from the graph. Reduction is as follows: delete s, add a dummy node d and an edge td. Call this graph G'. Claim: G' has a spanning tree with at least $k+1$ leaves iff G has a second-node-disjoint routing.

Note that, in any spanning tree on G', d is a leaf, and t is not. Therefore, if G' has a spanning tree with at least $k+1$ leaves then there exists a second-node-disjoint routing that uses these leaves (except d) as "second nodes". Conversely, if G has k second-node-disjoint st-paths, let U be the union of these paths, delete s from U, add node d and edge td. Then the resulting graph T is a subgraph of G' with at least $k+1$ leaves: k "second nodes" and d. T can be extended to a $(k+1)$-leaf spanning tree on G'. □

Since for st-SNDP undirected edge is the same as bi-directed, st-SNDP on directed graphs is NP-complete as well. We don't know whether it is fixed-parameter tractable.

It is an open question whether SNDP is fixed-parameter tractable, even on undirected graphs. We believe that under some reasonable conditions if G is large enough with respect to k then it is a positive instance of SNDP. More precisely, we believe that for any negative instance of SNDP one of the following holds:

- we can reduce it to a smaller instance.
- we can prove it negative, using some simple (polynomial-time) checks.
- its size is bounded by some function of k.

3.2 Connection to satisfiability of boolean formulas in CNF

Consider the construction in Lemma 2.1. In this construction, each source that corresponds to a variable has two possible routes to its sink (corresponding to "true" or "false"). Suppose we allow more than two routes. Then we get a simple generalization of SAT in CNF form where variables are allowed to have more than two values, and each clause is a logical "or" of terms $x \neq v$ where x is a variable and v is a possible value of x. We denote this problem MSAT. Note that MSAT can also be seen as a simple special case of the Constraint Satisfaction Problem [22].

The following lemma shows that MSAT is harder than SAT.

Lemma 3.4 MSAT *is $W[1]$-hard with respect to the number of variables, and NP-complete even if each clause contains at most two variables.*

Proof: For NP-completeness we reduce from an instance G of 3-Coloring to an instance F of MSAT. Each variable in F corresponds to a node in G and has 3 values, one value for each color. Each edge xy of G is expressed by three clauses in F: $(x \neq c \vee y \neq c)$ for each color c.

For $W[1]$-hardness we reduce from an instance G of Independent Set to an instance F of MSAT. There are k variables in F with n values each, representing a set of k vertices of G. Clauses on these variables ensure that the corresponding vertices are distinct and independent. Specifically, for each pair xy of distinct variables, clauses $(x \neq v \vee y \neq v)$ for each value v ensure distinctness, and clauses $(x \neq u \vee y \neq v)$ for each edge uv of G ensure independence. □

In this section let n be the number of variables, m be the number of clauses. The following algorithm is useful for small m.

Lemma 3.5 MSAT *can be solved in time $O(m^2 n\, 2^m)$ and space $O(2^m)$.*

Proof: Let F be an instance of MSAT on variables $x_1 \dots x_n$. Let $\mathcal{C}$ be the set of (all) clauses of F. Let $\mathcal{B}_i$ be a collection of all subsets S of $\mathcal{C}$ such that there exists a solution to MSAT on S restricted to $x_1 \dots x_i$. Then F has a solution iff $\mathcal{C} \in \mathcal{B}_n$. To compute $\mathcal{B}_n$ we can use dynamic programming with subproblems $\mathcal{B}_i$. Specifically, each $\mathcal{B}_i$ can be computed from $\mathcal{B}_{i-1}$ in time $O(m^2\, 2^m)$. □

Since MSAT is $W[1]$-hard in n we cannot hope to solve MSAT in time exponential in n only. Therefore we consider algorithms with running times of the form $O(c^k)$ where $k = m+n$. Our goal is to bring down the constant c. Note that k is the number of paths in the FEDP construction that produces MSAT.

Lemma 3.6 MSAT *can be solved in time $O((\frac{k}{n})^n)$, which is $O(1.445^k)$ in the worst case.*
Proof: The algorithm: pick any variable x, try all possible values of x, for each choice delete all satisfied clauses and recurse. If for some value of some variable there are no remaining clauses that use it, set this variable to this value and delete all satisfied clauses.

Let α_i be the number of choices at stage i, i.e. the number of possible of values of the variable that we picked. For each value there is a clause that uses it. Therefore, no matter what value we choose, at least $\alpha_i - 1$ clauses become satisfied. So $\sum_{i=1}^{n} \alpha_i \leq k$. So the total number of choices is $\prod_{i=1}^{n} \alpha_i \leq (\frac{k}{n})^n < 1.445^k$. □

In Appendix E we improve this algorithm as follows.

Theorem 3.7 MSAT *can be solved in time $O(1.26^k)$.*

4 Bottleneck-edge-disjoint paths and unsplittable flow

As in sections 2 and 3, we are given a directed graph $G = (V, E)$ and k pairs of terminals $s_i t_i$. In addition, each edge e has capacity $c_e > 0$ and every terminal pair $s_i t_i$ has *demand* d_i. We want to find a routing, i.e. a k-tuple of $s_i t_i$ paths satisfying certain conditions. Recall that the bottleneck capacity $c(p)$ of a path p is the capacity of its lowest capacity edge and that edges of p whose capacity is equal to $c(p)$ are the bottleneck edges of p.

In the UNSPLITTABLE FLOW problem, we require the routing be such that for every edge $e \in E$, the sum of demands of all terminal pairs whose path uses the edge e is at most c_e. The bottleneck edge disjoint paths problem (BEDP) is to find a routing so that (1) the bottleneck capacity of each $s_i t_i$ path is at least d_i and (2) no bottleneck edge of any path is used by any other path.

First we consider a very restricted special case of both BEDP and UNSPLITTABLE FLOW on acyclic graphs. Suppose all capacities and demands are either 1 (low) or $k+1$ (high) and that all edges leaving any s_i with demand $d_i = 1$ have capacity 1. It is easy to verify that any routing that is a valid unsplittable flow is also bottleneck disjoint and vice versa.

Theorem 4.1 *restricted-*BEDP *on directed acyclic graphs is $W[1]$-hard.*
Proof: We define a fixed-parameter reduction from k-CLIQUE to restricted-BEDP. Let (G, k) be the instance of k-CLIQUE, where $k \in \mathbb{N}$ and G is an undirected graph. Let $V(G) = \{1 \dots n\}$. We construct an equivalent instance (G', k') of restricted-BEDP, where $k' = \binom{k}{2} + 2$.

First we describe the part of the construction that depends only on n and k. We create n gadgets, one for each node in the graph G. There will be terminal pairs (st), $(s't')$ and $\binom{k}{2}$ terminal pairs $(s''t'')$. All high-capacity edges of G' are shown in Fig. 2 as solid lines. The only terminal pair with high demand is (st).

Each gadget G_i has an upper and lower part, both simple paths of high-capacity edges, of $2k+2$ and 5 edges, resp. For each G_i there are k parallel low-capacity edges $(s'' w_i^3)$ and $(w_i^2 t'')$.

Any high-capacity st path uses each edge $a_i b_i$ and either all of the lower part or all of the upper part of each gadget G_i. In other words, in each gadget the st path has to make a binary choice. The idea is that choosing the upper part of G_i corresponds to selecting node i to the k-clique. We will add a terminal pair $s't'$ and some low-capacity edges to make sure that *at most* k nodes are selected (Fig. 3):

- edges $(s'\, u_1^0)$ and $(v_n^k\, t')$.
- edges $(v_i^j\, u_{i+1}^j)$, where $1 \leq i \leq n-1$ and $0 \leq j \leq k$.

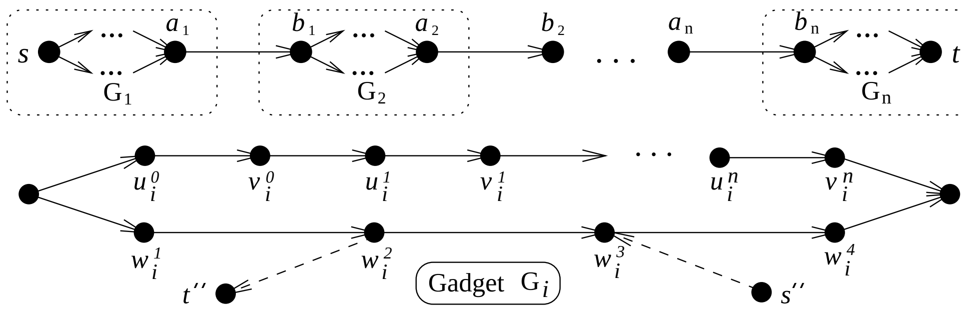


Figure 2: high-capacity edges.

- edges $(v_i^j\, u_{i+2}^{j+1})$, where $1 \le i \le n-2$ and $0 \le j \le k-1$.

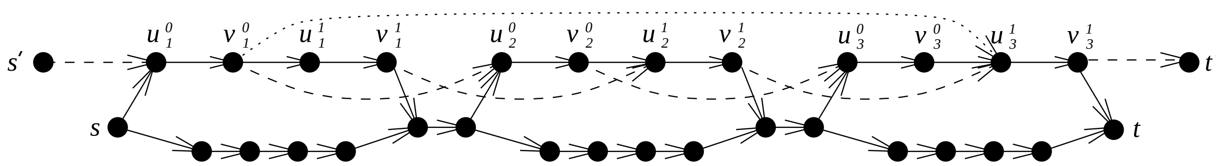


Figure 3: some low-capacity edges for $k = 2$, $n = 3$.

The $s't'$ path p' goes through the upper parts of the gadgets, maybe skipping some by choosing edges $(v_i^j\, u_{i+1}^{j+1})$. The idea is that at most k gadgets can be skipped in this fashion. Why? p' changes gadgets only through nodes v_i^j. Say in v_i^j path p' is *at level* j. Then each time p' skips a gadget, it goes one level up, and there are only $k+1$ levels.

Note that the st path p cannot use the upper part of a gadget G_i if p' path uses some of it, too. Therefore, p can choose the upper parts in at most k gadgets.

Finally we come to the part of the construction that encodes the edges of G. For each edge ij of G such that $i < j$, add a low-capacity edge $e_{ij} = (w_i^4\, w_j^1)$. This completes our construction.

Suppose G' is a positive instance of restricted-BEDP. Let S be the set of i such that the st path chose the upper part of G_i. We need to prove that S is a k-clique in G.

Since all edges $a_i b_i$ are used by the st path, any $s''t''$ path must use some edge e_{ij} such that $i, j \in S$. Since no two $s''t''$ paths can use the same edge e_{ij}, at least $\binom{k}{2}$ such edges must exist. Since $|S| \le k$ for each $i, j \in S$ such that $i < j$ there exists an edge e_{ij}, hence an edge ij in G. Therefore, S is a k-clique in G.

Conversely, given a k-clique S in G here is a routing that satisfies restricted-BEDP.

- the st path chooses the upper parts of G_i iff $i \in S$.
- the $s't'$ path skips all G_i's a.t. $i \in S$.
- for each $i, j \in S$, $i < j$, route an $s''t''$ path via e_{ij}.

Therefore, our construction is a reduction. It is fixed-parameter because there are only $\binom{k}{2} + 2$ sources in G'. □

Corollary 4.2 UNSPLITTABLE FLOW *and* BEDP *are* $W[1]$*-hard on directed acyclic graphs.*

Theorem 4.3 UNSPLITTABLE FLOW *on directed acyclic graphs can be solved in time $O(knm^k)$.*
Proof: The idea is from Fortune, Hopcroft and Wyllie [6].

Denote capacities by c_e, demands by d_i. For each source s_i add a node s'_i and an infinite-capacity edge $s'_i s_i$. Define a *pebbling game* as follows. There are k pebbles $p_1 \dots p_k$. Pebbles can be placed on edges. Initially each pebble p_i is on edge $s'_i s_i$. 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 c_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.

If p_i reaches t_i it is removed. The game is won iff all pebbles are removed. It is easy to see that the pebbling game has a winning strategy iff there is a solution for UNSPLITTABLE FLOW (paths in UNSPLITTABLE FLOW correspond to trajectories of pebbles). The crucial observation is that if some pebbles go through an edge e then at some moment all these pebbles sit 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. Each pebble can be on either of the m edges, or removed, so there are $(m+1)^k$ possible configurations. Since there are at most kn legal moves from each configuration, G_{state} contains at most $kn(m+1)^k$ edges. The algorithm is to search G_{state} to determine whether the winning configuration is reachable from the starting one. □

A modification of this algorithm works also for BEDP. However, the state space of the pebbling game will be much larger. Let C be the set of possible edge capacities, and let $d = |C|$ (note that $d \leq m$). Initially, each pebble chooses its weight $w_i \geq d_i$, $w_i \in C$. If a pebble p_i at any time gets to an edge with capacity equal to w_i, it is *marked*. The capacity constraint requires that at any time, if a pebble is on an edge with capacity equal to the weight of that edge, no other pebble can be on this edge at this time. A pebble can be removed only if it reaches its sink and is marked. BEDP has a solution if and only if there is a choice of w_i for every i and a sequence of moves that removes all pebbles. Hence:

Theorem 4.4 BEDP *on directed acyclic graphs can be solved in time $O(kn(2md)^k)$, where d is the number of distinct capacities.*

References

[1] C. Albuquerque, B.J. Vickers and T. Suda, "Network Border Patrol," *INFOCOM(1) 2000.*

[2] D.M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, **17**(1989) pp. 1-14.

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

[4] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," (1999) to appear in *IEEE/ACM Transactions on Networking.*

[5] S. Floyd, M. Handley, J. Padhye and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," *SIGCOMM 2000.*

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

[7] A. Frank, "Packing paths, circuits, and cuts - a survey," in: *Paths, Flows and VLSI-Layouts*, B. Korte, L. Lovász, H-J. Prömel, A. Schrijver, eds., Springer-Verlag (1990) 47-100.

[8] M.R. Garey and D.S.Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco (1979).

[9] V. Jacobson, "Congestion avoidance and control", *Proc. ACM SIGCOMM 1988.*

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

[11] R.M. Karp, E. Koutsoupias, C.H. Papadimitriou and S. Shenker, "Algorithmic problems in congestion control," *Proc. 41th IEEE FOCS*, 2000.

[12] F. Kelly, A. Maulloo, and D. Tan. "Rate control for communication networks: shadow prices, proportional fairness and stability." *J. Operational Research Society*, **49**(3) (1998), pp. 237-52.

[13] J. Kleinberg, "Single-source unsplittable flow," *Proc. 37th IEEE FOCS*, 1996.

[14] J. Kleinberg, "Decision algorithms for unsplittable flow and the half-disjoint paths problem," *Proc. 30th ACM Symp. on Theory of Computing*, 1998.

[15] J. Kleinberg, "Approximation Algorithms for Disjoint Paths Problems," *Ph.D. Thesis*, Dept. of EECS, MIT, 1996.

[16] J. Kleinberg, Y.Rabani, E.Tardos, "Fairness in routing and load balancing," *Proc. 40th IEEE FOCS*, 1999.

[17] S.G. Kolliopoulos and C. Stein, "Improved approximation algorithms for unsplittable flow problems," *Proc. 38h IEEE FOCS,* 1997.

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

[19] B. P. Lee, R. K. Balan, L. Jacob, W. K. G. Seah and A. L. Ananda, "TCP Tunnels: Avoiding Congestion Collapse," *LCN 2000.*

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

[21] Y. Shiloach, "A polynomial solution to the undirected two paths problem," *J. ACM*, **27** (1980) pp. 445-456.

[22] U. Schöning, "A probabilistic algorithm for k-SAT and Constraint Satisfaction Problems," *Proc. 40th IEEE FOCS,* 1999.

[23] A. Srinivasan, "Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems," *Proc. 38h IEEE FOCS,* 1997.

Appendix A: Motivating SNDP

We say that a source s_i is *well-behaved* if it does not increase its sending rate when it is greater than the receiving rate of sink t_i. Otherwise we call a source *misbehaved.*

The first-edge disjoint path model was motivated by the assumption that every edge has a capacity that is small enough for the edge to be flooded by a misbehaved source. Instead, we could imagine a network with high capacity links, where the bottleneck is the speed at which nodes can process and forward packets. In this model, a misbehaved source can flood the first node on its path to the corresponding sink. Such flooded node becomes unusable for any other path. Thus the *second-node disjoint path* (SNDP) problem is, given a graph G and a set of $s_i t_i$ pairs to find a path p_i from each source s_i to its sink t_i so that for every i, the second node on p_i (the one after s_i) is not used by any other path p_j.

Appendix B: Motivating BEDP

Let the *bottleneck capacity* of a path p be the minimum of capacities of its edges. Call edges of p with this capacity *bottleneck edges.*

A reasonable congestion control protocol may require that whenever a well-behaved source s_i sees that the receiving rate of its sink t_i is smaller that the sending rate of s_i, it decreases its sending rate. Suppose a source s_i misbehaves. Let c_b be the bottleneck capacity of the $s_i t_i$ path p_i. Since it takes time to resend packets, it is detrimental for s_i to send at rate higher than c_b.[1] If s_i keeps sending at rate c_b, all well-behaved sources whose path passes through any bottleneck edge of p_i must drop their sending rates to 0.

This motivates the following problem. Given a directed graph G, k pairs of terminals, and a requirement d_i for every $s_i t_i$ pair, find a path p_i from each s_i to t_i, so that the bottleneck capacity of p_i is at least d_i, and no other path uses any bottleneck edge of p_i. We call this the bottleneck edge disjoint paths (BEDP) problem.

Appendix C: Distinct-FEDP on acyclic graphs

We can speed up our algorithms for FEDP if the input graph is acyclic. Here we'll only consider the case when no two sources are located at the same node. The general case can be improved similarly.

Suppose the input graph G is acyclic. Then the reduced graph G_R is acyclic, too. Without loss of generality (wlog) we can rearrange s_i's so that if there is an $s_i s_j$ path then $i > j$. Sanity check: 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 wlog if the $s_i t_i$ path does not use w_{ij}.

Algorithm C.1 *Simple algorithm for* DISTINCT-FEDP *on acyclic graphs.*

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 wlog collapse all w_{1j} s.t. there is no edge $w_{1j} t_1$. Now once we know that the $s_1 t_1$

[1] s_i can find the bottleneck capacity by small additive increases of the sending rate: if c_b is not reached yet then after some time Δt all competing sources drop their sending rates s.t. the receiving rate of s_i reaches the sending rate.

path is routed through a specific edge $s_1 w$, we can wlog 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. □

Theorem C.2 DISTINCT-FEDP *on acyclic graphs can be solved in time $O(mk + (k+1)!)$*

Now we will define two heuristics that help prune the search tree.

First heuristic. Wlog delete an edge $s_1 t_i$, $i > 1$, if there is no $s_i s_1$ path.

In particular, let r_i be the smallest number r s.t. there is an edge $w_{rj} s_i$. Then we can wlog delete edges $s_1 t_i$ s.t. $1 < i < r_i$. This observation improves the bound on α_i as follows.

Lemma C.3 $\alpha_r \leq k - r - \alpha_i + 3$ *for each* $r = r_i$.

Proof: Fix i s.t. $\alpha_i > 1$. Let $r = r_i$, $w = w_{rj}$ s.t. there is an edge $w_{rj} s_i$. On stage i of the algorithm there are α_i non-terminals. Each of them is i-hard, therefore has an edge to a (distinct) sink $t_{i'}$ s.t. $i' \geq r$. Let T' be the set of these α_i sinks. After each of the α_i choices the subsequent reduction contracts all but one non-terminals w_{ij}, thus, in particular, adds edges wt for sinks in T but one, say $t \in T'$.

On stage r, each r-hard non-terminal must have an edge to a distinct "witness" sink. Then none of the sinks in $T' - t$ can be a witness sink for an i-hard $w_{rj} \neq w$. There are $\alpha_i - 1$ elements in $T - t$, hence at least $\alpha_i - 2$ elements in $T - t - t_r$. Thus, there are at most $k - r - (\alpha_i - 2)$ possible witnesses for r-hard $w_{rj} \neq w$. □

Second heuristic. Consider an edge $e = (w_{1j} t_i)$ to a 1-blockable sink t_i, $i \neq 1$. What difference can e possibly make? Obviously, e makes a difference only if edge $s_1 w_{1j}$ is not blocked, in which case for each edge ws_1 an edge wt_i gets added. However, adding edge wt_i is sometimes wlog, for example when it already existed, or when $w = w_{i'j'}$, $i' \neq i$ and t_i was i'-accessible. Therefore we can wlog delete e unless there is an edge ws_1, $w = w_{i'j'}$, s.t.

- there is no edge wt_i,
- (if $i' \neq i$ or there are no other two edge paths $s_i t_i$) there are edges entering $s_{i'}$
- (if $i' \neq i$) t_i is not i'-accessible

Special case. Suppose for each i there is exactly one i' s.t. there is an edge $w_{i'j} s_i$. Then in addition to Lemma C.3 the second heuristic gives a similar but sometimes stronger bound on α_i as follows: for each $r < k$

$$\alpha_r \leq k - r - \frac{1}{2} \sum_{r_i = r} (\alpha_i - 3)$$

Appendix D: Converting an FEDP instance to the normal form

Start with an instance of FEDP: directed graph G and k terminal pairs $s_1t_1 \dots s_kt_k$. We will show how to convert it to the normal form in time and space $O(mk)$, where m is the number of edges.

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. By splitting edges uv and $u'v$ we mean: delete them, create a new vertex v' with the same edges as v, and add edges uv and $u'v'$.

Reduction D.1 *Convert* FEDP *instance to the normal form.*

1. make sure S and T are disjoint: if $s_i = t_{i'}$ replace s_i by an edge.
2. make sure there are no edges s_iv, v terminal: else, replace v by an edge uv.
3. make sure there are no edges s_iv, $s_{i'}v$: else, split them.
4. take a transitive closure over non-blockable edges
5. restrict it to important nodes.
6. delete all edges between non-terminals and all edges leaving T.

Let R_v be the set of terminals reachable from v via non-blockable edges. In Red. D.1 steps 1-3 and 6 are trivial, and for steps 4-5 it suffices to compute R_v for all nodes v. This can be done by $2k$ depth-first searches, in total time $O(mk)$.

Let G_1 be the result of Red. D.1. G_1 can have more nodes than G because some vertices in G may be split. By keeping track of which vertices were split, we can extend R to $V(G_1)$ for free. There are at most m blockable edges in the input graph G, so there can at most m non-terminals in G_1. Thus, the size of R is now $O(mk)$.

We use R instead of adjacency matrix for G_1. Thus we can reduce the graph, for both DISTINCT-FEDP and (general) FEDP, in time $O(mk)$.

Appendix E: Solving MSAT in time $O(1.26^k)$

Without loss of generality the domains of the variables are pairwise disjoint. Say a value is *lonely* if it is used by just one clause. Say a variable is lonely if all its values are lonely. Say a clause is lonely if all values used by it are lonely.

The following reductions can be applied:

- if some clause C uses just one value v, delete v and all clauses that use it.
- if some variable x has just one value v, delete x and v.
- if a clause C is lonely, delete all values used by C, merge all variables in C into one, delete C.
- if a variable x is lonely, merge all clauses that use x into one, delete x.

To *resolve* a variable x means try each possible value of x, for each value delete all satisfied clauses, delete x and recurse. The recursive step of the algorithm is as follows:

1. If some variable has no values left, output "NO" and halt. If there are no more clauses, output "YES" and halt. If any of the reductions above applies, apply it and recurse.

2. If there is a variable v without lonely values, resolve it. Say v has α values. Then we make α choices and on each choice delete one variable and at least $2(\alpha - 1)$ clauses.

3. Else, pick any variable x. It has a lonely value v used by only one clause C. There exists another variable x' with a non-lonely value v' that also used by C. We resolve x'. There are two cases:

   (3a) if we set x' to v' then we also resolve x.
   (3b) else, clause C is satisfied by x', so we can set x to v and delete clauses satisfied by x.

Let $f(k)$ be the worst-case running time of this algorithm for an instance with $m+n=k$. Then in case 2 the running time is at worst

$$(\alpha - 1)\, f(k - 2\alpha + 1) \tag{1}$$

Let's analyze case 3. Say x has α values, x' has α' values. Since x is not lonely, it must have at least one non-lonely value, say v''.

(3a) we make α choices and after each choice delete at least $\alpha' - 1$ clauses satisfied by x' and at least $\alpha - 1$ clauses satisfied by x (actually, after all choices but $x = v''$ we delete at least α clauses satisfied by x).

(3b) we make $\alpha' - 1$ choices and delete at least α clauses satisfied by x (since value v'' is not lonely) and at least α' clauses satisfied by x' (since value v' is not lonely).

Thus, the running time is at worst

$$f(k - \alpha - \alpha') \;+\; (\alpha - 1)\, f(k - \alpha - \alpha' - 1) \;+\; (\alpha' - 1)\, f(k - \alpha - \alpha' - 2) \tag{2}$$

The constraints are that α and α' are at least 2.

Using (1) and (2), one can prove that case 3 never increases the worst-case running time, so that $f(k) = c^k$, where $c = 2^{1/3} \approx 1.26$.

Appendix F: One long calculation

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

$$\prod_{i=1}^{l} \sum_{j=0}^{k_i} \binom{x_i}{j} = O((ek)^k) \tag{3}$$

Proof: For each $j \le k_i$

$$\begin{aligned}
\binom{x_i}{j-1} &\le \binom{x_i}{j} \frac{1}{k+l} \\
\binom{x_i}{j} &\le \binom{x_i}{k_i} \left(\frac{1}{k+l}\right)^{k_i - j} \\
\sum_{j=0}^{k_i} \binom{x_i}{j} &\le \binom{x_i}{k_i} \frac{1}{1-\frac{1}{k+l}} \\
&\le \binom{x_i}{k_i} \left(1 + \frac{1}{k}\right) \\
\text{LHS of (3)} &\le e \prod_{i=1}^{l} \binom{x_i}{k_i}
\end{aligned}$$

If $k_i = 1$ then $\binom{x_i}{k_i} = x_i < 2k$, which only helps us. So wlog $k_i \ge 2$ for all i.

$$\begin{aligned}
\binom{a}{b} &< \frac{a^b}{b!} < \frac{a^b}{b^b e^{-b} \sqrt{2\pi b}} \\
\binom{x_i}{k_i} &< \frac{[e(k+l)]^{k_i}}{\sqrt{2\pi k_i}} \\
\prod_{i=1}^{l} \binom{x_i}{k_i} &< \frac{[e(k+l)]^k}{(4\pi)^{-l/2}} \\
&= (ek)^k f(\alpha),
\end{aligned}$$

where $l = \alpha k$, $\gamma = \sqrt{4\pi}$ and $f(\alpha) = \frac{1+\alpha}{\gamma^\alpha}$. It remains to prove that $f(\alpha) \le 1$ for all $\alpha \in [0,1]$. This is true because $f(0) = 1$ and for all $\alpha \in [0,1]$

$$f'(\alpha) = \frac{1 - (1+\alpha)\ln\gamma}{\gamma^\alpha} < 0$$

□

Appendix G: SNDP is equivalent to first-finite-edge-disjoint paths

Consider first-*finite*-edge-disjoint paths problem (FEDP$_\infty$), a generalization of FEDP that allows infinite capacities. This problem asks whether there exists a first-*finite*-edge-disjoint routing. Suppose we are given an instance G of FEDP$_\infty$ with k terminal pairs $s_i t_i$. We will show how to convert it to an equivalent instance of SNDP.

Let S be the set of source nodes, T be the set of sink nodes. Say an edge uv is *blockable* if there is an infinite-capacity path $s_i u$ from some source s_i. Let B_h, B_t be the sets of heads and tails of blockable edges, resp. Similarly to Appendix D we can make sure that

- S, T, B_h and B_t are pairwise disjoint
- endpoints of any two blockable edges are disjoint.

Let E' be the set of non-blockable edges. Let G' be a graph with sources, sinks and blockable edges. Take a transitive closure in E' as follows: for each blockable edge uv

- if there is a vr path in E', $r \in T \cup E_t$, add an infinite-capacity edge vr.
- if there is a ru path in E', $r \in S \cup E_h$, add an infinite-capacity edge ru.

Then FEDP$_\infty$ holds for G iff it holds for G'. Now contract all blockable edges in G' to get an equivalent instance of SNDP.

Conversely, given an instance G of SNDP, we can convert it to an equivalent instance of FEDP$_\infty$:

- make sure S and T are disjoint.
- delete all edges between sources and all edges of the form $s_i t_{i'}$, $i' \neq i$.
- make sure each non-terminal node in G has an edge from a source node.
- assign each edge of G infinite capacity.
- replace each node w of G which is not a source node by a finite-capacity edge uv such that
    - each edge rw is replaced by edge ru;
    - each edge wr is replaced by edge vr;
    - if sink t_i was located at w, place it at v.

Let G' be the resulting graph. Then SNDP holds for G iff FEDP$_\infty$ holds for G'.