Parametrisierte Komplexität des Min-Multicut Problems (Parameterized Complexity of the Min-Multicut Problem)

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch Westfälischen Technischen Hochschule Aachen

vorgelegt von

Name: Engelmann Vorname: Viktor

Geb. am: 7. September 1981 in: Köln

Erstprüfer: Prof. Dr. Rossmanith

Geilenkirchen, den 22. Juni 2010

Contents

List of Figures

Chapter 1

Introduction

The famous Max-Flow = Min-Cut theorem by Ford and Fulkerson is a fundamental result for network-theory and shows that the Min-Cut problem is solvable in polynomial time. Ford and Fulkerson also showed that, when all capacities are integral, there is a maximum flow that assigns integral flows to all edges. Ordinary (single-commodity) networks are already very versatile and can be used to solve many practical problems. multi-commodity networks have multiple source/drain pairs $\mathcal{P} = \{\{s_1, t_1\}, \cdots, \{s_l, t_l\}\}$ and the corresponding problems are Max-Multiflow, Max-Integer-Multiflow and Min-Multicut, which ask for a maximum *sum* of (integral) flows between the respective pairs and a minimum set of edges whose removal disconnects all sources from their respective drains.

Chapter 1 introduces the fundamental terms and techniques that are used. Chapter 2 introduces several problems for multi-commodity networks. Chapter 3 will grade Min-Multicut and Min-Multicut on several special graph classes in terms of classical complexity classes (P, NPC, APX, MaxSNP, PTAS, FPTAS). Chapter 4 discusses the fixed parameter tractability and intractability of Min-Multicut in several parameters. The results are sometimes tangent to the related problems from chapter 2, but the main focus is on Min-Multicut. The most important open question is the fixed parameter tractability in the parameter k (the allowed value of the multicut). Chapter 5 looks at the fixed parameter tractability of Min-Multicut in the most important parameters, when restricted to special graph classes. Chapter 6 examines many reduction rules w.r.t. k and possible generalizations of reduction rules for trees. Chapter 7 takes a look at possible approaches to developing an algorithm that is FPT in k or a W[1]-hardness proof and discusses several obstacles that one comes across on the quest for one of these results (like several plausible interrelations that one might

conjecture, but that are untrue). The bibliography is a treasure chest on its own, because it contains a huge collection of important works on the topic and many famous, fundamental works.

Since MIN-MULTICUT is a quite prominent problem, it has already been analyzed deeply and thus many properties have already been found. So apparently there aren't many results left to be found without going beyond the scope of a degree dissertation. All the more the writer is pleased to report fixed parameter tractability of MIN-MULTICUT in k on uniform grids (5.6), which is a new result.
The main goal of this elaboration is to give a broad overview over the topic and known results. To be a reference work and a starting point for further development. The focus will be on the basic ideas of the algorithms and theorems, because to spawn new ideas, it is more important to comprehend the existing ideas, than to have the full proofs in all detail (which can be found in the original works after all).
It is convenient to have the core ideas of the most important elaborations gathered in one location, because when one wades through the individual elaborations, one has to find them first and then read the same definitions, backgrounds and references over and over again. Also the notations are not always consistent (the notations here comply with the majority of elaborations) and many disquisitions are filled with details and related works, that steal focus from the core ideas.

1.1 Elementary graph theory

The most general definition of a ***graph*** is a 3-tuple $G = (V, E, f)$, containing a set of ***vertices*** V, a set of ***edges*** E and a function $f : E \to \mathfrak{P}_2(V)$. Figuratively speaking, edges constitute connections between vertices and f defines, which edge connects which vertices. Visual representations of graphs usually depict a ***vertex*** as dot and edges as lines between two of these dots. If two vertices are connected by an edge, they are ***adjacent*** and if an edge e connects a vertex v to another vertex, then v and e are ***incident***.
Often it is sufficient to define a graph as pair $G = (V, E)$ with $E \subseteq \mathfrak{P}_2(V)$. This definition does not formally allow multiple edges to connect the same pair of vertices, but it is still widely used, even in domains where graphs with these so called ***parallel edges*** can occur. Parallel edges are simply supposed to be distinguishable somehow because in most domains, a graph with parallel edges can be replaced by an equivalent[1] graph without parallel

[1] equivalence w.r.t. the respective domain

edges anyway.

$$
\begin{aligned}
&V = \{v_1, v_2, v_3, v_4, v_5, v_6\} \\
&E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \\
&f(e_1) = \{v_1, v_2\}\,, f(e_2) = \{v_1, v_3\} \\
&f(e_3) = \{v_3, v_4\},\ f(e_4) = \{v_3, v_5\} \\
&f(e_5) = \{v_2, v_4\},\ f(e_6) = \{v_4, v_6\} \\
&f(e_7) = \{v_5, v_6\}
\end{aligned}
$$

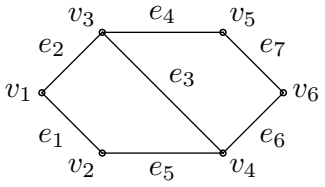


Figure 1.1: Formal and visual representation of a graph

The edges of a ***directed graph*** or ***digraph*** have a direction, they are not represented by a *set* of vertices, but by an *ordered sequence* of vertices called ***arcs*** or ***arrows***. A digraph can be represented as a 3-tuple (V, E, f) again, but here the codomain of f is $V \times V$ instead of $\mathfrak{P}_2(V)$. Again digraphs are often defined as pair (V, E) with $E \subseteq V \times V$.
Graphs without parallel edges are called ***multigraphs*** and digraphs without parallel arcs are called ***multi-digraphs***. Note that in the directed case, (x, y) and (y, x) are not considered parallel.

$$
\begin{aligned}
&V = \{v_1, v_2, v_3, v_4, v_5, v_6\} \\
&E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \\
&f(e_1) = (v_1, v_2),\ f(e_2) = (v_1, v_3) \\
&f(e_3) = (v_3, v_4),\ f(e_4) = (v_3, v_5) \\
&f(e_5) = (v_2, v_4),\ f(e_6) = (v_4, v_6) \\
&f(e_7) = (v_5, v_6)
\end{aligned}
$$

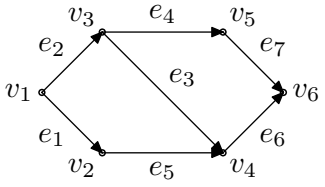


Figure 1.2: Formal and visual representation of a directed graph

A ***weighted (di)graph*** has an additional weight function $\rho : E \to \mathbb{R}$. Graphs without a weight function can usually be considered to have an implicit weight function $\rho : e \mapsto 1$ for all $e \in E$.

An ordered sequence of vertices $(v_1, v_2, \cdots, v_q)$ is called a (directed) ***walk*** from v_1 to v_q on a graph, if for all $i = 1, \cdots, q-1$ an edge $\{v_i, v_{i+1}\} \in E$ or an arc $(v_i, v_{i+1}) \in E$ exists. If the vertices are all pairwise different, then the sequence is called a (directed) ***path***. If all vertices are pairwise different, except $v_0 = v_q$, then the sequence is called a (directed) ***cycle*** or ***circle***.

A walk of length 0 (v) is not considered a cycle and a walk of length 2 (v, w, v) is only considered a cycle, if there are parallel edges or opposing arcs between v and w. A graph is ***connected***, if for all pairs of vertices v, w there is a path from v to w. A ***tree*** is an undirected, connected graph that contains no cycle. A directed graph that contains no directed cycle is called a ***DAG*** (***directed, acyclic graph***). Note that the direction plays a role in the directed case. The digraph in fig. 1.2 is a DAG, although the "same" graph without directions (fig. 1.1) is not a tree. A tree in which every edge is incident to one center vertex c is called a ***star***. Formally: $S_n = (\{c, x_1, x_2, \cdots, x_n\}, \{\{c, x_i\} | i = 1, \cdots, n\})$.

The ***neighborhood*** of a vertex v in an undirected graph is $N(v) := \{w \in V | \{v, w\} \in E\}$, the set of vertices that are adjacent to v. In directed graphs, we distinguish the ***positive neighborhood*** $N^+(v) := \{w \in V | (v, w) \in E\}$ (also called the ***successors***) and the ***negative neighborhood*** $N^-(v) := \{w \in V | (w, v) \in E\}$ (also called the ***predecessors***) of a vertex.

The (inbound / outbound) ***degree*** of a vertex x on a weighted (di)graph is defined as

$$d(v) := \sum_{w \in N(v)} \rho(\{w, v\}) \qquad \underbrace{= |N(v)|}_{\text{In unweighted multigraphs}}$$

$$d^+(v) := \sum_{y \in N^+(x)} \rho((x, y)) \qquad \underbrace{= |N^+(v)|}_{\text{In unweighted multi-digraphs}}$$

$$d^-(v) := \sum_{y \in N^-(x)} \rho((y, x)) \qquad \underbrace{= |N^-(v)|}_{\text{In unweighted multi-digraphs}}$$

The minimum and maximum (inbound / outbound) degree of a (di)graph G are

$$\begin{array}{rcl|rcl} \delta(G) & := & min\{d(x) | x \in V\} & \Delta(G) & := & max\{d(x) | x \in V\} \\ \delta^+(G) & := & min\{d^+(x) | x \in V\} & \Delta^+(G) & := & max\{d^+(x) | x \in V\} \\ \delta^-(G) & := & min\{d^-(x) | x \in V\} & \Delta^-(G) & := & max\{d^-(x) | x \in V\} \end{array}$$

A graph that can be drawn on a 2-dimensional plane such that no edges overlap, is called a ***planar graph***. The sets of points in the plane, that are reachable from each other without crossing an edge in a given drawing, are called ***countries***. The area that surrounds the drawn graph is also considered a country and is called the ***outer region***. The ***linegraph*** [91] $\mathcal{L}(G)$ of a graph $G = (V, E)$ is a graph (V', E') which has a vertex for every edge in G (formally this can be expressed as $V' = E$) and two of these verticess are adjacent in $\mathcal{L}(G)$ if the corresponding edges are incident

in G (formally: $E' = \{\{e, e'\} \in \mathfrak{P}_2(E) | e \cap e' \neq \emptyset\}$). The linegraph of a directed graph has an arc (e, e') if $e = (x, y), e' = (y, z)$ for some $y \in V$. If $V' \subseteq V$, then the ***induced subgraph*** of V' is $G[V'] := (V', E')$ with $E' = \{\{u, v\} \in E | u \in V', v \in V'\} = E \cap \mathfrak{P}_2(V')$.

1.2 Complexity theory in a nutshell

A ***(deterministic) Turing machine (DTM)***, named after Alan Turing [90], is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, \textvisiblespace, \overline{q})$ containing a set of ***states*** Q, an input-***alphabet*** Σ, a working-alphabet $\Gamma \supset \Sigma$, a transition function $\delta : (Q \backslash \{\overline{q}\}) \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$, a ***start-state*** $q_0 \in Q$, a selected symbol $\textvisiblespace \in \Gamma \backslash \Sigma$ that defines a separator and an ***end-state*** $\overline{q} \in Q$. Turing machines and computer programs are fundamentally equivalent, but Turing machines are very formal models, which is why they are well suited for formal analysis of computability and complexity theory.
A ***configuration*** of a Turing machine is a 3-tuple $(w, v, q) \in \Gamma^* \times \Gamma^* \times Q$. The transition function δ defines a configuration-alteration relation $\vdash$ as follows

$$\begin{array}{rcll}
(\varepsilon, \varepsilon, q_0) & \vdash & (\varepsilon, \textvisiblespace, q_0) & \\
(w, av, q) & \vdash & (w, bv, q') & \text{if} \quad \delta(q, a) = (q', b, N) \\
(w, av, q) & \vdash & (wb, v, q') & \text{if} \quad \delta(q, a) = (q', b, R) \\
(w, a, q) & \vdash & (wb, \textvisiblespace, q') & \text{if} \quad \delta(q, a) = (q', b, R) \\
(wc, av, q) & \vdash & (w, cbv, q') & \text{if} \quad \delta(q, a) = (q', b, L) \\
(\varepsilon, av, q) & \vdash & (\varepsilon, \textvisiblespace bv, q') & \text{if} \quad \delta(q, a) = (q', b, L)
\end{array}$$

for $w, v \in \Gamma^*$, $a, b, c \in \Gamma$, $q \in Q \backslash \{\overline{q}\}$, $q' \in Q$. Using this relation, a Turing machine $\mathcal{M}$ defines a (possibly partial) function $f_{\mathcal{M}} : \Sigma^* \to (\Gamma \backslash \{\textvisiblespace\})^*$ by

$$f_{\mathcal{M}}(w) = \begin{cases} w' & \text{if } (\varepsilon, w, q_0) \vdash^* (v', w'w'', \overline{q}) \text{ with } w' \in (\Gamma \backslash \{\textvisiblespace\})^*, w'' \in (\textvisiblespace \Gamma^*)^* \\ \perp & \text{otherwise} \end{cases}$$

where $\vdash^*$ is the transitive closure of $\vdash$ and $\perp$ is the symbol for an undefined result. A Turing machine $\mathcal{M}$ ***computes*** a function $f : \Sigma^* \to (\Gamma \backslash \{\textvisiblespace\})^*$, if $f_{\mathcal{M}}(w) = f(w)$ for all $w \in \Sigma^*$. A function $f : \Sigma^* \to (\Gamma \backslash \{\textvisiblespace\})^*$ is (Turing-) ***computable*** if there exists a Turing machine that computes f. If $f_{\mathcal{M}}(w) \in \{0, 1\}$ for all $w \in \Sigma^*$, then $L(\mathcal{M}) := \{w \in \Sigma^* | f_{\mathcal{M}}(w) = 1\}$ is the ***language*** of $\mathcal{M}$ and we say $\mathcal{M}$ ***decides*** $L(\mathcal{M})$. A language $\mathcal{L} \subseteq \Sigma^*$ is (Turing-) ***decidable***, if there exists a Turing machine $\mathcal{M}$ with $L(\mathcal{M}) = \mathcal{L}$. If $f_{\mathcal{M}}(w) = \perp$ is also possible, then we say $\mathcal{M}$ ***recognizes*** $L(\mathcal{M})$, but this is only defined for completeness; all upcoming languages are decidable.

The ***running-time*** of $\mathcal{M}$ on an input w is $\tau(\varepsilon, w, q_0)$ with $\tau(v, w, \overline{q}) = 0$, $\tau(v, w, q) = 1 + \tau(v', w', q')$ for $(v, w, q) \vdash (v', w', q')$. The ***worst-case*** running-time of $\mathcal{M}$ is a function

$$t : \mathbb{N} \to \mathbb{N} \text{ with } t(n) = max\ \{\tau(\varepsilon, w, q_0) | w \in \Sigma^n\}.$$

If there exist f, a, b, n_0 such that $t(n) \leq a \cdot f(n) + b$ for all $n \geq n_0$, then we say that f is an upper bound to the running-time or the running-time of $\mathcal{M}$ is in $O(f(n))$. O is one of the Landau symbols [70]. If there exists n_0 and a polynomial p such that $t(n) \leq p(n) \cdot f(n)$ for all $n \geq n_0$, then the running-time of $\mathcal{M}$ is in $O^*(f(n))$.

A ***nondeterministic Turing machine (NTM)*** has a relation

$$\delta \subseteq ((Q \backslash \{\overline{q}\}) \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$$

instead of the transition function. A configuration (w, v, q) can now have multiple configurations (w', v', q') with $(w, v, q) \vdash (w', v', q')$

$$\begin{array}{lcll}
(\varepsilon, \varepsilon, q_0) & \vdash & (\varepsilon, \text{␣}, q_0) & \\
(w, av, q) & \vdash & (w, bv, q') & \text{if} \quad ((q, a), (q', b, N)) \in \delta \\
(w, av, q) & \vdash & (wb, v, q') & \text{if} \quad ((q, a), (q', b, R)) \in \delta \\
(w, a, q) & \vdash & (wb, \text{␣}, q') & \text{if} \quad ((q, a), (q', b, R)) \in \delta \\
(wc, av, q) & \vdash & (w, cbv, q') & \text{if} \quad ((q, a), (q', b, L)) \in \delta \\
(\varepsilon, av, q) & \vdash & (\varepsilon, \text{␣}bv, q') & \text{if} \quad ((q, a), (q', b, L)) \in \delta
\end{array}$$

An NTM cannot compute a function, since an end-configuration $(v', w', \overline{q})$ is not necessarily unique, so the result might not be well-defined. An NTM can only decide a language $L(\mathcal{M}) = \{w \in \Sigma^* | f_{\mathcal{M}}(w) = 1\}$, where

$$f_{\mathcal{M}}(w) = \begin{cases} 1 & \text{if} \quad (\varepsilon, w, q_0) \vdash^* (v', 1w', \overline{q}) \text{ for some } v' \in \Gamma^*, w' \in (\text{␣}\Gamma^*)^* \\ 0 & \text{if} \quad (v', w', q') \vdash^* (v'', 0w'', \overline{q}) \text{ for all } (w', v', q') \text{ with} \\ & \quad (\varepsilon, w, q_0) \vdash^* (w', v', q') \\ \perp & \quad \text{otherwise} \end{cases}$$

The running-time of an NTM on an input w is

$$\tau(\varepsilon, w, q_0) = \begin{cases} \tau_1(\varepsilon, w, q_0) & \text{if } f_{\mathcal{M}}(w) = 1 \\ \tau_0(\varepsilon, w, q_0) & \text{if } f_{\mathcal{M}}(w) = 0 \\ \infty & \text{otherwise} \end{cases}$$

with $\tau_1(v', 1w', \overline{q}) = \tau_0(v', 0w', \overline{q}) = 0, \tau_1(v', 0w', \overline{q}) = \infty$

- $\tau_1(v', w', q') = 1 + min\ \{\tau_1(v'', w'', q'') | (v', w', q') \vdash (v'', w'', q'')\}$

- $\tau_0(v', w', q') = 1 + max\ \{\tau_0(v'', w'', q'')|(v', w', q') \vdash (v'', w'', q'')\}$.

worst case and upper bounds are defined for NTMs like for DTMs.

A ***problem*** over an alphabet Σ is a language $\mathcal{L} \subseteq \Sigma^*$. An ***instance*** of the problem is a word $w \in \Sigma^*$ and the question is whether $w \in \mathcal{L}$. For example the problem TREE contains all encodings of connected graphs that contain no cycles. A DTM or NTM $\mathcal{M}$ ***decides*** a problem $\mathcal{L}$, if $\mathcal{L} = L(\mathcal{M})$. A problem $\mathcal{L}$ is ***polynomial*** if there is a DTM $\mathcal{M}$ that decides $\mathcal{L}$ and that has a running-time in $O(p(|w|))$ for a polynomial p. A problem $\mathcal{L}$ is ***nondeterministically polynomial***, if there is an NTM $\mathcal{M}$ that decides $\mathcal{L}$ and that has a running-time in $O(p(|w|))$ for a polynomial p.
P is the class of all polynomial problems, ***NP*** is the class of all nondeterministically polynomial problems.

Let $\mathcal{L}$ a problem over Σ, $\mathcal{L}'$ a problem over Γ. A ***P-reduction*** [20, 59] is a function $g : \Sigma^* \to \Gamma^*$ that is computable in (deterministic) polynomial time and that has $w \in \mathcal{L}$ iff $g(w) \in \mathcal{L}'$. If such a function exists, we write $\mathcal{L} \leq_P \mathcal{L}'$

Lemma 1.2.1. *[59] If $\mathcal{L} \leq_P \mathcal{L}'$ and $\mathcal{L}' \in P$, then $\mathcal{L} \in P$*

Proof. Let $w' = g(w)$. Because of the polynomial running-time of the reduction, $|w'| \leq p(|w|)$ for a polynomial p. Since $\mathcal{L}' \in P$, there is an algorithm with a running-time $O(p'(|w'|))$ for a polynomial p'. The overall running-time is $O(p(|w|) + p'(|w'|)) \subseteq O(p(|w|) + p'(p(|w|)))$, which is polynomial in $|w|$. □

Analogous it can be shown that if $\mathcal{L} \leq_P \mathcal{L}'$ and $\mathcal{L}' \in NP$, then $\mathcal{L} \in NP$.

A problem $\mathcal{L}$ is ***NP-hard***, if $\mathcal{L}' \leq_P \mathcal{L}$ for all problems $\mathcal{L}'$ in NP. A problem is ***NP-complete*** or ***NPC***, if it is NP-hard and in NP. Since $\leq_P$ is transitive, it is sufficient to show membership in NP and $\mathcal{L}' \leq_P \mathcal{L}$ for only one NP-complete problem $\mathcal{L}'$ to prove NP-completeness of $\mathcal{L}$.
In 1971, Cook [20] proved that the satisfiability problem from propositional logic SAT is NP-complete. Since then, many more problems followed [59, 46]. An important conclusion [59] from Cooks theorem is that either all NP-complete problems have a deterministic polynomial algorithm (P=NP), or none (P≠NP). The question whether P=NP or P≠NP is the most important, open question of computer science. P≠NP is widely believed and highly likely, but it is not proven yet.

1.3 Network theory

A ***network*** is a 5-tuple (V, E, s, t, c) containing a set of vertices V, a set of edges or arcs E, two selected vertices or ***terminals*** s (***source***) and $t \neq s$ (***target***, also called ***drain*** or ***sink***) and a capacity-function $c : E \to \mathbb{R}^+$.

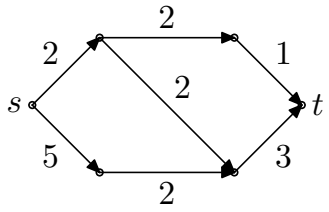


Figure 1.3: Example of a network

Networks are used to model for example logistic problems, communication channels or traffic. A vertex might stand for a transfer station, an edge between two vertices might stand for a traffic link between two cities and the capacity-function might stand for how many entities of some commodity can be transported through the traffic links in a given timespan. The source could stand for a factory, where the entities are produced and the drain could stand for a retail store, to which entities are delivered.
Many applications of network theory do not require different capacities for the edges. A network whose edges all have the same capacity (or equivalently: A network without a capacity-function and implicit capacities of 1 for all edges) is called a ***uniform network***.

1.3.1 Maximum flows

In directed networks, a ***network-flow*** is a function $f : E \to \mathbb{R}^+$ with

1. $f(e) \leq c(e)$ for all $e \in E$ (***capacity-limitation***)
2. $\underbrace{\sum_{y \in N^-(x)} f((y,x))}_{f^-(x)} = \underbrace{\sum_{y \in N^+(x)} f((x,y))}_{f^+(x)}$ for all $x \in V \backslash \{s,t\}$ (***flow-preservation***)
3. $f((x,s)) = 0$, $f((t,x)) = 0$ for all vertices $x \in V$

An edge with $f(e) = c(e)$ is called a ***saturated*** edge. In the example, a network-flow would stand for the amount of entities that are actually

transported through the traffic links. The flow-preservation says that entities are not retained and not produced at transfer stations.
The ***magnitude*** of a network-flow f is $\sum_{x \in N^+(s)} f((s,x)) = \sum_{x \in N^-(t)} f((x,t))$. This quantity equates to the amount of entities that are transported from

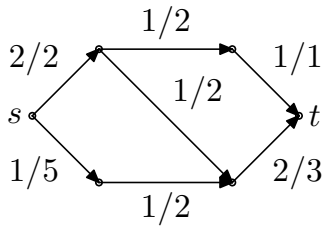


Figure 1.4: A network-flow with magnitude 3

factory s to store t. Naturally the question arises, what magnitude a network-flow can at most have, or in other words, how many entities can be transported from s to t in a given timespan. This question encourages the following definition: a network-flow is a maximum flow or a ***max-flow***, if there exists no network-flow that has a larger magnitude.
The flow in fig. 1.4 is not a maximum flow, because obviously it could be increased by increasing the flow on the lower arcs by 1, to obtain the flow in fig. 1.5.

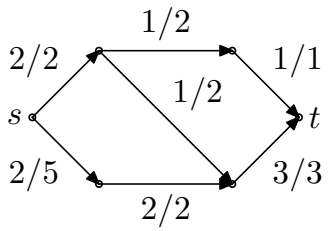


Figure 1.5: A network-flow with magnitude 4

For undirected networks, it makes no sense to map edges to numbers, because this gives no information about the direction in which the entities travel through the edge. Instead, flows on undirected networks map ordered pairs of vertices to numbers $f : V \times V \to \mathbb{R}^+$. This requires some changes in the conditions.

1. $f(x,y) + f(y,x) \leq c(\{x,y\})$ for all $\{x,y\} \in E$ (***capacity-limitation***)

2. $\sum_{y \in N(x)} f(y,x) = \sum_{y \in N(x)} f(x,y)$ for all $x \in V \backslash \{s,t\}$ (***flow-preservation***)

3. $f(x,s) = 0$, $f(t,x) = 0$ for all vertices $x \in V$

An undirected edge $\{x,y\}$ is called saturated, if $f(x,y) = c(\{x,y\})$ or $f(y,x) = c(\{x,y\})$.

Many disquisitions define $f(x,y) = -f(y,x)$ for undirected networks. They define the capacity-limitation as $|f(x,y)| + |f(y,x)| \leq 2 \cdot c(\{x,y\})$, flow-preservation as $\sum_{y \in N(x)} f(x,y) = 0$ and an edge is saturated if $|f(x,y)| = c(\{x,y\})$. This definition makes sense for single-commodity networks, but it makes it unnecessarily complicated to express the capacity-limitation in linear programs (which are needed for multi-commodity networks).
In the directed case, instead of the third condition, many disquisitions demand that s has no inbound arcs and t has no outbound arcs, but this definition already fails for undirected networks and it makes it unnecessarily complicated to model some circumstances with (directed) multi-commodity networks.

1.3.2 The Ford-Fulkerson algorithm

Let f a flow on an undirected network (V,E,s,t,c). An ***augmenting path*** is a sequence of vertices $(s = x_1, x_2, \cdots, x_q = t)$ such that $f(x_i, x_{i+1}) - f(x_{i+1}, x_i) < c(\{x, x_i\})$ for every $i = 1, \cdots, q-1$. On directed networks, an augmenting path is a sequence $(s = x_1, x_2, \cdots, x_q = t)$ such that for all $i = 1, \cdots, q-1$ one of the following holds:

- $(x_i, x_{i+1}) \in E$ and $f((x_i, x_{i+1})) < c((x_i, x_{i+}))$ (forward arc)
- $(x_{i+1}, x_i) \in E$ and $f((x_{i+1}, x_i)) > 0$ (backward arc)

If such a path exists (this can be tested in polynomial time, using Dijkstra's famous algorithm [29]), then the flow can be augmented by increasing the flow on the forward arcs and decreasing the flow on the backward arcs. Decreasing the flow on a backward arc (x_{i+1}, x_i) means that the flow coming over that arc is replaced by the flow coming over the augmenting path. The rest of the augmenting path defines a rerouting of the excessive flow in x_{i+1} (which must be rerouted to reconstitute the flow-preservation in x_{i+1}). This circumstance is illustrated in fig. 1.6

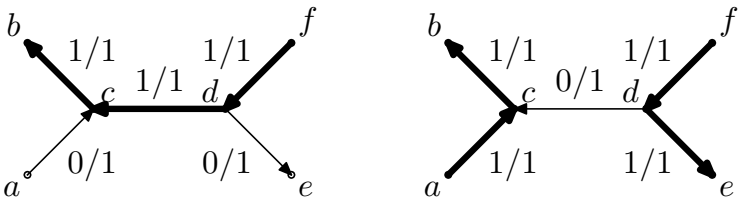


Figure 1.6: An augmenting path $(\cdots, a, c, d, e, \cdots)$ with a backward arc

Ford and Fulkerson proved [42] that a flow is a maximum flow if and only if there is no augmenting path. Therefore a maximum flow can be found using the algorithm by Ford and Fulkerson:

Initialize a flow f, such that $f(e) = 0$ for all $e \in E$
while $\exists$ augmenting path P **do**
    increase f along P
done

An interesting conclusion is that there exists a maximum flow that assigns integral values to all edges, provided all capacities are integral, because in every iteration, an integral amount is added to the flow on any edge. This means that Max-Integer-Flow = Max-Flow for such networks, which is an important property, because it might be impossible to transport entities partially through one transport link and partially through other transport links. To determine the max-integer-flow of a network with real capacities, one must only round all capacities down and determine the max-flow of the resulting network with integral capacities.

The algorithm by Ford and Fulkerson however is not polynomial, as fig. 1.7 shows.

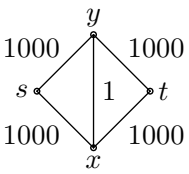


Figure 1.7: A network, where the Ford-Fulkerson algorithm might run exponentially long

In that example, the Ford-Fulkerson algorithm could decide to increase the initial flow by 1 along $s - x - y - t$ in the first iteration, then increase the

flow by 1 along $s-y-x-t$, then along $s-x-y-t$ again and so on. The algorithm would iterate 2000 times, the running-time would depend linearly on the capacities, which can be exponential in the input-length. It would only be polynomial, if the capacities were provided in a unary alphabet (the algorithm is a so called "pseudo-polynomial" algorithm).

1.3.3 The Edmonds-Karp algorithm

Edmonds and Karp modified the Ford-Fulkerson algorithm [34], so that it always selects a *shortest* augmenting path. This trick lowers the running-time of the algorithm to $O(|E|^2 \cdot n) \subseteq O(n^5)$. The reason for this is that in every iteration, at least one edge becomes saturated $f(x,y) = c(\{x,y\})$ or $f((x,y)) = c((x,y))$. It might become desaturated again (when it occurs backwards in an augmenting path), but since shortest augmenting paths are used, this cannot happen before the distance of x to s is increased, because otherwise the selected path would not have been a shortest path before. Since the distance of any vertex to s can be at most n, every edge can at most become saturated n times, so the algorithm selects at most $n \cdot |E|$ augmenting paths. Together with a running time of $O(|E|)$ for finding a shortest augmenting path, the total running-time is in

$$O(\underbrace{|E|}_{\text{find aug. path}} \cdot \underbrace{n}_{\text{at most } n \text{ times}} \cdot \underbrace{|E|}_{\text{per edge}})$$

Further improvements to the Ford-Fulkerson algorithm have running-times down to $O(n^3)$, where the main idea is to increase the flow on multiple shortest paths at once [30, 72, 39]. These algorithms become increasingly complicated, a completely different and much simpler approach is the preflow-push algorithm, which also has a running-time of $O(n^3)$ [61, 50].

1.3.4 Minimum cuts

A ***cut*** is a set of edges $C \subseteq E$ such that in $(V, E-C)$ there is no path from s to t. The ***value*** of a cut is $\sum_{e \in C} c(e)$.

A cut is a minimum cut or ***min-cut***, if there exists no cut that has a smaller value. The cut in fig. 1.8 is not a minimum cut, because the cut in fig 1.9 has a smaller value.

Minimum cuts are of practical interest, e.g. because they are a metric for the resistance of a supply chain against grid failure. The problem of finding a minimum cut was solved by the famous Max-Flow = Min-Cut theorem by Ford and Fulkerson [42]. A minimum cut can be found in polynomial time

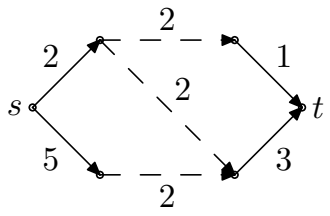


Figure 1.8: A cut with value 6

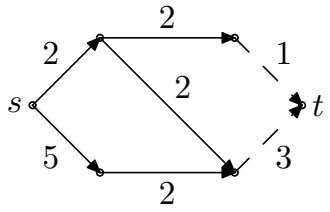


Figure 1.9: A cut with value 4

by constructing a max-flow f and selecting $S \subseteq V$ such that S contains all vertices that are reachable from s using only unsaturated edges. A minimum cut then is the set of edges that start in S and end in $V \backslash S$, see fig. 1.10.
Therefore, min-cuts can be found in polynomial time.

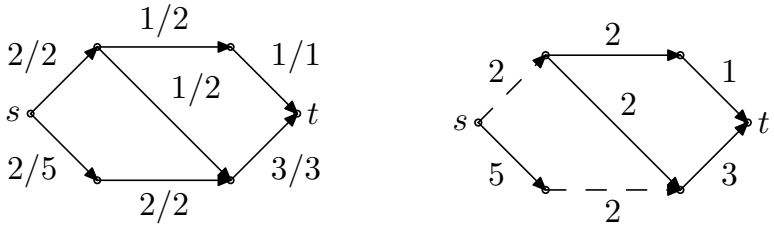


Figure 1.10: A cut with value 4 that has been found using the max-flow

1.3.5 Single-commodity multi-terminal networks

A ***single-commodity multi-terminal network*** is a 5-tuple (V, E, S, T, c) containing a set of vertices V, a set of edges or arcs E, two disjoint sets of selected vertices S and T (***sources*** and ***targets***) and a capacity-function $c : E \to \mathbb{R}^+$. The sources might stand for multiple factories that produce the same commodity and the drains might stand for multiple retail stores that

must be supplied.

It makes no difference whether entities of the same commodity are produced at multiple factories or at a single factory and delivered to the other "factories" first. So multiple sources can be replaced by a single new source s that is connected to the old sources with edges of sufficiently large capacity (e.g. $c(\{s, s'\}) = \sum_{e \in E} c(e)$ for all $s' \in S$). Analogous it makes no difference whether entities are sold at multiple retail stores or the "retail stores" forward them to a single store where they are sold. So multiple drains can be replaced by a single new drain t that is connected to the old drains with edges of sufficiently large capacity (e.g. $c(\{t', t\}) = \sum_{e \in E} c(e)$ for all $t' \in T$).

This implies that max-flow, max-integer-flow and min-cut of a single-commodity multi-terminal network (V, E, S, T, c) equate to the max-flow, max-integer-flow and min-cut of the ordinary network $(V \dot\cup \{s, t\}, E \dot\cup \{\{s, s'\}|s' \in S\} \dot\cup \{\{t', t\}|t' \in T\}, s, t, c')$ with

$$c'(e) = \begin{cases} c(e) & \text{if } e \in E \\ \sum_{e \in E} c(e) & \text{else} \end{cases}$$

Thus max-flow, max-integer-flow and min-cut of a single-commodity multi-terminal network can be found in polynomial time.

1.3.6 Productivity and demand

The demand of the stores can be less than the amount that could be produced and delivered. Also the factories might not be productive enough to produce the amount that is demanded and could be delivered. Productivity and

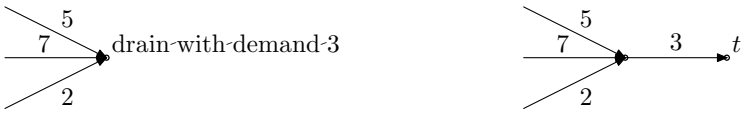


Figure 1.11: A drain with less demand than supply

demand can be modeled by splitting the corresponding source or drain into two vertices and connecting them with an edge or arc, whose capacity is set to the productivity or demand. One of these vertices becomes the new source or drain, the other one inherits the neighborhood of the old source or drain. See fig. 1.11 and 1.12.

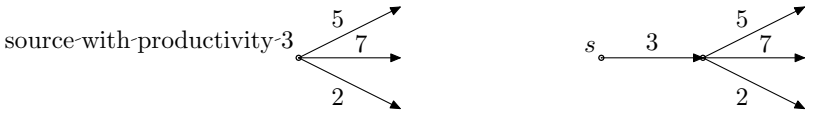


Figure 1.12: A source with less productivity than demand

1.4 Parameterized complexity theory

To solve problems in practice, one needs algorithms with polynomial running-times, but for NP-complete problems, achieving such an algorithm would imply P=NP. As mentioned before, it is not yet proven that P≠NP, so the existence of such an algorithm is not disproven. In theory, a polynomial algorithm might exist, but unless one is actually found (and thus P=NP is proven), one must cope with exponential running-times in practice.

Parameterized complexity theory is a relatively young branch of theoretical computer science, in which one tries to subdivide the instances of NP-complete problems into "simple" and "hard" instances, so that at least the "simple" instances can be solved in practice. This is done by looking at different properties (***parameters***) and trying to develop algorithms that are fast for the instances which have a small value for these parameters.

1.4.1 Fixed parameter tractability (FPT)

A problem is ***fixed parameter tractable*** [32, 41, 77], if it is solvable by an algorithm, whose running-time is in $O(f(k) \cdot p(|w|)) = O^*(f(k))$ for any instance w, where f is an arbitrary, total function with $f(x) < \infty$ for all x and p is a polynomial. k is some property of w, called the ***parameter***. For example Clique (the problem of finding a maximum set of vertices that are all pairwise adjacent) can be solved in $O(2^{\Delta(G)} \cdot \Delta(G)^2 \cdot n)$ by examining all subsets of the neighborhood of any vertex. We say that the problem is fixed parameter tractable or FPT *in* that parameter. The class of all fixed parameter tractable problems is called ***FPT***. Informally speaking, a parameter in which an NP-complete problem is FPT, is a cause of the exponential running-time, because the problem can be solved fast, except for instances where the property is large.
Note that fixed parameter tractability is not an inherent property of a problem. A problem can be FPT in one parameter, but ***fixed parameter intractable*** in a different parameter. For example Clique is FPT in Δ, but it is fixed parameter intractable in the size of a maximum clique (unless

FPT=W[1] see 1.4.7 [41, 15]).

If there exists an algorithm with a running-time of $O(f(k_1, \cdots, k_q) \cdot p(n))$, then we say that the problem is FPT in $\{k_1, \cdots, k_q\}$, although this isn't conform to the formal definition of a parameterized problem (see 1.4.2). It would be formally correct to say that the problem is FPT in $max\{k_1, \cdots, k_q\}$.

Lemma 1.4.1.1. *If a problem is FPT in parameters $\{k_1, \cdots, k_q\}$ and there is a set of parameters $\{k'_1, \cdots, k'_{q'}\}$ such that $max\{k_{i_1}, \cdots, k_{i_r}\} \leq f(k'_1, \cdots, k'_{q'})$ for some function f, then the problem is also FPT in*

$$(\{k_1, \cdots, k_q\} \backslash \{k_{i_1}, \cdots, k_{i_r}\}) \cup \{k'_1, \cdots, k'_{q'}\}$$

Proof. In the running-time of the FPT algorithm, every k_{i_j} can be replaced by $f(k'_1, \cdots, k'_{q'})$. □

For example the VERTEX COVER problem (the problem of finding a minimum set of vertices, such that every edge is incident to one of them) is FPT in the treewidth (see 1.4.5) tw and the treewidth of any graph is at most as large as the size of a minimum vertex cover β. Therefore VERTEX COVER is also FPT in $(\{tw\} \backslash \{tw\}) \cup \{\beta\} = \{\beta\}$.

Lemma 1.4.1.2. *If a problem is NP-hard already for constant values of some parameters $\{k_1, \cdots, k_q\}$, then the problem cannot be FPT in these parameters unless P=NP.*

Proof. Fixed parameter tractability would entail an $O(f(k_1, \cdots, k_q) \cdot p(n))$ algorithm, which is a polynomial running-time for constant values for $k_1, \cdots, k_q$, since then $f(k_1, \cdots, k_q)$ is also constant, so $O(f(k_1, \cdots, k_q) \cdot p(n)) = O(p(n))$. This means that the algorithm would be polynomial for the NP-hard problem and this means P=NP. □

An important technique to develop FPT algorithms is using ***bounded searchtrees***: If an algorithm gets $f(k)$ as argument, calls $g(f(k))$ recursions, each of which gets an argument that is at least 1 smaller than the argument of the parent, a parameter < 0 is not allowed and the running-time within every recursive call is polynomial $p(n)$, then the overall running time is in $O(g(f(k))^{f(k)} \cdot p(n))$.

1.4.2 Parameterized problems

To formalize the concept of the previous section: a ***parameterized problem*** over an alphabet Σ is a (not necessarily decidable) set of pairs of words and

integers $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$. An instance of a parameterized problem is a pair (w, k) with $w \in \Sigma^*, k \in \mathbb{N}$ and the question is whether $(w, k) \in \mathcal{L}$.

For example the problem CLIQUE contains all pairs (w, k), where w is an encoding of a graph that has a clique of size k. If for every w, there is at most one k with $(w, k) \in \mathcal{L}$, then $\mathcal{L}$ can be interpreted as a function $\Sigma^* \to \mathbb{N}$. A parameterized problem $\mathcal{L}$ is fixed parameter tractable, if there is an algorithm that decides whether $(w, k) \in \mathcal{L}$ in a running-time of $O(f(k) \cdot p(|w|))$.

CLIQUE, as defined above is not FPT, but it was FPT in Δ. To see that this circumstance can still be described by the above definition, one must keep in mind that $proj_1(\mathcal{L})$[2] itself can already be an encoding of a parameterized problem, so CLIQUE_Δ could be described as $\mathcal{L} \subseteq (\Sigma^* \times \mathbb{N}) \times \mathbb{N}$. Instances of this problem would have the form $((w, k), d)$ and $\mathcal{L}$ would contain the instances $((w, k), d)$ where w is an encoding of a graph G, $d = \Delta(G)$ and G has a clique of size k.

Any problem $\mathcal{L} \subseteq \Sigma^*$ can be turned into a parameterized problem $\mathcal{L}_\varphi$, using a function $\varphi : \Sigma^* \to \mathbb{N}$, by defining

$$\mathcal{L}_\varphi := \{(w, \varphi(w)) | w \in \mathcal{L}\}$$

A problem $\mathcal{L}$ is FPT *in* a function φ, if $\mathcal{L}_\varphi$ is FPT.

If a problem $\mathcal{L}$ is FPT in multiple functions $\varphi_1, \cdots, \varphi_a$ (i.e. there are a algorithms, one is fast for instances w with small values of $\varphi_1(w)$, one is fast for instances $\cdots$ and one is fast for instances w with small values of $\varphi_a(w)$) then one can decide whether $w \in \mathcal{L}$ by computing $\varphi_1(w), \varphi_2(w), \cdots, \varphi_a(w)$ and using these values to predict the running-times of the algorithms on w. Using this information the algorithm that is fastest for the concrete instance w can be selected.
For this method, we would like all these functions φ_i to be computable in polynomial time (so the running-times can be predicted sufficiently fast), but this is usually not mandatory. Many FPT-algorithms do not depend on the knowledge of the parameter. E.g. they can be run multiple times with successively increased parameter, so the running-time is

$$O\left(\sum_{j=1}^{\varphi_i(w)} f(i) \cdot p(|w|)\right) = O\left(\underbrace{\left(\sum_{j=1}^{\varphi_i(w)} f(i)\right)}_{\text{function in } \varphi_i(w)} \cdot \underbrace{p(|w|)}_{\text{polynomial in } |w|}\right)$$

[2] $proj_1$ is the projection to the first component

Other algorithms do not use the parameter. E.g. the parameter of the $O(2^{\Delta} \cdot \Delta^2 \cdot n)$ algorithm for CLIQUE is only needed to describe the running-time, but it is not used within the algorithm.

Multiple of such algorithms can be run in parallel and aborted as soon as one of them terminates. The running-time of this method is

$$O(a \cdot t \cdot o)$$

where a is the number of algorithms that are executed in parallel (usually constant, but it is sufficient that $a \leq p(|w|)$ for a polynomial p), t is the running-time of the algorithm that is fastest for the given instance w and o is the overhead from context-switching (constant on random access machines like computers and $\leq a \cdot t$ on (one-tape) Turing machines). So although the parameters can be unknown, the overall running-time stays FPT

$$O(\underbrace{(p(|w|)}_{a} \cdot \underbrace{f(\varphi_i(w)) \cdot p_i(|w|))}_{t} \underbrace{{}^{2}}_{o}) = O(\underbrace{f(\varphi_i(w))^2}_{\text{function in } \varphi_i(w)} \cdot \underbrace{p(|w|)^2 \cdot p_i(|w|)^2}_{\text{polynomial in } |w|})$$

1.4.3 FPT-reductions

Let $\mathcal{L}$ a parameterized problem over Σ and $\mathcal{L}'$ a parameterized problem over Γ. Analogous to P-reductions, an ***FPT-reduction*** is a function $g : (\Sigma^* \times \mathbb{N}) \to (\Gamma^* \times \mathbb{N})$ that is computable in FPT time and that has

1. $(w, k) \in \mathcal{L}$ iff $(w', k') \in \mathcal{L}'$
2. $k' \leq f(k)$ for a function f

for $g((w, k)) = (w', k')$. If such a function exists, we write $\mathcal{L} \leq_{FPT} \mathcal{L}'$.

Lemma 1.4.3.1. *If $\mathcal{L} \leq_{FPT} \mathcal{L}'$ and $\mathcal{L}'$ is FPT, then $\mathcal{L}$ is FPT. (in other words: FPT is closed under FPT-reductions)*

Proof. To solve an instance (w, k) of $\mathcal{L}$, one can compute $g((w, k))$ and run the FPT algorithm for $\mathcal{L}'$ on the result (w', k'). This method has an FPT running-time:
(w', k') can be computed in $O(f_1(k) \cdot p_1(|w|))$, so $|w'| \leq f_1(k) \cdot p_1(|w|) + |w|$. So the running-time of the FPT algorithm for $\mathcal{L}'$ will be

$$\begin{aligned}
& O(f_2(k') \cdot p_2(|w'|)) \\
\subseteq\ & O(f_2(f(k)) \cdot p_2(f_1(k) \cdot p_1(|w|) + |w|)) \\
\subseteq\ & O(f_2(f(k)) \cdot (f_1(k) \cdot p_1(|w|))^{d+1}) \text{ if } d \text{ is the degree of } p_2 \\
=\ & O(\underbrace{f_2(f(k)) \cdot f_1(k)^{d+1}}_{\text{function in } k} \cdot \underbrace{p_1(|w|)^{d+1}}_{\text{polynomial}})
\end{aligned}$$

Together with the running-time for the reduction, the overall running-time is in

$$\begin{aligned} & O(f_1(k) \cdot p_1(|w|) + f_2(f(k)) \cdot f_1(k)^{d+1} \cdot p_1(|w|)^{d+1}) \\ \subseteq\ & O(2 \cdot max\,\{f_1(k) \cdot p_1(|w|), f_2(f(k)) \cdot f_1(k)^{d+1} \cdot p_1(|w|)^{d+1}\}) \\ =\ & O(max\,\{f_1(k) \cdot p_1(|w|), f_2(f(k)) \cdot f_1(k)^{d+1} \cdot p_1(|w|)^{d+1}\}) \\ =\ & \underbrace{O(f_1(k) \cdot p_1(|w|))}_{FPT} \text{ or } \underbrace{O(f_2(f(k)) \cdot f_1(k)^{d+1} \cdot p_1(|w|)^{d+1})}_{FPT} \end{aligned}$$

which is an FPT running-time. □

Many disquisitions show fixed parameter intractability of a parameterized problem by giving an FPT-reduction from a fixed parameter intractable problem to it.

1.4.4 Kernelizations

A ***kernelization*** is a function $g : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that is computable in polynomial time and that has

- $(w, k) \in \mathcal{L}$ iff $(w', k') \in \mathcal{L}$
- $|w'| \leq f(k)$ for a function f

for $g((w, k)) = (w', k')$. We call (w', k') the ***(problem-)kernel*** of (w, k). Informally speaking, a kernelization is a polynomial algorithm that can solve "simple" sub-problems of an instance. The kernel is the part of the instance that the kernelization cannot solve, it is the "hard" part of the instance.

Lemma 1.4.4.1. *[77] A parameterized problem is fixed parameter tractable if and only if it is decidable and has a kernelization.*

Proof. Let $\mathcal{L}$ a decidable, parameterized problem with a kernelization g. Let (w, k) an instance of the problem and $(w', k') = g((w, k))$ with $|w'| \leq f(k)$.

Since the problem is decidable, there is an algorithm with some upper bound $O(h(|w|))$ to the running-time for any instance w. w' can be computed in polynomial time $O(p(|w|))$. Thereafter the decision algorithm can be applied in finite time $O(h(|w'|)) \subseteq O(h(f(k)))$. The total running-time is $O(p(|w|) + h(f(k)))$, which satisfies the definition of an FPT algorithm.

Let $\mathcal{L}$ a parameterized problem that is fixed parameter tractable i.e. there is an algorithm that decides the problem in $O(f(k) \cdot p(|w|))$. A kernelization

can be obtained that has the f from the running-time as limit for the kernel sizes. Instances (w, k) with $|w| \leq f(k)$ will be mapped to themselves, as their size is small enough. For instances (w, k) with $f(k) < |w|$, the running-time of the FPT algorithm becomes polynomial, because $f(k) \cdot p(|w|) \leq |w| \cdot p(|w|)$, which is polynomial. Therefore these instances can be solved in polynomial time, using the FPT algorithm and mapping them to a trivial instance $w_{yes} \in \mathcal{L}$ if the algorithm says $(w, k) \in \mathcal{L}$ or to $w_{no} \notin \mathcal{L}$ if the algorithm says $(w, k) \notin \mathcal{L}$.
The size of the kernel of (w, k) is bounded by $max\{f(k), |w_{yes}|, |w_{no}|\}$. □

Many disquisitions show fixed parameter tractability of a parameterized problem by defining a kernelization and proving an upper bound of $f(k)$ for $|w'|$.

If there exists a polynomial p such that $|w'| \leq p(k)$ for all (w, k), then the problem is in the important complexityclass ***POLYKERNEL***.

1.4.5 Treewidth

Informally speaking, the ***treewidth*** tw of a graph is a measurement for its similarity to a tree. This property seems inconspicuous, but over the last decades, it has proven to be one of the most important properties of a graph, at least from the viewpoint of computational complexity, because most graph-theoretic problems are FPT in the treewidth (see 1.4.6).

The formal definition as it is used nowadays goes back to Robertson and Seymour [81, 82, 83], although many equivalent definitions were known before [11]. Formally, the treewidth of a graph G is the smallest possible size of a ***tree-decomposition***, where a tree-decomposition is a pair (T, B) containing a tree T and a mapping $B : V(T) \to \mathfrak{P}(V(G))$. The sets of vertices from G, that are associated to the vertices of the tree are called ***bags***. If $e = \{x, y\}$ is an edge in G and there is a vertex v in T with $x, y \in B(v)$ (there is a bag that contains both x and y), then the edge is ***cleared***. A tree-decomposition must fulfill

1. $\bigcup_{v \in V(T)} B(v) = V(G)$ (every vertex is in a bag)

2. $\forall \{x, y\} \in E(G). \exists v \in V(T). \{x, y\} \subseteq B(v)$ (every edge is cleared)

3. if $x \in B(v_1) \cap B(v_2)$, then $x \in B(v_i)$ for all v_i on the (unique) path from v_1 to v_2 in T. (in other words: for every vertex v from G, the bags that contain v induce a subtree)

The ***size*** of a tree-decomposition (T, B) is one less than the cardinality of its largest bag $\max_{v \in V(T)} |B(v)| - 1$. The ***pathwidth*** is defined just like the treewidth, only that for the pathwidth, T must be a path.

An interesting interrelation is that the treewidth+1 coincides with the number of policemen that are needed to catch a (fast) fugitive in a graph [87], if the policemen always know, where he is. If they don't know this, then the number of needed policemen coincides with the pathwidth+1 [63, 35]. The strategy to catch the fugitive is to place the policemen on the vertices of one bag and move them successively to the vertices of an adjacent bag (depending on where the fugitive is). This strategy forces the fugitive more and more into a dead end (a leaf of the tree-decomposition).

In linear time, a tree-decomposition can be turned into a ***nice tree-decomposition*** without changing its width (Lemma 13.1.3 in [66]). In a nice tree-decomposition, T is a rooted tree such that

- every vertex of T has at most 2 children
- if a vertex x has 2 children y, z, then $B(x) = B(y) = B(z)$ (such a vertex x is called a ***join vertex***)
- a vertex x with 1 child y has either
  - $B(x) = B(y) \dot\cup \{v\}$ (such a vertex x is called an ***introduce vertex***) or
  - $B(x) \dot\cup \{v\} = B(y)$ (such a vertex x is called a ***forget vertex***)

In terms of the cops and fugitive game, a nice tree-decomposition corresponds to a strategy in which the policemen are moved one after the other and the decision in which direction to move can be split up into multiple decisions between pausing and then moving in one direction or pausing and then deciding between the other directions. Many FPT algorithms use dynamic-programming on a (nice) tree-decomposition. For this, it is important that for any graph $G = (V, E)$, there exists a nice tree-decomposition (T, B) with at most $4 \cdot |V|$ vertices in T (Lemma 13.1.3 in [66]).

1.4.6 Courcelle's theorem

Courcelle's theorem [24] is one of the most important theorems in parameterized complexity theory. It says that every graph-theoretic problem

that can be axiomatized in monadic second order (MSO_2) logic[3], is fixed parameter tractable in the treewidth. In [3], this result was generalized to extended MSO_2 ($ExtMSO_2$) logic.
$ExtMSO_2$ thereby contains formulas that are constructed inductively from

Atomic statements, membership and equality $adj(x, y)$, $inc(x, e)$ to express adjacency of two vertices x, y or incidence of a vertex x to an edge e. Also quantified variables can be tested for equivalence $x = y$ and membership in a set $x \in U$.

Propositional logic operators $\vee$, $\wedge$, $\neg$ to express disjunction, conjunction and negation of statements.

First order logic (FOL) quantifiers $\exists x \in X.\varphi(x)$, $\forall x \in X.\varphi(x)$ to express that a statement φ holds for at least one ($\exists$) or all ($\forall$) elements x of a set X.

Monadic second order (MSO) quantifiers $\exists X \subseteq Y.\varphi(X)$, $\forall X \subseteq Y.\varphi(X)$, to express that a statement φ holds for at least one or all subsets X of a set Y.

Extensions to MSO $max\ X \subseteq Y.\varphi(X)$, $min\ X \subseteq Y.\varphi(X)$ to quantify a maximum or minimum set, for which a statement φ holds.

The validity of Courcelle's theorem in logics with arithmetic extensions is subject matter of current research. Examples for $ExtMSO_2$ formulas for common graph-theoretic problems are

Clique	$max\, U \subseteq V.\forall u \in U.\forall v \in U.u = v \vee adj(u, v)$
Dominating Set	$min\, U \subseteq V.\forall v \in V.v \in U \vee \exists u \in U.adj(u, v)$
Vertex Cover	$min\, U \subseteq V.\forall e \in E.\exists v \in U.inc(v, e)$

Courcelle's theorem is usually proven by constructing tree-automata for the $ExtMSO_2$ formulas and applying them to tree-decompositions, whose computation is FPT in tw [10].

Lemma 1.4.6.1. *Let $\mathcal{L}$ a (parameterized) graph-theoretic problem. If for all instances (w, k) of the problem (w is an encoding of a graph G), an $ExtMSO_2$ formula φ_G can be constructed in $O(f(k_1, \cdots, k_q) \cdot p(n))$ such that $(w, k) \in \mathcal{L}$ iff G models φ_G, and whose length is bounded by $g(k_1, \cdots, k_q)$ for some f, g and parameters $k_1, \cdots, k_q$, then the problem is fixed parameter tractable in $\{tw, k_1, \cdots, k_q\}$ (k might be among the parameters $k_1, \cdots, k_q$).*

[3]The 2 in the index of ExtMSO refers to two-sorts, because the universe of the logical structure of a graph contains two sorts of elements: vertices and edges

Proof. To decide whether $(w,k) \in \mathcal{L}$, the formula φ_G can be constructed in $O(f(k_1,\cdots,k_q)\cdot p(|w|))$. Thereafter the automaton for φ_G can be constructed in a running-time that depends only on the length of the formula, so this step takes $O(h(g(k_1,\cdots,k_q)))$ steps for some h. Finally, the tree-decomposition of G can be obtained and checked by the automaton in $O(\tilde{f}(tw))$. The total running-time is

$$O(f(k_1,\cdots,k_q)\cdot p(n) + h(g(k_1,\cdots,k_q)) + \tilde{f}(tw))$$

□

Example: The question whether a graph G has a path of length k is FPT in $\{tw,k\}$, because an ExtMSO$_2$ formula with a size bounded by $O(k^2)$ would be

$$\exists x_1,\cdots,x_k \in V. x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \cdots \wedge x_{k-1} \neq x_k$$
$$\wedge adj(x_1,x_2) \wedge adj(x_2,x_3) \wedge \cdots \wedge adj(x_{k-1},x_k).$$

These results are very important, but they are only of theoretical interest. In practice, algorithms still have to be designed manually, since the automata are far too large to be used in practice. There exist short MSO$_2$ formulas, that correspond to automata with e.g. $2^{65536} \approx 10^{19728}$ states. In fact, the size of the automata is not bounded by any elementary function [45].

1.4.7 The weft-hierarchy

A parameterized problem $\mathcal{L}$ is in the complexity class ***W[i]***, if for every instance (w,k), a boolean circuit with multiple inputs and a single output can be constructed within FPT time, with the following properties:

- $(w,k) \in \mathcal{L}$ if and only if the circuit has a satisfying assignment that assigns *true* to exactly k inputs
- Every path from an input to the output contains at most c gates with fan-in ≤ 2 (***small gates***), where c is a constant for all instances
- Every path from an input to the output contains at most i gates with unbounded fan-in (***big gates***)

A boolean circuit whose maximum number of big gates on a path from an input to the output is i, has ***weft*** i [31]. Analogous to NP-hardness and NP-completeness, [41] a parameterized problem $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$ is ***W[i]-hard***, if every parameterized problem $\mathcal{L}' \in W[i]$ has $\mathcal{L}' \leq_{FPT} \mathcal{L}$ and ***W[i]-complete***, if it is W[i]-hard and in W[i].

$$FPT = W[0] \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[*] := \bigcup_{i \in \mathbb{N}} W[i]$$

Note that FPT=W[0], because within FPT time the instances of parameterized problems in FPT can be solved entirely and mapped to trivial circuits (without big gates), so $FPT \subseteq W[0]$. Conversely: if all instances of a parameterized problem can be mapped in time $O(f(k) \cdot p(n))$ to a weft 0 circuit (which has at most 2^c inputs and can thus be solved in constant 2^{2^c} steps), then all instances can be solved in $O(f(k) \cdot p(n) + 2^{2^c})$, which is an FPT running-time, so $W[0] \subseteq FPT$.

It is widely believed that $W[i] \neq W[j]$ for all $i \neq j$ and although this is not proven yet, a proof for W[i]-hardness of a problem $\mathcal{L}$ is considered a proof for $\mathcal{L} \notin W[j]$ for any $j < i$. In particular, showing W[1]-hardness of a parameterized problem is considered proof for fixed parameter intractability.

Chapter 2

Multi-commodity networks

A ***multi-commodity network*** is a 4-tuple $(V, E, \mathcal{P}, c)$ containing a set of vertices V, a set of edges or arcs E, a set of pairs of selected vertices $\mathcal{P} = \{(s_1, t_1), \cdots, (s_l, t_l)\} \subseteq V \times V$ or $\mathcal{P} = \{\{s_1, t_1\}, \cdots, \{s_l, t_l\}\} \subseteq \mathfrak{P}_2(V)$ and a capacity-function $c : E \to \mathbb{R}^+$. A multi-commodity network whose edges all have the same capacity (or equivalently: A multi-commodity network without a capacity-function and implicit capacities of 1 for all edges) is called a ***uniform multi-commodity network***. Usually it is assumed that $c(e) \geq 1$ for all edges e, so that the height of a searchtree can be bounded by k.

2.1 The Max-Multiflow problem

Let $(V, E, \mathcal{P}, c)$ an undirected multi-commodity network. A ***multiflow*** is a function $f : V \times V \times \{1, \cdots, l\} \to \mathbb{R}^+$ with

1. $\sum_{i=1}^{l} f(x, y, i) + f(y, x, i) \leq c(\{x, y\})$ for all $\{x, y\} \in E$ (***capacity-limitation***)
2. $\sum_{y \in N(x)} f(y, x, i) = \sum_{y \in N(x)} f(x, y, i)$ for all $i = 1, \cdots, l$ and all $x \in V \backslash \{s_i, t_i\}$ (***individual flow-preservation***)
3. $f(x, s_i, i) = 0$, $f(t_i, x, i) = 0$ for all $i = 1, \cdots, l$ and all vertices $x \in V$

For directed multi-commodity networks $(V, E, \mathcal{P}, c)$, a ***multiflow*** is a function $f : E \times \{1, \cdots, l\} \to \mathbb{R}^+$ with

1. $\sum_{i=1}^{l} f(e, i) \leq c(e)$ for all $e \in E$ (***capacity-limitation***)

2. $\sum_{y \in N^-(x)} f((y,x),i) = \sum_{y \in N^+(x)} f((x,y),i)$ for all $i = 1, \cdots, l$ and all $x \in V \backslash \{s_i, t_i\}$ (***individual flow-preservation***)

3. $f((x,s_i),i) = 0$, $f((t_i,x),i) = 0$ for all $i = 1, \cdots, l$ and all vertices $x \in V$

The crucial difference to single-commodity multi-terminal flows is that it is not sufficient to require that the magnitude of the inbound flow equals the magnitude of the outbound flow for every vertex, but it must be required that the inbound flow of every individual commodity equals the outbound flow of the same commodity for all commodities in all vertices. This is why an augmenting path does not necessarily exist for non-optimal multiflows. An augmenting path for one commodity cannot simply replace the flow of a different commodity and decrease the flow on a backward arc, because this would violate the flow-preservation of the involved commodities in the involved vertices (compare fig. 1.6 and fig. 2.1).

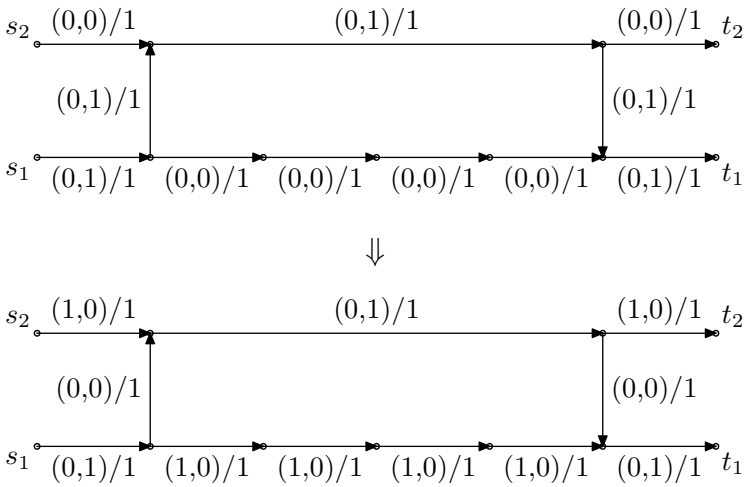


Figure 2.1: A non-maximum multiflow and a multiflow with violated flow-preservation after using a normal augmenting path

2.1.1 Linear programs

Despite this obstacle, MAX MULTIFLOW is still polynomial, because its instances can be modeled as linear programs [43] and solving linear programs is polynomial [58, 76]. Itai showed [57] that LINEAR PROGRAMMING $\leq_P$

MAX MULTIFLOW. For every pair of an undirected edge $\{x, y\}$ and a commodity i, two variables $X_{x,y,i}$ and $X_{y,x,i}$ are used which describe how many entities of commodity i travel through the edge in either direction. For directed arcs (x, y), only one variable $X_{x,y,i}$ is needed per commodity, since the entities can only travel in one direction through these arcs.

The capacity-limitation can be described as (canonical) linear constraints:

$$\sum_{i=1}^{l} X_{x,y,i} + X_{y,x,i} \leq c(\{x, y\}) \text{ for all edges } \{x, y\} \in E$$

$$\sum_{i=1}^{l} X_{x,y,i} \leq c((x, y)) \text{ for all arcs } (x, y) \in E$$

Also the flow-preservation can be described as (canonical) linear constraints:

$$\begin{aligned} \sum_{y \in N(x)} X_{y,x,i} - \sum_{y \in N(x)} X_{x,y,i} &\leq 0 \\ -\sum_{y \in N(x)} X_{y,x,i} + \sum_{y \in N(x)} X_{x,y,i} &\leq 0 \end{aligned}$$

for all $i = 1, \cdots, l$ and all vertices $x \in V \backslash \{s_i, t_i\}$.

$$\begin{array}{rcl|rcl} \sum\limits_{y \in N(s_i)} X_{y,s_i,i} & \leq & 0 & \sum\limits_{y \in N(t_i)} X_{t_i,y,i} & \leq & 0 \\ -\sum\limits_{y \in N(s_i)} X_{y,s_i,i} & \leq & 0 & -\sum\limits_{y \in N(t_i)} X_{t_i,y,i} & \leq & 0 \end{array}$$

for all $i = 1, \cdots, l$.

The objective function is to maximize the sum of the outbound flows of commodity i from s_i.

Example: Consider the multi-commodity network in fig. 2.2.

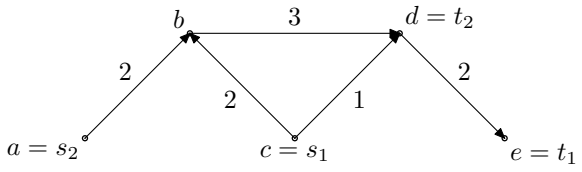


Figure 2.2: A multi-commodity network

Objective function: maximize $X_{c,b,1} + X_{c,d,1} + X_{a,b,2}$ under the condition

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{1} & 0 & \mathbf{-1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
\mathbf{-1} & 0 & \mathbf{1} & 0 & \mathbf{-1} & 0 & 0 & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & \mathbf{-1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & \mathbf{-1} & 0 & \mathbf{1} & 0 & \mathbf{-1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{-1} & 0 & \mathbf{-1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{-1} & 0 \\
0 & 0 & \mathbf{-1} & 0 & 0 & 0 & \mathbf{-1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{-1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{-1}
\end{pmatrix} \cdot \begin{pmatrix}
X_{a,b,1} \\ X_{a,b,2} \\ X_{b,d,1} \\ X_{b,d,2} \\ X_{c,b,1} \\ X_{c,b,2} \\ X_{c,d,1} \\ X_{c,d,2} \\ X_{d,e,1} \\ X_{d,e,2}
\end{pmatrix} \leq \begin{pmatrix}
2 \\ 3 \\ 2 \\ 1 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{pmatrix}$$

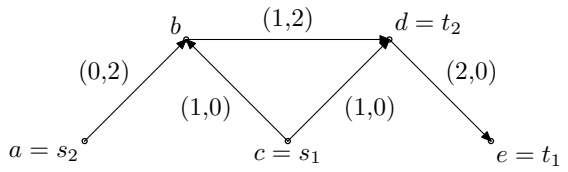
Figure 2.3: A linear program for the multi-commodity network in fig. 2.2

A linear program for this instance is shown in fig. 2.3.

The first 5 lines of the matrix model the capacity-limitation, followed by 2,4,2,4 and 2 lines, modeling the flow-preservation of vertex a, b, c, d and e respectively.

Using an LP-solver (which uses the simplex algorithm [27]), gives the result

```
Value of objective
function: 4.0
x1 = 0.0  x2 = 2.0
x3 = 1.0  x4 = 2.0
x5 = 1.0  x6 = 0.0
x7 = 1.0  x8 = 0.0
x9 = 2.0  x10 = 0.0
```



2.1.2 Revenue, transportation costs and minimal requirements

Using linear programs, it is even possible to incorporate different prices of the commodities, by using the prices as coefficients π_i to the outbound flows of commodity i in the objective function. This causes the solution to preempt flow of entities that generate more revenue.

Also different prices for using different edges can be modeled and it is even possible to model different prices for transporting entities of different commodities over the same edge (for example it is surely cheaper to transport cars on a train, than transporting toxic waste on the same train). This can be done by subtracting the loss from the objective function. The loss is the product of the price $\tau_{x,y,i}$ for transporting entities of commodity i from x to y and the amount of entities that are actually transported $X_{x,y,i}$. This will cause the solution to preempt cheaper routes.

New objective function: maximize

$$\sum_{i=1}^{l} \left(\underbrace{\pi_i \cdot \sum_{(s_i,z)\in E} X_{s_i,z,i}}_{\text{revenue from commodity } i} - \underbrace{\sum_{(x,y)\in E} \tau_{x,y,i} \cdot X_{x,y,i}}_{\text{transport costs for commodity } i} \right)$$

If store t_i needs at least d_i units of commodity i, then this can be expressed as an additional constraint in the linear program:

$$\sum_{x\in N^-(t_i)} -1 \cdot X_{x,t_i,i} \leq -1 \cdot d_i$$

2.2 The Max-Integer-Multiflow problem

The MAX-MULTIFLOW and MAX-INTEGER-MULTIFLOW problem, like the MAX-FLOW and MAX-INTEGER-FLOW problems, only differ in the codomain of the flow-function, which is $\mathbb{R}^+$ for MAX-MULTIFLOW, but $\mathbb{N}_0$ for MAX-INTEGER-MULTIFLOW.

Unfortunately, the MAX-INTEGER-FLOW = MAX-FLOW theorem from single-commodity networks with integral capacities does not hold for multi-commodity networks, as figure 2.6 shows. In fact, determining the magnitude of a maximum integer multiflow is NP-complete. There are several interesting

proofs for the NP-completeness of MAX-INTEGER-MULTIFLOW on instances with $l = 2$, one can be found in [38]. The first proof for the NP-completeness (without limitation on l) was invented by Donald E. Knuth in a private conversation with Richard M. Karp in 1974, who published the proof in 1975 [60]. It is also NP-complete on grids [68].

On uniform stars, MAX-INTEGER-MULTIFLOW is equivalent [49] to the (polynomial [33]) MAXIMUM MATCHING problem, using the same construction as in the reduction from MIN-MULTICUT ON STARS to VERTEX COVER (see 5.1): construct an auxiliary graph with the same set of vertices as the star, except for the center vertex, and add an edge $\{x, y\}$, if $\{x, y\} \in \mathcal{P}$. Having flow between x and y corresponds to selecting the edge $\{x, y\}$ for a matching (because of uniformity and integrality, every vertex can only have flow to one other terminal, like in a matching, every vertex can only be covered by at most one edge). If there is a terminalpair $\{c, x_i\}$, where c is the center vertex and x_i is some other vertex, then the flow between these vertices can be used without any loss - remove x_i and add 1 to the result.
On undirected, uniform trees, the max-integer-multiflow equates to the number of edge-disjoint paths: The path that is used by one flow must be edge-disjoint to all paths that are used by other flows. Conversely, every path in a set of disjoint paths allows an integer-flow of 1. Computing the number of disjoint paths is NP-complete in general [2], but polynomial on trees [49].

The algorithm to find a maximum-integer-multiflow in a uniform tree T uses dynamic programming: Select an arbitrary vertex r as root. Now select a height-1 subtree T' (a last vertex t with children on a longest path downwards and its children $x_1, \cdots, x_q$). As mentioned above, the maximum-integer-multiflow between the leaves of T' can be found by using a matching algorithm, where an edge $\{x_i, x_j\}$ is member of the matching iff there is flow (of size 1) from x_i to x_j.
This disregards flows, which might go through the edge between t and its parent t'. Using a flow that starts at x_i and goes through $\{t, t'\}$, can only pay off if the flow from x_i is not necessarily needed within T'. This is the case if there is a maximum matching in the auxiliary graph, which does not cover x_i. This can be tested in polynomial time e.g. by computing the size of a maximum matching for the auxiliary graph of $T' - x_i$. If it is less than the size of a maximum matching for the auxiliary graph of T', then using a flow from x_i through $\{t, t'\}$ cannot pay off.
Let X the set of vertices in T', whose usage can pay off. If there is a terminalpair $\{t, t''\}$ with $t'' \neq x_i$ for $i = 1, \cdots, q$, then also include t in

X, since in this case, a flow over $\{t, t'\}$ can also come from t. Let $\widetilde{X}$ the set of s_i and t_i labels that are attached to vertices in X. Delete T' from T and attach a new vertex τ to t' and give all labels in $\widetilde{X}$ to τ. A maximum-integer-multiflow on the resulting (smaller) tree determines which vertex from X to use (if any) by having a flow between some s_j and t_j, where one of them is a label of τ.

Consider the example in fig. 2.4: The max-integer-multiflow on the right

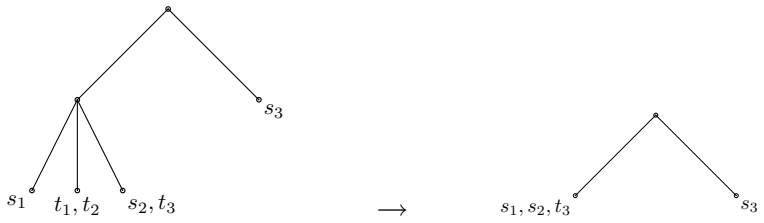


Figure 2.4:

tree is between s_3 and t_3. Therefore in T', the flow over $\{t, t'\}$ will come from the vertex labeled t_3 and within T', the flow will be used which prevails if t_3 is deleted.

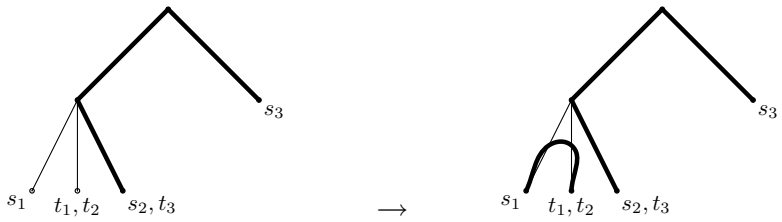


Figure 2.5:

On trees, MAX-INTEGER-MULTIFLOW becomes NP-complete already if $c(e) \in \{1, 2\}$ [49]. Garg, Vazirani and Yannakakis use many arguments for relationships between integer-multiflow problems on trees and matching problems. $c(e) \in \{1, 2\}$ is the point, where MAX-INTEGER-MULTIFLOW becomes equivalent to THREE-DIMENSIONAL MATCHING, which is a well known NP-complete problem [59, 46].

Since every integral multiflow is a multiflow, we have

$$max\text{-}integer\text{-}multiflow \leq max\text{-}multiflow$$

2.3 The Min-Multicut problem

A ***multicut*** is a set of edges $C \subseteq E$ such that in $(V, E-C)$ there is no path from an s_i to its corresponding t_i. The ***value*** of a multicut is $\sum_{e \in C} c(e)$.

A multicut is a minimum multicut or a ***min-multicut***, if there exists no multicut that has a smaller value.

The MAX-FLOW=MIN-CUT theorem from single-commodity networks does not hold for multi-commodity networks, as figure 2.6 shows. But we have

$$max\text{-}multiflow \leq min\text{-}multicut$$

because if a multicut C is removed from E, then there cannot be any flow left and successively putting $e \in C$ back into the network can increase the multiflow by at most $c(e)$ for every $e \in C$, so after putting all edges/arcs from C back in, there cannot be more flow than $\sum_{e \in C} c(e) = value(C)$ for any multicut C, in particular for a min-multicut.

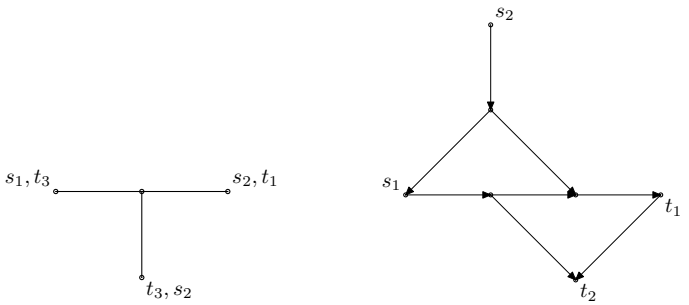


Figure 2.6: An undirected and a directed [57] uniform multi-commodity network with max-integer-multiflow = 1, max-multiflow = 1.5, min-multicut = 2

It is easy to see that MIN-MULTICUT on undirected networks $\leq_P$ MIN-MULTICUT on directed networks: simply replace every undirected edge $\{x, y\}$ with the gadget graph [48] in fig. 2.7. Obviously paths and cuts in the

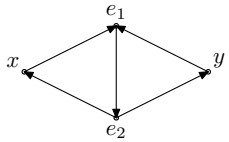


Figure 2.7:

undirected graph correspond to paths and cuts, that use the (e_1, e_2) arcs in the directed graph. From a multicut C for the undirected graph, a multicut for the directed graph can be obtained by cutting the (e_1, e_2) arcs from the gadget graphs that correspond to the edges in C. Conversely, if there is a multicut C for the directed graph, then there is a multicut C', that uses only the (e_1, e_2)-arcs. A multicut C' for the undirected graph contains all edges, whose (e_1, e_2) is in C. This reduction retains the size of the solution, so this is an FPT-reduction.

The dual problem [27] of the linear program for a MAX-MULTIFLOW instance is a linear program for a relaxed version of MIN-MULTICUT [47].

2.4 The Multiway-Cut and Multiway-Flow problems

Let (V, E, T, c) a 4-tuple containing a set of vertices V, a set of edges E, a set of terminals $T \subseteq V$ ($|T| =: l$) and a capacity-function $c : E \to \mathbb{R}^+$. The MULTIWAY-CUT problem (also called MULTITERMINAL-CUT) asks for a subset $C \subseteq E$ with a minimum $\sum_{e \in C} c(e)$, such that in $(V, E - C)$ there is no path from any $s \in T$ to any $t \neq s \in T$. Clearly MULTIWAY-CUT instances are a special case of MIN-MULTICUT, as an equivalent instance of MIN-MULTICUT can easily be constructed: $(V, E, \mathcal{P}, c)$ with $\mathcal{P} = \{(s, t) | s \in T, t \in T \backslash \{s\}\}$. (note: $|\mathcal{P}| = \binom{|T|}{2}$).

MULTIWAY-CUT is

- Polynomial on trees, using dynamic programming [19, 22]
- Polynomial on uniform DAGs [23] (see below)
- NP-complete and MaxSNP-hard on undirected uniform networks with $l = 3$ [26] (see 3.3.1)
- NP-complete and MaxSNP-hard on directed uniform networks with $l = 2$ [48] (see 3.3.2) (this seems like it was equivalent to MIN-CUT and hence polynomial so P=NP, but here all paths $s_1 - s_2$ *and* all paths $s_2 - s_1$ have to be cut)
- NP-complete on uniform planar graphs with $\Delta = 3$ and unbounded l [26]

- Not NP-complete on uniform planar graphs for bounded values of l (here it is solvable in $O((4l)^l n^{2l-1} \log(n))$ [26]), but because of the l in the exponent of n, this algorithm is not an FPT algorithm
- FPT in k [73] (see 4.2)

On (non-uniform) DAGs, splitting every $x \in T$ up into two terminals x_s, x_t, such that x_t inherits the negative neighborhood of x and x_s inherits the positive neighborhood of x, turns MULTIWAY-CUT instances into single-commodity cut instances (and thus MULTIWAY-CUT is polynomial on DAGs [23]) because *all* x_s have to be disconnected from *all* x_t, just like in single-commodity cut problems. Thereby it is crucial that the graph is acyclic, because otherwise a single-commodity cut would also include arcs to disconnect x_s from x_t, which is unnecessary in MULTIWAY-CUT. So actually MULTIWAY-CUT is polynomial on digraphs, if no terminal is on a cycle.

The MULTIWAY-FLOW problem (also called MULTITERMINAL-FLOW) asks for a function $f : V \times V \times T \to \mathbb{R}^+$, subjected to capacity-limitation and flow-preservation, such that $f(x,t,t) = 0$ for all $t \in T$ and $f(x,t_2,t_1) > f(t_2,y,t_1)$ is allowed for $t_1, t_2 \in T$. This means that flow from one terminal can go to any other terminal, but not back to its origin.
A maximum multiway-flow will never route a flow "through" a terminal $s_1 \to s_2 \to s_3$, because such a flow can be increased by splitting the flow up into two flows $s_1 \to s_2$ and $s_2 \to s_3$. This is why on directed networks, the terminals can be split like in the method for MULTIWAY-CUT on DAGs, which turns the problem on DAGs into single-commodity flow problems [23]. Thereby it is crucial that the graph is acyclic, because otherwise flow from x_s to x_t would be possible, which is forbidden in MULTIWAY-FLOW. Again, the argument also works for digraphs in which no terminal is on a cycle.

2.5 V-Cut problems

MIN-MULTICUT and MULTIWAY-CUT are the natural generalizations to the classic problems of single-commodity networks, which is why these problems get much attention. There are however several more problems on multi-commodity networks. MIN V-CUT and MIN MULTIWAY-V-CUT ask for a minimum set of *vertices* whose removal separates a given set of terminalpairs or all terminals within a set of terminals. Both of these problems have a restricted and an unrestricted version, where the ***unrestricted*** version allows deletion of terminals, which the ***restricted*** version does not allow.

Obviously UNRESTRICTED V-CUT $\leq_{FPT}$ RESTRICTED V-CUT, since every terminal t can be split into t_1, t_2 such that t_1 becomes the new terminal, t_2 inherits the neighborhood of t and an edge (or arcs in both directions) with sufficiently large capacity (or multiple parallel length-2 paths to retain uniformity and multigraph property) is added between t_1 and t_2. Deleting t in the unrestricted version now corresponds to deleting t_2 while it is forbidden to delete the terminal t_1. The edge-cut versions can easily be (P-, FPT-) reduced to unrestricted V-CUT versions, by operating on the linegraphs of the networks.

UNRESTRICTED V-CUT is

- Polynomial on uniform trees by always deleting the least common ancestor (LCA) as described in 5.3 [14].
- NP-complete on graphs of treewidth 2, because the linegraphs of trees with $\Delta = 3$ have treewidth 2 and MIN-MULTICUT is NP-complete on trees with $\Delta = 3$ (see 3.3.4)
- NP-complete on cacti (note: caci have treewidth 2), using the method presented in 3.3.5 [7]. See also [14]
- NP-complete on interval graphs [52]
- Fixed parameter intractable in tw [14]
- Fixed parameter tractable in l [73]

RESTRICTED V-CUT is

- Polynomial on interval graphs [52]
- NP-complete on trees [14]
- NP-complete on trees with $\Delta = 3$ and $pw = 2$ [52]
- FPT in k on uniform trees, by an algorithm that is very similar to the FPT algorithm for MIN-MULTICUT on uniform trees (5.3). Here, the LCA is not always deletable, since it might also be a terminal. In this case, the first deletable vertex on the paths from LCA to s_i or t_i can be deleted, which yields a searchtree of height k and a branching-factor of at most 2 [52]
- Fixed parameter intractable in tw [14]
- Fixed parameter intractable in l [52]

- Fixed parameter tractable in $\{l, tw\}$ [52]

Restricted, directed V-Cut and Unrestricted, directed V-Cut can be (P-, FPT-)reduced to directed Min-Multicut, by splitting every vertex x into x^+ and x^- such that x^+ inherits the positive neighborhood of x and x^- inherits the negative neighborhood. Turn every old arc into $k+1$ parallel length-2 paths and add the arc (x^-, x^+) for all $x \in V$.
Deleting (x^-, x^+) now corresponds to deleting x, as this destroys all paths through x and deleting the old arcs has become impossible, so only the new arcs (= the vertices) can be deleted. If x is a terminal, then

- for Unrestricted, directed V-Cut
  - make x^- the new s_i if $x = s_i$
  - make x^+ the new t_i if $x = t_i$
- for Restricted, directed V-Cut
  - do not split x.

Chapter 3

Classical complexity of Min-Multicut

3.1 Polynomial: uniform paths and cycles

On uniform paths, MIN-MULTICUT is polynomial [23]. Assume the path $(\{x_1, \cdots, x_n\}, \{\{x_i, x_i + 1\} | i = 1, \cdots, n-1\})$ is arranged on a horizontal line. Since the graph is undirected, it can be assumed w.l.o.g. that for all the terminalpairs $\{s_i, t_i\}$ t_i is to the right of s_i. The edge on the left of the leftmost t_i can be cut, because since s_i is on the left of t_i, there has to be at least one cut to the left of t_i. Since t_i is the leftmost t, cutting any edge further to the left can only downgrade the solution, because edges further to the left might not cut a terminalpair $\{s_j, t_j\}$ whose s_j is between t_i and the edge further to the left, so the edge on the left of t_i is an optimal choice. After cutting this edge, every disconnected terminalpair can be removed from $\mathcal{P}$ and the algorithm can start over on the remaining path on the right of t_i.
The following algorithm iterates over the vertices from left to right, collects all indices j of terminalpairs $\{s_j, t_j\}$ that are not disconnected yet in a set S. Whenever a t_j is met, the edge on its left is cut, unless $\{s_j, t_j\}$ is already disconnected, which is the case if $j \notin S$.

$S := \emptyset$
for $i = 1, \cdots, n$ **do**
    if $x_i = t_j$ for a $j \in S$ **then**
        cut $\{x_{i-1}, x_i\} \in E$
        $S := \emptyset$
    fi
    $S := S \cup \{j | \{s_j, t_j\} \in \mathcal{P}$ with $x_i = s_j\}$
done

The algorithm obviously has a running-time of $O(n^2)$, considering that for every x_i there can be at most $n-1$ vertices from which x_i must be disconnected. Actually every vertex only needs to be disconnected from at most 2 other vertices, namely the closest ones in both directions, that are together with x_i in $\mathcal{P}$, but removing all other terminalpairs takes $O(n^2)$ either.

On uniform cycles, a minimum multicut can easily be found by trying each edge as first cut and using the $O(n^2)$ algorithm for the remaining path, so the running-time is $O(n^3)$. This can be improved by excluding the edges that have already been tested as first cut, but the complexity class $O\left(\sum_{i=1}^{n} i^2\right) = O\left(\frac{n\cdot(n+1)\cdot(2\cdot n+1)}{6}\right) = O(n^3)$ stays the same.

3.2 Polynomial: undirected networks with $l = 2$

For *undirected* multi-commodity networks with $l = 2$, Hu showed [56] that MIN-MULTICUT = MAX-MULTIFLOW holds, but Itai later showed [57] that Hu's proof only works for integral and rational capacities. For real capacities it failed because Hu had proven that his algorithm for multiflow with $l = 2$ only terminated when the constructed multiflow defined a multicut. The problem with this proof was that his algorithm can run infinitely on multi-commodity networks with real capacities. Itai's $O(n^3)$ algorithm did not have this problem, so his algorithm was conclusive proof for the equality for real capacities.
On *directed* uniform multi-commodity networks with $l = 2$, MIN-MULTICUT is already NP-complete (see 3.3.2).

The following algorithm is Hu's algorithm. Itai's corrections are very much like Edmonds' and Karp's correction to Ford's and Fulkerson's algorithm. (for completeness it should be added that Hu's algorithm is older than Edmonds' and Karp's algorithm).

The algorithm starts by determining a max-flow for $\{s_1, t_1\}$. Of course the $s_1 - t_1$ flow might block edges that $s_2 - t_2$ needs to use (see fig. 2.1), so in the second phase, flow of commodity 1 is rerouted. This is done by selecting (not necessarily edge-disjoint) pairs of so called ***forward- and backward paths*** between s_2 and t_2.
The forward path is only allowed to use edges $\{x, y\}$ (w.l.o.g. the path visits

x before y) if

$$\underbrace{f(x,y,1) - f(y,x,1)}_{\text{1-flow from } x \text{ to } y} + \underbrace{f(x,y,2) - f(y,x,2)}_{\text{2-flow from } x \text{ to } y} < c(\{x,y\})$$

Note that flow from y to x counts negatively, so the above term can be smaller than $c(\{x,y\})$ by having flow from y to x, which can be reversed. The backward path is only allowed to use edges $\{x,y\}$ (w.l.o.g. the path visits x before y, so on the path, y is closer to s_2), if

$$\underbrace{f(x,y,1) - f(y,x,1)}_{\text{1-flow from } x \text{ to } y} + \underbrace{f(y,x,2) - f(x,y,2)}_{\text{2-flow from } y \text{ to } x} < c(\{x,y\})$$

Along the forward path, the flow of both commodities is increased. This can mean increasing flow on unsaturated edges, but it can also mean decreasing flow in the opposite direction. Since $s_1 \neq s_2$ or $t_1 \neq t_2$, increasing 1-flow on an $s_2 - t_2$ path violates the flow preservation. t_2 will have 1-overflow, while s_2 will have 1-underflow. This is counterbalanced through the backward path. Along the backward path $t_2 - s_2$, the flow of commodity 1 is increased and flow of commodity 2 is decreased. On edges that are shared by both paths, this means that

- 1-flow is increased in both directions (increasing it in one direction annulls the increased flow in the other direction)
- 2-flow is increased on the forward path and decreased on the backward path, which means that in both cases the 2-flow towards t_2 is increased

On edges that are only on the forward path, the flow of both commodities towards t_2 is increased, on edges that are only on the backward path, the 1-flow towards t_2 is decreased (or in other words, the 1-flow towards s_2 is increased). So on the cycles between the paths, the 1-flow is rerouted and 2-flow is split up and routed towards t_2 partially along the forward path and partially along the backward path.
In fig. 3.1, the changes along the forward path are indicated by the black arrows in the upper half of the picture, the changes along the backward path are indicated by the black arrows in the lower half. The gray arrows indicate how the flow is changed by these operations.

The algorithm does not guarantee integral flow of the commodities on every edge (Maximum Integer Multiflow was already NP-complete on (un)directed, uniform networks with $l = 2$), but it guarantees the flow of every commodity on every edge to be a multiple of $\frac{1}{2}$ if all capacities are

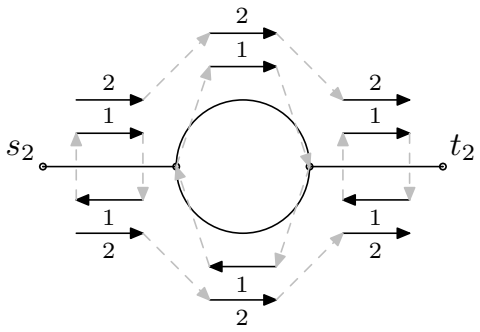


Figure 3.1: The changes along the forward- and backward path in Hu's algorithm

integral (this property is called ***half-integrality***). Integrality follows, if all capacities are even integers, because in this case all capacities can be halved, then a half-integral multiflow can be determined and doubled.

3.3 NP-completeness

Besides the proofs that are presented here, MIN-MULTICUT is also NP-complete for interval graphs [52], grids [9] and walls [14]. Membership in NP is obvious for every variant of MULTICUT, because a nondeterministic Turing machine can (nondeterministically) generate a solution and test it for validity in deterministic polynomial time, so only NP-hardness remains to be proven.

3.3.1 Uniform undirected networks with $l = 3$

For $l = 3$ MIN-MULTICUT on undirected, uniform multi-commodity networks is already NP-hard [25]:

Let (V, E) a graph. The MAX-CUT problem asks for a partitioning of $V = V_1 \dot\cup V_2$, such that $|\{\{x, y\} \in E | x \in V_1, y \in V_2\}|$ becomes maximal. MAX-CUT is NP-complete [59, 46] and it can be P-reduced to MULTIWAY-CUT with 3 terminals, which can trivially be P-reduced to MIN-MULTICUT with 3 terminalpairs.

Theorem 3.3.1.1. MAX-CUT $\leq_P$ UNIFORM MULTIWAY CUT WITH $l = 3$

Proof. Let (V, E) a graph. Add 3 vertices s_1, s_2, s_3 to the graph, which are the terminals that have to be disconnected from each other $T = \{s_1, s_2, s_3\}$. Replace every edge $\{x, y\}$ by two paths of length 3 (x, x_1, y_1, y)

and (x, x_2, y_2, y). Connect the terminals s_1, s_2, s_3 to the vertices of these paths in the following way:

1. s_1 is connected to x, x_2, y, y_1
2. s_2 is connected to x, x_1, y, y_2
3. s_3 is connected to x_1, x_2, y_1, y_2

The edges that are incident to s_i get capacity 4, but this definition can be changed to adding 4 paths of length 2, to work for uniform multigraphs. An (i, j)-separator is a set that disconnects s_1, s_2, s_3 from each other while keeping x connected to s_i and y connected to s_j. A $(1, 2)$-separator with value 27 is

$$\{\{s_1, y\}, \{s_1, y'\}, \{y, y'\}, \{s_2, x\}, \{s_2, x'\}, \{x, x'\}, \{s_3, x''\}, \{s_3, y''\}, \{x'', y''\}\}$$

and a $(2, 1)$-separator with value 27 is

$$\{\{s_1, x\}, \{s_1, x''\}, \{x, x''\}, \{s_2, y\}, \{s_2, y''\}, \{y.y'', \}, \{s_3, x'\}, \{s_3, y'\}, \{x', y'\}\}.$$

For every other pair (i, j), a minimum (i, j)-separator has value 28 or more. In particular the minimum $(1, 1)$- and $(2, 2)$-separators (consider that (i, i)-separators leave x and y connected) have value 28.

(V, E) has a max-cut of size k if and only if the resulting Multiway-Cut instance with 3 terminals has a Multiway-Cut of size $\leq 28 \cdot |E| - k$. The edges between the partitions of a max-cut of (V, E) correspond to the $(1, 2)$- or $(2, 1)$-separators and the partitions of the max-cut correspond to the sets of vertices that stay connected to s_1 and s_2 respectively. The connections between s_1, s_2, s_3 on the gadget graphs for the edges within V_1 and V_2 are destroyed by $(1, 1)$-separators or $(2, 2)$-separators respectively. □

3.3.2 Uniform directed networks with $l = 2$

Garg, Vazirani and Yannakakis reduced undirected Multiway-Cut with $l = 3$ (see 3.3.1) to Multiway-Cut on uniform digraphs with $l = 2$[1] (which can trivially be P-reduced to directed uniform min-multicut with $l = 2$). For this, replace every edge $\{x, y\}$ with the gadget graph from fig. 3.2 (this is the FPT-reduction from undirected to directed Min-Multicut from 2.3). The 3 old terminals s_1, s_2, s_3 now get connected to 2 new terminals s_1', s_2' as shown in fig. 3.3. Where the arcs between the old and the new terminals get

[1]This is no contradiction to the polynomiality of Min-Cut because in directed Multiway-Cut with $l = 2$, all the $s_1 - s_2$ paths, but also all $s_2 - s_1$ paths must be destroyed (and unlike in the undirected case, the paths do not necessarily coincide).

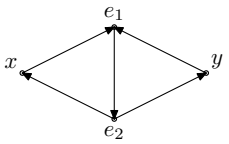


Figure 3.2:

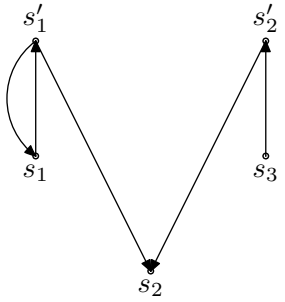


Figure 3.3:

a large capacity (or many multiple parallel paths of length 2), so that the paths between the new terminals can only be destroyed by destroying paths between the old terminals. Whenever there is a path between two of the old terminals left, there is a path in one direction between the new terminals.

3.3.3 Uniform, undirected stars

There is a simple P-reduction from Vertex Cover (which is NP-complete [59, 46]) to Min-Multicut instances on stars [49]. So Min-Multicut is already NP-hard for uniform, undirected stars, which is a very strong result, because stars are very simple structures.

Theorem 3.3.3.1. Vertex Cover $\leq_P$ Min-Multicut on stars

Proof. Let (w, k) with $w = (V, E)$ an instance of Vertex Cover. Now construct a star (V', E') and a set of terminalpairs $\mathcal{P}$. $V' = V \dot{\cup} \{c\}$, the new vertex c will be the center of the star; every old vertex will be connected to it $E' = \{\{c, x\} | x \in V\}$. The old edges are turned into terminalpairs $\mathcal{P} = \{\{s, t\} | \{s, t\} \in E\}$ (this means $\mathcal{P} = E$). To destroy an $s - t$ path, one can remove the edge $\{s, c\}$ or the edge $\{t, c\}$. Removing an edge $\{x, c\}$ corresponds to selecting $x \in V$ as part of a vertex cover.
Show that (V, E) has a vertex cover of size k if and only if $(V', E', \mathcal{P})$ has a

multicut of size k:

Let $C = \{x_1, \cdots, x_k\} \subseteq V$ a vertex cover of size k for (V, E), set $C' = \{\{x_1, c\}, \cdots, \{x_k, c\}\}$. Suppose there was a pair $\{s, t\} \in \mathcal{P}$ and a path (s, c, t) left in $(V', E' - C')$, then $\{c, s\}$ and $\{c, t\}$ are not in C'. Hence s and t are not in C, but $\{s, t\} \in \mathcal{P}$ means that there is an edge $\{s, t\}$ in E, which would not be covered by C, so C is not a vertex cover of (V, E). $\lightning$

Let $C' = \{\{c, x_1\}, \cdots, \{c, x_k\}\}$ a multicut of size k for $(V', E', \mathcal{P})$, set $C = \{x_1, \cdots, x_k\}$. Suppose there was an edge $\{x, y\} \in E$ that was not covered by C, then x and y are not in C. Hence $\{c, x\}$ and $\{c, y\}$ are not in C', but $\{x, y\} \in E$ means that there is a terminalpair $\{x, y\} \in \mathcal{P}$, which would not be cut by C', so C' would not be a multicut for $(V', E', \mathcal{P})$. $\lightning$ □

The NP-hardness on stars implies NP-hardness for caterpillars, trees, cacti, planar, bipartite and perfect (also known as Berge) graphs. Note that the reduction retains the size of a solution, so this is also an FPT-reduction.

3.3.4 Uniform, undirected trees with $\Delta = 3$

The problem 3SAT or 3CNF-SAT asks for the satisfiability of propositional logic formulas in conjunctive normal form (CNF) with exactly 3 literals per clause $\bigwedge(x \vee y \vee z)$. This is a famous, NP-complete problem [46] and there is a P-reduction to Min-Multicut on trees with $\Delta = 3$ by Călinescu, Fernandes and Reed [14].

Theorem 3.3.4.1. 3SAT $\leq_P$ Min-Multicut on uniform, undirected trees with $\Delta = 3$

Proof. Let φ a 3CNF formula with n variables and m clauses. For every variable x_i, a gadget graph with 3 vertices is constructed: a root with 2 children that are labeled with x_i and $\overline{x_i}$. For each of these gadgets, a terminalpair $\{x_i, \overline{x_i}\}$ will be part of $\mathcal{P}$. For every clause $C_j = (x \vee y \vee z)$, a gadget graph with 5 vertices is constructed: a root with 2 children, one of which is labeled z, the other one is an internal vertex w_j, that has x and y as children. For these gadgets, two terminalpairs $\{x, y\}$ and $\{z, w_j\}$ are part of $\mathcal{P}$. Finally, the roots of the gadget graphs are connected by an arbitrary binary tree and all the literals in the clause-gadgets become terminalpairs with their respective vertex from the variable-gadgets.

It is easy to see that $2m + n$ cuts are already needed to separate the terminalpairs of the gadget graphs alone, because every variable-gadget requires 1 cut and every clause-gadget requires 2 cuts.

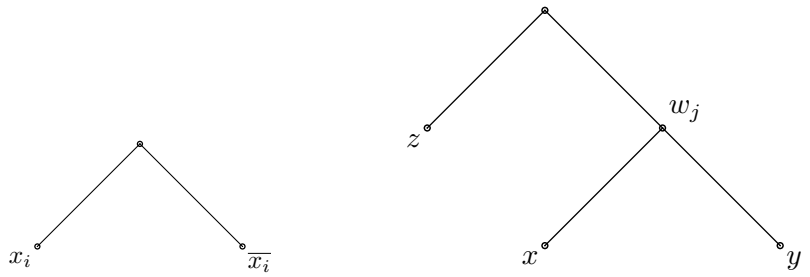


Figure 3.4: Gadgets for variable x_i and clause $C_j = (x \vee y \vee z)$

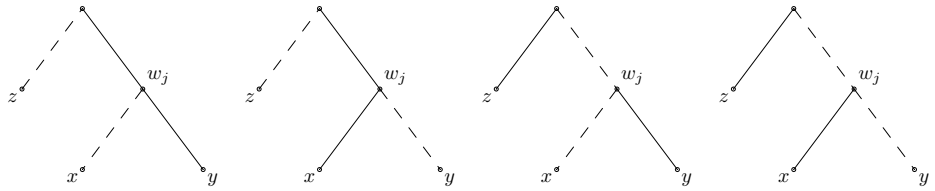


Figure 3.5: The possible cuts for $\{z, w_j\}, \{x, y\} \in \mathcal{P}$ with 2 cuts

These $2m + n$ cuts can be arranged so that no further cuts are required, if and only if φ is satisfiable. As you can see in fig. 3.5, every separation of $\{z, w_j\}$ and $\{x, y\}$ that uses only 2 cuts, leaves a path between the root and one literal. This literal will satisfy the clause in a satisfying assignment. It can only be disconnected from its corresponding vertex in the variable-gadget by using the one cut of the variable-gadget on the edge to the vertex that has the same sign as the literal from the clause-gadget. This cut in the variable-gadget corresponds to the truth assignment to that variable. □

If the roots were not connected by a binary tree, but by a path, then Δ would be 4 (still constant), but the pathwidth would become 2, because 3 policemen are needed to catch a fugitive without knowing where he is: one blocks the path, one occupies w_j and the last one successively visits the leaves. If the fugitive has not been found, then proceed to the next vertex of the path.

3.3.5 DAGs, 13-layer digraphs, caterpillars with $\Delta = 5$

Bentz generalized the proof for NP-completeness of Min-Multicut on trees with $\Delta = 3$ (3.3.4) so that it works for many other Multicut versions and graph classes [7]. The general method requires only gadget graphs for variables and clauses and a component Q that connects every variable-gadget

to every clause-gadget.

- The gadget-graph of a variable x_i must have two vertices x_i, $\overline{x_i}$ and a set of terminalpairs $\mathcal{P}' \subseteq \mathcal{P}$ such that there are two possible deletions[2], that both disconnect all terminalpairs in $\mathcal{P}'$, but one disconnects x_i from Q and keeps $\overline{x_i}$ connected to Q, the other one disconnects $\overline{x_i}$ from Q and keeps x_i connected to Q. No single deletion may disconnect all terminalpairs in $\mathcal{P}'$ and both x_i and $\overline{x_i}$ from Q.

- The gadget graph of a clause $C_j = x \vee y \vee z$ contains three vertices x, y, z and a set of terminalpairs $\mathcal{P}' \subseteq \mathcal{P}$. There are three[3] sets of two elements d_x, d_y, d_z. The removal of the elements of any set must disconnect all terminalpairs in $\mathcal{P}'$, but d_α leaves α connected to Q and disconnects the other ones from Q. No deletion of two elements may disconnect all terminalpairs in $\mathcal{P}'$ and all of x, y, z from Q.

- All the literals in the clause-gadgets must become terminalpairs with their respective vertex from the variable-gadgets.

If these conditions are satisfied, then the argumentation for trees with $\Delta = 3$ is applicable in the same way to show NP-completeness of the given MULTICUT version on the given graph class.

The gadgets for MIN-MULTICUT on uniform DAGs are shown in fig. 3.6. For variable-gadgets, a terminalpair $(\overline{x_i}, x_i')$ is in $\mathcal{P}$. For clause-gadgets, the terminalpairs (z', w_j) and (x', y) are added. Q can be a directed path of length $n + m$, where the variable-gadgets are attached to the first n vertices of the path, followed by the m clause-gadgets. The resulting DAG has $\Delta^+ = 2$ and $\Delta^- = 2$.

A ***q*-layer digraph** [55] is a digraph whose vertex set can be partitioned into q partitions $V = V_1 \dot\cup \cdots \dot\cup V_q$ such that every arc (x, y) has $x \in V_i$ and $y \in V_{i+1}$ for some i. Of course every q-layer digraph is a DAG. The DAGs constructed before are 13-layer digraphs if for Q, a single vertex is used and the diagonal arcs that go from upper left to lower right, are subdivided twice. The first 4 layers are for the variable gadgets, the fifth layer is Q, followed by 8 layers for the clause gadgets.

[2]It depends on the MULTICUT version, which elements can be deleted

[3]This is actually a slight generalization of Bentz' result, because he stayed closer to Călinescu, Fernandes and Reed. He demanded four sets, two of which preserve a connection between z and Q. Also he demanded exactly one terminalpair in variable-gadgets and exactly two terminalpairs in clause-gadgets and he demanded the sets to be disjoint.

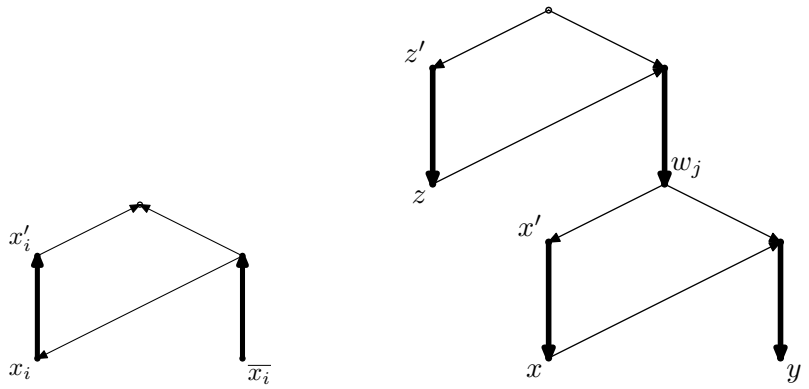


Figure 3.6: Gadgets for variable x_i and clause $C_j = (x \vee y \vee z)$

The proof for NP-completeness of MIN-MULTICUT on uniform caterpillars with $\Delta = 5$ by Guo, Hüffner, Kenar, Niedermeier and Uhlmann [52] is older than Bentz' generalization, but it is actually an application of Bentz' result, since they specify gadget graphs and then "manually" argue in the same way as Călinescu, Fernandes and Reed.

3.4 Approximability

MAX-MULTIFLOW=MIN-MULTICUT does not hold, but much attention was paid to the gap between these values. The results in this area always came along with approximation algorithms whose quality equal the boundaries on the gap. Most of the algorithms work by searching a multicut, using the result of a linear program for the relaxed problem. Leighton and Rao [71] were the first who found an approximation algorithm. It has a quality of $O(\log(n))$ and works for uniform multi-commodity networks. This result was generalized by Klein, Agrawal, Ravi and Rao [64] who used a similar technique to obtain an approximation for general multi-commodity networks, which has a quality of $O(\log(C) \cdot \log(D))$ where C is the sum of all capacities and D is the sum of the minimal requirements (2.1.2). Tragoudas [89] improved the quality to $O(\log(n) \cdot \log(D))$. Garg, Vazirani and Yannakakis [47] improved this further to $O(\log(l) \cdot \log(D))$. Finally Plotkin and Tardos [79] showed an $O(\log(l)^2)$ approximation.

For planar networks, Klein, Plotkin and Rao [65] found an $O(\log(D))$ approximation, which was improved by Plotkin and Tardos [79] to $O(\log(l))$ and finally Tardos and Vazirani [37] found an $O(r^3)$ approximation for $K_{r,r}$-

free graphs, which is an $O(3^3) = O(1)$ approximation for planar graphs, since planar graphs are $K_{3,3}$-free [69]. For trees, Garg, Vazirani and Yannakakis [49] presented a 2-approximation algorithm that simultaneously is a $\frac{1}{2}$-approximation for MAX-INTEGER-MULTIFLOW on trees. A simple 2-approximation algorithm for uniform trees is mentioned in 5.3.

To answer the question for a polynomial time approximation scheme (PTAS) or even a fully polynomial time approximation scheme (FPTAS), the complexity class ***MaxSNP*** is needed. MaxSNP is defined [78] using Fagin's characterization [40] of NP as the class of problems that are axiomatizable in second-order logic with a formula of the form

$$\exists S.\forall \overline{x}.\exists \overline{y}.\psi(\overline{x},\overline{y},G,S)$$

where the first $\exists$ quantifies a structure, the other 2 quantifiers quantify vectors and ψ is a quantifier-free formula. ***Strict NP*** (***SNP***) is restricted to formulas of the form

$$\exists S.\forall \overline{x}.\psi(\overline{x},G,S)$$

although this is no actual restriction, since the existential quantifiers can be replaced by function symbols in the quantified structure S (Skolem normalform). MaxNP and MaxSNP_0 now don't ask for the existence of a structure S that satisfies ψ for *all* assignments to x, but that satisfies ψ for a maximum number of assignments to x.

MaxNP : $\max_S |\{\overline{x}|\exists \overline{y}.\psi(\overline{x},\overline{y},G,S)\}|$

$\mathbf{MaxSNP}_0$: $\max_S |\{\overline{x}|\psi(\overline{x},G,S)\}|$

The class MaxSNP is the closure of MaxSNP_0 under L-reductions, which are defined as follows: Let $\mathcal{L}$ an optimization-problem over Σ with optimal solutions $opt : \Sigma^* \to \mathbb{R}$ and $\mathcal{L}'$ an optimization-problem over Γ with optimal solutions $opt' : \Gamma^* \to \mathbb{R}$. A function $g : \Sigma^* \to \Gamma^*$ is an ***L-reduction***, if there are two constants α, β, such that

- $opt(w) \leq \alpha \cdot opt'(g(w))$ for all $w \in \Sigma^*$
- From a solution for $g(w)$ with cost c, a solution for w can be constructed in polynomial time with cost at most $opt(w) + \beta(c - opt'(g(w)))$

This definition regards minimization-problems. If at least one of $\mathcal{L}$ and $\mathcal{L}'$ is a maximization-problem, then there are slightly different definitions in [78]. Note that L-reductions preserve polynomial approximability. From

a polynomial approximation for $\mathcal{L}'$ with quality $1+\varepsilon$, a polynomial approximation for $\mathcal{L}$ can be constructed with quality $1+\varepsilon\alpha\beta$.

Min-Multicut is MaxSNP-hard, because the reduction in 3.3.3.1 is an L-reduction with $\alpha = 1$ and $\beta = 1$ and Vertex Cover is MaxSNP-hard, because its special case Vertex Cover-B is MaxSNP-hard [78]. The MaxSNP-hardness implies [4] that there is no PTAS (unless P=NP) and hence no FPTAS for Min-Multicut and not even for Min-Multicut on stars. This is bad news, since the existence of an FPTAS would have entailed fixed parameter tractability in k [13, 5]. Călinescu, Fernandes and Reed found a PTAS for Unrestricted Min-V-Cut on uniform networks with bounded treewidth which extends to Min-Multicut with bounded degree and bounded treewidth, since the linegraph of a network with bounded degree and bounded treewidth, has bounded treewidth [14].

A recent result by Chawla, Krauthgamer, Kumar, Rabani and Sivakumar [16] says that Min-Multicut is not even approximable to within a *constant* ratio or even a ratio of $\Omega(\log(\log(n)))$, assuming Khot's [62] unique games conjecture is true.

For further details refer to [8], which is a bibliography on Min-Multicut and Max-Multiflow, that mostly contains works on approximability.

Chapter 4

Parameterized complexity of Min-Multicut

4.1 Fixed parameter intractability

Using lemma 1.4.1.2 and the NP-completeness of MIN-MULTICUT on uniform stars (3.3.3), fixed parameter tractability of MIN-MULTICUT can be ruled out for many sets of possible parameters. Denotations are taken from [91]. (Nontrivial), uniform stars $(\{c, x_1, \cdots, x_n\}, \{\{c, x_i\} | i = 1, \cdots, n\})$ simultaneously have

- Treewidth $tw = 1$, as they are trees
- Pathwidth $pw = 1$, because any ordering of $\{c, x_1\}, \cdots, \{c, x_n\}$ is a path-decomposition
- No cycles $\nu = 0$, because trees by definition have no cycles
- A maximum path length of $\Lambda = 2$ between any terminalpair, since every vertex is either the central vertex or connected to it, so there exists a path ≤ 2 between any pair of vertices and paths in trees are unique
- A diameter $dm = 2$, because there is no pair of vertices with minimum distance > 2, as their maximum distance is 2
- A radius of $r = 1$, because the central vertex does not have a distance > 1 to any vertex
- A 2-coloring, because trees are bipartite, so $\chi = 2$

- No complete 3-coloring, because a second and third color (after the color for c) could not be connected, as all edges contain c. Hence the achromatic number is $\psi = 2$
- A pseudoachromatic number of $\psi_s = 2$, also because no complete 3-coloring exists.
- No clique > 2, because that would imply a cycle, so $\omega = 2$ for all trees
- A maximum matching of size $\alpha_0 = 1$, containing $\{c, x_i\}$ for some i. No matching can contain more edges, because they are all incident to c
- A vertex cover of size $\beta = 1$ (containing just c)
- A dominating set of size $\gamma = 1$ (containing just c)
- A bondage number of $b = 1$, because in stars, the deletion of any edge makes it necessary to include the incident end-vertex into a dominating set
- An inclusion-maximal irredundance set of size $ir = 1$ (containing just c)
- A vertex separator of size $\sigma = 1$ (removing a neighbor of an end-vertex separates the end-vertex from all other vertices and all trees have at least 2 end-vertices)
- An edge separator of size $\lambda = 1$ (removing an edge that is incident to an end-vertex separates the end-vertex from the other vertices and all trees have at least 2 end-vertices)
- Vertices of degree 1 (every tree contains 2 end-vertices), so the minimum degree is $\delta = 1$
- An average degree of $\varphi = \frac{2|E|}{n-1} = 2$
- An index of $\mu = 0$, because trees have index 0
- Exactly one spanning tree, which is the star itself, so $z = 1$

Also stars have many parameters close to n, so that their distances to n are also parameters in which MIN-MULTICUT simultaneously is not FPT. stars have

- A vertex of degree $\Delta = n - 1$, namely c

- A smallest clique-decomposition of size $\Theta = n-1$, because a maximum clique has size 2 and therefore all n edges have to be in different cliques for all trees
- A smallest edge coloring of size $\chi' = n-1$, because all edges are incident to c, so all edges must have different colors
- A total coloring of size $\chi_T = n$, containing the $n-1$ colors necessary for the edges, one additional color for c (all edges are incident to c, so no color can be reused) and the vertex x_i can be colored with the same color as edge $\{c, x_{(i \bmod n)+1}\}$
- A maximum independent set of size $\alpha = n-1$, containing all vertices except c
- An edge cover of size $\beta_0 = n-1$, because every x_i can only be covered by the edge $\{c, x_i\}$
- All vertices except c are end-vertices, so $|\Gamma| = n-1$

These facts yield

Theorem 4.1.1. Min-Multicut *is not FPT in any subset of*

$$\{tw, pw, \nu, \Lambda, dm, r, \chi, \psi, \psi_s, \omega, \alpha_0, \beta, \gamma, b, ir, \sigma, \lambda, \delta, \varphi, \mu, z$$

$$n-\Delta, n-\Theta, n-\chi', n-\chi_T, n-\alpha, n-\beta_0, n-|\Gamma|\}$$

Also using lemma 1.4.1.2, the NP-completeness on trees with $\Delta = 4$ and $pw = 2$ (3.3.4) implies that Min-Multicut is not FPT in any subset of

$$\{\Delta, tw, pw, \nu, \chi, \omega, \sigma, \lambda, \delta, \varphi, \mu, z, n-\Theta\}$$

The Min-Multicut instances, that are constructed in 3.3.1 have[1]

- $l = 3$
- Every vertex has distance ≤ 3 to s_3, so the diameter is $dm \leq 6$
- The radius is $r \leq 6$, because $r \leq dm$ [91]
- The vertices that are adjacent to the terminals s_i have degree 2, so $\delta \leq 2$

[1]Assuming the case where the terminals s_i are connected to the other vertices using 4 parallel length 2 paths instead of 1 edge with capacity 4

- When the MAX-CUT instance has n vertices and m edges, then the constructed MIN-MULTICUT instance has $n + 4m + 3 + 48m$ vertices and $6m + 96m$ edges, so the average degree is

$$\begin{aligned} \varphi &= \frac{2 \cdot (6m + 96m)}{n + 4m + 3 + 48m - 1} \\ &= \frac{204m}{n + 52m + 2} \\ &\leq \frac{204m}{52m} \\ &= \frac{51}{13} \\ &< 4 \end{aligned}$$

- A vertex separator of size $\sigma \leq 2$ because $\sigma \leq \delta$. Removing all adjacent vertices of a vertex with degree δ separates that vertex from the rest graph.

- An edge separator of size $\lambda \leq 2$ because $\lambda \leq \delta$. Removing all incident edges of a vertex with degree δ separates that vertex from the rest graph.

So using lemma 1.4.1.2, it can be concluded that MIN-MULTICUT is not FPT in any subset of

$$\{l, dm, r, \delta, \varphi, \sigma, \lambda\}$$

4.2 Fixed parameter tractability in $\{l, k\}$

D. Marx provided an $O^*(k^k 4^{k^3})$ FPT algorithm [73] for MULTIWAY V-CUT and modified it to solve MINIMUM V-CUT in $O^*(4^{k(l-1)} k^k 4^{k^3})$, which is a running-time that is FPT in $\{l, k\}$. MIN-MULTICUT can be FPT-reduced to MINIMUM V-CUT by working on the linegraph.
The algorithm for MULTIWAY V-CUT is a bounded searchtree algorithm that selects an arbitrary terminal $t \in T$ that is not yet separated from $T - t$ and branches over the $O(k4^{k^2})$ important separators (see below) that have a size of at most k. In every branch k is decreased by at least 1, so this algorithm has a running-time of $O^*((k4^{k^2})^k) = O^*(k^k 4^{k^3})$.

Let S an (X, Y)-separator (a set of vertices whose removal disconnects all vertices in X from all vertices in Y). An (X, Y)-separator S' ***dominates*** S if $|S'| \leq |S|$ and $R(X, S) \subsetneq R(X, S')$, where $R(X, S)$ denotes the set of all

vertices that are reachable from $X - S$ in $(V - S, E)$. This means that S' is "closer" to Y and not larger than S. An ***important*** (X, Y)***-separator*** is an exclusion-minimal, undominated (X, Y)-separator.

Theorem 4.2.1. *For any X, Y there are at most 4^{k^2} important (X, Y)-separators of size k (independent of the graph)*

Proof. Induction

Induction foundation $k = 1$ A separator of size 1 can only be a cut-vertex. Only the cut-vertex that is closest to Y and farthest from X is important because it dominates all other cut-vertices between X and Y.

Induction hypothesis For a fixed k there are at most $4^{(k-1)^2}$ important (X, Y)-separators of size $k - 1$.

Induction step Let S an important (X, Y)-separator of size k and H another important (X, Y)-separator of size k. The question is how many possibilities there are for H.

Case 1 $Z := S \cap H \neq \emptyset$, then $H \backslash Z$ is an important $(X \backslash Z, Y \backslash Z)$-separator in $(V \backslash Z, E)$ of size $\leq k - 1$, so the induction hypothesis says there are at most $4^{(k-1)^2}$ candidates for H per Z and since $Z \subsetneq S$, there are at most 2^k candidates for Z, so there are at most $2^k 4^{(k-1)^2}$ such H.

Case 2 $S \cap H = \emptyset$. Divide H into $H_1 = H \cap R(X, S)$ and $H_2 = H \cap R(Y, S)$ and divide S into $S_1 = S \cap R(X, H)$ and $S_2 = S \cap R(Y, H)$ (see fig. 4.1).

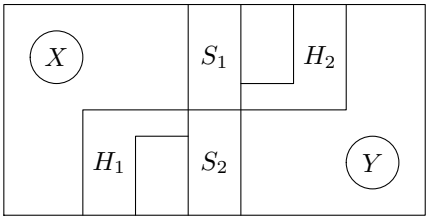


Figure 4.1: The partitionings $H = H_1 \dot\cup H_2, S = S_1 \dot\cup S_2$

If $H_1 = \emptyset$ then $R(X, S) \cup S \subsetneq R(X, H)$, (in the picture H would be on the right of S) and hence H would dominate S. Conversely, if $H_2 = \emptyset$ then $R(X, H) \cup H \subsetneq R(X, S)$, ($H$ would be on the left

of S) and hence S would dominate H. Therefore, $H_1 \neq \emptyset$ and $H_2 \neq \emptyset$ and in particular $|H_1| \leq k-1$ and $|H_2| \leq k-1$.

H_1 is an $(X \cup S_1, S_2)$-separator in $(V \cap R(X,S) \cup S, E)$ which in the picture is the left part of the graph up to and including S. H_2 is an $(S_1, Y \cup S_2)$-separator in $(V \cap R(Y,S) \cup S, E)$ which in the picture is the right part of the graph up to and including S. These are both important separators, since if H_1 was dominated, then the dominating separator could be united with H_2 to obtain a separator that would dominate H and if H_2 was dominated, then the dominating separator could be united with H_1 to obtain a separator that would dominate H, so in both cases H would not be important.

Since H_1 and H_2 are important separators of size $\leq k-1$, the induction hypothesis says there are at most 4^{i^2} such H_1 (where $i = |H_1|$) and $4^{(k-i)^2}$ such H_2 per partitioning $S = S_1 \dot\cup S_2$ and there are at most 2^k such partitionings because $|S| = k$.

Altogether the number of important (X,Y)-separators is at most

$$\begin{aligned}
& \overbrace{1}^{S} + \overbrace{2^k 4^{(k-1)^2}}^{\text{Case 1}} + \overbrace{\sum_{i=1}^{k-1} 2^k 4^{i^2} 4^{(k-i)^2}}^{\text{Case 2}} \\
\leq\ & 1 + 2^k 4^{(k-1)^2} + (k-1) 2^k 4^{(k-1)^2+1} \\
\leq\ & k 2^k 4^{(k-1)^2+1} \\
\leq\ & 4^k 4^{(k-1)^2+1} \\
=\ & 4^{k+k^2-2k+2} \\
\leq\ & 4^{k^2}
\end{aligned}$$

□

The constructions in the cases of the induction step yield an algorithm to enumerate all important separators (so it is possible to branch over them): Select an initial separator S_0 using flow algorithms. Construct an important separator S by removing redundant vertices (vertices s for which $S - s$ is still a separator), removing subsets of S_0 and using flow algorithms again. Then enumerate subsets $Z \subsetneq S$ and recursively enumerate the important $(X \backslash Z, Y \backslash Z)$-separators of size $k - |Z|$ in $(V \backslash Z, E)$ and unite them with Z. Finally enumerate subsets $S_1 \subsetneq S$ and recursively enumerate the important

$(X \cup S_1, S_2)$-separators H_1 in $(V \cap R(X,S) \cup S, E)$ of size $\leq k$ and unite each of them with each important $(S_1, Y \cup S_2)$-separator H_2 in $(V \cap R(Y,S) \cup S, E)$ of size $k - |H_1|$.

For every $t \in T$ there exists a minimum multiway-cut that contains an important $(\{t\}, T-t)$-separator: Such a separator can be obtained by "pushing" any minimum multiway-cut S towards $T-t$ and removing redundant vertices. Thereby no path $t_i - t_j$ with $t_i, t_j \in T-t$ can be connected again, because this would also establish paths $t - t_i$ and $t - t_j$, so the resulting important $(\{t\}, T-t)$-separator would not separate t from $T-t$. Therefore branching over all $\sum_{i=1}^{k} 4^{i^2} \leq k4^{k^2}$ important separators is complete.

The modified algorithm for MINIMUM V-CUT selects an arbitrary terminalpair $\{s_i, t_i\} \in \mathcal{P}$ and branches over the important $(\{s_i\}, T \cup \{t_i\})$-separators for some $T \subseteq \left(\bigcup_{j=1}^{l} \{s_j, t_j\} \right) \backslash \{s_i\}$. As before, there are $O(k4^{k^2})$ possibilities per subset T and since $|\bigcup_{j=1}^{l} \{s_j, t_j\}| \leq 2l$, there are $2^{2l-2} = 4^{l-1}$ sets T with $s_i \notin T, t_i \in T$, so there are $O(4^{l-1}k4^{k^2})$ branches, which results in a total running-time of $O^*((4^{l-1}k4^{k^2})^k) = O^*(4^{k(l-1)}k^k4^{k^3})$. Marx mistakenly said the running-time was $O^*(2^{kl}k^k4^{k^3})$, which stems from the erroneous assumption $|\bigcup_{j=1}^{l} \{s_j, t_j\}| = l$ that leads to 2^l subsets instead of $2^{2l} = 4^l$.

4.3 Fixed parameter tractability in $\{tw, l\}$

An ExtMSO$_2$ formula can be constructed from any instance of MIN-MULTICUT:

$$min\, C \subseteq E.\ \bigwedge_{\{s_i,t_i\} \in \mathcal{P}} \neg StillConnected(s_i, t_i, C)$$

where s_i and t_i are "*StillConnected*" in $(V, E-C)$ iff every partitioning $V = U \dot\cup \overline{U}$ with $s_i \in U$ and $t_i \in \overline{U}$ has an edge $\notin C$, that connects U and $\overline{U}$:

$$\forall U \subseteq V.(s_i \in U \wedge t_i \notin U) \rightarrow \underbrace{\exists x \in U.\exists y \notin U.\exists e \in E.inc(x,e) \wedge inc(y,e)}_{\text{exists edge } e \text{ between } U \text{ and } \overline{U}} \wedge e \notin C$$

The length of the whole formula is bounded by $O(l)$, so lemma 1.4.6.1 says that MIN-MULTICUT is FPT in $\{tw, l\}$.

Guo, Hüffner, Kenar, Niedermeier and Uhlmann found an algorithm that is FPT in $\{tw, l\}$, that does not rely on Courcelle's theorem [52]. This algorithm[2] is based on the observation that a multicut C is a separator such that s_i and t_i are in different components of $(V, E - C)$ for all $\{s_i, t_i\} \in \mathcal{P}$. Components can also be understood as color-classes in a vertex-coloring, so the algorithm enumerates all possible colorings $L_S : S \to \{1, \cdots, |S|\}$ of the terminals with $|S|$ colors, where $S = \bigcup \mathcal{P}$ (the set of distinct terminals in $\mathcal{P}$, $|S| \leq 2 \cdot l$). Of course colorings that assign the same color to both vertices of an $\{s_i, t_i\} \in \mathcal{P}$ are unwanted. Testing this costs $O(|\mathcal{P}|) = O(l)$ per coloring. From these at most $|S|^{|S|} \leq (2 \cdot l)^{2 \cdot l}$ initial colorings, a ***coloring extension*** is searched, which is a coloring $f : V \to \{1, \cdots, \alpha\}$ of all vertices, where $\alpha \leq |S|$ is the number of colors that are actually used in the initial coloring. From such a coloring, a multicut can be obtained by cutting all edges, whose incident vertices have different colors. These edges are called ***bad edges***. So the goal is to find a coloring extension which has a minimum number of bad edges.

This coloring extension is searched, using dynamic-programming on a (w.l.o.g. nice) tree-decomposition (T, B). Let T_x the subtree of T rootet at $x \in T$ and let $G_x = (V_x, E_x)$ the subgraph that is induced by the vertices in the bags of T_x: $G_x = G\left[\bigcup_{y \in T_x} B(y)\right]$.
For every vertex x of the tree T, create a table A_x with $\alpha^{|B(x)|}$ rows, one for each coloring of the vertices in bag $B(x)$. A_x will be used to cache the minimum number of bad edges in the subgraph G_x, given a coloring f of the vertices in the bag of x, so $A_x : (B(x) \to \{1, \cdots, \alpha\}) \to \mathbb{N}$.

For leaves x, $V_x = B(x)$, so $A_x(f)$ will just be the number of bad edges in $B(x)$, if $B(x)$ is colored according to f. So for leaves, the table can be filled by just enumerating the colorings f and counting the bad edges.

$$A_x(f) := \begin{cases} \infty & \text{if } f(v) \neq L_S(v) \text{ for a } v \in S \cap B(x) \\ |\{\{u, v\} \in E_x | f(u) \neq f(v)\}| & \text{otherwise} \end{cases}$$

Note that colors are assigned to *all* vertices, including the terminals. If a

[2]Actually, they gave an algorithm for RESTRICTED V-CUT and only mentioned the changes that are required to make it work for MIN-MULTICUT. Here, the algorithm with applied changes is presented.

coloring assigns a different color to a terminal, than the initial coloring, then this coloring f is illegal and thus will get weight $A_x(f) = \infty$, so it will not be used later.

After filling the tables of the leaves, the tables of the other vertices x are filled in a bottom up order, by enumerating the colorings f of the bags and computing $A_x(f)$.

- **Forget vertices** x with child y: Such vertices have $V_x \dot\cup \{v\} = V_y$ for a $v \notin V_x$. Every coloring of V_y is a coloring of V_x with an additional color for v, so the minimum number of bad edges in G_x, given coloring f is the minimum number of bad edges in G_y, given the same coloring of V_x, extended by some color for v.

  $$A_x(f) := min\{A_y(f_{v\mapsto c})|c \in \{1, \cdots, \alpha\}\}$$

  where

  $$f_{v\mapsto c}(x) := \begin{cases} c & \text{if } x = v \\ f(x) & \text{otherwise} \end{cases}$$

  Running time to fill the table: $O(\alpha^{|B(x)|} \cdot \alpha) \subseteq O((2 \cdot l)^{tw+1})$.

- **Introduce vertices** x with child y: In this case, $V_x = V_y \dot\cup \{v\}$. This time, every coloring f of V_x is a coloring of *all* vertices in V_y which has a fixed, additional color c for $v \notin V_y$, so the minimum number of bad edges in G_x, given f is the minimum number of bad edges in G_y given f (restricted to $B(y)$) plus the number of bad edges between v and $B(y)$ (because of rule 3 of tree-decompositions, all neighbors of v in G_y are members of $B(y)$)

  $$A_x(f) := \begin{cases} \infty & \text{if } v \in S \text{ and } c \neq L_S(v) \\ A_y(f|_{B(y)}) + |\{u \in B(y) \cap N(v) | f(u) \neq c\}| & \text{otherwise} \end{cases}$$

  Running time to fill the table: $O(\alpha^{|B(x)|} \cdot |B(y)|) \subseteq O((2 \cdot l)^{tw} \cdot tw)$.

- **Join vertices** x with children y and z: Here, $B(x) = B(y) = B(z)$, so without manipulations, f is a coloring for all involved bags and $V_x = V_y \cup V_z$. The minimum number of bad edges in G_x, given f is the sum of the minimum number of bad edges in G_y and G_z given f, but the bad edges in $E_y \cap E_z$ are counted twice, so this number has to be subtracted. By rule 3 of tree-decompositions, $E_y \cap E_z \subseteq E_x$, so in $O(|B(x)|^2)$ this value can be determined.

  $$A_x(f) := A_y(f) + A_z(f) - |\{\{u, v\} \in E_x | f(u) \neq f(v)\}|$$

  Running time to fill the table: $O(\alpha^{|B(x)|} \cdot |B(x)|^2) \subseteq O((2 \cdot l)^{tw} \cdot tw^2)$.

By definition, the root r of the nice tree-decomposition has $G_r = G$, so the minimum number of bad edges in G is the minimal entry in A_r. An actual coloring can be obtained by tracing the tables back down. The overall running-time of the algorithm is

$$O(\underbrace{(2\cdot l)^{2\cdot l}\cdot(l}_{\text{enum init colorings}} + \underbrace{(2\cdot l)^{tw+1}\cdot tw^2}_{\text{fill tables}}\cdot \underbrace{|V|}_{\text{vertices in } T\ (1.4.5)}\ )) \subseteq O((2\cdot l)^{2\cdot l+tw+1}\cdot tw^2\cdot|V|)$$

which is a running-time that is FPT in $\{tw, l\}$.

4.4 Fixed parameter tractability in $\{tw((V, E \cup \mathcal{P}))\}$

Gottlob and Lee showed [51] that a uniform MIN-MULTICUT instance $(V, E, \mathcal{P})$ can be transformed into a logical structure $\mathfrak{A} = (U, v, e, adj, inc, p)$ with

- universe $U = V \cup E$
- unary predicates (sets) $v(x) \Leftrightarrow x \in V$ and $e(x) \Leftrightarrow x \in E$
- binary predicates (relations)
  - $adj(x, y) \Leftrightarrow \{x, y\} \in E$
  - $inc(e, x) \Leftrightarrow e \in E, x \in e$
  - $p(x, y) \Leftrightarrow \{x, y\} \in \mathcal{P}$

A minimum multicut can then be found using an ExtMSO_2 formula with *constant* length:

$$min\, C.\forall s.\forall t.p(s,t) \rightarrow \neg StillConnected(s,t,C)$$

The running-time is now FPT in the treewidth of the Gaifman graph of $\mathfrak{A}$ (see definition 4, lemma 5 and theorem 20 in [44]). The Gaifman graph $\mathcal{G}(\mathfrak{A})$ is the same as (V, E), except that the binary relation p adds edges between s_i and t_i for all $\{s_i, t_i\} \in \mathcal{P}$ and inc adds paths of length 2 between all adjacent vertices. The treewidth of $\mathcal{G}(\mathfrak{A})$ is $\leq tw((V, E \cup \mathcal{P})) + 1$. The additional $s_i - t_i$ edges are expressed in $E \cup \mathcal{P}$ and at most one additional policeman is needed to catch the fugitive. To be precise, one additional policeman is needed if and only if (V, E) is a tree, because the fugitive can be chased in the same way on $\mathcal{G}(\mathfrak{A})$ as on $(V, E \cup \mathcal{P})$. Whenever an edge is cleared,

the fugitive could be caught by a third policeman (which is available unless (V, E) is a tree) if he had gone to the length-2 path that is parallel to the cleared edge. So, the running-time is FPT in $tw((V, E \cup \mathcal{P}))$. It should be noted that $tw((V, E \cup \mathcal{P})) \leq tw((V, E)) + l$, because a fugitive can be caught by placing a policeman on every s_i and otherwise chasing him like in (V, E). So this result implies fixed parameter tractability in $\{tw, l\}$ (4.3), but not vice versa.

4.5 Fixed parameter tractability in $\{l, \Delta\}$ and $\{\Lambda, k\}$

There cannot be more than Δ paths between any terminalpair $\{s_i, t_i\} \in \mathcal{P}$. Hence MIN-MULTICUT$= k \leq l \cdot \Delta$ (see also lemma 6.2.4), so lemma 1.4.1.1 implies that MIN-MULTICUT is FPT in $(\{l, k\} \backslash \{k\}) \cup \{l, \Delta\} = \{l, \Delta\}$.

If Λ is the length of a longest path between an s_i and its t_i, then the problem is FPT in $\{k, \Lambda\}$, because branching over all edges of an arbitrary $s_i - t_i$ path would define a recursion-tree with branching factor Λ and height k, so the running-time would be in $O^*(\Lambda^k)$. Since $\Lambda \leq n$, this is at most a running-time of $O^*(n^k)$.

Chapter 5

Fixed parameter tractability on special graph classes

5.1 Uniform stars

In 3.3.3.1, the NP-completeness of MIN-MULTICUT on stars had been proven by a P-reduction from VERTEX COVER. The construction also works vice versa (this is also proven in [47] already) and it is also an FPT-reduction in both directions:

Theorem 5.1.1. MIN-MULTICUT ON UNIFORM STARS $\leq_{FPT}$ VERTEX COVER

Proof. Let $(V', E', \mathcal{P})$ with $V' = \{c, x_1, \cdots, x_n\}, E' = \{\{c, x_i\} | i = 1, \cdots, n\}$ an instance of the undirected MIN-MULTICUT problem on a star. Note that w.l.o.g. we can assume that the center vertex c is not member of any terminalpair in $\mathcal{P}$, because otherwise lemma 6.3.2.1 is applicable. Show that the graph $(V, E) := (\{x_1, \cdots, x_n\}, \mathcal{P})$ has a vertex cover of size k if and only if $(V', E', \mathcal{P})$ has a multicut of size k.

Let $C = \{x_{i_1}, \cdots, x_{i_k}\} \subseteq V$ a vertex cover of size k for (V, E). Set $C' = \{\{x_{i_1}, c\}, \cdots, \{x_{i_k}, c\}\}$. Suppose there was a pair $\{s, t\} \in \mathcal{P}$ and a path (s, c, t) left in $(V', E' - C')$, then $\{c, s\}$ and $\{c, t\}$ are not in C'. Hence s and t are not in C, but $\{s, t\} \in \mathcal{P}$ means that there is an edge $\{s, t\}$ in (V, E), which would not be covered by C, so C is not a vertex cover for (V, E). $\lightning$

Let $C' = \{\{c, x_{i_1}\}, \cdots, \{c, x_{i_k}\}\}$ a multicut of size k for $(V', E', \mathcal{P})$, set $C = \{x_{i_1}, \cdots, x_{i_k}\}$. Suppose there was an edge $\{x, y\} \in E$ that was not covered by C, then x and y are not in C. Hence $\{c, x\}$ and $\{c, y\}$ are not

in C', but $\{x, y\} \in E$ means that there is a terminalpair $\{x, y\} \in \mathcal{P}$, which would not be cut by C', so C' is not a multicut for $(V', E', \mathcal{P})$. $\lightning$ □

Vertex Cover is well analyzed and there are many fast FPT algorithms. The current record is held by Chen, Kanj and Xia [17] with a running-time of $O^*(1.2738^k)$ and a kernelization with kernel-sizes $\leq 2k$ by Nemhauser and Trotter [75]. These algorithms can be applied to Min-Multicut on uniform stars with the same running-times (except for the reduction, which is computable in linear time, depending on the used data structures) and kernel-sizes.

5.2 Stars with real capacities ≥ 1

Min-Multicut on stars with real capacities corresponds to Vertex Cover with weighted vertices. Some rules from the uniform case don't apply anymore, like when there is a vertex x, which is member of only one terminalpair $\{x, y\} \in \mathcal{P}$, then $\{c, y\}$ can be cut (which corresponds to selecting neighbors of vertices of degree 1 for Vertex Cover). This rule cannot be applied anymore, unless $c(\{c, y\}) \leq c(\{c, x\})$ (see fig. 6.2).

This case is still FPT, because a bounded searchtree algorithm can be used to solve it. In every recursion, choose an arbitrary terminalpair $\{x, y\}$ and branch over the two involved edges $\{x, c\}$ and $\{y, c\}$. The branching vector is $(c(\{x, c\}), c(\{y, c\}))$ where both are at least 1. Thus, the recursion-tree has at most 2^k nodes and the running-time within every node is obviously polynomial, because only the terminalpairs $\{z, x\}$ or $\{z, y\}$ need to be removed from $\mathcal{P}$ for all $z \in V$. Therefore the algorithm has a running-time of $O(2^k \cdot p(n))$.

Not all, but some improvements can still be derived from the Vertex Cover algorithms. E.g. in the second branch you can cut all edges from $\{\{c, z\} \in E | \{z, x\} \in \mathcal{P}\}$, because in that case, it is decided that $\{x, c\}$ will not be part of the multicut, so all terminalpairs $\{z, x\}$ must be separated by $\{z, c\}$. If there is a terminalpair $\{x, y\} \in \mathcal{P}$ and x and y are both not part of any other terminalpair (this corresponds to two connected vertices of degree 1 in Vertex Cover), then the cheaper edge can be cut. Therefore it can be assumed that there is a vertex x that is part of more than one terminalpair, so branching over $\{x, c\}$ and all members of $\{\{y, c\} | \{y, x\} \in \mathcal{P}\}$ (which corresponds to x and $N(x)$ in Vertex Cover) gives a branching vector of at least $(1, 2)$ with a branching factor of 1.618, so this yields a running-time of $O(1.618^k p(n))$.

5.3 Uniform trees

In a rooted tree, the ***least common ancestor*** (***LCA***) of two vertices a, b is the vertex v on the (unique) path between a and b that has the least distance to the root. Figuratively speaking, when the vertices are arranged from top (root) to bottom, then the LCA of a, b is the lowest vertex x, for which the paths to a and b are both downwards.

The following algorithm for MIN-MULTICUT on uniform trees was invented by Guo and Niedermeier [53], it is a bounded searchtree algorithm with branching factor 2 and tree-height k. The running-time within every node of the recursion-tree is polynomial, so the overall running-time is $O(2^k p(n))$:

Select an arbitrary vertex r as root. Select a vertex x, which is the LCA w.r.t. r of a terminalpair $\{s_i, t_i\} \in \mathcal{P}$ such that there is no terminalpair whose LCA has a larger distance to r. If $x = s_i$ or $x = t_i$, then cut the edge from the $s_i - t_i$ path that is incident to x. Otherwise the $s_i - t_i$ path has the form $(s_i, \cdots, s', x, t', \cdots, t_i)$. In this case, branch over cutting $\{s', x\}$ and cutting $\{x, t'\}$ and start over recursively with parameter $k - 1$. Note that selecting both $\{s', x\}$ and $\{x, t'\}$, instead of branching over them gives a polynomial time approximation algorithm of quality 2.

The correctness and completeness of the algorithm follows from the fact that x is a least common ancestor of maximum distance to the root. This implies that no subtree below x contains both terminals of a terminalpair. Therefore cutting an edge closer to x can only improve the solution, because every terminalpair that has one terminal in the same subtree of x as s_i and that can be cut by an edge deeper down, can also be cut by $\{x, s'\}$ and every terminalpair that has one terminal in the same subtree of x as t_i and that can be cut by an edge deeper down, can also be cut by $\{x, t'\}$. Since $s_i - t_i$ is a terminalpair, at least one edge in one of the subtrees must be cut, so there exists an optimal solution that contains $\{x, s'\}$ or $\{x, t'\}$.

Notice that when $x = s_i$ or $x = t_i$, then there is only one choice of the edge, so in these cases there is only one successor in the recursion tree, which means that in this case the running-time is only half the time needed in the other case. Hence it is a good idea to try every vertex as root (it can be chosen differently in every recursion) and test whether this situation arises. The improvement can be amplified, because it is not really necessary that x is an LCA of *maximum distance* to r, it is sufficient if there is an LCA x with

$x = s_i$ or $x = t_i$ such that there is no terminalpair $\{s_j, t_j\}$ within the same subtree as t_i or s_i respectively. This improvement costs only a polynomial factor in the running-time and might often save exponential running-time, so the sacrifice is probably worthwhile.

On trees there is only one path between any pair of terminals. Therefore, if k was greater than l, then every terminalpair could be disconnected by an individual edge and the instance would be solvable. Hence $k \leq l$ can be assumed and thus lemma 1.4.1.1 implies that MIN-MULTICUT on trees is also FPT in $(\{l, k\} \backslash \{k\}) \cup \{l\} = \{l\}$.

5.4 Trees with integral capacities

When an edge has capacity $> k$, then obviously it cannot be part of a multicut with total value $\leq k$. Hence edges of capacity $> k$ can be excluded, so we can assume all edges to have capacities between 1 and k.

The algorithm by Guo and Niedermeier exploits the fact that cutting an edge closer to the LCA x cannot downgrade the solution. When the tree is not uniform, this is not true anymore, because an edge closer to x might have a larger capacity than an edge deeper down. In a private conversation, P. Rossmanith noticed that for integral capacities there are only k different kinds of capacities and among the edges with the same capacity, it is still better to choose the edge on the $s_i - x$ or $t_i - x$ path that is closest to x. So the algorithm by Guo and Niedermeier can still be applied, but there are up to k different edges to be tried on both subtrees of x, which yields a branching vector of $(1, 1, 2, 2, 3, 3, \cdots, k, k)$ on nodes of the recursion-tree that have parameter k. A trivial running-time bound is $O((2k)^k p(n))$, but since the later branches reduce the parameter more, this is far from the actual running-time.

Lemma 5.4.1. *A tree in which a vertex with parameter k has the branching vector*

$$(\underbrace{1, \cdots, 1}_{i \text{ times}}, \underbrace{2, \cdots, 2}_{i \text{ times}}, \cdots, \underbrace{k, \cdots, k}_{i \text{ times}})$$

has exactly $(i+1)^k$ vertices.

Proof. induction:

Induction foundation A vertex with parameter 0 has no successors, so $T_i(0) = 1 = (i+1)^0$.

Induction hypothesis $T_i(j) = (i+1)^j$ for all $j = 1, \cdots, k$ for a fixed k

Induction step $k \to k+1$. The number of vertices in the tree is

$$\begin{aligned} T_i(k+1) &= 1 + \sum_{j=1}^{k+1} i \cdot T_i(k+1-j) \\ &= 1 + i \cdot \sum_{j=0}^{k} T_i(j) \\ &= 1 + i \cdot \sum_{j=0}^{k} (i+1)^j \text{ (induction hypothesis)} \\ &= 1 + i \cdot \frac{(i+1)^{k+1} - 1}{i} \text{ (geometric sum)} \\ &= (i+1)^{k+1} \end{aligned}$$

□

So this shows that the algorithm actually has a running-time of $O(3^k p(n))$, because the recursion-tree has such branching vectors with $i = 2$. In practice, the running-times will be better, because even if all different capacities appear (which will also not be the case, usually), an edge of capacity c only needs to be tested, if there was no edge with a smaller capacity closer to x, because cutting an edge deeper in the subtree can only pay off, if it is cheaper than the edges above it. So of all $k!^2$ orders in which the capacities can appear in the two subtrees, only one $(k, k-1, k-2, \cdots, 1)$ and $(k, k-1, k-2, \cdots, 1)$ actually causes the 2 branches for all $i = 1, \cdots, k$.

Niedermeier and Guo found an $O(3^d mn^2)$ algorithm for non-uniform trees, where d is the maximum number of $s_i - t_i$-paths that pass a vertex or edge [54].

5.5 Uniform directed acyclic graphs

Directed Feedback Vertex Set and Feedback Arc Set are problems that ask for a minimum set of vertices or arcs, whose removal destroys all directed cycles in a digraph. Chen Liu and Lu [18] (see also [80]) reduce Directed Feedback Vertex Set and Feedback Arc Set to uniform Constrained Multicut on DAGs and present an $O(k4^k n^3)$ algorithm to solve that problem. Unfortunately Constrained Multicut

is a variant of MULTICUT that has a set of pairs *of sets* as terminalpairs $\mathcal{P} = \{(S_1, T_1), \cdots, (S_l, T_l)\}$. This alone would be no problem, because a terminal*set*pair (S_i, T_i) could be replaced by multiple terminalpairs $\{(s,t)|s \in S_i, t \in T_i\} = S_i \times T_i$, but the task is not to separate all $s \in S_i$ from all $t \in T_i$, but to separate all $s \in S_i$ from all $t \in \bigcup_{j=1}^{i} T_j$, which seems closer related to MULTIWAY-CUT.

An immediate algorithm for MULTICUT on a uniform DAG $(V, E, \mathcal{P})$ that might come to ones mind is:
For each $(s_i, t_i) \in \mathcal{P}$ add $k+1$ arcs (t_i, s_i) to E. The resulting digraph (V, E') has cycles that contain s_i and t_i and that can only be destroyed by removing arcs from E, which separate s_i from t_i, so this would result in an instance of DIRECTED FEEDBACK ARC SET. Indeed a feedback arc set of size $\leq k$ for the resulting graph would also be a multicut for $(V, E, \mathcal{P})$, but fig. 5.1 shows that the converse is not necessarily true. $(V, E, \mathcal{P})$ can have a multicut that is not a feedback arc set for (V, E').

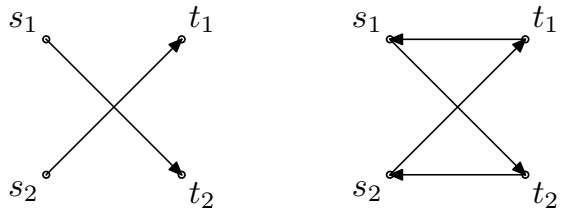


Figure 5.1: (V, E) has a multicut of size 0, but (V, E') has no feedback arc set of size < 1

5.6 Uniform grids

An $m \times n$-***grid*** is a graph with $m \cdot n$ vertices $V = \{x_{i,j} | i = 1, \cdots, m, j = 1, \cdots, n\}$ and $2 \cdot m \cdot n - n - m$ edges $E = \{\{x_{i,j}, x_{i+1,j}\}, | i = 1, \cdots, m-1, j = 1, \cdots, n\} \cup \{\{x_{i,j}, x_{i,j+1}\} | i = 1, \cdots, m, j = 1, \cdots, n-1\}$.

Theorem 5.6.1. MIN-MULTICUT *is fixed parameter tractable in* $\{k\}$ *on uniform grids.*

Proof. If $m \leq k$ and $n \leq k$, then $|V| \leq k^2$ and thus every time-bounded decision algorithm for MIN-MULTICUT has an FPT running-time for such instances. Therefore, only instances with $m > k$ or $n > k$ remain to be

examined. Assume w.l.o.g. $n \geq m$ (otherwise the grid can be rotated), which implies $n > k$. The grid cannot be cut through horizontally, since that would require at least $n > k$ cuts.

Assume there is one edge known to be a member of a min-multicut. For an $s_i - t_i$ path, the absence of this edge is no obstacle, because this one edge can easily be bypassed, unless more edges nearby are also cut (see fig 5.2). The removed edge can be bypassed on the bold path, so one of these 3 edges must also be cut or cutting the other edge was useless. Figuratively speaking,

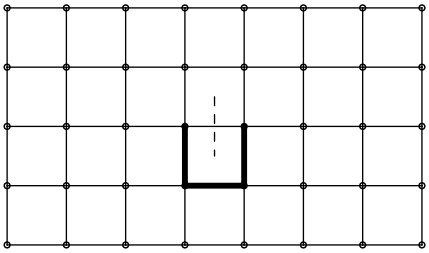


Figure 5.2:

one can start cutting with a pair of scissors in one square and continuously cut one of the edges that adjoin the rectangle that has been entered last. This approach has a searchtree with branching-factor 3 and (since at most k edges can be cut) height k, so this method will have a running-time of $O^*(3^k)$.

Now examine the possibilities, where such a cut can be started and how it can end. The examination starts with an algorithm for a situation that has multiple assumptions and discusses the changes that are required, whenever an assumption is dropped.

- Select an arbitrary terminalpair $\{s_i, t_i\} \in \mathcal{P}$. Assumptions:
  - The grid is large ($m > k$ and $n > k$, so that the grid cannot be cut through vertically, either)
  - s_i and t_i are both far away from all borders (more than k steps)
  - no other cuts are nearby (all other cuts are more than k steps away from s_i and t_i)

  In this case, the cut has to enclose s_i or t_i (such cuts are called ***island cuts***). Assume it has to enclose s_i. The cut must cross the horizontal line on which s_i lies, somewhere. If the cut crossed the horizontal line

more than k steps to the right of s_i, then more than k cuts would be required to reach s_i, to enclose it. The case where the cut has to enclose t_i is analogous, so for such terminalpairs, there are only $2 \cdot k$ choices for a first cut, from which the $O^*(3^k)$ cutting algorithm can start, so the running-time is $O^*(2 \cdot k \cdot 3^k)$.

- The grid is large, s_i and t_i can be close to a border, no other cuts are nearby.
  In this case, the cut can utilize the border to separate s_i from t_i (such cuts are called ***border cuts***). If the cut reaches the border, then the "pair of scissors" must be reset to the starting point of the cut, to search for the cutting-path to the border in the other direction. If s_i or t_i is close to the right border, then the terminalpair may be cut by a border cut to the right border. So in this case, the horizontal line on the right of the terminal may be not crossed by a cut. Hence, the first k edges on the horizontal line to the *left* of the terminal must be tried as first cut. If the terminal is also less than k steps away from the left border[1], then all edges on the horizontal line must be tried, because a border cut to the left border would also be possible. The running-time in this case is $O^*(4 \cdot k \cdot 3^k)$.

- The grid is large, s_i and t_i can be close to a border, other cuts can be nearby.
  In this case, the continuous cutting algorithm can come across another cut and share edges with it (such cuts are called ***archipelago cuts***), so when a different cut is met, then the cutting can continue from any rectangle that the other cut crosses and any rectangle, that another cut crosses, which is reachable from it, moving only over edges that are cut. Since all other cuts together cross at most k rectangles, there are at most $2 \cdot k$ possibilities to continue cutting at this point, but it can be shown that it is only necessary to try the edges that go into the same connected component that contains the current terminalpair. Also it can be shown that groups of other cuts never have to be met more than once.
  If a rectangle is reached that is connected to the outer region (this is the case if the other cut is a border cut or shares edges with a border cut or shares edges with a cut that shares edges with a border cut etc.), then the pair of scissors can also be reset to the starting point, as if the border had been reached. Infact, it can be shown that in

[1] In this case, we would have $m \leq n \leq 2 \cdot k \Rightarrow |V| \leq 4k^2$, so this case would be FPT anyway

this situation, it is useless not to reset. Also, the search for the first cut must be modified, because the horizontal line may be crossed by a different cut already. In this case, the continuous cutting algorithm can also start at any rectangle, that the other cut crosses and any further cut, that is reachable from it, crossing only edges that are already cut. All in all this, case can have a running-time up to $O^*((2 \cdot k)^k)$.

- The grid is flat ($m \leq k < n$), s_i and t_i are close to the upper and lower borders, other cuts can be nearby.
  In this case, s_i and t_i can be separated by a cut that is more than k steps away from s_i and t_i. They can be separated by a cut that goes from top to bottom (such cuts are called ***tectonic cuts***). Now if m is only a little bit smaller than k, then this is no problem. To be precise, if $m \geq \frac{k}{c}$, then there can be at most c tectonic cuts, so choosing the number of tectonic cuts $i \leq c$ and their starting points on the bottom border (from which the 3^k algoritm can be started) before running the algorithms from above costs only a factor of $\sum_{i=1}^{c} |V|^i = \frac{|V|^{c+1}-1}{c-1}$ in the running-time.

To understand the general algorithm, think back to the polynomial algorithm for paths (3.1). W.l.o.g. assume that for all terminalpairs $\{s_j, t_j\} \in \mathcal{P}$, t_j is the terminal with the larger x coordinate. From now on, $\{s_i, t_i\} \in \mathcal{P}$ will not be chosen arbitrarily anymore. Choose a (still connected) terminalpair whose t_i is farthest to the left. Possibly $\{s_i, t_i\}$ must be cut by an island- border- or archipelago cut, so the methods from above must also be tried. To find a tectonic cut, distinguish two cases:

- **Tectonic cuts that do not cross other cuts**: Like in the polynomial algorithm for paths, there has to be a cut to the left of t_i (unless s_i has the same x-coordinate as t_i, but in this case, the terminals are closer together than k steps, so the correct cut would also be found by the search for a border cut) and any continuous cut on the left of t_i can be straightened and moved to the right without downgrading the solution, until t_i is met. It might pay off to move some cuts further to the right, but since at least one cut has to be on the left of t_i, there is an optimal multicut that contains the edge on the left of t_i if $\{s_i, t_i\}$ has to be cut by a tectonic cut. The continuous cutting algorithm could be started from there, but infact such a cut would have been found by the search for a border cut already, so this case is already covered.

The only change to the algorithm, that is required for this case, is the choice of $\{s_i, t_i\}$.

Note: if the edge on the left of t_i was already cut, then any tectonic cut further to the left, that uses no other cuts, could be improved by moving it to the right and using this cut. So in that case, any tectonic cut that crosses no other cuts, would not be subset of any optimal solution.

– **Tectonic cuts that cross other cuts**: Instead of using a tectonic cut close to t_i, it might be cheaper to use a tectonic cut that crosses other cuts (that can be far away from s_i and t_i). Luckily there are at most k other cuts, at which the continuous cutting algorithm can be started.

The running-time of the algorithm is the product of the running-time of the searches for the individual cuts. The search for an individual cut starts with $3 \cdot k$ branches and has 3 branches, as long as no other cut is encountered. In that case, there are up to k branches and since there are at most k individual cuts, the running-time is bounded by $O^*((3k)^k)$. □

Although this is an FPT running-time, it is quite unpleasant, but keep in mind that only the first cuts and the archipelago cuts cause many branches. The sum of all cuts must be $\leq k$, so either there are not many other cuts that can be reused (so other cuts will not often be encountered) or the other cuts are not large (so if other cuts are encountered, then there are not many branches), or there are not many cuts left to separate s_i from t_i (so the rest of the recursion tree will not have a large height). Also, a search for a subsequent cut is started only in branches where the cutting algorithm has found a full cut for the last terminalpair. So usually the running-time will stay closer to 3^k.

Uniform grids have $\Delta \leq 4$, so if $k \geq 4 \cdot l$, then lemma 6.2.4 says that the instance is solvable. Hence $k \leq 4 \cdot l$ can be assumed and therefore lemma 1.4.1.1 together withtheorem 4.2.1 implies that Min-Multicut is also FPT in $(\{l, k\} \backslash \{k\}) \cup \{l\} = \{l\}$ on uniform grids.

The uniformity was only used to be able to move tectonic cuts to the right without downgrading the solution. In large grids, tectonic cuts are impossible, so it can also be concluded that Min-Multicut ist FPT in $\{k\}$ on large grids with real capacities ≥ 1.

An interesting fact is that cutting an edge in a planar graph is the same as contracting an edge in its dual graph (see definition 11.6 in [91]).

Chapter 6

Reduction rules

If it is certain that an edge e will be part of the multicut, then it can just be removed and k can be decreased by $c(e)$. If it is certain that an edge e will not be part of the multicut, then it is easy to see that e can be contracted, if the graph is undirected. In directed graphs, this is not the case, because contracting an arc might create paths that did not exist before, as fig. 6.1 shows. Redirecting the inbound arcs of x to the positive neighborhood of y

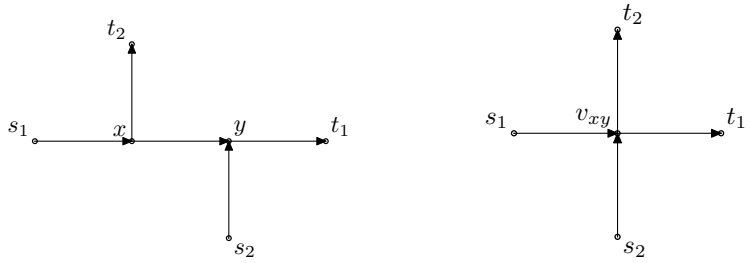


Figure 6.1: The left network has a multicut of size 1 without $\{x, y\}$, but the right network has no multicut of size < 2.

might increase the size of a minimum multicut. So the only possibility to deal with excluded arcs seems to be marking them as excluded, which is also what is done in literature.

Lemma 6.0.2. *In undirected networks, if there are two vertices x, y such that the min-cut of (V, E, x, y, c) has size $> k$, then x and y can be fused (repaced by a new vertex z that inherits the neighbors of x and y)*

Proof. If after cutting any set of edges with weight $\leq k$, there is a path $v - x$ and a path $y - w$ left, then there is a path $v - x - y - w$ because with a cut of weight $\leq k$, not all paths $x - y$ can be destroyed because the min-cut is

$> k$. Therefore, fusing x and y does not change reachability w.r.t. any cut $\leq k$. □

In directed graphs, the previous lemma is not applicable that easily because again, fusing vertices in directed graphs might create paths that did not exist before. In directed graphs, vertices can be fused as follows: If there is a sequence of vertices $(v_1, v_2, \cdots, v_{q-1}, v_q = v_1)$ such that for every $i = 1, \cdots, q-1$ there is either an excluded arc $(v_i, v_{i+1}) \in E$ or a min-cut of (V, E, v_i, v_{i+1}, c) has size $> k$, then all vertices $v_1, \cdots, v_{q-1}$ can be fused. Such a sequence can be found in polynomial time [21] by computing the strongly connected components of an auxiliary digraph.
Whenever there is a pair of vertices (x, y) such that a min-cut (V, E, x, y, c) has size $> k$, an excluded arc can be added, to simplify further computations.

In the following, some new terminology is used. If $\{x, y\} \in \mathcal{P}$, then x and y are ***$\mathcal{P}$-neighbors***. If $\{x, y\} \in \mathcal{P}$ and y is a leaf with neighbor y', then x and y' are ***quasi-$\mathcal{P}$-neighbors***. A set of leaves, that have the same neighbor is a ***group***, $Gr(x)$ is the group of leaves, that contains x (including x). If $\{x, y\} \in \mathcal{P}$ and x and y are in the same group, then $\{x, y\}$ is called a ***group request***. Let T a tree. The ***internal tree*** T' is the tree that is obtained from T by deleting all leaves of T.

- I_1 is the set of leaves of T'
- I_2 is the set of vertices, that have degree-2 in T'
- I_2^+ is the set of I_2-vertices, that are adjacent to leaves in T. $I_2^- = I_2 \backslash I_2^+$
- I_3 is the set of vertices that have degree ≥ 3 in T'
- L_1, L_2 and L_3 are the leaves of T whose neighbor is in I_1, I_2 or I_3 respectively
- L_2^{bad} is the set of L_2 leaves which have a group request. $L_2^{good} = L_2 \backslash L_2^{bad}$

A ***caterpillar component*** is an inclusion-maximal, connected set of I_1 and I_2 vertices together with the leaves attached to them. The I_1 and I_2 vertices are the ***backbone*** of the caterpillar component.

6.1 Necessary criteria

Lemma 6.1.1. *If max-flow of $(V, E, s_i, t_i, c) > k$ for an $i \in \{1, \cdots, l\}$, then the instance is not solvable.*

Proof. For any $C \subseteq E$ with $\sum_{e \in C} c(e) \leq k$, there is a path $s_i - t_i$ in $(V, E - C)$, because otherwise C would be a cut for $s_i - t_i$ with a value smaller than the magnitude of the max-flow between s_i and t_i, which contradicts MAX-FLOW = MIN-CUT. □

Lemma 6.1.2. *If $(v, v) \in \mathcal{P}$ for some vertex v, then the instance is not solvable.*

Proof. Even if $C = E$ is used as cut, v is reachable from v, since the path (v) starts in v, ends in v and uses no edges that are not in $E \backslash C = \emptyset$. □

6.2 Sufficient criteria

All criteria in this section implicitly assume $s_i \neq t_i$ for all $\{s_i, t_i\} \in \mathcal{P}$ (see lemma 6.1.2).

Lemma 6.2.1. *If $\sum_{i=1}^{l}$ max-flow of $(V, E, s_i, t_i, c) \leq k$, then the instance is solvable.*

Proof. The union of all $s_i - t_i$ cuts is a sufficiently small multicut. □

This property can be computed in $O(n^5)$ for general (di)graphs with positive real capacities, using an $O(n^3)$ flow-algorithm $|\mathcal{P}| \leq n^2$ times.

Lemma 6.2.2. *For undirected graphs: if*

$$\sum_{i=1}^{l} min\{d(s_i), d(t_i)\} \leq k$$

then the instance is solvable.

Proof. The precondition of 6.2.1 is satisfied, because every terminalpair can be cut, using all incident edges of s_i or t_i. □

Lemma 6.2.3. *For digraphs: if*

$$\sum_{i=1}^{l} min\{d^+(s_i), d^-(t_i)\} \leq k$$

then the instance is solvable.

Proof. The precondition of 6.2.1 is satisfied, because every terminalpair can be cut, using all outbound arcs of s_i or all inbound arcs of t_i. □

Lemma 6.2.4. *If $\Delta(G) \leq \frac{k}{l}$, then the instance is solvable.*

Proof. The precondition of lemma 6.2.2 is satisfied:

$$\begin{aligned}\sum_{i=1}^{l} min\{d(s_i), d(t_i)\} &\leq \sum_{i=1}^{l} d(s_i) \\ &\leq \sum_{i=1}^{l} \Delta(G) \\ &= l \cdot \Delta(G) \\ &\leq l \cdot \frac{k}{l} \\ &= k\end{aligned}$$

□

Lemma 6.2.5. *If $\Delta^+(G) \leq \frac{k}{l}$ or $\Delta^-(G) \leq \frac{k}{l}$, then the instance is solvable.*

Proof. Analogous to lemma 6.2.4, the precondition of lemma 6.2.3 is satisfied. □

Lemma 6.2.6. *For uniform (di)graphs: if $|E| \leq k$, then the instance is solvable.*

Proof. E is a sufficiently small multicut. □

Lemma 6.2.7. *For non-uniform (di)graphs: if $\sum_{e \in E} c(e) \leq k$, then the instance is solvable.*

Proof. E is a sufficiently small multicut. □

Lemma 6.2.8. *For non-uniform (di)graphs: if $|E| \cdot max\,\{c(e) | e \in E\} \leq k$, then the instance is solvable.*

Proof. E is a sufficiently small multicut. □

Lemma 6.2.9. *For uniform undirected graphs: if $\binom{n}{2} \leq k$, then the instance is solvable. For uniform digraphs: if $2\binom{n}{2} \leq k$, then the instance is solvable.*

Proof. E is a sufficiently small multicut. □

Lemma 6.2.10. *If max-flow of $(V \dot\cup \{s,t\}, E \dot\cup \{(s,s_i),(t_i,t) | (s_i,t_i) \in \mathcal{P}\}, s, t, c') \leq k$, with*

$$c'(e) = \begin{cases} c(e) & \text{if } e \in E \\ \sum_{e \in E} c(e) & \text{otherwise} \end{cases}$$

then the instance is solvable.

Proof. It is even possible to separate all s_i from *all* t_j with k cuts. □

6.3 Generalizations of reduction rules for trees

The first 8 reduction rules in this section come from Guo and Niedermeier [53] and guarantee kernels of size $O(k^{3k})$ on uniform trees. The last 3 rules are by Bousquet, Daligault, Thomassé and Yeo [12] and guarantee kernels of size $O(k^6)$ on uniform trees. Rule number 2, 4 and 5 are used by both parties, but they are named differently. Here they carry both names under which they are used.

6.3.1 Idle edge

Lemma 6.3.1.1. *If an edge $e = \{x, y\} \in E$ is not on any path from an s_i to t_i, then no optimal multicut C contains e.*

Proof. If C is a multicut and $e \in C$, then putting e back into E cannot reestablish a path from an s_i to its t_i, because this would mean that e is on a path from this s_i to this t_i. So there is no $s_i - t_i$ path in $(V, E - (C - e))$, which means that $C - e$ is a smaller multicut than C and hence C is no optimal multicut. □

To test whether an edge $\{x, y\}$ is idle, it must be tested, whether there is a terminalpair $\{s_i, t_i\} \in \mathcal{P}$, such that there is a path from s_i to t_i, containing $\{x, y\}$.

Lemma 6.3.1.2. *In undirected graphs, there exists a path $s_i - t_i$, containing $\{x, y\}$ if and only if there are 2 (vertex-)disjoint paths $s_i - x$ and $y - t_i$ or 2 (vertex-)disjoint paths $s_i - y$ and $x - t_i$.*

Proof. If there exist 2 disjoint paths $(s_i = x_1, \cdots, x_q = x)$ and $(y = y_1, \cdots, y_r = t_i)$ or $(s_i = y_1, \cdots, y_q = y)$ and $(x = x_1, \ldots, x_r = t_i)$, then there is a path $(s_i = x_1, \cdots, x_q = x, y = y_1, \cdots, y_r = t_i)$ or $(s_i = y_1, \cdots, y_q = y, x = x_1, \cdots, x_r = t_i)$ from s_i to t_i, containing $\{x, y\}$. The (vertex-)disjointness ensures that the constructed walk is indeed a path.

If there is a path from s_i to t_i, containing $\{x, y\}$, then either x occurs in the path first or y occurs in the path first, so the path has the form $(s_i, \cdots, x, y, \cdots, t_i)$ or $(s_i, \cdots, y, x, \cdots, t_i)$. In the first case, there are the 2 disjoint paths $s_i - x$ and $y - t_i$. In the second case, there are the 2 disjoint paths $s_i - y$ and $x - t_i$. □

Lemma 6.3.1.3. *In directed graphs, there exists a path $s_i - t_i$, containing (x, y) if and only if there are 2 (vertex-)disjoint paths $s_i - x$ and $y - t_i$.*

Proof. If there exist 2 disjoint paths $(s_i = x_1, \cdots, x_q = x)$ and $(y = y_1, \cdots, y_r = t_i)$, then there is a path $(s_i = x_1, \cdots, x_q = x, y = y_1, \cdots, y_r = t_i)$ from s_i to t_i, containing (x, y). The (vertex-)disjointness ensures that the constructed walk is indeed a path.

If there is a path from s_i to t_i, containing (x, y), then x occurs in the path first, so the path has the form $(s_i, \cdots, x, y, \cdots, t_i)$. There are the 2 disjoint paths $s_i - x$ and $y - t_i$. □

Thus, to identify an idle edge, an algorithm for 2 disjoint paths can be used $2 \cdot |\mathcal{P}| \leq 2 \cdot n^2$ times. For digraphs, the algorithm must be applied only $|\mathcal{P}| \leq n^2$ times. The k Disjoint Paths problem is FPT in k [85, 84], so for constant $k = 2$, this can be tested in $O(f(2) \cdot p(n)) = O(p(n))$. See also [88].

To identify all idle edges, this must be tested for all edges or all arcs, so the total running-time is

$$O(|E| \cdot 2 \cdot |\mathcal{P}| \cdot p(n)) \subseteq O\left(\frac{n^2}{2} \cdot 2 \cdot n^2 \cdot p(n)\right) = O(n^4 \cdot p(n))$$

for undirected graphs and

$$O(|E| \cdot |\mathcal{P}| \cdot p(n)) \subseteq O(n^2 \cdot n^2 \cdot p(n)) = O(n^4 \cdot p(n))$$

for directed graphs.

6.3.2 Unit path / Unit request

Lemma 6.3.2.1. *If there is an edge $e = \{s_i, t_i\} \in E$, then every multicut C contains e.*

Proof. If $e \notin C$, then there is a path (s_i, t_i) using e in $(V, E - C)$ and hence C is no multicut. □

The argument stays valid for non-uniform (di)graphs. There is an obvious $O(n^4)$ algorithm to find such edges and depending on the used data structures, they can also be found in $O(n^2)$.

This reduction rule is a special case of the overloaded edge rule (6.3.6).

6.3.3 Dominated edge

If there are edges $e, e' \in E$ such that every $s_i - t_i$ path, which contains e, contains e' too, then e can be excluded. This is the case because if a multicut C contains e, then $(C-e)\cup\{e'\}$ is a multicut that is at least as good as C, so excluding e from the search does not make solvable instances unsolvable and excluding an edge can never make an unsolvable instance solvable. This holds true for uniform (di)graphs, but for non-uniform (di)graphs, $c(e) \geq c(e')$ must also be satisfied, because as you can see in fig. 6.2, abandoning

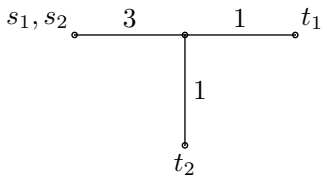


Figure 6.2: The edges of weight 1 are a multicut with total weight 2, whereas the dominated edge rule without the $c(e) \geq c(e')$ requirement would exclude these edges, leaving only the edge of weight 3 to cut.

this requirement can cause non-optimal results. In non-uniform (di)graphs, cutting multiple cheap edges can be overall cheaper, than cutting less, but more expensive edges.

To find all dominated edges, it can be tested for all $O(|E| \cdot (|E| - 1)) \subseteq O(n^4)$ pairs of edges e, e', whether e' dominates e.

Lemma 6.3.3.1. *Let $(V, E, \mathcal{P})$ an instance of the* Min-Multicut *problem. An edge or arc e' dominates an edge or arc e if and only if e is idle in $(V, E - e', \mathcal{P})$.*

Proof. Let e' dominate e. This by definition means that all $s_i - t_i$ paths, which contain e, contain e' also. Therefore all $s_i - t_i$ paths that contain e, are destroyed by removing e' and therefore there is no $s_i - t_i$ path in $(V, E - e', \mathcal{P})$ that contains e, which by definition means that e is idle in $(V, E - e', \mathcal{P})$.

Let e be idle in $(V, E-e', \mathcal{P})$. This by definition means that no s_i-t_i path in $(V, E-e', \mathcal{P})$ contains e. This implies that no s_i-t_i path in $(V, E, \mathcal{P})$ that contains e, is not destroyed by removing e' and hence e' is on every $s_i - t_i$ path that contains e and this by definition means that e' dominates e. (Note: this also shows that idle edges are dominated by all other edges) □

So to find all dominated edges, idleness can be tested $O(|E|^2)$ times. Idleness can be tested in $O(|\mathcal{P}| \cdot p(n))$, see 6.3.1, so the overall running-time for the dominated edge rule is

$$O(|E|^2 \cdot |P| \cdot p(n)) \subseteq O(n^6 \cdot p(n))$$

for uniform (di)graphs.

For non-uniform (di)graphs, this is also true, but it is only half the truth, because an edge e might be dominated by a set E' of edges, which have $\sum_{e' \in E'} c(e') \leq c(e)$ and e is idle in $(V, E - E', \mathcal{P})$. This property (full dominated edge) however does not seem to be FPT in k unless Min-Multicut is FPT in k.

A different generalization of the rule is the following:

Lemma 6.3.3.2. *If an $\{s_i, t_i\} \in \mathcal{P}$ is a separator and there is a component X of $(V - \{s_i, t_i\}, E, \mathcal{P})$ such that there is no terminal in X, then every min-cut of $(X \cup \{s_i, t_i\}, E \cap \mathfrak{P}_2(X \cup \{s_i, t_i\}), s_i, t_i, c)$ is subset of a min-multicut.*

Proof. Let C a min-multicut. Successively delete the edges from the network, that are not inside X. In the resulting instance, every remaining $s_j - t_j$ path passes through X and thus it passes s_i and t_i. Therefore, $\{s_i, t_i\}$ dominates all other terminalpairs, so all of them can be removed from $\mathcal{P}$. The resulting network has exactly one terminalpair ($\{s_i, t_i\}$), so it is a single-commodity network and using a min-cut C' for it instead of $C \cap E(X)$ would also ultimately disconnect the last terminalpair, so $(C \backslash E(X)) \cup C'$ would be a multicut that would be smaller than the min-multicut C, unless $|C \cap E(X)| = |C'|$, in which case $C \cap E(X)$ is also a min-cut for the constructed network. □

6.3.4 Dominated path / Inclusion

The dominated path rule for trees says that, if there are two distinct terminalpairs $\{s_i, t_i\}$ and $\{s_j, t_j\}$ such that the $s_i - t_i$ path contains s_j and t_j, then $\{s_i, t_i\}$ can be removed from $\mathcal{P}$, because any multicut for $(V, E, \mathcal{P} - \{s_i, t_i\})$ must cut $\{s_j, t_j\}$, which also destroys the $s_i - t_i$ path. This concept is independent of capacities, so the rule also works for non-uniform trees. In general, there can be more than one path between s_i and t_i, so the existence of a path $(s_i, \cdots, s_j, \cdots, t_j, \cdots, t_i)$ or $(s_i, \cdots, t_j, \cdots, s_j, \cdots, t_i)$ is not a sufficient criterion anymore.

In general graphs, $\{s_i, t_i\}$ can be removed, if *every* $s_i - t_i$ path crosses both s_j and t_j. This can be tested in polynomial time by testing whether there is an $s_i - t_i$ path in $(V - s_j, E)$ or in $(V - t_j, E)$. If in one case such a path exists, then the rule is not applicable, because if (w.l.o.g.) there is an $s_i - t_i$ path in $(V - s_j, E)$, then this path does not cross both terminals, so separating $\{s_j, t_j\}$ does not automatically separate $\{s_i, t_i\}$ and thus $\{s_j, t_j\}$ does not dominate $\{s_i, t_i\}$.
These two tests can be done using Dijkstra's algorithm twice, so this takes $O(2 \cdot n \cdot \log(n))$. To find all dominated paths, one can test for all $O(|\mathcal{P}|^2) \subseteq O(n^4)$ pairs of terminalpairs whether one dominates the other, which yields a running-time of $O(2 \cdot |\mathcal{P}|^2 \cdot n \cdot \log(n) \cdot 2) \subseteq O(n^6)$.
From the properties it follows that s_j and t_j must be cut vertices. Both separate the component that contains s_i from the component that contains t_i.

Different $s_i - t_i$ paths might be dominated by different terminalpairs, so the general rule would be that a terminalpair $\{s_i, t_i\}$ is dominated by a set of terminalpairs $\mathcal{P}' \subseteq \mathcal{P}$, iff every $s_i - t_i$ path contains s_j and t_j for some $\{s_j, t_j\} \in \mathcal{P}'$. Note that every path needs to contain both terminals of one terminalpair, but the terminalpair may differ between different $s_i - t_i$ paths. Constrained to trees, the generalized version is equivalent to the original rule for trees, so for trees, this is not a true generalization and thus, the polynomiality of the rule for trees is retained.

It is easy to see that, if a graph has a structure as in fig. 6.3, then $\{s_i, t_i\}$ is dominated by $\mathcal{P}' = \{\{s_{i_j}, t_{i_j}\} | j = 1, \cdots, q\}$. The example in fig. 6.4, due to

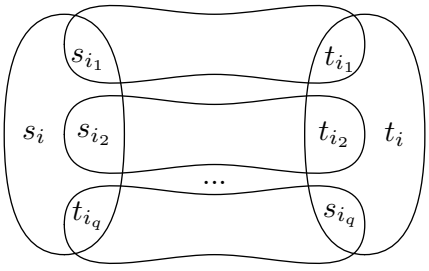


Figure 6.3: Structure of a graph that has a generalized dominated path

J. Kneis, shows an instance that has a dominated path $\{s_1, t_1\}$, but not such a structure. So having a structure as in fig. 6.3 is only a sufficient, but not a necessary criterion. For general graphs, this structure is still a generalization

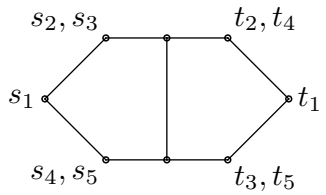


Figure 6.4: A graph that has a dominated path, but the generalized rule does not apply

of the first rule for general graphs and it can be tested in polynomial time. Because of rule unit path, it can be assumed that the components between the s_{i_j} and t_{i_j} are not empty, so in component j, there exists an $x \in V$ such that all $s_i - x$ and $t_i - x$ paths contain s_{i_j} or t_{i_j}. This means that $\{s_{i_j}, t_{i_j}\}$ is an $(\{s_i, t_i\}, \{x\})$ separator. To test whether the rule applies, find an $\{s_{i_j}, t_{i_j}\} \in \mathcal{P}$ which separates $\{s_i, t_i\}$ from some $x \in V$. Successively remove such s_{i_j} and t_{i_j} from V until either s_i is disconnected from t_i (then the rule is applicable and $\{s_i, t_i\}$ can be deleted from $\mathcal{P}$) or no such $\{s_{i_j}, t_{i_j}\}$ can be found anymore (in this case, the rule is not applicable).

6.3.5 Disjoint paths / Disjoint requests

If there are $k+1$ or more edge-disjoint paths between terminalpairs, then the instance is not solvable. Let C a multicut with $|C| \leq k$. According to the pigeonhole principle [36], at least 2 paths have to be cut by the same edge $e \in C$. Therefore e is on both of these paths, which is a contradiction to the assumed edge-disjointness of the paths. This property is computable in polynomial time for trees (see 2.2) so the rule is feasible for this case, but in general, computing the property is NP-complete [2], even for planar graphs [74].
Although the k-Disjoint Paths problem is FPT in k [85, 84], this does not help because these algorithms require $l = k$. The general case appears to be as hard as Min-Multicut, but there is actually an even stronger rule that is applicable in polynomial time on general graphs.

In 2.2, 2.3 and 3.4 it was discussed that

$$max\text{-}imf \leq max\text{-}mf \leq min\text{-}mc \leq max\text{-}mf \cdot \log(l)$$

and it is clear that the number of disjoint paths is $\leq max\text{-}imf$ because on every disjoint path, there can be a separate integral flow. So it is possible

to use linear programming (see 2.1.1) to determine the $max\text{-}multiflow$ and conclude insolvability if $k < max\text{-}mf$ (which is a situation that arises earlier than $k <$ number of disjoint paths) or solvability if $k \geq max\text{-}mf \cdot \log(l)$.

6.3.6 Overloaded edge

If there is an edge $e \in E$ and at least $k+1$ terminalpairs that have a path of length 2 containing e, then e can be cut. Actually Niedermeier and Guo say that e can be contracted, k can be reduced by 1 and all terminalpairs can be removed that are separated by e. This may lead to the same result for trees, but cutting e is formally more accurate and removing a terminalpair from $\mathcal{P}$, just because *one* path is destroyed, leads to false results in general graphs. Contracting an arc can also change the result in directed trees (remember fig. 6.1).
It might seem questionable, whether this rule holds true for general graphs, where additional paths for the terminalpairs can exist. To see that the rule indeed holds true, one must consider that if e was not cut, then the other $k+1$ (or more) edges of the paths would have to be cut.

In non-uniform (di)graphs, less than k such terminalpairs may be sufficient for an edge to be overloaded, because the sum of the other capacities can be greater than k, which already makes the above statement true. So for non-uniform, undirected graphs, an edge $\{x, y\}$ is overloaded iff

$$\sum_{z \in \{z | \{x,z\} \in \mathcal{P} \wedge \{y,z\} \in E\}} c(\{y,z\}) + \sum_{z \in \{z | \{y,z\} \in \mathcal{P} \wedge \{x,z\} \in E\}} c(\{x,z\}) > k$$

For non-uniform, directed graphs, an arc (x, y) is overloaded iff

$$\sum_{z \in \{z | (x,z) \in \mathcal{P} \wedge (y,z) \in E\}} c((y,z)) + \sum_{z \in \{z | (z,y) \in \mathcal{P} \wedge (z,x) \in E\}} c((z,x)) > k$$

These properties can clearly be computed in polynomial time.

The basic idea behind this rule is: not cutting the edge would be like contracting it and then there would be more than k disjoint paths / multiflow $> k$. So this rule can be generalized in the following way: If the network violates any necessary criterion after contraction of e, then e must be part of a min-multicut. This rule is polynomial as long as all tested neccessary criteria are polynomial (and the number of tested criteria is polynomial). This includes the generalized test for disjoint paths. In directed networks, an arc must be part of a min-multicut, if after excluding it (remember the

discussion behind Lemma) there would be a sequence of vertices $(v_1, \cdots, v_q)$ with $v_1 = v_q$ such that for every $i = 1, \cdots, q-1$ either there is an excluded arc (v_i, v_{i+1}) or the min-cut of (V, E, v_i, v_{i+1}, c) is larger than k and fusing $v_1, \cdots, v_{q-1}$ would lead to a network that violates a necessary criterion.

Note: $\{v, v\} \notin \mathcal{P}$ is a necessary criterion. Therefore the unit path rule is implied by the generalized version, so it can be removed.

6.3.7 Overloaded caterpillar

If there is a vertex v and at least $k+2$ vertices $u_1, \cdots, u_{k+2}$ that are within the same caterpillar-component C, but in a different caterpillar-component than v and $\{\{v, u_i\} | i = 1, \cdots, k+2\} \subseteq \mathcal{P}$, then remove (one of) the terminalpairs $\{v, u_i\}$ with maximum distance between v and u_i.

The reason for this is that because of the pigeonhole principle, at least two of the remaining $k+1$ terminalpairs have to be cut by the same edge. All edges that are shared by any two of the terminalpairs, are on the common path, so this cut also disconnects $\{v, u_i\}$.

This rule is a special case of the common factor rule (6.3.10), because the common factors of the paths $\{\{v, u_j\} | j = 1, \cdots, k+2\} \backslash \{\{v, u_i\}\}$ are all subsets of the path $\{v, u_i\}$, so the overloaded caterpillar rule can be omitted.

Niedermeier and Guo only required $k+1$ such paths, which would be a difference to common factor, since the common factor rule would require $k+2$ such paths, but the example in fig. 6.5 shows, that the rule is false if only $k+1$ paths are demanded. The paths $\{s'_1, t'_1\}, \cdots, \{s'_k, t'_k\}$ require k cuts. After

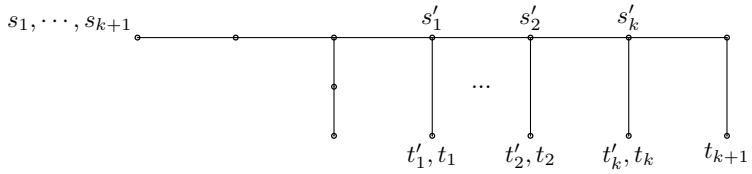


Figure 6.5: An unsolvable instance that becomes solvable, if $\{s_{k+1}, t_{k+1}\}$ is removed

these, no cut is left to disconnect $\{s_{k+1}, t_{k+1}\}$. The overloaded caterpillar rule would delete $\{s_{k+1}, t_{k+1}\}$ from $\mathcal{P}$, turning this unsolvable instance into a solvable one.

6.3.8 Overloaded L_3 leaves

Niedermeier and Guo define this rule as follows: If there is a vertex v with terminalpairs $\{v, u_1\}, \cdots, \{v, u_{k+1}\}$, where $u_1, \cdots, u_{k+1}$ are leaves within the same L_3-group, then remove $\{v, u_1\}, \cdots, \{v, u_{k+1}\}$ from $\mathcal{P}$ and instead put $\{v, u\}$ into $\mathcal{P}$.

The basic idea behind this rule is: not disconnecting v from u would be like fusing u and v, so the edges $\{u, u_1\}, \cdots, \{u, u_{k+1}\}$ would become $k+1$ disjoint paths. So v must be disconnected from u and $\{u, v\}$ dominates $\{v, u_1\}, \cdots, \{v, u_{k+1}\}$. This concept can be generalized in the following way: if there are two vertices v, u such that the network violates any necessary criterion if v and u are fused, then $\{v, u\}$ can be added to $\mathcal{P}$ and all terminalpairs that are dominated afterwards can be removed from $\mathcal{P}$. This rule is polynomial as long as all tested neccessary criteria are polynomial (and the number of tested criteria is polynomial). This includes the generalized test for disjoint paths.

On trees, the generalized rule is stronger than the original overloaded L_3 leaves rule, because not all the disjoint paths need to be single edges $\{u, u_i\}$, u is not necessarily an I_3 vertex and the disjoint paths can go through different subtrees. Also the generalized version covers situations where $\{v, u\}$ also dominates terminalpairs in the opposite direction $\{u, v_1\}, \cdots, \{u, v_q\}$.

The generalized version works for non-uniform graphs, because the generalized disjoint paths rule works for non-uniform graphs, so it is not necessary to have $> k$ disjoint paths $u - v_i$ or $v - u_i$, it is sufficient to have $> k$ flow between $u - v_i$ and $v - u_i$ total.

6.3.9 Unique direction

If there is a leaf s'' with $N(s'') = \{s\}$, $s' \in N(s)$ and for all terminalpairs $\{s'', t\}$ that contain s'', the path $s'' - t$ has the form $(s'', s, s', \cdots, t)$, then contract $\{s, s''\}$ (it can be assumed that $\{s, s''\} \notin \mathcal{P}$, because of rule unit path). The reason for this is that $\{s, s'\}$ dominates $\{s, s''\}$.

For inner vertices s with degree 2, there is a different rule[1]: Let $N(s) = \{s', s''\}$. If for all terminalpairs $\{s, t\} \in \mathcal{P}$ that contain s, the path $s - t$ has the form $(s, s', \cdots, t)$, then contract $\{s, s''\}$. This is again possible because

[1]Due to confusing notation, it is hard to see that the vertices must have degree 2, but in an e-mail conversation with Jean Daligault, it was clearified that this is meant.

$\{s, s'\}$ dominates $\{s, s''\}$.

In non-uniform networks, $c(\{s, s'\}) \leq c(\{s, s''\})$ has to be required again, but since this rule is a special case of the dominated edge rule, it can be omitted.

6.3.10 Common factor

The rule for trees is the following: If there is a terminalpair $\{s_i, t_i\} \in \mathcal{P}$ and $k+1$ terminalpairs $\{s_{i_1}, t_{i_1}\}, \cdots, \{s_{i_{k+1}}, t_{i_{k+1}}\} \in \mathcal{P}$ such that for every two of these terminalpairs $\{s_{i_x}, t_{i_x}\}, \{s_{i_y}, t_{i_y}\}$, all edges which are on the path between s_{i_x} and t_{i_x} and also on the path between s_{i_y} and t_{i_y} (the edges in the "common factor") are also on the path between s_i and t_i, then remove $\{s_i, t_i\}$ from $\mathcal{P}$. If the common factor of two pairs is empty, this also counts. To cut the $k+1$ terminalpairs with k cuts, two of them have to be cut by the same edge (pigeonhole principle). This edge has to be part of the common factor and thus it also disconnects $\{s_i, t_i\}$.
It must be explicitly demanded that the terminalpairs intersect the $s_i - t_i$ path. The argument would stay valid without this requirement, but the upcoming algorithm would not find external terminalpairs and this weaker rule is sufficient for the proof of the polynomial kernel sizes.

As to the running-time: in general, the applicability of the rule sounds like it is related to the Clique problem, which is W[1]-hard in k [41, 15]. Create an auxiliary graph with one vertex for every terminalpair and connect vertices of terminalpairs for which the common factor is a subset of the $s_i - t_i$ path. The rule is applicable, iff the resulting graph has a clique of size $k+1$, but if the original graph is a tree, then the applicability can also be tested, using a matching algorithm.
Informally, a set of terminalpairs X has to be found, which all intersect the $s_i - t_i$ path such that the edges through which they enter and leave the path, are all distinct. Create an auxiliary graph with a vertex for every edge, that is incident to a vertex on the $s_i - t_i$ path, but that is not itself part of the path. Connect the vertices of two edges e, e', if there exists a terminalpair $\{s_j, t_j\}$ such that e and e' are both on the $s_j - t_j$ path. The construction so far does not regard terminalpairs $\{s_j, t_j\} \in \mathcal{P}$ that have one terminal among the vertices of the $s_i - t_i$ path (no terminalpair can have both terminals on the $s_i - t_i$ path because such a terminalpair would dominate $\{s_i, t_i\}$). Collect these terminalpairs in a set $Y \subseteq \mathcal{P}$. Such terminalpairs can be part of X without any loss. Only from the auxiliary graph, the vertex has to be deleted that stands for the edge through which the $s_j - t_j$ path leaves the

$s_i - t_i$ path.
In the original graph, rule common factor is applicable to $\{s_i, t_i\}$ iff the auxiliary graph has a matching of size $k - |Y| + 1$ and since the MAXIMUM MATCHING problem is polynomial [33], the applicability of the rule in trees is polynomial.

To incorporate external terminalpairs, one could also subtract a max-multiflow from $k - |Y| + 1$, since external terminalpairs also consume cuts, leaving less cuts for the terminalpairs intersecting the $s_i - t_i$ path. A problem with this approach however is that these external terminalpairs might have some intersection with the terminalpairs whose path intersect the $s_i - t_i$ path, so depending on these intersections it might be better to peempt some terminalpairs in the selection of the maximum matching.

The rule seems untransferable to general graphs. The common factors of terminalpairs can be computed in polynomial time on general graphs, because for every edge $\{x, y\}$ it can be tested if there is an $s_i - x - y - t_i$ or $s_i - y - x - t_i$ path and an $s_j - x - y - t_j$ or $s_j - y - x - t_j$ path (see 6.3.1.2). But a generalization of the rule would require *every* $s_i - t_i$ path to cross *every* edge of the common factor of *every* pair of terminalpairs for *some* set of terminalpairs. Maybe a tree-decomposition could be utilized to test this property, but it is unclear what should be done, if a common factor is not a path and the conditions for the applicability seem so odd, that the rule probably would hardly ever be applicable.

6.3.11 Dominating wingspan

For caterpillars: Let x an L_2^{good} vertex, attached to $x' \in I_2^+$, so all paths to his $\mathcal{P}$-neighbors pass the caterpillars backbone. The ***wingspan*** of x is the path between its closest quasi-$\mathcal{P}$-neighbors in both directions x_l and x_r (if in one direction there is no $\mathcal{P}$-neighbor, then unique direction would be applicable). The ***size*** of a wingspan is the number of L_2^{good} leaves pending from the vertices of the wingspan.
The dominating wingspan rule now says: if rule common factor applies to the path between x_l and x_r, then the edge incident to x can be contracted.

As before, there has to be a cut within the edges of the $x_l - x_r$ path, w.l.o.g. it is on the left of x'. Since x_l is a *closest* quasi-$\mathcal{P}$-neighbor, this cut destroys all paths to terminalpairs over the left edge, so only paths to the right remain (there are no group-requests by definition of L_2^{good}). Now, for every multicut that contains $\{x, x'\}$, an equally good multicut can be

obtained by using the edge on the right of x' instead of $\{x, x'\}$.
The rule can also be applied, if both terminals of a terminalpair are on the $x_l - x_r$ path, because here again, there has to be one cut between x_l and x_r and the argument works as before. The argument also works, if x is an L_2^{bad} leaf and it has exactly one group request and no $\mathcal{P}$-neighbors in one direction of the caterpillars backbone.

In general trees, there might be an I_3 vertex x'_l or x'_r between x' and a closest quasi-$\mathcal{P}$-neighbor in at least one direction. In this case, there might be quasi-$\mathcal{P}$-neighbors in different subtrees of x'_l or x'_r. So it has to be made sure that still all paths in that direction are destroyed by a cut. This can be done by using x'_l instead of x_l or x'_r instead of x_r respectively. Also, if all $\mathcal{P}$-neighbors of x in that direction are in the same subtree of x'_l or x'_r, then the left or right boundary can be moved further into that subtree. In this case, the other subtrees of x'_l or x'_r might even contribute to the set of terminalpairs for the common factor rule.

If the quasi-$\mathcal{P}$-neighbors are in multiple directions, then take the closest quasi-$\mathcal{P}$-neighbors from all directions $z_1, \cdots, z_q$. If common factor applies to all $\{z_i, b\}$, where b is the farthest I_1 or I_2 vertex in the other direction, then $\{x, x'\}$ can also be contracted. This is the case because as before, there has to be a cut between b and z_i for $i = 1, \cdots, q$. Either one of these cuts is between b and the first I_3 vertex on the paths to $z_1, \cdots, z_q$ (so the argument can be applied as before) or all cuts are behind the first I_3 vertex (so they disconnect x from all $\mathcal{P}$-neighbors in that direction and again, $\{x, x'\}$ can be replaced in any multicut by the first incident edge of x' in the other direction).

Given its relationship to common factor, it is also unclear how this rule can be transferred to general graphs.

6.4 Polynomial kernels in uniform trees

The proof works by showing boundaries on the sizes of all partitions of V from the beginning of chapter 6.

Lemma 6.4.1. *For general graphs: at most $k^2 + k$ components are attached to any cut-vertex (for trees, this means $\Delta \leq k^2 + k$).*

Proof. Let $v \in V$ a cut-vertex. There can be at most k components in $(V - v, E)$ that contain an edge of a multicut $\leq k$ (pigeonhole principle). For each of these components C, there can be at most k other components

attached to v, that contain a $\mathcal{P}$-neighbor of a vertex in C, because otherwise the generalized L_3-leaves rule would apply. So all further components attached to v contain only idle edges and hence the component would be removed. □

- $|I_1| \leq k$ because every I_1-vertex has a group request (otherwise its edge to the internal tree would dominate all its other incident edges or there would be unit paths or idle edges). The group requests of different I_1 vertices are edge disjoint, so if $|I_1| > k$, then rule disjoint paths would apply.

- $|L_1| \leq k^2 + k$ because every L_1-leaf has a group request (to an L_1-leaf without group request, either the unique direction rule applies or it has a unit path or idle edge) and if an L_1-leaf had $> k$ group requests, then overloaded edge would apply. There are at most k L_1-leaves whose incident edge is part of a multicut $\leq k$ and each one of them has at most k group requests.

- $|L_2^{bad}| \leq k^2 + k$ for the same reason.

- $|I_3| \leq |I_1| \leq k$ because in any tree, the number of leaves is larger than the number of vertices with degree > 2 (induction). This also holds for the internal tree T'.

- There are at most $2 \cdot k - 1$ caterpillar components. Every caterpillar resides between I_3 or I_1 vertices. Select an arbitrary $r \in I_3$ as root. Map every caterpillar C to the I_1 or I_3 vertex, that C meets from above. This is a bijection and since $|I_1 \cup I_3| \leq 2 \cdot k$, there are at most $2 \cdot k$ caterpillars, but there is no caterpillar above r.

- $|L_3| \leq k^3 + k^2$ because $\Delta \leq k^2 + k$ (Lemma 6.4.1), which also holds for the $\leq k$ vertices in I_3.

- $|L_2^{good}| \in O(k^4)$ complicated. The proof basically says that if there was a large wingspan, then it would dominate many terminalpairs so the dominating wingspan rule would apply. If there are only small wingspans, then there cannot be many L_2^{good} leaves in one caterpillar, because this would entail that there are many wingspans, which pairwise do not intersect, so the disjoint request rule would apply. The claim then follows from the fact that there are at most $2 \cdot k - 1$ caterpillars.

- $\underline{|I_2^+| \leq |L_2^{good}| \in O(k^4)}$ because every I_2^+ vertex (by definition) has an L_2 leaf attached to it and every L_2^{good} leaf is attached to exactly one I_2^+ vertex. By the pigeonhole principle, this implies $|I_2^+| \leq |L_2^{good}|$.
- $\underline{|I_2^-| \in O(k^6)}$ complicated.

Chapter 7

Conclusion and perspectives

See [91] for denotation.

MIN-MULTICUT is

- Polynomial on
  - Uniform paths 3.1
  - Uniform cycles 3.1
  - Uniform DAGs, if MULTIWAY-CUT 2.4
  - Uniform trees, if MULTIWAY-CUT ..... [19]
  - Undirected multi-commodity networks with $l \leq 2$ 3.2
  - Directed "multi"-commodity networks with $l = 1$ 1.3.4
- NP-complete on
  - Uniform, undirected networks with $l \geq 3$ 3.3.1
  - Uniform, directed networks with $l \geq 2$ 3.3.2
  - Uniform, undirected stars 3.3.3
  - Uniform, undirected trees with $\Delta \geq 3$ 3.3.4
  - Uniform, undirected grids ..... [9]
  - Interval graphs ..... [52]
  - Uniform caterpillars with $\Delta = 5$ 3.3.5
  - Uniform, directed, acyclic graphs with $\Delta^+ = \Delta^- = 2$ 3.3.5
  - Uniform 13-layer digraphs 3.3.5

- Approximable to within
  - A factor of $\log(l)$ ..... [47]
  - A factor of $O(r^3)$ on $K_{r,r}$-free graphs ..... [37]
  - A constant factor on planar graphs ..... [37]
  - A factor of 2 on trees ..... [49]
  - Arbitrary factors (PTAS) on uniform with bounded Δ and tw [14]
- Not Approximable to (unless P=NP) within
  - Arbitrary factors (no PTAS, no FPTAS) 3.4
  - Constant factors unless Khot's unique game conjecture is false [16]
  - A factor of $O(\log(\log(n)))$ unless Khot's unique game conjecture is false ..... [16]
- FPT in all supersets of
  - $\{l, k\}$ 4.2
  - $\{l, tw\}$ 4.3
  - $\{tw((V, E \cup \mathcal{P}))\}$ 4.4
  - $\{l, \Delta\}$ 4.5
  - $\{k, \Lambda\}$ 4.5
- Not FPT (unless P=NP) in any subset of
  - $\{\Delta, tw, pw, \nu, \chi, \omega, b, \sigma, \lambda, \delta, \varphi, \mu, z\}$ 4.1
  - $\{tw, pw, \nu, \Lambda, dm, r, \chi, \psi, \psi_s, \omega, \alpha_0, \beta, \gamma, b, ir, \sigma, \lambda, \delta, \varphi, \mu, z,$ $n - \Delta, n - \Theta, n - \chi', n - \chi_T, n - \alpha, n - \beta_0, n - |\Gamma|\}$ 4.1
  - $\{l, dm, r, \delta, \varphi, \sigma, \lambda\}$ 4.1
- FPT in $\{k\}$ on
  - Trees with integral capacities 5.4
  - Large grids with real capacities ≥ 1 ..... 5.6
- FPT in $\{k\}$ and $\{l\}$ on
  - Uniform stars 5.1
  - Stars with real capacities ≥ 1 ..... 5.2

  - Uniform trees 5.3
  - Uniform grids 5.6

- FPT in $\{d\}$ on non-uniform trees, where d is the maximum number of $s_i - t_i$-paths that pass a vertex or edge ..... [54]

- FPT in $\{l\}$ on planar graphs ..... [6]

On uniform trees, Min-Multicut has kernels of size $O(k^6)$, so Min-Multicut on uniform trees is in the important complexityclass POLYKERNEL.
Min-Multicut is in the complexityclass XP (because of the $O(\Lambda^k) \subseteq O(n^k)$ running time of the algorithm in 4.5) and in $W[P] \supseteq W[t] \forall t \in \mathbb{N}$ (because to find a solution, an NTM can nondeterministically write k integers between 1 and n^2, which requires $k \cdot \log(n^2) = k \cdot 2 \cdot \log(n)$ uses of indeterminism to select k edges). For the definitions of XP and W[P], see [41].

The reduction rules for trees can be generalized

	undirected		directed		trees		
	u	n	u	n	$\mathbb{N}$	$\mathbb{R}^+$	
Idle edge	P	P	P	P	P	P	×
~~Unit path~~	-	-	-	-	-	-	-
Simple dominated edge	P	P	P	P	P	P	×
Full dominated edge	P		P				×
Simple dominated path	P	P	P	P	P	P	×
Generalized dominated path	P	P	P	P	P	P	×
Full dominated path					P	P	×
Disjoint paths	P	P	P	P	P	P	✓
Overloaded edge	P	P	P	P	P	P	✓
~~Overloaded caterpillar~~	-	-	-	-	-	-	-
Overloaded L_3 leaves	P	P	P	P	P	P	✓
~~Unique direction~~	-	-	-	-	-	-	✓
Common factor	?	?	?	?	?	?	
Dominating wingspan	?	?	?	?	?	?	

u	uniform
n	non-uniform
P	a generalized version of the rule is applicable in polynomial time
-	unit path is a special case of generalized overloaded edge unique direction is a special case of dominated edge overloaded caterpillar is a special case of common factor
?	it is unclear whether there is an analogue in general graphs to the rule
empty	the complexity of application of the rule is unknown
✓, ×	is the generalization, restricted to trees, more general than the original rule for trees?

The running-time of $O^*(\Lambda^k) \subseteq O(n^k)$ of the branching algorithm in 4.5 shows that for no constant k the problem is NP-complete, so lemma 1.4.1.2 cannot be applied to show fixed parameter intractability in k.

The greatest obstacle is that about every conjecture that would help parameterizing the problem in k, is untrue. To quantify this statement, here are several plausible conjectures that would be helpful, but that are untrue:

Conjecture 7.1. *Every / some min-multicut contains a minimum cut of some terminalpair.*

Disproof.
In fig. 7.1, the min-cuts of $s_1 - t_1$ are $\{\{s_1, t_1\}, \{t_1, t_2\}, \{t_1, s_2\}\}$ and

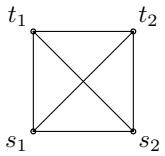


Figure 7.1:

$\{\{s_1, t_1\}, \{s_1, s_2\}, \{s_1, t_2\}\}$ and the min-cuts of $s_2 - t_2$ are symmetric. The minimum multicuts are $\{\{s_1, t_1\}, \{s_2, t_2\}, \{s_1, s_2\}, \{t_1, t_2\}\}$ and $\{\{s_1, t_1\}, \{s_2, t_2\}, \{s_1, t_2\}, \{s_2, t_1\}\}$. So every min-cut is diagonal to every min-multicut. □

Conjecture 7.2. *Every / some min-multicut contains an edge that is part of a min-cut for some terminalpair.*

Disproof.
In fig. 7.2, the min-cut of terminalpair $s_i - t_i$ is solely $\{e_i\}$. The unique min-multicut is $\{e'_1, e'_2, e'_3\}$, so every min-cut is disjoint to every min-multicut. □

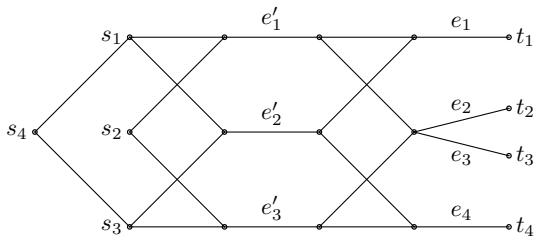


Figure 7.2:

Conjecture 7.3. *A min-multicut of every minor of (V, E) is contained in / has a common edge with every / some min-multicut for (V, E).*

Disproof.
In fig. 7.3, the unique min-multicut of the left network is $\{x, y\}$. Every

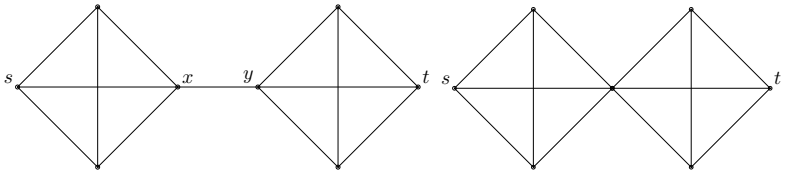


Figure 7.3:

multicut of the right network contains at least 3 edges. This gap can be arbitrarily larger, by using arbitrarily large complete subgraphs. This example also shows that min-multicuts of minors don't necessarily become smaller. But to be fair, this conjecture is true as long as there is a min-multicut, whose edges are not contracted in the construction of the minor. □

Conjecture 7.4. *Every / some min-multicut contains an edge that is part of a min-multicut for some spanning tree.*

Disproof.
In fig. 7.4, the min-multicut contains only the edge adjacent to t_1, t_2, but a spanning tree might be cut along the parallel paths in between. □

Conjecture 7.5. *Every min-multicut for some subset of $\mathcal{P}$ is subset of a min-multicut for $\mathcal{P}$*

Disproof.
In fig. 7.5, the min-multicut contains only the edge adjacent to t_1, t_2, but $\{s_1, s_2\}$ is a min-(multi)cut for $\mathcal{P} = \{\{s_2, t_2\}\}$. □

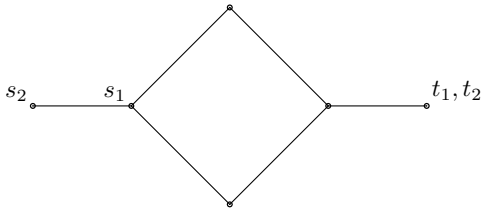


Figure 7.4:

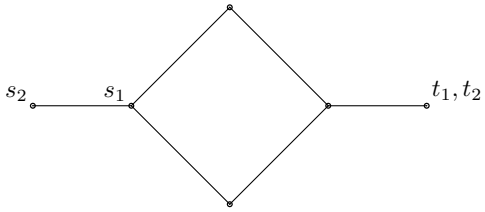


Figure 7.5:

Conjecture 7.6. *There is a subset $\mathcal{P}'$ of $\mathcal{P}$ with size $\leq 2^k$, for which a particular min-multicut remains a minimum multicut.*

Disproof.
The conjecture is true. However this does not help, because the converse is not true. Using e.g. color-coding to find this subset and computing a min-multicut for it (using Marx' algorithm, because with $l \leq 2^k$ its running-time becomes $O^*(4^{k(2^k-1)}k^k4^{k^3})$) can result in a set of edges that is a min-multicut for $\mathcal{P}'$, but not for $\mathcal{P}$, as fig. 7.5 shows. □

Despite all the obstacles, there is hope for solving the question whether Min-Multicut is FPT or W[1]-hard in k.

Many of the reduction rules for trees (see section 6) apply to the block-cutvertex graph (see definition 8.3 in [91]) of a general graph, so maybe the number of blocks in a general graph can be limited by a function in k. So the number of blocks might be a suitable property to limit the height of a recursion tree of the compression-phase in an interative compression.

The disjoint path rule (6.3.5) implies that there are at most k paths between a terminalpair and lemma 6.3.6 implies that the graph has no complete subgraph K_q with $q > k$. This seems to limit the number of crosslinks between the paths, so there might only be $\binom{k}{2}$ classes of significant edges, which would result in an $O^*((k^2)^k) = O^*(k^{2k})$ algorithm. On the

other hand, there seems to be no reason why the equivalence classes of one terminalpair should have any impact on the cuts for any other terminalpair.

If there is an $s_i - t_i$ path with length $\leq k$, then one of the edges on that path has to be selected, which means there are k sub-problems with parameter $k-1$, so this branching will not violate an FPT running-time, if it can be reached in the branches. Therefore Min-Multicut is FPT in k if and only if it is FPT in k on instances, where the shortest $s_i - t_i$ path has length $> k$. Lemma 6.3.6 implies that the graph has no complete subgraph K_q with $q > k$. These properties together seem to imply that the graph may be sparse, which might be a helpful property. On the other hand, the problem is not FPT in tw, so sparseness might not help at all.

The argument for the fixed parameter tractability of $k-$Disjoint Paths is that a graph with large treewidth contains a large grid, from which a vertex can be removed (which reduces the treewidth) without destroying any of the k paths, if they exist. Therefore the treewidth can be limited by a function in k. This argument seems to stay valid for Min-Multicut, so it seems reasonable to say that Min-Multicut is FPT in k if and only if it is FPT in $\{k, tw\}$, but every vertex of the contained grid may be a terminal. Removing a terminal should not be a good idea, but then again, a network that contains a large grid with many terminals might not have a multicut of size k, so this might not be a problem.

A promising approach to solving Min-Multicut instances in FPT time is a 2-dimensional iterative compression: One could arrange sub-problems of an instance in an $(l-1) \times (|E|-|V|-1)$ table. $(V, E, \mathcal{P}, c)$ could be put in the upper right corner and for every step to the left, one edge from a cycle could be removed (in a fixed order). For every step downwards, a terminalpair could be removed. The bottom row would contain single-commodity Min-Cut instances, which can be solved in polynomial time. The left column would contain Min-Multicut instances on trees, which is FPT in k. So the solutions for the left column and bottom row can be computed in sufficiently short time and it might be possible to fill the other cells successively by combining the solutions from below and from the left.

The common factor rule (6.3.10) sounds like it was related to Clique, but on trees, the paths that are used together can be found using matching algorithms. Maybe it would be possible to construct a Multicut instance from a Clique instance G so that common factor is applicable (which may be the case if the solution is sufficiently small) iff G has a clique of size k.

This would imply W[1]-hardness, since CLIQUE is W[1]-hard in k [41, 15].

One approach to a W[1]-hardness proof may be cutting edges in the implicationgraph of ANTIMONOTONE-W[1] instances (which exist because ANTIMONOTONE-W[1] instances are weighted 2-SAT instances), but this is only a vague idea.

Bibliography

[1] Isolde Adler. Lecture notes: Baumzerlegungen, algorithmen und logik. 2009.

[2] Matthew Andrews, Julia Chuzhoy, and Lisa Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. In FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, Murray Hill, NJ, USA, October 23-25, 2005, pages 226–244. IEEE Computer Society, 2005. doi: 10.1109/SFCS.2005.41, isbn: 0-7695-2468-0.

[3] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. Journal of Algorithms, 12(2):308–340, 1991. doi: 10.1016/0196-6774(91)90006-K.

[4] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. Journal of the Association for Computing Machinery, 45(3):501–555, 1998. doi: 10.1145/278298.278306.

[5] Christina Bazgan. Schémas d'approximation et complexité paramétrée. PhD thesis, 1995.

[6] Cédric Bentz. Edge disjoint paths and multicut problems in graphs generalizing the trees. Technical Report No 948, CEDRIC, 2005.

[7] Cédric Bentz. On the complexity of the multicut problem in bounded tree-width graphs and digraphs. Discrete Applied Mathematics, 156(10):1908–1917, 2008. doi: 10.1016/j.dam.2007.09.013.

[8] Cédric Bentz, Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. A bibliography on multicut and integer multiflow problems. Technical report, 2004.

[9] Cédric Bentz, Marie-Christine Costa, and Frédéric Roupin. Maximum integer multiflow and minimum multicut problems in two-sided uniform grid graphs. Journal of Discrete Algorithms, 5(1):36–54, 2007. doi: 10.1016/j.jda.2006.03.009.

[10] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, San Diego, CA, USA, May 16-18, 1993, pages 226–234. ACM, 1993. doi: 10.1145/167088.167161, isbn: 0-89791-591-7.

[11] Hans L. Bodlaender. Treewidth: Structure and algorithms. In Giuseppe Prencipe and Shmuel Zaks, editors, SIROCCO 2007: Structural Information and Communication Complexity, 14th International Colloquium, Castiglioncello, Italy, June 5-8, 2007, Proceedings, volume 4474 of Lecture Notes in Computer Science, pages 11–25. Springer, 2007. isbn: 978-3-540-72918-1.

[12] Nicolas Bousquet, Jean Daligault, Stéphan Thomassé, and Anders Yeo. A polynomial kernel for multicut in trees. In Susanne Albers and Jean-Yves Marion, editors, STACS '09: 26th International Symposium on Theoretical Aspects of Computer Science, Proceedings, Freiburg, Germany, February 26-28, 2009, volume 3 of Leibniz International Proceedings in Informatics, pages 183–194. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. doi: 10.4230/LIPIcs.STACS.2009.1824, isbn: 978-3-939897-09-5.

[13] Liming Cai and Jianer Chen. On fixed-parameter tractability and approximability of np optimization problems. Journal of Computer and System Sciences, 54(3):465–474, 1997. doi: 10.1006/jcss.1997.1490.

[14] Gruia Călinescu, Cristina G. Fernandes, and Bruce Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. Journal of Algorithms, 48(2):333–359, 2003. doi: 10.1016/S0196-6774(03)00073-7.

[15] Marco Cesati. Compendium of parameterized problems, 2006.

[16] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. In CCC '05: Proceedings of the 20th Annual IEEE Conference on Computational Complexity, San Jose, CA, USA, June

11-15, 2005, volume 15, pages 94–114. IEEE Computer Society, 2006. doi: 10.1109/CCC.2005.20, isbn: 0-7695-2364-1.

[17] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In MFCS '06: Mathematical Foundations of Computer Science 2006, 31st International Symposium, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings, volume 4162 of Lecture Notes in Computer Science, pages 238–249. Springer, 2006. doi: 10.1007/11821069_21, isbn: 3-540-37791-3.

[18] Jianer Chen, Yang Liu, and Songiian Lu. Directed feedback vertex set problem is fpt. In Erik Demaine, Gregory Z. Gutin, Daniel Marx, and Ulrike Stege, editors, Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs, Freiburg, Germany, July 08-13, 2007, number 07281 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[19] Sunil Chopra and Mendu Rammohan Rao. On the multiway cut polyhedron. Networks, 21(1):51–89, 1991. doi: 10.1002/net.3230210106.

[20] Stephen Arthur Cook. The complexity of theorem-proving procedures. In STOC '71: Proceedings of the third annual ACM symposium on Theory of computing, Shaker Heights, OH, USA, May 03-05, 1971, pages 151–158. ACM, 1971. doi: 10.1145/800157.805047.

[21] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. Introduction to Algorithms. MIT Press, 2001. isbn: 0-262-03293-7.

[22] Marie-Christine Costa and Lucas Létocart. Polynomial algorithms to solve the multiway cut and integer flow problems on trees. In ECCO XV: Conference of the European Chapter on Combinatorial Optimisation, Lugano, Switzerland, May 30 - June 1, 2002, CEDRIC report, 2002.

[23] Marie-Christine Costa, Lucas Létocart, and Frederic Roupin. Minimal multicut and maximal integer multiflow: A survey. European Journal of Operational Research, 162(1):55–69, 2005. doi: 10.1016/j.ejor.2003.10.037.

[24] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics, pages 193–242. Elsevier, 1990. isbn: 0-444-88074-7.

[25] W. H. Cunningham. The optimal multiterminal cut problem. In Fred Roberts, Frank Hwang, and Clyde Monma, editors, Reliability of Computer and Communication Networks, volume 5 of DIMACS Series in discrete mathematics and theoretical computer science, pages 105–120. ACM, 1991. isbn: 0-8218-6592-7.

[26] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. SIAM Journal on Computing, 23(4):864–894, 1994. doi: 10.1137/S0097539792225297.

[27] George B. Dantzig. Linear Programming and Extensions. Princeton University Press, 1963.

[28] Reinhard Diestel. Graphentheorie. Springer Verlag, 2000. isbn: 978-3540676560.

[29] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.

[30] Yefim Dinitz. Algorithm for solution of a problem of maximum flow. Soviet Mathematics Doklady (English translation from Networks with power estimation, Doklady Akademii Nauk), 11:1277–1280, 1970.

[31] Rodney G. Downey and Michael Ralph Fellows. Fixed-parameter tractability and completeness i: Basic results. SIAM Journal on Computing, 24(4):873–921, 1995. doi: 10.1137/S0097539792228228.

[32] Rodney G. Downey and Michael Ralph Fellows. Parameterized Complexity. Springer-Verlag, 1998. isbn: 0-387-94883-X.

[33] Jack R. Edmonds. Paths, trees and flowers. Canadian Journal of Mathematics, (17):449–467, 1965.

[34] Jack R. Edmonds and Richard Manning Karp. Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the Association for Computing Machinery, 19(2):248–264, 1972. doi: 10.1145/321694.321699.

[35] John A. Ellis, I. H. Sudborough, and Jonathan S. Turner. The vertex separation and search number of a graph. Information and Computation, 113(1):50–79, 1994. doi: 10.1006/inco.1994.1064.

[36] Paul Erdös and R. Rado. A partition calculus in set theory. Bulletin of the American Mathematical Society, 62(5):427–489, 1956. doi: 10.1090/S0002-9904-1956-10036-0.

[37] Éva Tardos and Vijay V. Vazirani. Improved bounds for the max-flow min-multicut ratio for planar and $k_{r,r}$ -free graphs. Information Processing Letters, 47(2):77–80, 1993. doi: 10.1016/0020-0190(93)90228-2.

[38] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. SIAM Journal on Computing, 5(4):691–703, 1976. doi: 10.1137/0205048.

[39] Shimon Even and Robert Endre Tarjan. Network flow and testing graph connectivity. SIAM Journal on Computing, 4(4):507–518, 1975. doi: 10.1137/0204043.

[40] Ron Fagin. Generalized first-order spectra and polynomial time recognizable sets. In Richard Manning Karp, editor, Complexity of Computations, volume 7 of SIAM–AMS Proceedings, pages 43–73. AMS, 1974.

[41] Jörg Flum and Martin Grohe. Parameterized Complexity Theory. Springer-Verlag, 2006. isbn: 3-540-29952-1.

[42] Lester Randolph Ford and Delbert Ray Fulkerson. Maximal flow through a network. Canadian Journal of Mathematics, 8:399–404, 1956.

[43] Lester Randolph Ford and Delbert Ray Fulkerson. Flows in networks. Princeton University Press, 1962. isbn: 0-691-07962-5.

[44] Markus Frick. Easy Instances for Model Checking. PhD thesis, 2001.

[45] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. Annals of Pure and Applied Logic, 130(1):3–31, 2004. doi: 10.1016/j.apal.2004.01.007.

[46] M. R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979. isbn: 0-7167-1044-7.

[47] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. SIAM Journal on Computing, 25(2):698–707, 1993. doi: 10.1137/S0097539793243016.

[48] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway cuts in directed and node weighted graphs. In Serge Abiteboul and Eli Shamir, editors, ICALP '94: Proceedings of the 21st International Colloquium on Automata, Languages and Programming, Jerusalem, Israel, July 11-14, 1994, Proceedings, volume 820 of Lecture Notes in Computer Science, pages 487–498. Springer-Verlag, 1994. doi: 10.1007/3-540-58201-0_92, isbn: 3-540-58201-0.

[49] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica, 18(1):3–20, 1997. doi: 10.1007/BF02523685.

[50] Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. Journal of the Association for Computing Machinery, 35(4):921–940, 1988. doi: 10.1145/48014.61051.

[51] Georg Gottlob and Stephanie Tien Lee. A logical approach to multicut problems. Information Processing Letters, 103(4):136–141, 2007.

[52] Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for vertex multicut in interval and bounded treewidth graphs. European Journal of Operational Research, 186(2):542–553, 2008. doi: 10.1016/j.ejor.2007.02.014.

[53] Jiong Guo and Rolf Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. Networks, 46(3):124–135, 2005. doi: 10.1002/net.20081.

[54] Jiong Guo and Rolf Niedermeier. Exact algorithms and applications for tree-like weighted set cover. Journal of Discrete Algorithms, 4(4):608–622, 2006. doi: 10.1016/j.jda.2005.07.005.

[55] Mohammad Taghi Hajiaghayi and Tom Leighton. On the max-flow min-cut ratio for directed multicommodity flows. Theoretical Computer Science, 352(1):318–321, 2006. doi: 10.1016/j.tcs.2005.10.037.

[56] T. C. Hu. Multi-commodity network flows. Operations Research, 11(3):344–360, 1963. doi: 10.1287/opre.11.3.344.

[57] Alon Itai. Two-commodity flow. Journal of the Association for Computing Machinery, 25(4):596–611, 1978. doi: 10.1145/322092.322100.

[58] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. Combinatorica, 4(4):373–396, 1984.

[59] Richard Manning Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, Complexity of Computer Computations, Proceedings of a symposium on the complexity of computer computations, Yorktown Heights, NY, USA, March 20-22, 1972, pages 85–103. Plenum Press, 1972. isbn: 0-306-30707-3.

[60] Richard Manning Karp. On the complexity of combinatorial problems. Networks, 5(1):45–68, 1975.

[61] A. V. Karzanov. Determining the maximum flow in a network by the method of preflows. Soviet Mathematics Doklady, 15:434–437, 1974.

[62] Subhash Khot. On the power of unique 2-prover 1-round games. In STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing 2002, Montreal, Quebec, Canada, May 19 - 21, 2002, pages 767–775. ACM, 2002. doi: 10.1145/509907.510017, isbn: 1-58113-495-9.

[63] Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. Theoretical Computer Science, 47(2):205–218, 1986. doi:10.1016/0304-3975(86)90146-5.

[64] Philip N. Klein, Ajit Agrawal, R. Ravi, and Satish Rao. Approximation through multicommodity flow. In FOCS '90: Proceedings of the 31st Annual Symposium on Foundations of Computer Science, St. Louis, MO, USA, October 22-24, 1990, pages 726–737. IEEE Computer Society, 1990. doi: 10.1109/FSCS.1990.89595, isbn: 0-8186-2082-X.

[65] Philip N. Klein, Serge A. Plotkin, and Satish Rao. Planar graphs, multicommodity flow, and network decomposition. In STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, San Diego, CA, USA, May 16-18, 1993, pages 682 – 690. ACM, 1993. doi: 10.1145/167088.167261, isbn: 0-89791-591-7.

[66] Antonius J. J. Kloks. Treewidth, Computations and Approximations, volume 842 of Lecture Notes in Computer Science. Springer, 1994. isbn: 3-540-58356-4.

[67] Joachim Kneis and Alexander Langer. A practical approach to courcelle's theorem. In MEMICS '08: Proceedings of the International

Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, Znojmo, Czech Republic, November 14-16, 2008, volume 251 of Electronic Notes in Theoretical Computer Science, pages 65–81, 2009. doi: 10.1016/j.entcs.2009.08.028.

[68] M. E. Kramer and J. van Leeuwen. Leeuwen, the complexity of wire-routing and minimum area layouts for arbitrary vlsi circuits. In Franco P. Preparata, editor, Advances in computing research, vol. 2: VLSI theory, pages 129–146. JAI Press, 1984. isbn: 0-89232-461-9.

[69] Kazimierz Kuratowski. Sur le probléme des courbes gauches en topologie. Fundamenta Mathematicae, 15:271–283, 1930.

[70] Edmund Landau. Handbuch der Lehre von der Verteilung der Primzahlen. Teubner, 1909. 2 volumes. Reprinted by Chelsea, New York, 1953.

[71] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms. In FOCS '88: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, White Plains, NY, USA, October 24-26, 1988, pages 101–111. IEEE Computer Society, 1988. doi: 10.1109/SFCS.1988.21958, isbn: 0-8186-0877-3.

[72] Vishv M. Malhotra, M. Pramodh Kumar, and S. N. Maheshwari. An $O(|V|^3)$ algorithm for finding maximum flows in networks. Information Processing Letters, 7(6):277–278, 1978.

[73] Dániel Marx. Parameterized graph separation problems. Theoretical Computer Science, 351(3):394–406, 2006. doi: 10.1016/j.tcs.2005.10.007.

[74] Matthias Middendorf and Frank Pfeiffer. On the complexity of the disjoint paths problems. Combinatorica, 13(1):97–107, 1993.

[75] G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. Mathematical Programming, (8), 1975.

[76] Yu. E. Nesterov and A. S. Nemirovskii. An interior-point method for generalized linear-fractional programming. Mathematical Programming, 69(1):177–204, 1995. doi: 10.1007/BF01585557.

[77] Rolf Niedermeier. Invitation to fixed-parameter algorithms. Oxford University Press, 2006. isbn: 0-198-56607-7.

[78] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. In STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing Chicago, IL, USA, May 02 - 04, 1988, pages 229–234. ACM, 1988. doi: 10.1145/62212.62233, isbn: 0-89791-264-0.

[79] Serge A. Plotkin and Éva Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. In STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, San Diego, CA, USA, May 16-18, 1993, pages 691–697. ACM, 1993. doi: 10.1145/167088.167263, isbn: 0-89791-591-7.

[80] Igor Razgon and Barry O'Sullivan. Directed feedback vertex set is fixed-parameter tractable. In Erik Demaine, Gregory Z. Gutin, Daniel Marx, and Ulrike Stege, editors, Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs, Freiburg, Germany, July 08-13, 2007, number 07281 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[81] Neil Robertson and Paul D. Seymour. Graph minors i. excluding a forest. Journal on Combinatorial Theory, Series B, 35(1):39–61, 1983. doi: 10.1016/0095-8956(83)90079-5.

[82] Neil Robertson and Paul D. Seymour. Graph minors – a survey. In I. Anderson, editor, Surveys in Combinatorics, pages 153–171. Cambridge University Press, 1985. isbn: 0521315247.

[83] Neil Robertson and Paul D. Seymour. Graph minors ii. algorithmic aspects of tree-width. Journal of Algorithms, 7(3):309–322, 1986. doi: 10.1016/0196-6774(86)90023-4.

[84] Neil Robertson and Paul D. Seymour. Graph minors. xiii: the disjoint paths problem. Journal of Combinatorial Theory, Series B, 63(1):65–110, 1995. doi: 10.1006/jctb.1995.1006.

[85] Neil Robertson and Paul D. Seymour. Graph minors. xx. wagner's conjecture. Journal of Combinatorial Theory, Series B, 92(2):325–357, 2004. doi: 10.1016/j.jctb.2004.08.001.

[86] Peter Rossmanith. Lecture notes: Parametrisierte algorithmen. 2007.

[87] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. Journal on Combinatorial Theory, Series B, 58(1):22–33, 1993. doi: 10.1006/jctb.1993.1027.

[88] Torsten Tholey. Solving the 2-disjoint paths problem in nearly linear time. Theory of Computing Systems, 39(1):51–78, 2006. doi: 10.1007/s00224-005-1256-9.

[89] Spyros Tragoudas. VLSI partitioning approximation algorithms based on multicommodity flow and other techniques. PhD thesis, 1991.

[90] Alan Mathison Turing. Computability and lambda-definability. Journal of Symbolic Logic, 2(4):153–163, 1937.

[91] Lutz Volkmann. Fundamente der Graphentheorie. Springer-Verlag, 1996. isbn: 3-211-82774-9.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Geilenkirchen, den 22. Juni 2010