The Parameterised Complexity Of Integer Multicommodity Flow

Hans L. Bodlaender
Utrecht University, The Netherlands

Isja Mannens
Utrecht University, The Netherlands

Jelle J. Oostveen
Utrecht University, The Netherlands

Sukanya Pandey
Utrecht University, The Netherlands

Erik Jan van Leeuwen
Utrecht University, The Netherlands

Abstract

The Integer Multicommodity Flow problem has been studied extensively in the literature. However, from a parameterised perspective, mostly special cases, such as the Disjoint Path problem, have been considered. Therefore, we investigate the parameterised complexity of the general Integer Multicommodity Flow problem. We show that the decision version of this problem on directed graphs for a constant number of commodities, when the capacities are given in unary, is XNLP-complete with pathwidth as parameter and XALP-complete with treewidth as parameter. When the capacities are given in binary, the problem is NP-complete even for graphs of pathwidth at most 13. We give related results for undirected graphs. These results imply that the problem is unlikely to be fixed-parameter tractable by these parameters.

In contrast, we show that the problem does become fixed-parameter tractable when weighted tree partition width (a variant of tree partition width for edge weighted graphs) is used as parameter.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases multicommodity flow, parameterised complexity, XNLP-completeness, XALP-completeness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.6

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2310.05784

Funding *Isja Mannens*: The research of Isja Mannens was supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 853234).
Jelle J. Oostveen: The research of Jelle Oostveen was supported by the NWO grant OCENW.KLEIN.114 (PACAN).

1 Introduction

The Multicommodity Flow problem is the generalisation of the textbook flow problem where instead of just one commodity, multiple different commodities have to be transported through a network. The problem models important operations research questions (see e.g. [32]). Although several optimisation variants of this problem exist [32], we consider only the variant where for each commodity, a given amount of flow (the demand) has to be sent from the commodity's source to its sink, subject to a capacity constraint on the total

amount flow through each arc. The nature and computational complexity of the problem is highly influenced by the graph (undirected or directed, its underlying structure) and the capacities, demands, and flow value (integral or not, represented in unary or binary). When the flow values are allowed to be fractional, the problem can be trivially solved through a linear program (see e.g. [22, 25]).

We focus on Integer Multicommodity Flow, where all the given capacities and demands are integers and the output flow must be integral. The Integer Multicommodity Flow problem is widely studied and well known to be NP-hard even if all capacities are 1, on both directed and undirected graphs, even when there are only two commodities [11]. On directed graphs, it is NP-hard even for two commodities of demand 1 [14]. These strong hardness results have led to a range of heuristic solution methods as well as a substantial body of work on approximation algorithms. For surveys, see e.g., [1, 32, 33].

An important special case of Integer Multicommodity Flow and the main source of its computational hardness is the Edge Disjoint Paths problem. It can be readily seen that Integer Multicommodity Flow is equivalent to Edge Disjoint Paths when all capacities and demands are 1. Indeed, all aforementioned hardness results stem from this connection. The Edge Disjoint Paths problem has been studied broadly in its own right (see e.g. the surveys by Frank [16] and Vygen [31]), including a large literature on approximation algorithms. See, amongst others [21, 30] for further hardness and inapproximability results. On undirected graphs, Edge Disjoint Paths is fixed-parameter tractable parameterised by the number of source-sink pairs [28, 24].

Investigation of the parameterised complexity of Edge Disjoint Paths has recently been continued by considering structural parameterisations. Unfortunately, the problem is NP-hard for graphs of treewidth 2 [26] and even for graphs with a vertex cover of size 3 [13]. It is also W[1]-hard parameterised by the size of a vertex set whose removal leaves an independent set of vertices of degree 2 [19]. From an algorithmic perspective, Ganian and Ordyniak [19] showed that Edge Disjoint Paths is in XP parameterised by tree-cut width. Zhou et al. [34] give two XP algorithms for Edge Disjoint Paths for graphs of bounded treewidth: one for when the number of paths is small, and one for when a specific condition holds on the pairs of terminals. Ganian et al. [20] give an FPT algorithm parameterised by the treewidth and degree of the graph. Friedrich et al. [17, 18] give approximation algorithms for multicommodity flow on graphs of bounded treewidth.

These results naturally motivate the question: *What can we say about the parameterised complexity of the general Integer Multicommodity Flow problem under structural parameterisations?* We are unaware of any explicit studies in this direction. We do note that the result of Zhou et al. [34] implies an XP algorithm on graphs of bounded treewidth for a bounded number of commodities if the capacities are given in unary. We are particularly interested in whether this result can be improved to an FPT algorithm, which is hitherto unknown.

Our Setting and Contributions

We consider the Integer Multicommodity Flow problem for a small, fixed number of commodities. In particular, Integer ℓ-Commodity Flow is the variant in which there are ℓ commodities. Furthermore, we study the setting where some well-known structural parameter of the input graph, particularly its pathwidth or treewidth, is small.

Our main contribution is to show that Integer 2-Commodity Flow is unlikely to be fixed-parameter tractable parameterised by treewidth and or by pathwidth. Instead of being satisfied with just a W[t]-hardness result for some t or any t, we seek stronger results using the recently defined complexity classes XNLP and XALP. An overview of our results can be found in Table 1.

Table 1 Overview of our results for Integer 2-Commodity Flow. para-NP-complete = NP-complete for fixed value of parameter. (1) = capacities of arcs inside bags can be arbitrary, capacities of arcs between bags are bounded by weighted tree partition width. (2) Approximation, see Theorem 1.11; conjectured in FPT. For the undirected case, the same results hold, except that for the para-NP-completeness for the parameters pathwidth and treewidth, we need a third commodity.

Parameter	unary capacities	binary capacities
pathwidth	XNLP-complete	para-NP-complete
treewidth	XALP-complete	para-NP-complete
weighted tree partition width	FPT (1)	FPT (1)
vertex cover	(2); in XP	(2); open

XNLP is the class of parameterised problems that can be solved on a non-deterministic Turing machine in $f(k)|x|^{O(1)}$ time and $f(k)\log|x|$ memory for a computable function f, where $|x|$ is the size of the input x. The class XNLP (under a different name) was first introduced by Elberfeld et al. [9]. Bodlaender et al. [2, 5, 7] showed a number of problems to be XNLP-complete with pathwidth as parameter. In particular, [2] gives XNLP-completeness proofs for several flow problems with pathwidth as parameter.

In this work, we prove XNLP-completeness (and stronger) results for Integer ℓ-Commodity Flow. These give a broad new insight into the complexity landscape of Integer Multicommodity Flow. We distinguish how the capacities of arcs and edges are specified: these can be given in either unary or binary. First, we consider the unary case:

▶ **Theorem 1.1.** *Integer 2-Commodity Flow with capacities given in unary, parameterised by pathwidth, is XNLP-complete.*

▶ **Theorem 1.2.** *Undirected Integer 2-Commodity Flow with capacities given in unary, parameterised by pathwidth, is XNLP-complete.*

These hardness results follow by reduction from the XNLP-complete Chained Multicoloured Clique problem [6], a variant of the perhaps more familiar Multicoloured Clique problem [12]. We follow a common strategy in such reductions, using vertex selection and edge verification gadgets. However, a major hurdle is to use flows to select vertices and verify the existence of edges to form the sought-after cliques. To pass this hurdle, we construct gadgets that use Sidon sets as flow values, combined with gadgets to check that a flow value indeed belongs to such a Sidon set.

For the parameter treewidth, we are able to show a slightly stronger result. Recently, Bodlaender et al. [6] introduced the complexity class XALP, which is the class of parameterised problems that can be solved on a non-deterministic Turing machine that has access to an additional stack, in $f(k)|x|^{O(1)}$ time and $f(k)\log|x|$ space (excluding the space used by the stack), for a computable function f, where $|x|$ again denotes the size of the input x. Many problems that are XNLP-complete with pathwidth as parameter are XALP-complete with treewidth as parameter. This also holds for the Integer Multicommodity Flow problem:

▶ **Theorem 1.3 (♣).** *Integer 2-Commodity Flow with capacities given in unary, parameterised by treewidth, is XALP-complete.*

The reduction is from the XALP-complete Tree-Chained Multicoloured Clique problem [7] and follows similar ideas as the above reduction. Combining techniques of the proofs of Theorems 1.2 and 1.3 gives the following result.

▶ **Theorem 1.4 (♣).** *Undirected Integer 2-Commodity Flow with capacities given in unary, parameterised by treewidth, is XALP-complete.*

Assuming the *Slice-wise Polynomial Space Conjecture* [27, 5], these results show that XP-algorithms for Integer 2-Commodity Flow or Undirected Integer 2-Commodity Flow for graphs of small pathwidth or treewidth cannot use only $f(k)|x|^{O(1)}$ memory. Moreover, the results imply these problems are $W[t]$-hard for all positive integers t.

If the capacities are given in binary, then the problems become even harder.

▶ **Theorem 1.5.** *Integer 2-Commodity Flow with capacities given in binary is NP-complete for graphs of pathwidth at most* 13*.*

▶ **Theorem 1.6 (♣).** *Undirected Integer 3-Commodity Flow with capacities given in binary is NP-complete for graphs of pathwidth at most* 18*.*

Finally, we consider a variant of the Integer Multicommodity Flow problem where the flow must be *monochrome*, i.e. a flow is only valid when no edge carries more than one type of commodity. Then, we obtain hardness even for parameterisation by the vertex cover number of the graph, for both variants of the problem.

▶ **Theorem 1.7 (♣).** *Integer 2-Commodity Flow with Monochrome Edges is NP-hard for binary weights and vertex cover number* 6*, and W[1]-hard for unary weights when parameterised by the vertex cover number.*

▶ **Theorem 1.8 (♣).** *Undirected Integer 2-Commodity Flow with Monochrome Edges is NP-hard for binary weights and vertex cover number* 6*, and W[1]-hard for unary weights when parameterised by the vertex cover number.*

To complement our hardness results, we prove two algorithmic results. Bodlaender et al. [2] had given FPT algorithms for several flow problems, using the recently defined notion of weighted tree partition width as parameter (see [3, 2]). Weighted tree partition width can be seen as a variant of the notion of *tree partition width* for edge-weighted graphs, introduced by Seese [29] in 1985 under the name *strong treewidth*. See Section 2 for formal definitions of these parameters. The known hardness for the vertex cover number [13] implies that Edge Disjoint Paths is NP-hard even for graphs of tree partition width 3. Here, we prove that:

▶ **Theorem 1.9.** *The Integer ℓ-Commodity Flow problem can be solved in time $2^{2^{\mathrm{b}^{3\ell\mathrm{b}}}} n^{O(1)}$, where* b *is the breadth of a given tree partition of the input graph.*

▶ **Theorem 1.10.** *The Undirected Integer ℓ-Commodity Flow problem can be solved in time $2^{2^{\mathrm{b}^{3\ell\mathrm{b}}}} n^{O(1)}$, where* b *is the breadth of a given tree partition of the input graph.*

For the standard Integer 2-Commodity Flow problem with the vertex cover number $\mathrm{vc}(G)$ of the input graph G as parameter, we conjecture that this problem is in FPT. As a partial result, we can give the following approximation algorithms:

▶ **Theorem 1.11 (♣).** *There is a polynomial-time algorithm that, given an instance of Integer 2-Commodity Flow on a graph G with demands d_1, d_2, either outputs that there is no flow that meets the demands or outputs a 2-commodity flow of value at least $d_i - O(\mathrm{vc}(G)^3)$ for each commodity $i \in [2]$.*

▶ **Theorem 1.12 (♣).** *There is a polynomial-time algorithm that, given an instance of* Undirected Integer 2-Commodity Flow *on a graph G with demands d_1, d_2, either outputs that there is no flow that meets the demands or outputs a 2-commodity flow of value at least $d_i - O(\mathrm{vc}(G)^3)$ for commodity $i \in [2]$.*

Proofs of theorems marked by ♣ appear in the full version. For other theorems, we are only able to provide proof sketches in the limited space.

2 Preliminaries

In this paper, we consider both directed and undirected graphs. Graphs are directed unless explicitly stated otherwise. Arcs and edges are denoted as vw (an arc from v to w, or an edge with v and w as endpoints).

We use the interval notation for intervals of integers, e.g., $[-1, 3] = \{-1, 0, 1, 2, 3\}$. We simplify this notation for intervals that start at 1, i.e. $[k] = [1, k]$. Moreover, we use $\mathbb{N} = \{1, 2, \ldots\}$ and $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.

A *Sidon set* is a set of positive integers $\{a_1, a_2, \ldots, a_n\}$ such that all pairs have a different sum, i.e., when $a_i + a_{i'} = a_j + a_{j'}$ then $\{i, i'\} = \{j, j'\}$. Sidon sets are also Golomb rulers and vice versa – in a Golomb ruler, pairs of different elements have unequal differences: if $i \neq i'$ and $j \neq j'$, then $|a_i - a_{i'}| = |a_j - a_{j'}|$, then $\{i, i'\} = \{j, j'\}$. A construction by Erdös and Turán [10] for Sidon sets implies the following, cf. the discussion in [8].

▶ **Theorem 2.1.** *A Sidon set of n elements in $[4n^2]$ can be found in $O(n\sqrt{n})$ time and logarithmic space.*

We now formally define our flow problems. A *flow network* is a pair (G, c) of a directed (undirected) graph $G = (V, E)$ and a function $c : E \to \mathbb{N}_0$ that assigns to each arc (edge) a non-negative integer *capacity*. We generally use $n = |V|$ and $m = |E|$.

For a positive integer ℓ, an *ℓ-commodity flow* in a flow network with sources $s_1, \ldots, s_\ell \in V$ and sinks $t_1, \ldots, t_\ell \in V$ is a ℓ-tuple of functions $f^1, \ldots, f^\ell : E \to \mathbb{R}_{\geq 0}$, that fulfils the following conditions:

- **Flow conservation.** For all $i \in [\ell]$, $v \notin \{s_i, t_i\}$, $\sum_{wv \in E} f^i(wv) = \sum_{vw \in E} f^i(vw)$.
- **Capacity.** For all $vw \in E$, $\sum_{i \in [\ell]} f^i(vw) \leq c(vw)$.

An ℓ-commodity flow is an *integer ℓ-commodity flow* if for all $i \in [c]$, $vw \in E$, $f^i(vw) \in \mathbb{N}_0$. The *value for commodity i* of an ℓ-commodity flow equals $\sum_{s_i w \in E} f^i(s_i w) - \sum_{w s_i \in E} f^i(w s_i)$. We shorten this to "flow" when it is clear from context what the value of ℓ is and whether we are referring to an integer or non-integer flow.

The main problem considered in the paper now is as follows:

Integer ℓ-Commodity Flow
Input: A flow network $G = (V, E)$ with capacities c, sources $s_1, \ldots, s_\ell \in V$, sinks $t_1, \ldots, t_\ell \in V$, and demands $d_1, \ldots, d_\ell \in \mathbb{N}$.
Question: Does there exist an integer ℓ-commodity flow in G which has value d_i for each commodity $i \in [\ell]$?

The Integer Multicommodity Flow problem is the union of all Integer ℓ-Commodity Flow problems for all non-negative integers ℓ.

For undirected graphs, flow still has direction, but the capacity constraint changes to:

- **Capacity.** For all $vw \in E$, $\sum_{i \in [\ell]} f^i(vw) + f^i(wv) \leq c(vw)$.

The undirected version of the Integer ℓ-Commodity Flow problem then is as follows:

Undirected Integer ℓ-Commodity Flow
Input: An *undirected* flow network $G = (V, E)$ with capacities c, sources $s_1, \ldots, s_\ell \in V$, sinks $t_1, \ldots, t_\ell \in V$, and demands $d_1, \ldots, d_\ell \in \mathbb{N}$.
Question: Does there exist an integer ℓ-commodity flow in G which has value d_i for each commodity $i \in [\ell]$?

We use the well-known parameters *treewidth* and *pathwidth* without giving an explicit definition. For the parameter (weighted) tree partition width, refer to [2] (see also [3]). We also use these parameters for directed graphs. In that case, the direction of edges is ignored.

The classes XNLP and XALP were defined in the introduction. XNLP-hardness and XALP-hardness are defined with respect to pl-reductions. The main difference with the more standard parameterized reductions is that the computation of the reduction must be done with logarithmic space. In most cases, existing parameterized reductions are also pl-reductions; logarithmic space is achieved by not storing intermediate results but recomputing these when needed.

Our hardness results stem from two variants of Multicoloured Clique (see [12]):

Chained Multicoloured Clique
Input: A graph $G = (V, E)$, a partition of V into $V_1, \ldots, V_r$, such that $|i - j| \leq 1$ for each edge $uv \in E(G)$ with $u \in V_i$ and $v \in V_j$, and a function $c\colon V \to [k]$.
Parameter: k.
Question: Is there a set of vertices $W \subseteq V$ such that for all $i \in [r-1]$, $W \cap (V_i \cup V_{i+1})$ is a clique, and for each $i \in [r]$ and $j \in [k]$, there is a vertex $v \in W \cap V_i$ with $c(v) = j$?

Tree-Chained Multicoloured Clique
Input: A graph $G = (V, E)$, a tree partition $(\{V_i \mid i \in I\}, T = (I, F))$ with T a tree of maximum degree 3, and a function $c\colon V \to [k]$.
Parameter: k.
Question: Is there a set of vertices $W \subseteq V$ such that for all $ii' \in F$, $W \cap (V_i \cup V_{i'})$ is a clique, and for each $i \in I$ and $j \in [k]$, there is a vertex $v \in W \cap V_i$ with $c(v) = j$?

▶ **Theorem 2.2** (From [6] and [7]). *Chained Multicoloured Clique is XNLP-complete, and Tree-Chained Multicoloured Clique is XALP-complete.*

3 Hardness Results – Unary Capacities

We prove our hardness results for Integer Multicommodity Flow with unary capacities, parameterised by pathwidth. We aim to reduce from Chained Multicoloured Clique. It is good to know that all constructions will have disjoint sources and sinks for the different commodities. We will set the demands for each commodity equal to the total capacity of the outgoing arcs from the sources, which is equal to the total capacity of the incoming arcs to the sinks. Thus, the flow over such arcs will be equal to their capacity. Furthermore, throughout this section, our constructions will have two commodities. We name the commodities 1 and 2, with sources s_1, s_2 and sinks t_1, t_2, respectively.

We first introduce two types of gadgets: subgraphs that fulfil certain properties and that are used in the hardness constructions. Given an integer a, the *a-Gate gadget* (see Figure 1) either can move 1 unit of flow from one commodity from left to right, or at most a units

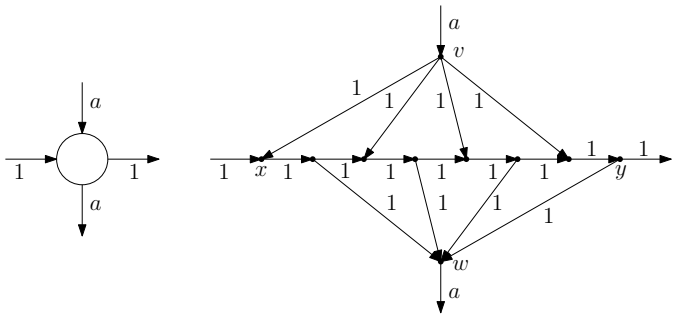


Figure 1 The a-Gate gadget. Left: the schematic representation of the gadget, with its entry and exit arcs. Right: the full construction for $a = 4$, with arcs labelled by their capacities.

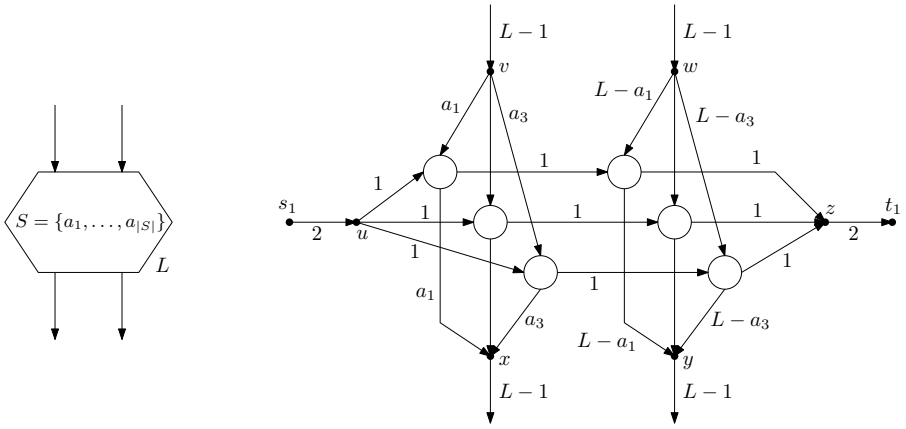


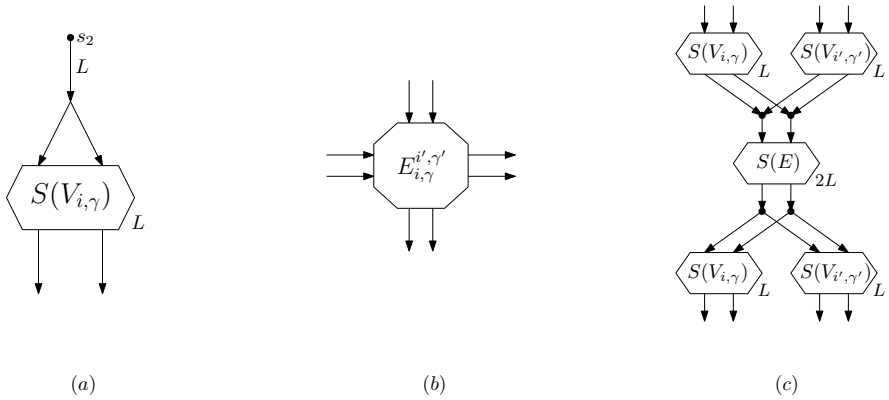
Figure 2 The (S, L)-Verifier gadget. Left: a schematic representation of the gadget, with its entry and exit arcs. The value on the bottom-right of the schematic representation denotes the sum of the incoming flows to the gadget. Right: the graph that realises the gadget for $S = \{a_1, a_2, a_3\}$, with arcs labelled by their capacities (the unlabelled arcs have capacities a_2 and $L - a_2$ respectively; their labels are omitted for clarity).

of flow from the other commodity from top to bottom, but not both. Hence, it models a form of choice. This gadget will grow in size with a, and thus will only be useful if the input values are given in unary. Given a set S of integers and a large integer L (larger than any number in S), the (S, L)-*Verifier* (see Figure 2) is used to check if the flow over an arc belongs to a number in S. The a-Gate gadget is used as a sub-gadget in this construction. In our reduction, later, we will use appropriately constructed sets S to select vertices or to check for the existence of edges.

Both types of gadget have constant pathwidth. The arcs incoming and outgoing of the gadget are called the *entry arcs* and *exit arcs* respectively.

Our hardness construction will be built using only Verifier gadgets as subgadgets. The entry arcs and exit arcs of this gadget are meant to transport solely flow of commodity 2. Hence, in the remainder, it helps to think of only commodity 2 being transported along the edges, to focus on the exact value of that flow. This value indicate which vertex is selected or whether two selected vertices are adjacent.

▶ **Theorem 1.1.** *Integer 2-Commodity Flow with capacities given in unary, parameterised by pathwidth, is XNLP-complete.*



(a) (b) (c)

Figure 3 (a) A Vertex selector gadget. (b) A schematic representation of an Edge check gadget for $V_{i,\gamma}$ and $V_{i',\gamma'}$. (c) The construction of an Edge check gadget.

Proof sketch. Proof of membership in XNLP follows immediately from a dynamic programming algorith. For the hardness, we reduce from Chained Multicolour Clique (see Theorem 2.2). Suppose we have an instance of Chained Multicolour Clique, with a graph $G = (V, E)$, colouring $c : V \to [k]$, and partition $V_1, \ldots, V_r$ of V.

Build a Sidon set with $|V|$ numbers by applying the algorithm of Theorem 2.1. Following the same theorem, the numbers are in $[4|V|^2]$. Set $L = 4|V|^2 + 1$ to be a "large" integer. To each vertex $v \in V$, we assign a unique element of the set S, denoted by $S(v)$. For any subset $V' \subseteq V$, let $S(V') = \{S(v) \mid v \in W\}$. For any subset $E' \subseteq E$, let $S(E') = \{S(u) + S(v) \mid uv \in E'\}$.

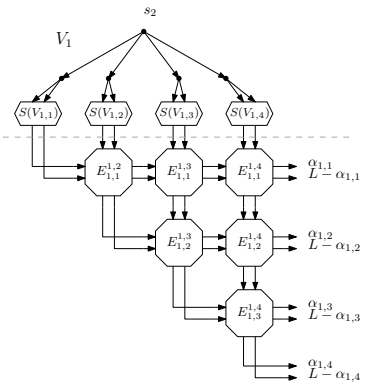
We now describe several (further) gadgets that we use to build the full construction. Let $V_{i,\gamma}$ be the vertices in V_i with colour γ. Each set $V_{i,\gamma}$ is called a *class*. For each class $V_{i,\gamma}$, we use a *Vertex selector gadget* (see Figure 3) to select the vertex from $V_{i,\gamma}$ that should be in the solution to the Chained Multicoloured Clique instance. We select some $v \in V_{i,\gamma}$ if and only if the left branch receives $S(v)$ flow and the right branch receives $L - S(v)$ flow.

For each pair of incident classes, we construct an *Edge check gadget* (see Figure 3). That is, we have an Edge check gadget for all classes $V_{i,\gamma}$ and $V_{i',\gamma'}$ with $|i - i'| \leq 1$, and $\{i, \gamma\} \neq \{i', \gamma'\}$. An Edge check gadget will check if two incident classes have vertices selected that are adjacent. If the entry arcs have flow (of commodity 2) of value $S(v)$, $L - S(v)$, $S(w)$, and $L - S(w)$ consecutively, then there is a valid flow if and only if $vw \in E$. Note that the sum $S(v) + S(w)$ is unique, because S is a Sidon set, and thus so is $2L - (S(v) + S(w))$. Hence, the only way for the flow to split up again and leave via the exit arcs is to split into $S(v)$, $S(w)$, $L - S(v)$, and $L - S(w)$; otherwise, it cannot pass the $(S(V_{i,\gamma}), L)$-Verifier or the $(S(V_{i',\gamma'}), L)$-Verifier at the bottom of the Edge check gadget. Hence, the exit arcs again have flow of values $S(v)$, $L - S(v)$, $S(w)$, and $L - S(w)$ consecutively, just like the entry arcs.

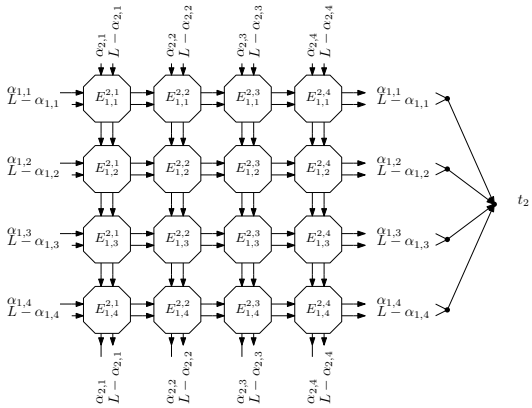
With these gadgets in hand, we now describe the global structure of the reduction. For each class $V_{i,\gamma}$, we first create a Vertex selector gadget (as in Figure 3).

We then create Edge check gadgets to check, for any $i \in [r]$, that the selected vertices in $V_{i,\gamma}$ for all $\gamma \in [k]$ are adjacent in G. The construction is shown in Figure 4a. We call this the *Triangle gadget* for V_i.

Next, we create Edge check gadgets to check, for any $i \in [r-1]$, that the selected vertices in $V_{i,\gamma}$ and $V_{i+1,\gamma'}$ for all $\gamma, \gamma' \in [k]$ are indeed adjacent in G. The construction is shown in Figure 4b. We call this the *Square gadget* for V_i and V_{i+1}.



(a) The Triangle gadget for V_1 consists of Edge check gadgets to enforce the selected vertices form a clique in V_1. Here, $k = 4$. The Vertex selector gadgets for each class $V_{1,\gamma}$ are also shown. The $\alpha_{1,\gamma}$'s denote the amount of flow selected in the corresponding $(S(V_{1,\gamma}), L)$-Verifiers.



(b) The Square gadget for V_1 and V_2 has Edge check gadgets to enforce the selected vertices form a clique between V_1 and V_2. Here, $k = 4$.

Figure 4

Then, we connect the Vertex selector, Triangle and Square gadgets as in Figure 5. We now set the demand for commodity 1 to the sum of the capacities of the outgoing arcs of s_1 (which is equal to the sum of the capacities of the incoming arcs of t_1). We set the demand for commodity 2 to the sum of the capacities of the outgoing arcs of s_2 (which is equal to the sum of the capacities of the incoming arcs of t_2). This completes the construction.

▷ Claim 3.1 (♣). The constructed graph has pathwidth at most $8k + O(1)$.

▷ Claim 3.2. The given Chained Multicolour Clique instance has a solution if and only if the constructed instance of Integer 2-Commodity Flow has a solution.

Proof sketch. For the forward direction, assume there exists a chained multicolour clique W in G. Recall that one vertex is picked per $V_{i,\gamma}$ class by definition and thus W has size rk. If $v \in V_{i,\gamma} \cap W$ for some $i \in [r], \gamma \in [k]$, then in the Vertex selector gadget of $V_{i,\gamma}$, we send $S(v)$ units of flow of commodity 2 to the left and $L - S(v)$ units of flow to the right into the Verifier gadget (see Figure 3). In any Verifier gadget, we route the flow so that it takes the path with capacity equal to the flow. This flow is then routed through all Edge check gadgets of the Triangle and Square gadgets, in the manner presented above in the description of Edge check gadgets. Since W is a chained multicolour clique, the corresponding edge exist in E and the flow can indeed pass through the $(S(E), 2L)$-Verifier gadget of each Edge check gadget. All flow of commodity 1 is routed through the unused gates in the Verifier gadgets, which is possible as we only use one vertical path per gadget for the flow corresponding to a vertex or edge (see Figure 2). It follows that we use all arcs from s_1 and to t_1 to capacity.

For the other direction, suppose there is a 2-commodity flow in the constructed graph with all arcs from s_1 and s_2 and to t_1 and t_2 used to capacity. Since the constructed graph is acyclic, we can apply induction on its topological ordering to show that flow of commodity 1 never leaves a Verifier gadget downwards. Then the construction of the Verifier gadget ensures that that the amount flow of commodity 2 that passes through it always corresponds to some $\alpha \in S$ for the associated set S of the gadget, and the left and right exit arcs carry α and $L - \alpha$ units of flow of commodity 2 respectively. Now, the arc from s_2 in a Vertex selector

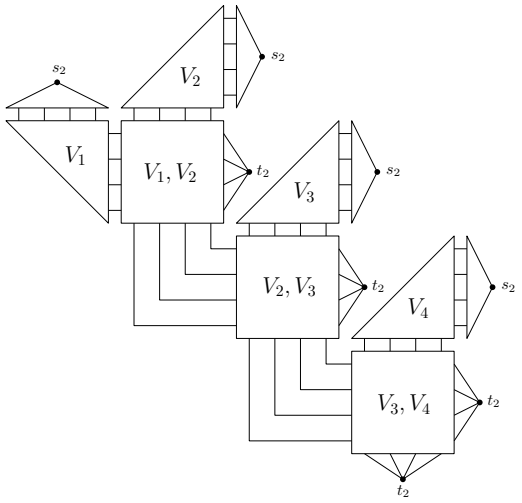


Figure 5 Overview of the complete structure of the reduction for $r = 4$. Triangles represent a structure as in Figure 4a, and squares a structure as in Figure 4b. Directions are not drawn, but clear from Figure 4a and 4b. The labels inside each block (say V_i or V_i, V_{i+1}) denote that flow corresponding to vertices of this set (i.e. V_i or V_i and V_{i+1}) is flowing in a block. Note that all points labelled s_2, t_2 are indeed the same vertex.

gadget for some class $V_{i,\gamma}$ must have L flow of commodity 2 and this must be split in α and $L - \alpha$, with $\alpha \in S(V_{i,\gamma})$. We place the corresponding vertex v in the chained multicoloured clique. In any Edge check gadget, the flow of $\alpha \in S(V_{i,\gamma})$, $L - \alpha$ and $\beta \in S(V_{i',\gamma'})$,$L - \beta$ combines to a unique sum $\alpha + \beta$ and $2L - (\alpha + \beta)$, and assures that the edge between the corresponding vertices is present. The flow must split back up into α, $L - \alpha$ and $\beta, L - \beta$ by the unique sum due to the fact that S is a Sidon set. We get that the chosen vertices indeed form a chained multicolour clique. ◁

Finally, using Theorem 2.1 and standard log-space techniques, the constructed graph with its capacities can be built with $O(f(k) + \log n)$ space, for some computable function f. ◀

To show the XALP-hardness of the Integer 2-Commodity Flow parameterised by treewidth, we reduce from Tree-Chained Multicolour Clique in a similar, but more involved manner (see Figure 6 for an illustration) and obtain Theorem 1.3.

We now reduce from the case of directed graphs to the case of undirected graphs in a general manner, by modification of a transformation by Even et al. [11, Theorem 4]. In this way, both our hardness results (for parameter pathwidth and for parameter treewidth) can be translated to undirected graphs.

▶ **Lemma 3.3.** *Let G be a directed graph of an Integer 2-Commodity Flow instance with capacities given in unary. Then in logarithmic space, we can construct an equivalent instance of Undirected Integer 2-Commodity Flow with an undirected graph G' with $\mathrm{pw}(G') \leq \mathrm{pw}(G) + O(1)$, $\mathrm{tw}(G') \leq \mathrm{tw}(G) + O(1)$, and unit capacities.*

Proof sketch. Given a directed graph $G = (V, E)$, demands d_1 and d_2, and capacity function $c : E \to \mathbb{N}_0$, we construct an instance G', d_1' and d_2', and $c' : E(G') \to \{0, 1\}$. To the graph G, we add four new vertices $\overline{s_1}, \overline{s_2}, \overline{t_1}, \overline{t_2}$ as new sources and sinks. We connect $\overline{s_i}$ to s_i and $\overline{t_i}$ to t_i by d_i parallel undirected edges of capacity 1, for each $i \in \{1, 2\}$. Next, for each arc $uv \in E$ of capacity p, we create p parallel undirected edges between u and v of capacity 1 each. Then,

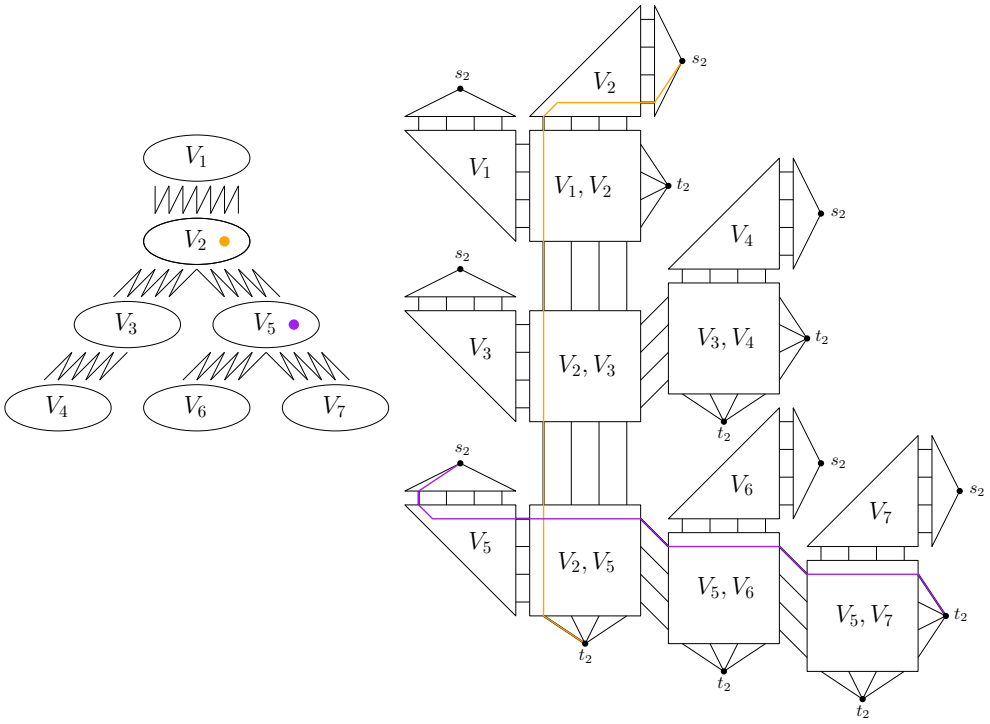


Figure 6 Overview of the structure of the reduction of Theorem 1.3. Left: the structure of the input tree partition. Right: the structure of the reduction. Triangles represent a structure as in Figure 4a, and squares a structure as in Figure 4b. Directions are not drawn, but clear from Figure 4a and 4b. The labels inside each block (say V_i or V_i, V_{i+1}) denote that flow corresponding to vertices of this set (i.e. V_i or V_i and V_{i+1}) is flowing in a block. Note that all points labelled s_2, t_2 are indeed the same vertex. Flow paths corresponding to a selected vertex in V_2 (orange) and one in V_5 (purple) are drawn as an example.

we replace each of these p undirected edges by the Diamond gadget of Figure 7(a). This is the graph G'. In G', the demands on the two commodities are $d'_1 = d_1 + e^*$ and $d'_2 = d_2 + e^*$, where e^* is the number of edge gadgets in G' (i.e. the sum of all capacities in c).

The pathwidth of G' is $\mathrm{pw}(G) + O(1)$ and the treewidth is $\mathrm{tw}(G) + O(1)$. Moreover, the demands d_1 and d_2 are met in G if and only if the demands d'_1 and d'_2 are met in G'. The construction can be done in logarithmic space: while scanning G, we can output G'. ◀

Theorem 1.2 and 1.4 follow immediately from this lemma and Theorem 1.1 and 1.3.

4 Hardness Results – Binary Capacities

Our previous reduction strategy relied heavily on a-Gate gadgets, which have size linear in a, and thus only work in the case a unary representation of the capacities is given. For the case of binary capacities, we can prove stronger results by reducing from 2-Partition. However, we need a completely new chain of gadgets and constructions.

We define three different types of (directed) gadgets. Since we use binary capacities, our goal is to double flow in an effective manner. For a given integer a, the *a-Doubler gadget* receives a flow and sends out $2a$ flow of the same commodity. This gadget is obtained by combining two other gadgets: the a-Switch and the Doubling a-Switch. The *a-Switch gadget* changes the type of flow; that is, it receives a flow from one commodity, but sends out a

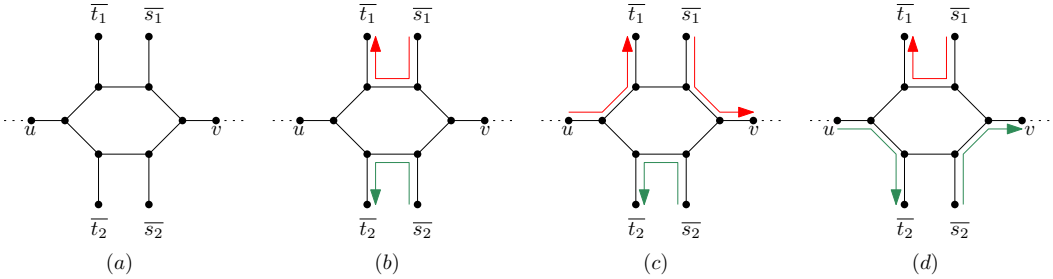


Figure 7 The transformation of Lemma 3.3 from directed to undirected graphs. Every arc uv with capacity c is replaced by c parallel copies of gadget (a), where $\overline{t_1}, \overline{s_1}, \overline{t_2}, \overline{s_2}$ are the same for every gadget, for all arcs. All capacities are 1. The remaining figures illustrate that the gadget either transports no flow (b), a flow of commodity 1 (c), or a flow of commodity 2 (d).

flow from the other commodity. The *Doubling a-Switch* is similar, but sends out $2a$ flow. All three types of gadgets have constant size, even in the binary setting. Moreover, any a-Doubler gadget has pathwidth 5.

A crucial property of the a-Doubler gadget are the following. It has two (vertical) entry arcs and two exit arcs. If the left entry arc carries a units of flow of commodity 2 and the right entry arc carries 0 units of flow of commodity 2, then the left exit arc carries $2a$ units of flow of commodity 2 and the right exit arc carries 0 units of flow of commodity 2. The same property holds with left and right swapped.

▶ **Theorem 1.5.** *Integer 2-Commodity Flow with capacities given in binary is NP-complete for graphs of pathwidth at most* 13.

Proof sketch. Membership in NP is trivial. To show NP-hardness, we transform from Partition. Recall Partition problem asks, given positive integers $a_1, \ldots, a_n$, to decide if there is a subset $S \subseteq [n]$ with $\sum_{i \in S} a_i = B$, where $B = \sum_{i=1}^{n} a_i/2$. This problem is well known to be NP-complete [23].

Create the sources s_1, s_2 and the sinks t_1, t_2. Create two vertices b_1, b_2, both with an arc of capacity B to t_2.

For each a_i, we build a Binary gadget that either sends a_i units of flow to b_1 or a_i units of flow to a vertex b_2, in each case of commodity 2. This will indicate whether or not a_i is in the solution set to the Partition instance. This gadget is constructed as follows (see Figure 8 for the case when $a_i = 13$). Consider the binary representation $a_i^p, \ldots, a_i^0$ of a_i. That is, $a_i = \sum_{j=0}^{p} 2^j a_i^j$. For each $j \in [p]$ such that $a_i^j = 1$, we create a column of chained Doubler gadgets. For each $j' < j$, create a $2^{j'}$-Doubler gadget and identify its entry arcs with the exit arcs of the $2^{j'-1}$-Doubler gadget (see Figure 8). Then the left exit arc of the (final) 2^{j-1}-Doubler gadget is directed to b_1, while the right exit arc is directed to b_2.

Create two directed paths P_i^1, P_i^2 of $2\sum_{j=0}^{p} a_i^j$ vertices each (see Figure 8). We consider the vertices of each of these paths in consecutive pairs, one pair for each a_i^j that is equal to 1. For each $j \in [p]$ such that $a_i^j = 1$, create a vertex v_i^j with an arc from s_2, an arc to the first vertex of the pair on P_i^1 corresponding to a_i^j, and an arc to the first vertex of the pair on P_i^2 corresponding to a_i^j. Then, add an arc from the second vertex of the pair on P_i^1 corresponding to a_i^j to the left entry arc of the 1-Doubler gadget of the jth column of gadgets and an arc from the second vertex of the pair on P_i^2 corresponding to a_i^j to the right entry arc of the 1-Doubler gadget of the jth column of gadgets. Finally, create a vertex u_i

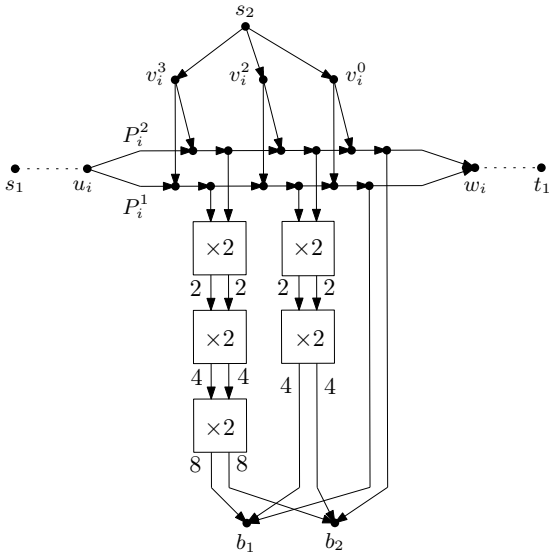


Figure 8 Example of the Binary gadget for $a_i = 13$ and its associated paths P_i^1 and P_i^2 and the ends u_i and w_i. Since $13 = 2^3 + 2^2 + 2^0$, we have a column with three Doubler gadgets (indicated by the square boxes), a column with two Doubler gadgets, and one with no Doubler gadgets. The vertices v_i^3, v_i^2 and v_i^0 are also drawn. Arcs are labelled by their capacities, but unlabelled arcs have capacity 1. If 1 unit of flow of commodity 1 is sent from s_1 to t_1, then it must pick one of P_i^1, P_i^2 to go through. Hence, the gadget ensures that either a_i units of flow of commodity 2 are sent to b_1 through the left entry and exit arcs of the Doubler gadgets, or a_i units of flow of commodity 2 are sent to b_2 through the left entry and exit arcs of the Doubler gadgets.

with an arc to the first vertex of P_i^1 and to the first vertex of P_i^2 and create a vertex w_i with an arc from the last vertex of P_i^1 and the last vertex of P_i^2. This completes the description of the Binary gadget.

We now chain the Binary gadgets. For each $i \in [n-1]$, add an arc from w_i to u_{i+1}. Add an arc from s_1 to u_1 and from w_n to t_1. These arcs all have capacity 1.

We set the demand for commodity 1 to the sum of the capacities of the outgoing arcs of s_1 (which is equal to the sum of the capacities of the incoming arcs of t_1). We set the demand for commodity 2 to the sum of the capacities of the outgoing arcs of s_2 (which is equal to the sum of the capacities of the incoming arcs of t_2). This completes the construction.

▷ Claim 4.1 (♣). The constructed graph has pathwidth at most 13.

▷ Claim 4.2. The given Partition instance has a solution if and only if the constructed instance of Integer 2-Commodity Flow has a solution.

Proof sketch. Let $S \subseteq [n]$ be a solution to the Partition instance. For each $i \in [n]$, we do the following. If $i \in S$, then we send flow of commodity 2 from s_2 to b_1, through left entry and exit arcs of the Doubler gadgets in the Binary gadget corresponding to a_i. To reach this left side of the Doubler gadgets, the flow passes through vertices and arcs of P_i^1. We can thus send flow of commodity 1 from u_i to w_i via P_i^2. Otherwise, if $i \notin S$, we send flow of commodity 2 from s_2 to b_2, through right entry and exit arcs of the Doubler gadgets in the Binary gadget corresponding to a_i. To reach this right side of the Doubler gadgets, the flow passes through vertices and arcs of P_i^2. We send flow of commodity 1 from u_i to w_i via P_i^1.

By the properties of the Doubler gadget, b_1 will receive a_i units of flow of commodity 2 if $i \in S$ and b_2 will receive a_i units of flow of commodity 2 if $i \notin S$. Since S is a solution to Partition, both b_1 and b_2 receive B units of flow of commodity 2, which they can then pass on to t_2. Moreover, we observe that we can send 1 unit of flow from s_1 to t_1 via the paths P_i^1 and P_i^2, using $i \notin S$ and $i \in S$ respectively.

In the other direction, we see that the flow of commodity 1 starting at u_1 takes a path which is a union of $P_i^{j_i}$ paths, for $i \in [n]$ and $j_i \in \{1, 2\}$. In particular, this flow does not "leak" into any Doubler gadget. Then, similar to the above, let $S \subseteq [n]$ be the set of indices i for which the flow of commodity 2 through the Binary gadget corresponding to a_i arrives at b_1. We can see that S is a valid solution to the Partition instance. ◁

Finally, as each a-Doubler has constant size, the gadget for some a_i has size $O(\log^2(a_i))$, which is polynomial in the input size. Hence, the construction as a whole has size polynomial in the input size. Moreover, it can clearly be computed in polynomial time. ◀

We now reduce from the case of directed graphs to the case of undirected graphs in a general manner. We define a new gadget similar to the one for the unary case. However, we note that there we required a copies of the gadget if the capacity of an arc is a, which is not feasible in the case of binary capacities. Also note that increasing the capacities of the gadget by Even et al. [11, Theorem 4], here Figure 7, invalidates the gadget, as any under-capacity edge would allow flow in the other direction. Hence, we need a different gadget, the Directed edge gadget, which we do not describe in detail here.

▶ **Lemma 4.3 (♣).** *Let G be a directed graph of an Integer ℓ-Commodity Flow instance with a path decomposition of width w, such that each bag contains the sources and sinks of commodities $1, \ldots, \ell$. Then in polynomial time, we can construct an equivalent instance of Undirected Integer $\ell + 1$-Commodity Flow of pathwidth at most $w + 5$.*

By combining Lemma 4.3 and Theorem 1.5, we obtain Theorem 1.6.

5 Algorithm for Parameter Weighted Tree Partition Width

We give an FPT-algorithm for Integer ℓ-Commodity Flow parameterised by weighted tree partition width. This algorithm assumes that a tree partition of the input graph is given. There is an algorithm by Bodlaender et al. [4] that for any graph G and integer w, runs in time $\text{poly}(w) \cdot n^2$ and either outputs a tree partition of G of width $\text{poly}(w)$ or outputs that G has no tree partition of width at most w. By some simple tricks, this can be expanded to approximate weighted tree partition width as well, at the expense of a slightly worse polynomial in w. An approximately optimal tree partition of this form would be sufficient as input to our algorithm.

▶ **Theorem 1.9.** *The Integer ℓ-Commodity Flow problem can be solved in time $2^{2^{\mathrm{b}^{3\ell b}}} n^{O(1)}$, where* b *is the breadth of a given tree partition of the input graph.*

Proof. We will describe a dynamic-programming algorithm on a given tree partition $(T, (B_x)_{x \in V(T)})$. Let $r \in V(T)$ be some node, that we will designate as the root of the tree T. For convenience, we first attach a node to every leaf, with an empty bag.

We will create a table τ, where every entry is indexed by a node x of the tree partition and a collection $\mathbf{f}_x$ of functions f_x^i, one function for every commodity $i \in [\ell]$. We will refer to $\mathbf{f}_x$ as a flow profile and use the superscript i to refer to the flow function for commodity i in the profile. The function

$$f_x^i : B_{p(x)} \to [-\mathrm{b}, \mathrm{b}],$$

where $p(x)$ the parent node of x, indicates for every $v \in B_{p(x)}$ the net difference between the amount of flow of commodity i that v receives from (indicated by a positive value) or sends to (indicated by a negative value) the vertices in the bag B_x, in the current partial solution. That is, $f_x^i(v)$ models the value of $\sum_{u \in B_x}(f^i(uv) - f^i(vu))$, where f denotes the current partial solution. Notice that this sum has value in $[-\mathrm{b}, \mathrm{b}]$, as the sum over all capacities of edges between bags B_x and $B_{p(x)}$ is at most b. The content of each table entry will be a boolean that indicates whether there exists a partial flow on the graph considered up to x that is consistent with the indices of the table entry.

We will build the table τ, starting at the leaves of the tree, for which we assumed the corresponding bags to be empty sets, and working towards the root. If x is a leaf in the tree partition, we set $\tau[x, \{\emptyset, \ldots, \emptyset\}] = \text{True}$, where we denote by $\emptyset$, the unique function with the empty set as domain. Otherwise, x is some node with children $y_1, \ldots, y_t$. We will group these children y_i in equivalence classes ξ, defined by the equivalence relation $y \sim y'$ if and only if $\tau[y, \mathbf{f}_x] = \tau[y', \mathbf{f}]$ for every flow profile $\mathbf{f}$. Note that there are at most $2^{\mathrm{b}^{\ell(2\,\mathrm{b}+1)}}$ such equivalence classes, with at most $\mathrm{b}^{\ell(2\,\mathrm{b}+1)}$ possible flow profiles $\mathbf{f}_{y_j} = (f_{y_j}^1, \ldots, f_{y_j}^\ell)$ for every child y_j of x.

We will now describe an integer linear program that determines the value of $\tau[x, \mathbf{f}_x]$ for a given flow profile $\mathbf{f}_x$. We define a variable $X_{\xi,\mathbf{g}}$ as the number of sets in class ξ whose in- and outflow we choose to match flow profile $\mathbf{g}$[1]. We also define a variable Y_e^i for each edge inside the bag B_x or between B_x and its parent bag, which indicates the flow of commodity i on this edge. We will denote by $N^{in}(v)$ and $N^{out}(v)$ the set of in-neighbours and out-neighbours of v, respectively, restricted to $B_x \cup B_{p(x)}$. We now add constraints for the following properties, for every commodity $i \in [\ell]$. Flow conservation for all vertices v in the bag B_x, that are not a sink/source for commodity i:

$$\sum_{u \in N^{in}(v)} Y_{uv} + \sum_{\xi,\mathbf{g}} X_{\xi,\mathbf{g}} \cdot g^i(v) = \sum_{u \in N^{out}(v)} Y_{uv}.$$

The flow of commodity i from a source s_i (if $s_i \in B_x$):

$$-\sum_{u \in N^{in}(s_i)} Y_{us_i} + \sum_{u \in N^{out}(s_i)} Y_{us_i} - \sum_{\xi,\mathbf{g}} X_{\xi,\mathbf{g}} \cdot g^i(s_i) = d_i$$

The flow of commodity i to a sink t_i (if $t_i \in B_x$):

$$\sum_{u \in N^{in}(t_i)} Y_{ut_i} - \sum_{u \in N^{out}(t_i)} Y_{ut_i} + \sum_{\xi,\mathbf{g}} X_{\xi,\mathbf{g}} \cdot g^i(t_i) = d_i$$

The desired flow to a vertex v in the parent bag:

$$\sum_{u \in N^{in}(v) \setminus B_{p(x)}} Y_{uv} - \sum_{u \in N^{out}(v) \setminus B_{p(x)}} Y_{uv} = f_x^i(v).$$

Edge capacities and non-negative flow:

$$0 \leq \sum_{i=1}^{\ell} Y_e^i \leq c(e)$$

[1] Throughout the proof, if we sum over pairs $\xi, \mathbf{g}$, we only sum over flow profiles that are valid for bags in ξ. Alternatively, we can set any invalid $X_{\xi,\mathbf{g}}$ to 0 beforehand.

The number of flow profiles of each type from each class matches the number of bags in that class:

$$\sum_{g:B_x \to [-\,\mathrm{b},\mathrm{b}]} X_{\xi,\mathbf{g}} = |\xi|.$$

$X_{\xi,\mathbf{g}}$ must be a non-negative integer:

$$X_{\xi,\mathbf{g}} \in \mathbb{N}_0$$

We then use an algorithm of Frank and Tardos [15, Theorem 5.3] to solve the ILP in time $N^{2.5N+o(N)}$, where N is the number of variables in the ILP. This number is dominated by the number of variables $X_{\xi,\mathbf{g}}$, of which there are $O(2^{\mathrm{b}^{\ell(2\,\mathrm{b}+2)}})$. We thus find a running time of $2^{\mathrm{b}^{\ell(2\,\mathrm{b}+2)}(2.5\cdot 2^{\mathrm{b}^{\ell(2\,\mathrm{b}+2)}}+o(2^{\mathrm{b}^{\ell(2\,\mathrm{b}+2)}}))}$. If the ILP has a feasible solution, we set $\tau[x,\mathbf{f}_x] = \text{True}$; otherwise, we set $\tau[x,\mathbf{f}_x] = \text{False}$. We solve $\mathrm{b}^{\ell(2\,\mathrm{b}+1)}$ such ILP's per bag in the decomposition and thus find a total running time of $O(2^{2^{\mathrm{b}^{3\ell b}}})$

Once we reach the root bag, we use a similar ILP to compute the flow on the root bag, finding a final solution. We find a total running time of $2^{2^{\mathrm{b}^{3\ell b}}} n^{O(1)}$. ◀

Note that with some minor changes to the ILP (flow variables can be negative and there is no distinction between in/out edges), this proof also works in the undirected case, proving Theorem 1.10

6 Conclusions

The parameterised complexity analysis of integer multicommodity flow shows that the problem is already hard for several natural parameterisations, e.g., treewidth and pathwidth, even when there are only two commodities. The XNLP- and XALP-completeness imply that the problems have XP algorithms but which are likely also to use $\Omega(n^{f(k)})$ space by the Slice-wise Polynomial Space Conjecture. Moreover, the XNLP- and XALP-completeness results imply that the problems are W[t]-hard.

We end the paper with some open problems. A number of cases for undirected graphs remain unresolved. We conjecture that for several such cases, the complexity results will be analogue to the directed case. A notable open case is Undirected Integer 2-Commodity Flow, which we conjecture is NP-complete for graphs with a pathwidth bound, but Theorem 1.6 only gives the result with three commodities.

We also conjecture that Integer 2-Commodity Flow is fixed parameter tractable with the vertex cover number as parameter, possibly by using a dynamic programming algorithm that only needs to investigate solutions that are "close" to the approximate solution found by Theorem 1.11.

Finally, we believe that the problem may be interesting to investigate on certain graph classes, for example planar graphs of bounded treewidth or in general on graphs of treewidth or pathwidth below the bounds given by our hardness results.

References

1 Cynthia Barnhart, Niranjan Krishnan, and Pamela H. Vance. Multicommodity flow problems. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 2354–2362. Springer US, 2009. doi:10.1007/978-0-387-74759-0_407.

2 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. In Michael A. Bekos and Michael Kaufmann, editors, *Proceedings 48th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2022*, volume 13453 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2022. doi:10.1007/978-3-031-15914-5_7.

3 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. *arXiv*, abs/2202.06838, 2022. arXiv:2202.06838.

4 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. On the parameterized complexity of computing tree-partitions. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.7.

5 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. In Holger Dell and Jesper Nederlof, editors, *Proceedings 17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.8.

6 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michal Pilipczuk. On the complexity of problems on tree-structured graphs. In Holger Dell and Jesper Nederlof, editors, *Proceedings 17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.6.

7 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *Proceedings 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 193–204. IEEE, 2022. doi:10.1109/FOCS52979.2021.00027.

8 Hans L. Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020*, volume 180 of *LIPIcs*, pages 4:1–4:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.4.

9 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.

10 P. Erdös and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, s1-16(4):212–215, 1941. doi:10.1112/jlms/s1-16.4.212.

11 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976. doi:10.1137/0205048.

12 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

13 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. *Math. Program.*, 171(1-2):433–461, 2018. doi:10.1007/s10107-017-1199-3.

14 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

15 A. Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, January 1987. doi:10.1007/BF02579200.

16 András Frank. Packing paths, circuits, and cuts – a survey. In Bernhard Korte, Lásló Lovász, Hans Jürgen Prömel, and Alexander Schrijver, editors, *Paths, Flows, and VLSI-Layout*, pages 47–100. Springer-Verlag, Berlin, 1990.

17 Tobias Friedrich, Davis Issac, Nikhil Kumar, Nadym Mallek, and Ziena Zeif. Approximate max-flow min-multicut theorem for graphs of bounded treewidth. *arXiv*, abs/2211.06267, 2022. arXiv:2211.06267.

18 Tobias Friedrich, Davis Issac, Nikhil Kumar, Nadym Mallek, and Ziena Zeif. A primal-dual algorithm for multicommodity flows and multicuts in treewidth-2 graphs. In Amit Chakrabarti and Chaitanya Swamy, editors, *Proceedings 25th International Conference on Approximation Algorithms for Combinatorial Optimization Problems and 26th International Conference on Randomization and Computation APPROX/RANDOM 2022*, volume 245 of *LIPIcs*, pages 55:1–55:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.55.

19 Robert Ganian and Sebastian Ordyniak. The power of cut-based parameters for computing edge-disjoint paths. *Algorithmica*, 83(2):726–752, 2021. doi:10.1007/s00453-020-00772-w.

20 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. On structural parameterizations of the edge disjoint paths problem. *Algorithmica*, 83(6):1605–1637, 2021. doi:10.1007/s00453-020-00795-3.

21 Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, F. Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computing and System Sciences*, 67(3):473–496, 2003. doi:10.1016/S0022-0000(03)00066-7.

22 George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Trans. Algorithms*, 4(1):13:1–13:17, 2008. doi:10.1145/1328911.1328924.

23 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

24 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.

25 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin, Heidelberg, 2000.

26 Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is np-complete for series-parallel graphs. *Discret. Appl. Math.*, 115(1-3):177–186, 2001. doi:10.1016/S0166-218X(01)00223-2.

27 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

28 Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

29 Detlef Seese. Tree-partite graphs and the complexity of algorithms. In Lothar Budach, editor, *5th International Conference on Fundamentals of Computation Theory, FCT 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1985. doi:10.1007/BFb0028825.

30 F. Bruce Shepherd and Adrian R. Vetta. The inapproximability of maximum single-sink unsplittable, priority and confluent flow problems. *Theory of Computation*, 13(1):1–25, 2017. doi:10.4086/toc.2017.v013a020.

31 Jens Vygen. Disjoint paths. Technical Report 94816, Research Institute for Discrete Mathematics, University of Bonn, 1998.

32 I-Lin Wang. Multicommodity network flows: A survey, part I: Applications and formulations. *International Journal of Operations Research*, 15(4):145–153, 2018. URL: `http://www.orstw.org.tw/ijor/vol15no4/IJOR2018_vol15_no4_p145_p153.pdf`.

33 I-Lin Wang. Multicommodity network flows: A survey, part II: Solution methods. *International Journal of Operations Research*, 15(4):155–173, 2018. URL: `http://www.orstw.org.tw/ijor/vol15no4/IJOR2018_vol15_no4_p155_p173.pdf`.

34 Xiao Zhou, Syurei Tamura, and Takao Nishizeki. Finding edge-disjoint paths in partial k-trees. *Algorithmica*, 26(1):3–30, 2000. `doi:10.1007/s004539910002`.