# An Efficient Reduction of a Gammoid to a Partition Matroid

**Marilena Leichter**
Department of Mathematics, Technische Universität München, Germany

**Benjamin Moseley**
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

**Kirk Pruhs**
Computer Science Department, University of Pittsburgh, PA, USA

**Abstract**

Our main contribution is a polynomial-time algorithm to reduce a $k$-colorable gammoid to a $(2k-2)$-colorable partition matroid. It is known that there are gammoids that can not be reduced to any $(2k-3)$-colorable partition matroid, so this result is tight. We then discuss how such a reduction can be used to obtain polynomial-time algorithms with better approximation ratios for various natural problems related to coloring and list coloring the intersection of matroids.

**2012 ACM Subject Classification** Theory of computation → Network optimization

**Keywords and phrases** Matroid, Gammoid, Reduction, Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2021.62

**Funding** *Marilena Leichter*: Supported in part by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF) and by the Deutsche Forschungsgemeinschaft (DFG), GRK 2201.
*Benjamin Moseley*: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.
*Kirk Pruhs*: Supported in part by NSF grants CCF-1535755, CCF-1907673, CCF-2036077 and an IBM Faculty Award.

## 1 Introduction

Our main contribution is a polynomial-time algorithm to reduce a $k$-colorable gammoid to a $(2k-2)$-colorable partition matroid. Before elaborating on the statement of this result, we first give the necessary definitions, and the most relevant prior work. After stating the result we then explain some of the algorithmic ramifications.

### 1.1 Definitions

A set system is a pair $M = (S, \mathcal{I})$ where $S$ is a universe of $n$ elements and $\mathcal{I} \subseteq 2^S$ is a collection of subsets of $S$. Sets in $\mathcal{I}$ are called **independent** and the rank $r$ is the maximum cardinality of a set in $\mathcal{I}$. A partition $C_1, C_2, \ldots, C_k$ of $S$ into independent sets is a $k$-coloring of $M$. The **coloring number** of $M$ is the smallest $k$ such that a $k$-coloring exists.

If each element $s \in S$ has an associated list $A_s$ of allowable colors, then a list coloring is a coloring $C_1, C_2, \ldots, C_j$ such that if an $s \in S$ is in $C_i$ then $i \in A_s$. The **list coloring number** of $M$ is the smallest $k$ that guarantees that if for $s \in S$ it is the case that if $|A_s| \geq k$ then a list coloring exists.

If $R \subseteq S$ then the restriction of $M$ to $R$, denoted by $M \mid R$, is a set system where the universe is $S \cap R$ and where a set $I \subseteq S$ is independent if and only if $I \subseteq R$ and $I \in \mathcal{I}$.


A hereditary set system is a set system where if $A \subseteq B \subseteq S$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$. A matroid is an hereditary set system with the additional properties that $\emptyset \in \mathcal{I}$ and if $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|A| < |B|$ then there exists an $s \in B \setminus A$ such that $A \cup \{s\} \in \mathcal{I}$. The intersection of matroids $(S, \mathcal{I}_1), \ldots, (S, \mathcal{I}_\ell)$ on common universe is a hereditary set system with universe $S$ where a set $I \subseteq S$ is independent if and only if for all $i \in [1, \ell]$ it is the case that $I \in \mathcal{I}_i$.

A **gammoid** is a matroid that has a graphical representation $(D = (V, E), S, Z)$, where $D = (V, E)$ is a directed graph, $S \subseteq V$ is a collection of source vertices and $Z \subseteq V$ is a collection of sink vertices. In the gammoid that is represented by $D$ a set $I \subseteq S$ is in $\mathcal{I}$ *if and only if* there exists $|I|$ vertex-disjoint paths from the vertices in $I$ to some subcollection of vertices in $Z$. A **partition matroid** is a type of matroid that can be represented by a partition $\mathcal{X}$ of $S$. In the partition matroid that is represented by the partition $\mathcal{X}$ a set $Y \subseteq S$ is in $\mathcal{I}$ *if and only if* $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$. [1]

A matroid $N$ is a reduction (also called weak map) of a matroid $M$, with the same universe, if and only if every independent set in $N$ is also an independent set in $M$. If $N$ is a partition matroid, then we say that there exists a **partition reduction** from $M$ to $N$. [5] defined the following decomposability concept, which generalizes partition reduction.

▶ **Definition 1.** *A matroid $M = (S, \mathcal{I})$ is $(b, c)$-**decomposable** if $S$ can be partitioned into sets $X_1, X_2, \ldots, X_\ell$ such that:*

- *For all $i \in [\ell]$, it is the case that $|X_i| \leq c \cdot k$, where $k$ is the coloring number of $M$.*
- *For a set $Y = \{v_1, \ldots, v_\ell\}$, consisting of one representative element $v_i$ from each $X_i$, the matroid $M \mid Y$ is $b$ colorable.*

If $b = 1$ then $X_1, X_2, \ldots, X_\ell$ represents a partition matroid. Thus $(1, c)$-decomposability means there exists a partition reduction where the coloring number increases by at most a factor of $c$.

## 1.2 Prior Work

There are two prior, independent, papers in the literature that are directly relevant to our results. [3] showed that any gammoid $M$ admits a $(1, (2 - \frac{2}{k}))$-decomposition. This proof is constructive, and can be converted into an algorithm. The resulting algorithm is essentially a local search algorithm that selects a neighboring solution in the dual matroid in such a way that an auxiliary potential function always decreases. But there seems to be little hope of getting a better than exponential bound on the time, at least using techniques from [3] as the potential can be exponentially large. Further [3] shows that no better bound is achievable.

[5] gave a polynomial-time algorithm to construct a $(18, 1)$-decomposition of a gammoid. The reduction was shown by leveraging prior work on unsplittable flows [6]. Both paper [3, 5] also observed that partition reductions are relatively easily obtainable for other common types of combinatorial matroids. In particular transversal matroids are $(1, 1)$-decomposable [3, 5], graphic matroids are $(1, 2)$-decomposable [3, 5] and paving matroids are $(1, \lceil \frac{r}{r-1} \rceil)$-decomposable if they are of rank $r$ [3].

The main algorithmic result from [5] is:

▶ **Theorem 2** ([5]). *Consider matroids $M_1, M_2, \ldots, M_\ell$ defined over a common universe, where matroid $M_i$ has coloring number $k_i$. There is a polynomial-time algorithm that, given a $(b_i, c_i)$-decomposition of each matroid $M_i$, computes a coloring of the intersection of $M_1, M_2, \ldots, M_\ell$ using at most $\left(\prod_{i \in [k]} b_i\right) \cdot \left(\sum_{i \in [k]} c_i\right) k^*$ colors, where $k^* = \max_{i \in [l]} k_i$.*

[1] Technically one can generalize this to let there be a separate upper bound for each $X_i$ on the number of elements $Y$ can obtain from $X_i$, but in this paper we only consider partition matroids where the bound is one.

Combining Theorem 2 with the decompositions in [3, 5] one obtains $O(1)$-approximation algorithms for problems that can be expressed as coloring problems on the intersection of $O(1)$ common combinatorial matroids. Several natural examples of such problems are given in [5]. [1] showed that for two matroids $M_1$ and $M_2$ over a common universe with coloring numbers $k_1$ and $k_2$, the coloring number $k$ of $M_1 \cap M_2$ is at most $2\max(k_1, k_2)$. The proof in [1] uses topological arguments that do not directly give an algorithm for finding the coloring. The list coloring number of a single matroid equals its coloring number [9]. For the intersection of two matroids [3] observed that a list coloring could be efficiently computed by partition reducing each of the matroids. A consequence of the results in [3] is a constructive proof that the list coloring number of $M_1 \cap M_2$ is at most $2\max(k_1, k_2)$ if $M_1$ and $M_2$ are each one of the standard combinatorial matroids. Further a consequence of the results in [5] is an efficient algorithm to compute such a list coloring if $M_1$ and $M_2$ are each one of the standard combinatorial matroids besides a gammoid.

For hereditary set systems the coloring number is equal to the set cover number. Set cover has been studied extensively in the field of approximation algorithms. The greedy algorithm has an approximation ratio of $H_n \approx \ln n$ and this is essentially optimal assuming $P \neq NP$ [10, 11].

## 1.3 Our Main Result and Its Algorithmic Applications

We are now ready to state our main result:

▶ **Theorem 3.** *A partition reduction from a $k$-colorable gammoid to a $(2k-2)$-colorable partition matroid can be computed in polynomial time given a directed graph $D$ that represents $M$ as input.*

Recall that [3] showed that the $(2k-2)$ bound is tight.

Combining our main result, Theorem 3, with Theorem 2 from [5] we obtain significantly better approximation guarantees for matroid intersection coloring problems in which one of the matroids is a gammoid. One example is given by the following problem. Initially assume that the input consists of a directed graph $D$ with a designated file server location (a sink) and a collection of clients requesting files from the server at various locations in the networks (the sources). The goal is to as quickly as possible get every client the file that they want from the server, where in each time step one can service any collection of clients for which there exist disjoint paths to the server. This is a matroid coloring problem that can be solved exactly in polynomial-time [4]. Now assume that additionally the input identifies the company to which each client is employed by, and for each company there is a Service Level Agreement (SLA) that upper bounds on how many clients from that company can be serviced in one time unit. Now, the problem becomes a matroid intersection coloring problem, where the intersecting matroids are a gammoid and a partition matroid. Using the $(18, 1)$-decomposition of a gammoid and Theorem 2 from [5] one obtains a polynomial-time 36-approximation algorithm. However, combining the $(2k-2)$-partition reduction of a gammoid from Theorem 3 with Theorem 2 from [5] we now obtain a polynomial-time 3-approximation algorithm.[2]

Another algorithmic consequence is an efficient algorithm to list coloring the intersection $M_1 \cap M_2$ of a $k_1$-colorable matroid $M_1$ and a $k_2$-colorable matroid $M_2$ if the list of allowable colors for each element has cardinality at least $2\max(k_1, k_2)$, and each of the matroids is

[2] This is because Theorem 2 is a $(1, 2)$-decomposition of a gammoid and a partition matroid is in itself a $(1, 1)$-decomposition, so Theorem [5] states the approximation is 3.

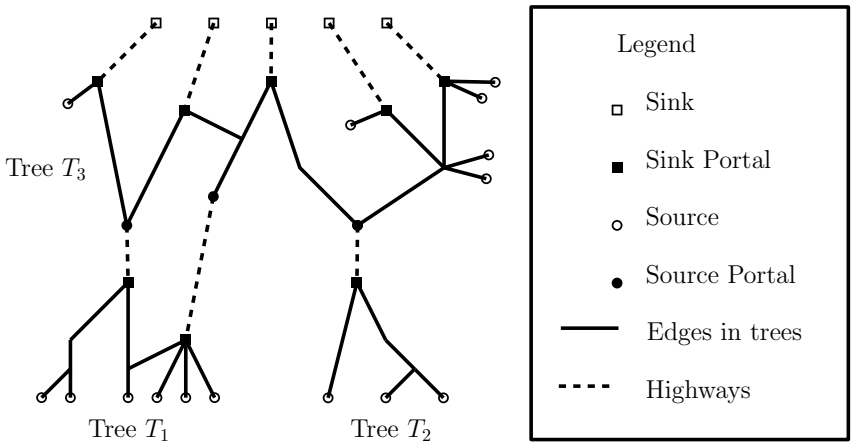



**Figure 1** Example of trees created. Here $k = 3$. Source portals are matched to sink portals along a path not in the trees. All sink portals will have $k$ units of flow entering them and source portals have $k$ units leaving.

either a graphic matroid, paving matroid, transversal matroid, or gammoid. Casting this into the context our running file server example implies that additionally each client has a list of allowable times when the file transfer may be scheduled. Our partition reduction of a gammoid then yields an efficient algorithm to find a feasible schedule as long as the cardinality of allowable times for each client is at least $2\max(k_1, k_2)$, where $k_1$ is time required if the network had infinite capacity (so only the SLA constraints come into play), and $k_2$ is the time required if the SLA allowed infinitely many file transfers (so only the network capacity constraints come into play).

Set cover is a canonical algorithmic problem. So there is considerable interest in discovering examples of natural special types of set cover instances where $o(\log n)$ approximation is possible. For example, several geometrically based types of instances are known, for example covering points in the plane using a discs [7], where a polynomial time approximation scheme is known. Our results provide another example of such a natural special case, namely when the sets come from the intersection of a small number of standard combinatorial matroids.

Theorem 3 and its proof reveal structural properties of gammoids that would seem likely to be of use to address future research on gammoids.

## 1.4 Overview of Techniques

Given a graphic representation of a gammoid, an optimal coloring can be computed in polynomial-time [4]. By superimposing the source-sink paths for the various color classes one can obtain a flow $f$ from the sources to the sinks that moves at most $k$ units of flow over any vertex. Using standard cycle-canceling techniques [2] one can then convert $f$ to what we call an acyclic flow. A flow $f$ is acyclic if for every undirected cycle $C$ in $D$ at least one edge in $C$ either has flow $k$ or has no flow. Thus by deleting edges that support no flow in $f$, as they are unnecessary, we are left with a forest $\mathcal{T}$ of edges that have flow in the range $[1, k-1]$ and a collection of disjoint paths, which we call highways, that have flow $k$. See Figure 1.

Now each part $X$ in the computed partition $\mathcal{X}$ will be entirely contained in one tree $T \in \mathcal{T}$, and the parts $X$ in a tree $T \in \mathcal{T}$ are computed independently of other trees in $\mathcal{T}$. There can be four types of vertices in $T$: (1) sources $s$ that have outflow 1, (2) source portals

$\tilde{s}$, which are vertices that have a highway directed into them, and which have outflow $k$ in $T$, (3) sink portals $\tilde{z}$, which are vertices that have a highway directed out of them, and which have inflow $k$ in $T$, and (4) normal vertices. Again see Figure 1.

We give a recursive partitioning algorithm for forming the parts $X$ in a tree $T \in \mathcal{T}$. On each recursive step our algorithm first identifies a single part $X$ of at most $2k-2$ sources and an associated sink portal that are in some sense near each other on the edge of $T$. The algorithm then removes these sources and sink portal from $T$, and reconnects disconnected sources back into appropriate places in $T$. The algorithm then recurses on this new tree $T$.

Most of the proof that our partitioning algorithm produces a $(1, 2-\frac{2}{k})$-decomposition focuses on routing individual trees in $\mathcal{T}$. So let $Y$ be a collection of sources such that for all $X \in \mathcal{X}$ $|Y \cap X| \leq 1$.

The first key part is proving that as the partitioning algorithm recurses on a tree $T$, it is always possible to route both the flow coming into $T$, and the flowing emanating within $T$, out of $T$, without routing more than $k$ units of flow through any vertex in $T$. Note that as the algorithm recurses the tree $T$ loses a sink portal (which reduces the capacity of the flow that can leave $T$ by $k$) and loses up to $2k-2$ sources (which means there is less flow emanating in $T$ that has to be routed out).

The second key part is to prove that there is a vertex-disjoint routing from the source portals in $T$ and the sources in $T \cap Y$ to the sink portals in $T$. To accomplish we trace our partitioning algorithm's recursion backwards. So in each step a new collection $X$ of sources and a sink portal is added back into $T$. We then prove by induction that no matter how the previously considered sources in $Y$ were routed, there is always a feasible way to route the chosen source in $Y \cap X$ to a sink portal in $T$. Then we finish by observing that unioning the routings constructed within the trees with the highways gives a feasible routing for $Y$.

## 2 Preliminaries

This section introduces notation and other necessary definitions. Let $D = (V, E)$ be a directed graph that represents a gammoid. Let $S \subseteq V$ be a set of sources, and $Z \subseteq V$ be the collection of sinks. We may assume without loss of generality that:

- Each vertex $v \in V$ has either out-degree 1 or in-degree 1.
- Each source $s \in S$ has in-degree 0 and out-degree 1.
- Each sink $z \in Z$ has in-degree 1 and out-degree 0 and $|Z| = r$.
- If $uv$ is an edge in $E$ then $vu$ is not an edge in $E$.

We assume without loss of generality that all color classes have full rank, that is $|S| = rk$. This can be assumed by adding dummy sources to $S$.

▶ **Definition 4.** *A **feasible flow** in a digraph $D$ from a collection $S' \subset S$ is a collection of paths $\{p^s \mid s \in S'\}$ such that (1) $p^s$ is a simple path from $s$ to some sink, and (2) no vertex or edge in $D$ has more than $k$ such paths passing through it.*

*A **feasible routing** in a digraph $D$ from a collection $S' \subset S$ is a collection of paths $\{p^s \mid s \in S'\}$ such that (1) $p^s$ is a simple path from $s$ to some sink, and (2) no vertex or edge in $D$ has more than one such path passing through it.*

## 3 The Partition Reduction Algorithm

This section gives the Partition Reduction Algorithm. First, we define a corresponding flow graph. Using a Cycle-Canceling Algorithm, we decompose the flow graph into a collection of trees. Then we algorithmically create the partitions from the local structure in these trees. The analysis of the algorithm is deferred to the next section.

## 3.1 Defining the Flow Graph

Given the digraph $D$ we can compute a minimum $k$ such that $M$ is $k$-colorable in polynomial time using a polynomial-time algorithm for matroid intersection [8]. Further we can compute the collection of resulting color classes $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. So $\mathcal{C}$ is a partition of the sources $S$, and for each $C_i \in \mathcal{C}$ there exist $r$ vertex-disjoint paths $p_i^1, \ldots, p_i^r$ in the digraph $D$ from the $r$ sources $C_i$ to $Z$. We create an $f$ where the flow $f(u,v)$ on each edge $(u,v)$ is initialized to the number of paths $p_i^j$ that traverse $(u,v)$, that is

$$f(u,v) = \sum_{i=1}^{k} \sum_{j=1}^{r} \mathbb{1}_{(u,v)\in p_i^j}$$

A flow $f$ is acyclic if for every undirected cycle $C$ in $D$ at least one edge in $C$ either has flow $k$ or has no flow in $f$. An arbitrary flow can be converted acyclic by finding cycles in a residual network $D^r$. This is standard [2], but for completeness we define it here.

For every directed edge $(u,v)$ with $f(u,v) < k$ there exists a forward directed edge $(u,v)$ in $D^r$ with capacity $c_r(u,v) := k - f(u,v)$. For every directed edge $(u,v)$ with $f(u,v) > 0$ there exists a backward directed edge $(v,u)$ in $D^r$ with capacity $c_r(v,u) := f(u,v)$. An augmenting cycle in $D^r$ is a simple directed cycle with strictly more than two edges.

**Cycle-Canceling Algorithm.** While there exists an augmenting cycle $C$ do the following:

- Let $c := \min_{(u,v)\in C} c_r(u,v)$ be the minimum capacity of an edge in $C$.
- For each forward edge $(u,v) \in C$, increase $f(u,v)$ by $c$.
- For each backward edge $(u,v) \in C$, decrease $f(u,v)$ by $c$.

As every iteration increases the number of edges that have flow $k$ in $f$ or that have no flow in $f$ by 1, the Cycle-Canceling Algorithm terminates after at most $|E|$ iterations. The following observations are straight-forward.

▶ **Observation 5.** *The following properties hold when the Cycle-Canceling Algorithm terminates:*

- *$f$ is a feasible flow of $kr$ units of flow from all the sources.*
- *Every undirected cycle $C$ in $D$ contains at least one edge with flow $k$ in $f$ or one edge with no flow in $f$.*
- *The collection of edges in $D$ that has flow strictly between 0 and $k$ in $f$ forms a forest.*
- *The collection of edges in $D$ with flow $k$ in $f$ are a disjoint union of directed paths, which we will call **highways**.*

## 3.2 Properties of the Acyclic Flows

We now give several definitions and straightforward observations about our acyclic flow $f$ that will be useful in our algorithm design and analysis.

▶ **Definition 6.**

- *A vertex $v$ is a source portal if its in-degree in $D$ is 1, and it has $k$ units of flow passing through it in $f$.*
- *A vertex $v$ is a sink portal if its out-degree in $D$ is 1, and it has $k$ units of flow passing through it in $f$.*
- *Let $\mathcal{T}$ be the forest consisting of edges in $D$ that have flow in $f$ strictly between 0 and $k$.*
- *For a tree $T \in \mathcal{T}$ and a vertex $v \in T$ define $T_v$ to be the forest that results from deleting the vertex $v$ from $T$.*

▶ **Observation 7.** *Each sink $z \in Z$ is in a tree $T \in \mathcal{T}$ that consists solely of $z$.*

**Proof.** By assumption, the sink $z$ has in-degree 1 in $D$ and all color classes $\mathcal{C}$ have full rank. Hence, $k$ units of flow are entering $z$ through a unique edge. ◀

As our partition reduction algorithm partitions each tree $T \in \mathcal{T}$ independently, it will be notationally more convenient to fix an arbitrary tree $T \in \mathcal{T}$, and make some definitions relative to this fixed $T$, and make some observations that must hold for any such $T$. To a large extent these observations are intended to show that the Figure 2 is accurate.

▶ **Definition 8.**
- *Let $\tilde{S}$ be the collection of source portals in tree $T$.*
- *Let $\tilde{Z}$ be the collection of sink portals in tree $T$.*
- *A normal vertex is a vertex that is none of a source, a sink, a source portal, nor a sink portal.*

▶ **Definition 9.**
- *A **feasible flow** in $T$ from a collection $S' \subset S$ is a collection of paths, one path $p^s$ for each $s \in S'$ and $k$ paths $p_1^{\tilde{s}}, \ldots, p_k^{\tilde{s}}$ for each source portal $\tilde{s} \in \tilde{S}$ such that (1) $p^s$ is a simple path from $s$ to some sink portal, (2) each $p_i^{\tilde{s}}$ is a simple path from $\tilde{s}$ to a sink portal, and (3) no vertex or edge in $T$ has more than $k$ such paths passing through it.*
- *A **feasible routing** in $T$ from a collection $S' \subset S$ is a collection of paths, one path $p^s$ for each $s \in S'$ and one path $p^{\tilde{s}}$ for each source portal $\tilde{s} \in \tilde{S}$ such that (1) $p^s$ is a simple path from $s$ to some sink portal, (2) $p^{\tilde{s}}$ is a simple path from $\tilde{s}$ to a sink portal, and (3) no vertex or edge in $T$ has more than one such paths passing through it.*

This following observation holds for trees in $\mathcal{T}$ initially and gives intuition for the structure of $\mathcal{T}$. We remark that this observation may not hold throughout the execution of our algorithm for all trees.

▶ **Observation 10.** *The number of sources in $T$ is an integer multiple of $k$.*

**Proof.** This follows from the fact that each source portal $\tilde{s} \in T$ has exactly $k$ units of flow coming into $T$ via $\tilde{s}$ in the flow $f$ and each sink portal $\tilde{z} \in T$ has exactly $k$ units of flow leaving $T$ via $\tilde{z}$ in $f$. ◀

▶ **Definition 11.**
- *For two vertices $u, v \in T$, let $P(u, v)$ be the unique undirected path from $u$ to $v$ in $T$.*
- *The **backbone** $B$ of $T$ is the subgraph of $T$ consisting of the union of all paths in between pairs of sink portals in $T$, that is $B = \bigcup_{\tilde{y} \in \tilde{Z}} \bigcup_{\tilde{z} \in \tilde{Z}} P(\tilde{y}, \tilde{z})$.*
- *For the backbone $B$, let $B_v$ be the induced forest that results from deleting $v$ from $B$.*
- *A vertex $v$ in a backbone $B$ is a **branching vertex** if either:*
  - *$v$ is not a sink portal and the forest $B_v$ contains at least two trees that each contain exactly one sink portal, or*
  - *$v$ is a sink portal and the forest $B_v$ contains at least one tree that contains exactly one sink portal.*
- *Let $\mathcal{H}$ be the forest that results from deleting the edges in $B$ from the tree $T$.*
- *For two vertices $u, v \in B$, let $S(P(u, v))$ be the sources $s \in S$ such that there exists a tree $H \in \mathcal{H}$ such that $s \in H$ and such that $H$ contains a vertex $w \in P(u, v)$. Intuitively these are the sources in trees in $\mathcal{H}$ hanging off vertices of the path $P(u, v)$.*
- *Let $S(v)$ denote $S(P(v, v))$.*

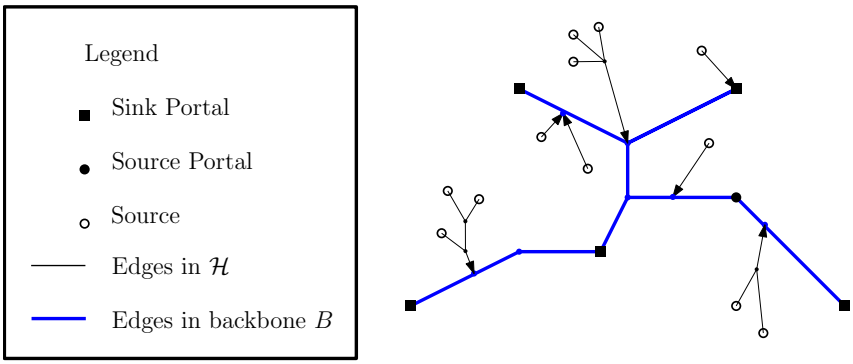



**Figure 2** Backbone of a tree.

▶ **Observation 12.** *If $\tilde{s} \in \tilde{S}$ is a source portal in $T$ then $\tilde{s}$ is in the backbone $B$ and $\deg^+_B(\tilde{s}) \geq 2$, that is $\tilde{s}$ has out-degree at least 2 in $B$.*

**Proof.** By definition $\tilde{s}$ has a unique incoming edge, which is saturated in $f$, at least one outgoing edge in $T$ that is not saturated in $f$. Hence, $\deg^+_B(\tilde{s}) \geq 2$. By flow conservation, there has to be at least two directed paths from $\tilde{s}$ to two different sink portals in $T$. This implies that $\tilde{s}$ is in the backbone $B$. ◀

▶ **Observation 13.** *If $B$ contains at least two sink portals, then $B$ contains a branching vertex $v$.*

**Proof.** Consider an arbitrary vertex $v \in B$. If $v$ is not a branching vertex, then there must be a subtree $T' \in B_v$ that contains two sink portals. One can then recurse on $T'$ to find a branching vertex. ◀

▶ **Observation 14.** *If $s \in S$ is a source in $T$ then $s$ is not in the backbone $B$.*

**Proof.** As $s$ has out-degree 1 in $D$, it can not be on any path between sink portals in $T$. ◀

▶ **Observation 15.** *For each tree $H \in \mathcal{H}$ it must be the case that all edges in $H$ are directed towards the unique vertex $w$ in $H$ that is also in $B$.*

**Proof.** This follows from the fact that $H \setminus \{w\}$ can not contain a sink portal. ◀

▶ **Observation 16.** *Assume that $T$ has at least two sink portals. Let $v$ be a branching vertex. Let $T'$ be a tree in the forest $B_v$ that contains exactly one sink portal $\tilde{z}$. Then the following must hold:*

- $T' = P(v, \tilde{z}) \setminus \{v\}$.
- *If $T'$ contains a source portal $\tilde{s}$, then* $\deg^+_B(\tilde{s}) = 2$.
- *The path $T'$ contains at most one vertex $y$ such that* $\deg^+_B(y) = 2$.

**Proof.** The first statement follows from the definition of $B$ and the fact that $T'$ only contains one sink portal. The second statement follows since every vertex on a path other than its endpoints has degree two. For the last statement assume to reach a contradiction that there were two such $y$'s, $y_1$ and $y_2$ with $y_1$ being closer to $v$ in $B$. Then the flow leaving $y_1$ toward $\tilde{z}$ could not be feasibly routed through $y_2$. ◀

## 3.3 Description of the Partition Reduction Algorithm

Given the collection of trees $\mathcal{T}$, our Partition Reduction Algorithm returns a partition $\mathcal{X}$ of the sources in $S$. The algorithm iterates through the trees $T$ in $\mathcal{T}$ and partitions the sources in $T$ based on their locality in $T$. So let us consider a particular tree $T \in \mathcal{T}$.

The algorithm performs the first listed case below that applies, with the base cases being checked before the other cases. In the non-base cases the tree $T$ will be modified, and the algorithm called tail-recursively on the modified tree. We will show after the algorithm description that the algorithm maintains the invariant that there is a feasible flow on the tree $T$ throughout the recursion.

**Base Case A.** If $T$ contains no sources then the recursion terminates, and the algorithm moves to the next tree in $\mathcal{T}$.

**Base Case B.** Otherwise if $T$ contains at most $2k-2$ sources and no source portal then these sources are added as a part $X$ in $\mathcal{X}$. The recursion terminates, and the algorithm then moves to the next tree in $\mathcal{T}$.

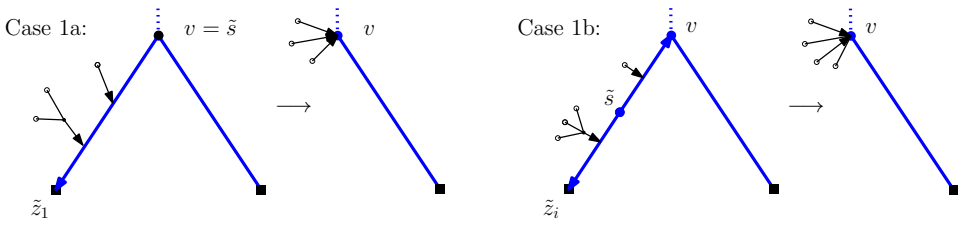
We perform the following recursively on $T$ if neither base case holds. Let $v$ be an arbitrary branching vertex in $B$. We will show this must exist in Observation 17.

Let $\tilde{z}_1$ be a sink portal in some tree $T_1$ in $T_v$ that only contains one sink portal. If $v$ is not a sink portal, let $\tilde{z}_2$ be a sink portal in some tree $T_2$, where $T_1 \neq T_2$, in $T_v$ that only contains one sink portal. If $v$ is a sink portal let $\tilde{z}_2 = v$.

The algorithm's cases are broken up as follows. Case 1 is executed when there is a source portal at $v$ or in $T_1$ or in $T_2$. Case 2 is executed when there is a vertex of out-degree 2 in $T_1$ or $T_2$ and there is no source portal. Case 3 is everything else.

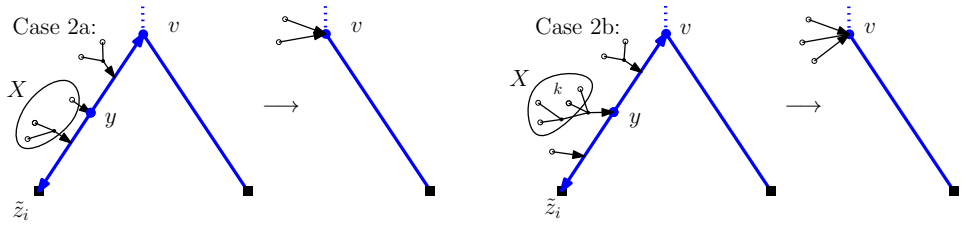
**Recursive Case 1a.** The vertex $v$ is a source portal. In this case $T$ is modified as follows: (1) for each source $s \in T_1$ a directed edge $(s, v)$ is added to $T$, (2) $v$ is converted into a normal vertex, and (3) all the nonsources in $T_1$ are deleted from $T$. The algorithm then recurses on this new $T$.

**Recursive Case 1b.** In this case for some $i \in \{1, 2\}$ the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a source portal. In this case $T$ is modified as follows: (1) for each source $s \in T_i$ a directed edge $(s, v)$ is added to $T$, and (2) all the nonsources in $T_i$ are deleted from $T$. The algorithm then recurses on this new $T$.

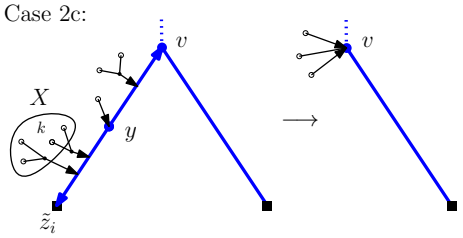



**Recursive Case 2a.** In this case for some $i \in \{1, 2\}$, the path $P(v, \tilde{z}_i) \setminus \{v\}$ contains a vertex $y$ with $\deg_B^+(y) = 2$ and $|S(P(y, \tilde{z}_i))| \leq 2k-2$. Add the sources in $S(P(y, \tilde{z}_i))$ as a part $X$ to $\mathcal{X}$. The tree $T$ is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge $(s, v)$ is added to $T$, and (2) the sources in $X$ and all the nonsources in $T_i$ are deleted from $T$. The algorithm then recurses on this new $T$.

**Recursive Case 2b.** In this case for some $i \in \{1,2\}$, the path $P(v,\tilde{z}_i)\setminus\{v\}$ contains a vertex $y$ with $\deg^+_B(y)=2$ and $|S(y)|=k$. In this case the algorithm adds the $k$ sources in $S(y)$ as a part $X$ to $\mathcal{X}$. The tree $T$ is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge $(s,v)$ is added to $T$, and (2) the sources in $X$ and all the nonsources in $T_i$ are deleted from $T$. The algorithm then recurses on this new $T$.

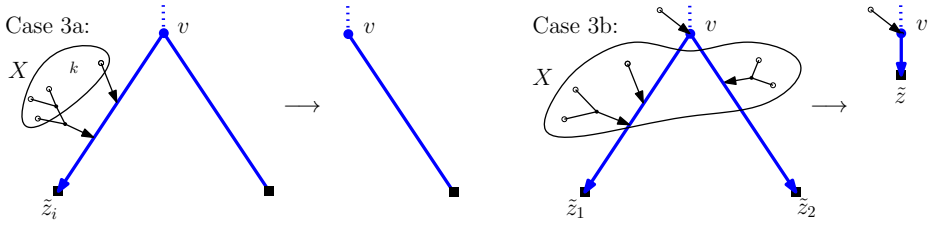



**Recursive Case 2c.** In this case for some $i \in \{1,2\}$, the path $P(v,\tilde{z}_i)\setminus\{v\}$ contains a vertex $y$ with $\deg^+_B(y)=2$ and $|S(P(y,\tilde{z}_i)\setminus\{y\})|=k$. In this case the algorithm adds the $k$ sources in $S(P(y,\tilde{z}_i)\setminus\{y\})$ as a part $X$ to $\mathcal{X}$. The tree $T$ is then modified as follows: (1) for each source $s \in T_i - X$ a directed edge $(s,v)$ is added to $T$, and (2) the sources in $X$ and all the nonsources in $T_i$ are deleted from $T$. The algorithm then recurses on this new $T$.




**Recursive Case 3a.** In this case for some $i \in \{1,2\}$ $T_i$ contains exactly $k$ sources. Add the sources in $T_i$ as a part $X$ to $\mathcal{X}$. The tree $T$ is modified by deleting $T_i$. The algorithm then recurses on this new $T$.

**Recursive Case 3b.** The set of sources in $T_1 \cup T_2$ are added as a part $X$ in $\mathcal{X}$. The tree $T$ is modified by deleting the vertices in $T_1$ and $T_2$. Add a new sink portal $\tilde{z}$ together with a directed edge $(v,\tilde{z})$ to $T$. The algorithm then recurses on this new $T$.




## 4 Analysis of the Partition Reduction Algorithm

Our goal is to show that the partition matroid, represented by the partition constructed from the trees, indeed corresponds to a feasible partition reduction from the gammoid. The analysis has the following key components.

- Every tree $T$ has a corresponding feasible flow throughout the algorithm.
- Every part $X$ of the partition has size at most $2k-2$ and all sources are in some part.
- Any collection of sources $Y$ such that $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$ is in $\mathcal{I}$ and, therefore, can each route a unit of flow to the sink in $D$.

## 4.1 The Trees Always Have a Feasible Flow

This section's goal is to show that each tree has a feasible flow as defined in Definition 8 throughout the execution of the algorithm. We will later use this to prove that our partition indeed represents a partition matroid that corresponds to a feasible partition reduction in the following section.

We begin by showing various invariants hold for each tree when a feasible flow exists. In particular, this will show that a branching vertex exists if any of the recursive cases are executed. Moreover, arriving at Cases (3a) and (3b) ensure the existence of $T_1$ and $T_2$. All together this with the fact that each tree has a feasible flow will establish that the algorithm always has a case to execute if $\mathcal{T}$ is non-empty.

This observation shows a branching vertex exists if neither base case holds.

▶ **Observation 17.** *Fix a tree $T \in \mathcal{T}$ during the execution of the algorithm and say $T$ supports a feasible flow as defined in Definition 8. If neither of the base cases apply then $T$ contains at least two sink portals. Moreover a branching vertex must exist in $T$ in this case.*

**Proof.** This observation holds because $T$ must contain either more than $2k-2$ sources or a source portal along with at least one source. In either case, we require two sink portals to support the strictly more than $k$ units of flow from these sources and source portal. A branching vertex must then exist by Observation 13. ◀

▶ **Observation 18.** *Fix a tree $T \in \mathcal{T}$ during execution of the algorithm and say $T$ supports a feasible flow as defined in Definition 8. Say that $T$ has a branching vertex $v$ with a tree $T_i$ containing exactly one sink $\tilde{z}_i$. Moreover say that there is no vertex with out-degree 2 in $P(v, \tilde{z}_i)$. It is the case that $P(v, \tilde{z}_i)$ is a directed path from $v$ to $\tilde{z}_i$.*

**Proof.** No vertex with out-degree 2 is in $P(v, \tilde{z}_i)$. Thus, $P(v, \tilde{z}_i)$ is either a path from $v$ to $\tilde{z}_i$ or from $\tilde{z}_i$ to $v$. Sink portals always have out-degree 0 in $T$, so the observation follows. We note that, sink portals have out-degree 0 in $T$ initially and are never given outgoing edges by the algorithm. ◀

The next observation shows that a branching vertex is not a sink portal when Cases (3a) or (3b) are executed.

▶ **Observation 19.** *Fix a tree $T \in \mathcal{T}$ during execution of the algorithm and say $T$ supports a feasible flow as defined in Definition 8. Say that $T$ has a branching vertex $v$ and a corresponding tree $T_i$ with exactly one sink $\tilde{z}_i$. If $P(v, \tilde{z}_i) \setminus \{v\}$ does not contain a vertex of out-degree 2 in $B$, then $v$ is not a sink portal.*

**Proof.** Observation 18 implies that $P(v, \tilde{z}_i)$ is a directed path form $v$ to $\tilde{z}_i$. Sink portals always have out-degree 0 in $T$, so the observation follows. We note that, sink portals have out-degree 0 in $T$ initially and are never given outgoing edges by the algorithm. ◀

The previous observations guarantee the algorithm always has a case to execute if a feasible flow exists in all trees. The next lemma guarantees the existence of a feasible flow.

▶ **Lemma 20.** *Fix any tree $T$ during the execution of the algorithm. There must be a feasible flow in $T$ as described in Definition 8.*

## 4.2 Bounding the Size of the Parts in the Partition

This section shows that every source is in some part $X$ in $\mathcal{X}$ and that every $X \in \mathcal{X}$ has size at most $2k-2$. Thus we have a valid partition with each part having the desired size.

▶ **Lemma 21.** *It is the case that $|X| \leq 2k-2$ for all $X \in \mathcal{X}$. Moreover, every source in $S$ is in some set in $\mathcal{X}$.*

**Proof.** It is easy to see that every source in $S$ is in some set in $\mathcal{X}$. This is because sources are always contained in some tree of $\mathcal{T}$ until they are added to a set placed in $\mathcal{X}$ and the algorithm stops once there is no tree in $\mathcal{T}$.

Now we show how to bound the size of sets in $\mathcal{X}$. Cases (1a) and (1b) do not add a set to $\mathcal{X}$. Cases (2b), (2c) and (3a) add a set to $\mathcal{X}$ of size $k$ by definition. Case (2a) adds a set of size $2k-2$.

Case (3b) is more interesting. Consider the execution of this case on a tree $T$ with branching vertex $v$. Let $\tilde{z}_1$ and $\tilde{z}_2$ be the corresponding sinks. We know $v$ is not a sink portal by Observation 19 and therefore $T_1$ and $T_2$ both exist. By Observation 18, the paths from $v$ to $\tilde{z}_1$ and $\tilde{z}_2$ are directed paths from $v$ to $\tilde{z}_1$ and from $v$ to $\tilde{z}_2$. Hence, neither $T_1$ nor $T_2$ contains more than $k$ sources. Since case (3a) does not hold, $T_1$ and $T_2$ has strictly less than $k$ sources. Thus, there are at most $2k-2$ sources in the set added to $\mathcal{X}$. ◀

## 4.3 Routing Sources within a Tree

In this section, we show that the algorithm is indeed a partition reduction from a $k$-colorable gammoid to a $(2k-2)$-colorable partition matroid. That is, we show that every set $Y$, that is independent in the partition matroid represented by $\mathcal{X}$ is also independent in the gammoid. More specifically, for any set $Y$ where $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$ it is the case that $Y \in \mathcal{I}$ or, equivalently, there is a feasible routing in the digraph $D$ from the sources in $Y$. The key to this will be Lemma 22 which essentially states that there exists a feasible routing in each tree $T \in \mathcal{T}$.

▶ **Lemma 22.** *Let $T$ be an arbitrary tree in $\mathcal{T}$. Let $\mathcal{X}_T$ be the parts in $\mathcal{X}$ that are also in $T$. For a set $Y$ with $|Y \cap X| \leq 1$ for all $X \in \mathcal{X}$, let $Y_T$ be the subset of sources in $Y$ that are also in $T$. Then there is a feasible routing $R$ in $T$ from $Y_T$.*

▶ **Lemma 23.** *Let $Y \subset S$ such that for all $X \in \mathcal{X}$ it is the case that $|X \cap Y| \leq 1$. Then there exists a feasible routing from $Y$ in the digraph $D$.*

**Proof.** This follows from Lemma 22 and the fact there is a unique highway into each source portal in each tree and a unique highway leaving each sink portal in every tree. ◀

## References


1 Ron Aharoni and Eli Berger. The intersection of a matroid and a simplicial complex. *Transactions of the American Math Society*, 2006.

2 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.

3 Kristóf Bérczi, Tamás Schwarcz, and Yutaro Yamaguchi. List colouring of two matroids through reduction to partition matroids. *arXiv preprint arXiv:1911.10485*, 2019.

4 Jack Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards*, 69B:67–72, 1965.

5 Sungjin Im, Benjamin Moseley, and Kirk Pruhs. The matroid intersection cover problem. *Operations Research Letters*, 49(1):17–22, 2020.

6 Jon M. Kleinberg. Single-source unsplittable flow. In *Symposium on Foundations of Computer Science*, pages 68–77, 1996.

7 Nabil Hassan Mustafa and Saurabh Ray. PTAS for geometric hitting set problems via local search. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 17–22, 2009.

8 James Oxley. *Matroid Theory*. Oxford graduate texts in mathematics. Oxford University Press, 2006.

9 Paul D Seymour. A note on list arboricity. *Journal of Combinatorial Theory, Series B*, 72(1):150–151, 1998.

10 Vijay Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

11 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.