Available online at www.sciencedirect.com
ScienceDirect
Discrete Optimization 00 (2012) 1–28
Discrete Optimization

Branch Decomposition Heuristics for Linear Matroids

Jing Ma[a,1], Susan Margulies[c,1], Illya V. Hicks[e,1], Edray Goins[g,1]

[a] *jm11.rice@gmail.com*
[b] *Department of Management Science and Engineering, Stanford University*
[c] *margulies@math.psu.edu*
[d] *Department of Mathematics, Pennsylvania State University*
[e] *ivhicks@rice.edu*
[f] *Department of Computational and Applied Math, Rice University*
[g] *egoins@math.purdue.edu*
[h] *Department of Mathematics, Purdue University*

Abstract

This paper presents three new heuristics which utilize classification, max-flow, and matroid intersection algorithms respectively to derive near-optimal branch decompositions for linear matroids. In the literature, there are already excellent heuristics for graphs, however, no practical branch decomposition methods for general linear matroids have been addressed yet. Introducing a "measure" which compares the "similarity" of elements of a linear matroid, this work reforms the linear matroid into a similarity graph. Then, the classification method, the max-flow method, and the mat-flow method, all based on the similarity graph, are utilized on the similarity graph to derive separations for a near-optimal branch decomposition. Computational results using the methods on linear matroid instances are shown respectively.

Keywords: linear matroid, branchwidth, branch decomposition, classification, max-flow, matroid intersection

1. Introduction

Branch-decomposition and its associated connectivity invariant branchwidth were introduced by Robertson and Seymour [1] as part of their graph minors project and played a fundamental role in their proof of Wagner's conjecture. It has been shown that branch decompositions open algorithmic possibilities for solving NP-hard problems for graphs. Theoretical work that used (tree) decompositions to solve some NP-complete problems can be found in Bern et al. [2], Courcelle [3], Arnborg et al. [4], and Borie et al. [5]. Most of the above theoretical work can be applied to branch decompositions. In particular, Courcelle [3] showed that several NP-complete problems can be solved in polynomial time using dynamic-programming techniques on input graphs with bounded branchwidth. The original result is about bounded treewidth, the invariant associated with tree decompositions of graphs–another byproduct of Robertson and Seymour's

proof of Wagner's conjecture. In contrast, the result is equivalent for bounded branchwidth, since the branchwidth and treewidth of a graph bound each other by a constant factor [1].

When using branch decompositions to solve graph problems, branch decompositions with small width (i.e., order of the branch decomposition) are always desirable. In contrast, finding optimal branch decompositions (branch decompositions with the smallest width possible) in general is NP-hard [6]. As a result, a branch of researchers has been focusing on developing practical heuristics that produce branch decompositions with small widths. In particular, there is the eigenvector heuristic based on spectral graph theory proposed by Cook and Seymour [7] and Hicks' diameter method along with a hybrid of the two [8].

Encouraged by the benefits of branch decompositions for solving NP-hard graph problems, researchers generalized the branch decomposition to any finite set where a symmetric submodular function is defined [9], intending to attack classes of NP-hard problems modeled on structures other than graphs. For example, Cunningham and Geelen [10] proposed a branch decomposition-based algorithm to solve integer programs with a non-negative constraint matrix. In contrast, unlike branch decompositions for graphs, there has been much less work to derive near-optimal branch decompositions for the general setting. Hence, the focus of this paper is to develop near-optimal branch decomposition heuristics for linear matroids.

Most NP-hard graph problems can be modeled on linear matroids, by displaying the problems in the form of matrices (e.g. node-edge incidence matrices). Moreover, the large class of integer programming problems can be easily modeled on linear matroids, simply by associating the constraint matrix of the integer program with a linear matroid. Thus, branch decomposition theories can be applied to these graph and integer program problems via linear matroid easily. Whenever in these occasions, the width of the branch decomposition is desired to be as small as possible.

Taking the Cunningham-Geelen algorithm for example, the algorithm has a complexity of $O((d+1)^{2k}mn + m^2n)$, where m and n are the number of rows and columns of the constraint matrix, k is the width of the input branch decomposition, and d is a numeric value of the matrix respectively. Thus, the solving time of the integer program using the Cunningham-Geelen algorithm is subject to change according to the width of the input branch decomposition exponentially. Generally, the smaller the width of the input branch decomposition, the faster the branch decomposition based algorithm performs. In contrast, a result of Seymour and Thomas [6] in conjunction with a result by Hicks and McMurray [11] (independently proven by Mazoit and Thomassé [12] as well) implies that finding an optimal branch decomposition of the linear matroid for a general matrix is NP-hard. Thus, to apply branch decomposition-based algorithms, finding near-optimal branch decompositions is of practical importance.

Another area of motivation for practical algorithms for the branchwidth of linear matroids is the relationship between the branchwidth of a binary matroid and the rankwidth of a fundamental graph of the matroid. Given a graph, rankwidth is the branchwidth of the cut-rank function related to the graph [13]. Rankwidth is related to cliquewidth with the fact that graphs have bounded rankwidth if and only if the graphs have bounded cliquewidth [13]. In addition, Oum [14] has shown that the branchwidth of a binary matroid is exactly one more than the rankwidth of its fundamental graph. Hence, work in this area offer practical algorithms for computing the rankwidth of bipartite graphs.

Though the broad applications of branch decompositions, there are not many practical works for finding the near-optimal branch decomposition for a general setting such as linear matroids, in spite of success such as the eigenvector method and the diameter method for branch decompositions of graphs. In a previous work by Oum and Seymour [9], the authors constructed an

algorithm to estimate the branchwidth of a symmetric submodular function within a factor of 3. For example, the algorithm decides either the branchwidth is at least 6 or finds a branch decomposition with width at most 15 for an instance with branchwidth being 5 [9]. Hence, the error factor of three of Oum and Seymour's method shows that it is not practical when applied to branch decomposition-based algorithms. New practical methods are needed to derive branch decompositions of linear matroids.

Inspired by the work of Belkin and Niyogi [15, 16] and its application to image processing by Szlam et al. [17], we reform the linear matroid into a similarity graph. Their work considered the data set as a finite weighted graph where the nodes of the graph represent elements in the data set and two nodes are connected if and only if their corresponding data points are "similar" enough under the similarity measure. In our work, basing on the similarity graph, three different methods are developed and implemented.

In the classification method, the Laplacian-Eigenmaps-based partially labeled classification theory by Belkin and Niyogi [15, 16] is introduced to derive branch decompositions. In their work, Belkin and Niyogi consider the problem of classification under the assumption that the data resides on a low-dimensional manifold within a high-dimensional representation space. Drawing on the correspondence between the graph Laplacian, the Laplace Beltrami operator on the manifold, and connections to the heat equation, they proposed an algorithmic framework to classify a partially labeled data set in a principled manner. The active role of graph Laplacian in heuristics or approximation algorithms for problems whose exact solution is NP-complete or coNP-complete has been strongly utilized in the literature. In the 1980s, Alon [18] showed the significance of the eigenvector corresponding to the second smallest eigenvalue of the graph Laplacian (see Chung [19] for definition of graph Laplacian), and supplied an efficient algorithm for approximating the expanding properties of a graph. In the work of Shi and Malik [20], they explored the eigenvector with the second smallest generalized eigenvalue of the graph Laplacian to partition graphs. In Cook and Seymour's eigenvector branch decomposition heuristic for graphs, they use the eigenvector with the second smallest eigenvalue of the weighted graph Laplacian before calling the max-flow algorithm to identify the source and sink nodes [7]. Classification, or partially labeled classification has previously been extensively applied in areas such as machine learning, data mining, pattern recognition, but not yet in branch decomposition techniques as an exclusive procedure without max-flow. Referring to Belkin and Niyogi's algorithmic framework, we construct a low-dimensional representation for the original high-dimensional elements of the linear matroid, where eigenvectors of the Laplacian of the similarity graph are the basis, and build classifiers for partitioning the element set of the linear matroid.

The max-flow method inherits the ingredients of the previous branch decomposition heuristics [7, 8] and tree decomposition heuristics [21]. Based on the similarity graph, our work contracts the similarity graph into its corresponding graph minors as the heuristic evolves, and picks the small neighborhoods of a diameter pair (nodes having their distance equal to the diameter of the minor) minimizing the number of paths between the pair. Each separation induced by the output cut in the max-flow algorithm becomes a separation in the output branch decomposition.

The mat-flow method is similar to the max-flow method in that it utilizes the small neighborhoods of the nodes of the similarity graph to define subsets of the ground set of the input matroid. In contrast, the mat-flow method uses the subsets derived from the similarity graph to solve a series of matroid intersection problems on the minors of the input matroid to produce separations for the branch decomposition. This technique is based upon the matroid version of Menger's theorem, proved by Tutte [22].

Throughout this paper, we assume that all graphs are simple and undirected unless otherwise

stated. One is referred to the books by Diestel [23] and Oxley [24] if unfamiliar with graph theory and matroid theory, respectively. In addition, we will use the term "node" to describe the nodes of a graph in general and the term "vertex" for branch decomposition trees. Fundamental definitions are given in Section 2. In Section 3, The Pushing Lemma and The Strong Closure Corollary are stated and proved. Then, a complete description of the branch decomposition heuristics: the classification method, the max-flow method, and the mat-flow method follows in Sections 4 to 7. Computational results using the methods on linear matroid instances are reported and compared in Section 9. Concluding remarks and directions for future work are given in Section 10.

2. Preliminaries

In this section, we discuss foundation definitions that will be utilized in the proposed algorithms. One necessary term is a minor of a graph. A graph H is a *minor* of a graph G if H can be achieved from a subgraph G by a series of edge contractions (i.e., replacing an edge uv and its ends by one node x). Other useful terms that are necessary to explain the algorithms are diameter pairs and eccentricity pairs. The diameter of a graph G is the smallest number ψ such that the shortest path between any nodes of G has distance at most ψ where the edges all have length equal to one. A *diameter pair* of the graph G is a pair of nodes having their distance equal to ψ. Similarly, one defines the *eccentricity* of a node w to be the smallest number $\eta(w)$ such that the shortest path between w and any other node of G is at most $\eta(w)$. Furthermore, the pair (w, v) forms an *eccentricity pair* for node w if the distance of any w, v-path is $\eta(w)$.

2.1. Branch Decompositions

Although the paper focuses on the branchwidth of linear matroids, we deem it relevant to define branch decompositions and branchwidth in terms of submodular set functions because we offer theoretical results in this general setting.

Given some finite set E and some function f over E to the non-negative integers, the function f is *symmetric* if $f(X) = f(E \setminus X)$ for any subset $X \subseteq E$. The function f is *submodular* if $f(X \cap Y) + f(X \cup Y) \leq f(X) + f(Y)$ for any $X, Y \subseteq E$. For a submodular function f over a finite set E, let $\lambda : 2^E \to \mathbb{Z}_+$ be defined as $\lambda(X) = f(X) + f(E \setminus X) - f(E) + 1$ for $X \subseteq E$. Note that the function λ is submodular and symmetric, that is, $\lambda(X \cap Y) + \lambda(X \cup Y) \leq \lambda(X) + \lambda(Y)$ for all $X, Y \subseteq E$ and $\lambda(X) = \lambda(E \setminus X)$ for any $X \subseteq E$. This function λ is called a *connectivity function* of f. A partition (X, Y) of E is called a *separation*, and its order is defined to be $\lambda(X)$.

A *tree* is an acyclic and connected graph. A vertex of degree 1 in the tree is called a *leaf*, otherwise, the vertex is a *non-leaf vertex or inner vertex*. Given some finite set E and a submodular function f defined over E, let T be a tree having $|E|$ leaves and in which every non-leaf vertex has degree at least three. Associate with each leaf w of T one of the elements of E, say $\tau(w)$, in such a way that each element of E is associated with a distinct leaf (τ is a bijection). The pair comprised of the tree T and the function τ, is a *partial branch decomposition* of f. If each inner vertex of T has degree exactly three, then T is called *cubic* and the pair (T, τ) is a *branch decomposition*. Under the conditions where T is cubic with at least $|E|$ leaves and τ is an injective function (one-to-one), then the pair (T, τ) is called an *injective branch decomposition*. Furthermore, if T is cubic but τ is a surjective function and T has at most $|E|$ leaves, then the pair (T, τ) is called a *surjective branch decomposition*. We will denote a *relaxed branch decomposition* to describe any of the four types of branch decompositions (exact, partial, injective, surjective). Notice that a partial branch decomposition is a relaxation on the degree requirement of the tree T and injective and surjective branch decompositions are relaxations on the function τ.

For a relaxed branch decomposition (T, τ), if T' is a connected subgraph of T and $X \subseteq E$ is the set given by the union of the labels (by τ) for the leaves of T', then we say that (T, τ) *displays* X or that X is *displayed* in (T, τ) by the component T'. Also, the separation $(X, E \setminus X)$ is *displayed* in a relaxed branch decomposition (T, τ) if both X and $E \setminus X$ are displayed in (T, τ). If T' is a leaf (i.e., $|X| = 1$), then the corresponding separation $(X, E \setminus X)$ is called a *leaf separation*. Obviously, there are $|E|$ such leaf separations in all for any relaxed branch decomposition except surjective branch decompositions. For a relaxed branch decomposition (T, τ), the *width* of an edge e of T is defined to be $\lambda(X)$ where X is the set displayed in (T, τ) by one of the components of $T \setminus e$, and without any confusion, the width of e can be written as $\lambda(e)$. Accordingly, the *width* of the relaxed branch decomposition (T, τ), denoted $width(T, \tau)$, is the maximum among the widths of its edges. Notice that any injective branch decomposition of width k can be transformed into a branch decomposition of width k by deleting unlabeled (by τ) leaves and contracting edges. Also, given a surjective branch decomposition $(\hat{T}, \hat{\tau})$ and a branch decomposition (T, τ) of a submodular function f over a finite set where every separation associated with $(\hat{T}, \hat{\tau})$ is displayed in (T, τ), there is an injection from the edges of $\hat{T}$ to those of T and there is an injection from the vertices of $\hat{T}$ to those of T. Both transformations will be beneficial in Section 3. The *branchwidth* of submodular function f is the minimum among the widths of all branch decompositions of f. Next, we prove a lemma that will also be beneficial in Section 3.

Lemma 1. *Let $(\hat{T}, \hat{\tau})$ be a relaxed branch decomposition of f over E with $width(\hat{T}, \hat{\tau}) \leq k$, for some integer $k > 0$. Also, let $\hat{\lambda}$ be the corresponding connectivity function and let (A_1, B_1) and (A_2, B_2) be displayed in $(\hat{T}, \hat{\tau})$ where $A_1 \cap A_2 = \emptyset$. In addition, define $A = A_1 \cup A_2$ and $B = E \setminus A$. If (X_1, X_2) is displayed in $(\hat{T}, \hat{\tau})$, then $\hat{\lambda}(B \cap X_1) \leq k$ or $\hat{\lambda}(B \cap X_2) \leq k$.*

Proof: First, notice that if $X_i \cap A_j \neq \emptyset$ where $i \in \{1, 2\}$, $j \in \{1, 2\}$, then either $X_i \subseteq A_j$ or $A_j \subseteq X_i$. Secondly, notice that if $X_i \subseteq A_j$ where $i \in \{1, 2\}$, $j \in \{1, 2\}$, then $\hat{\lambda}(B \cap X_i) \leq k$. Hence, without loss of generality we can assume that $A_1 \subseteq X_1$ and $\hat{\lambda}(A \cap X_1) \geq \hat{\lambda}(A)$. By submodularity, we have that $\hat{\lambda}(B \cap X_2) = \hat{\lambda}(A \cup X_1) \leq \hat{\lambda}(X_1) \leq k$. ∎

2.2. Tree building

Given any partial branch decomposition, any of the leaf separations can always be induced by deleting an edge incident with a leaf from the partial branch decomposition tree. Thus, the *star graph* (a tree with only one inner vertex and all the other vertices adjacent to it) with a bijection from the element set to its leaves, is a trivial partial branch decomposition. Given a star graph with inner vertex v as an initial partial branch decomposition, if one finds a reasonable partition (X,Y) of the set D, denoting the edges incident with the vertex v; split v into two vertices x and y, making X incident with x and Y incident to y; and connect x and y by an edge e, then one would have another partial branch decomposition. Notice that the leaves and bijection of the first partial branch decomposition are exactly the same leaves and bijection as this new partial branch decomposition. This procedure is called *splitting a vertex*. In this new partial branch decomposition, all the previous separations obtained in the star graph are preserved, and a new separation (X,Y) can be induced by deleting e from the new partial branch decomposition. In addition, it is easy to see that the maximum degree has been reduced after such a procedure. This gives a hint of how to proceed in order to produce branch decompositions.

After a closer look at the partial branch decompositions, one may find, as described above, (T_{i+1}, τ) which preserves all the separations which can be displayed in (T_i, τ). Thus, the width of

(T_{i+1}, τ) is always greater or equal to the width of (T_i, τ). In an algorithmic point of view, at each iteration i, a vertex with "large" degree (more than 3) is picked and split in a way that hopefully keeps the width of (T_{i+1}, τ) small.

In the first iteration, a separation (A, B) is found such that $|A|, |B| \geq 2$ and a new partial branch decomposition (T_2, τ) is created where v is replaced by vertices x and y and edge (x, y) and x has the leaves corresponding to A and y has the leaves corresponding to B. The above split is called an *initial split*, and later splits are called *sequential splits*.

2.2.1. k-split

In the case of conducting a k-split of the vertex v of degree greater than three, let D be the edge set incident with v in (T_i, τ). Also, suppose the heuristic partitions D into k nonempty subsets $X_1, X_2, ...$ and X_k. For every $1 \leq j \leq k$ where $|X_j| \geq 2$, generate a new vertex x_j such that v and x_j are adjacent and replace v with x_j for the edges in X_j. We will keep the edges in X_j incident with v for any $|X_j| = 1$ (see Figure 1 for example of 2-split and 3-split). The new tree T_{i+1} is formed by the above procedure. Moreover, (T_{i+1}, τ) is a partial branch decomposition. One may notice that in the case of a 2-split, vertex v turns out to be of degree 2 in T_{i+1}, thus to keep (T_{i+1}, τ) a partial branch decomposition, one needs to contract one of the edges of v before the next iteration. We say that (T_j, τ) is *extended* from (T_i, τ) for any pair of i and j, such that $i < j$ (see Cook and Seymour [7]).

Figure 2 shows a sample tree building process which terminates in 3 iterations. The size of the element set is 6. Using a 2-split and a 3-split, one goes from the star graph to the third partial branch decomposition. Note that, every inner vertex of T_3 has degree 3; (T_3, τ) is a branch decomposition. For the case in Figure 2, we have $width(T_1, \tau) \leq width(T_2, \tau) \leq width(T_3, \tau)$.

2.3. Matroid

Let E be a finite set and $\mathcal{I}$ be a family of subsets of E, called *independent sets*. We define $M = (E, \mathcal{I})$ to be a *matroid* if the following axioms are satisfied:

M0 $\emptyset \in \mathcal{I}$

M1 If $J' \subseteq J \in \mathcal{I}$, then $J' \in \mathcal{I}$.

M2 For every $H \subseteq E$, every maximal independent subset of H has the same cardinality.

For a matroid $M = (E, \mathcal{I})$, a *dependent* set is a subset of E that is not a member of $\mathcal{I}$. A *circuit* of a matroid is a minimally dependent subset (i.e., deleting any member results in an independent set). In order to describe the mat-flow method, matroid minors have to be defined as well. Let $M = (E, \mathcal{I})$ be a matroid and let $Y \subseteq E$. Define $\mathcal{I}' = \{J \mid J \subseteq (E \setminus Y), J \in \mathcal{I}\}$. The matroid $M' = (E \setminus Y, \mathcal{I}')$ is the matroid arising by *deleting* Y and is denoted by $M \setminus Y$. Now, consider J_Y, a maximal independent subset of Y in $\mathcal{I}$. Define $\hat{\mathcal{I}} = \{J \mid J \subseteq (E \setminus Y), J \cup J_Y \in \mathcal{I}\}$. The matroid $\hat{M} = (E \setminus Y, \hat{\mathcal{I}})$ is the matroid arising by *contracting* Y and is denoted by M/Y. Hence, a matroid $\bar{M}$ is a minor of a matroid M if $\bar{M}$ can be achieved by a series of contracting or deleting elements of the ground set of M (order does not matter). For more explanations and examples of matroids, please refer to Oxley [24].

Now, we would like to introduce the linear matroid. Let $\mathbb{F}$ be a field and $\mathbf{A} = (\mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_n)$ be a matrix over $\mathbb{F}$. Let the element set E be the column indices of $\mathbf{A}$ and $\mathcal{I} = \{J \subseteq E :$ the columns indexed by elements of J are linearly independent$\}$. Then, this $M = (E, \mathcal{I})$ is a matroid, and a matroid of such kind is a *linear matroid*. A *circuit* of a linear matroid is the set of indices of a minimal set of linearly dependent columns. Further, we define λ to be the

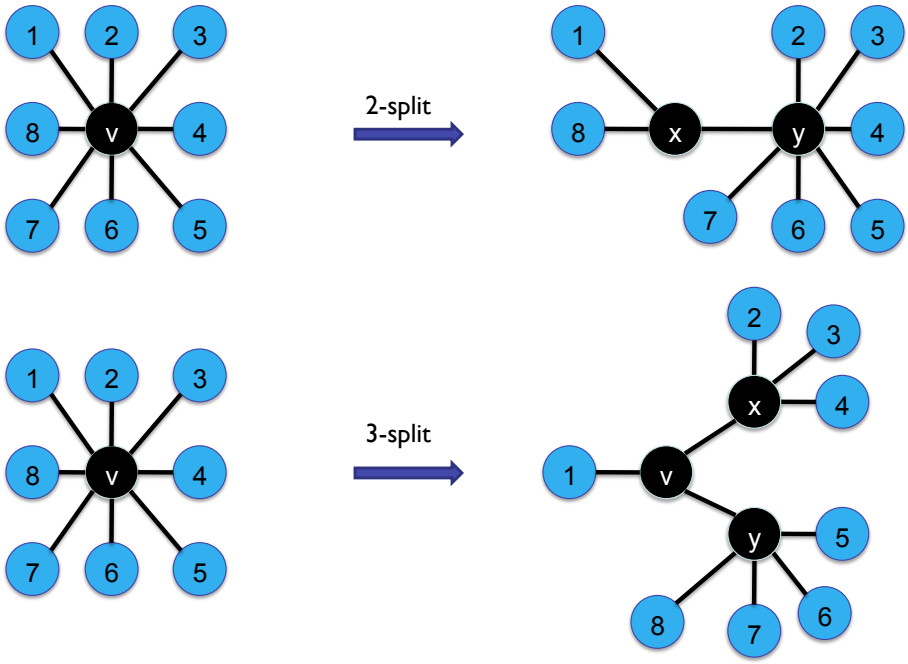

Figure 1: 2-split and 3-split

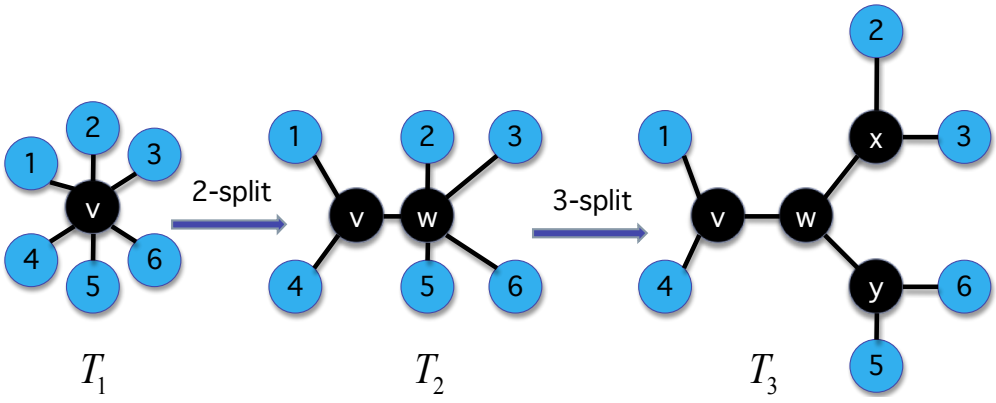

Figure 2: Sample Tree Construction

connectivity function of any matroid where f is the rank function of the matroid, denoted by r (i.e., $\lambda(X) = r(X) + r(E \setminus X) - r(E) + 1$ for $X \subseteq E$).

3. Pushing

Given some matroid, define that the partial branch decomposition (T, τ) is *k-extendible* if there exists a branch decomposition (T^*, τ^*) with $width(T^*, \tau^*) \leq k$ such that every separation displayed in (T, τ) is displayed in (T^*, τ^*). Without loss of generality, we assume k to be the smallest width of the branch decomposition from which the current partial branch decomposition can be extended to. Given some matroid, define (T_1, τ) to be the partial branch decomposition comprised of only one non-leaf vertex (i.e., T_1 is a star). Certainly (T_1, τ) is k-extendible when k equals to the branchwidth. In the i-th iteration, we desire to choose X and Y such that if (T_i, τ) is k-extendible then so is (T_{i+1}, τ). If (T_{i+1}, τ) is also k-extendible, then the vertex split from (T_i, τ) to (T_{i+1}, τ) is a greedy one. Repeatedly choosing separations with small orders, without regard to other considerations, turns out to be too short-sighted in practice. Indeed, there might always be a "safe" way to split. By "safe" we mean that if the current partial branch decomposition is k-extendible, then the new partial branch decomposition will be k-extendible as well. We do not know how to check directly if a given (T, τ) is k-extendible, thus, we need theoretical results, The Pushing Lemma and The Strong Closure Corollary, to maintain the k-extendibility. The name "pushing" is originated from Cook and Seymour [7], but their pushing lemma was proved only for graphs. Hence, we have extended their result for any partial branch decomposition of a submodular function over a finite set.

Let D be the edge set incident with v in (T_i, τ), and edges $e_1, e_2 \in D$ be distinct edges. Consider the 2-split of vertex v which partitions D into 2 subsets $X = \{e_1, e_2\}$ and $Y = D \backslash X$, makes edges in X incident with a new vertex x and Y incident with a new vertex y and connects x and y. This 2-split of v yields a new partial branch decomposition (T_{i+1}, τ). Let $\lambda(e_1 \cup e_2)$ be the short form representing the order of the separation induced by deleting the edge xy from (T_{i+1}, τ). Also, recall that $\lambda(e_1)$ and $\lambda(e_2)$ are the orders of the separations induced by deleting e_1 from (T_i, τ) and deleting e_2 from (T_i, τ) respectively.

Lemma 2. [The Pushing Lemma] *Let f be any submodular function over the set E to the non-negative integers and let λ be its connectivity function and let (T_i, τ) be a partial branch decomposition of f. Let v be a vertex with degree more than three in T_i, D be the set of edges in T_i incident with the vertex v and edges $e_1, e_2 \in D$ be distinct edges. Suppose*

$$\lambda(e_1 \cup e_2) \leq \max\{\lambda(e_1), \lambda(e_2)\}, \tag{1}$$

holds. Then taking the partition $X = \{e_1, e_2\}$, $Y = D \setminus X$ of D for the 2-split of vertex v yields a partial branch decomposition (T_{i+1}, τ) which is k-extendible if (T_i, τ) is k-extendible, for some integer $k > 0$.

> **Proof:** Let A_1 be the set displayed by the component of $T_i \setminus e_1$ in (T_i, τ) not containing v, A_2 be the set displayed by the component of $T_i \setminus e_2$ in (T_i, τ) not containing v, $A = A_1 \cup A_2$ and $B = E \setminus A \neq \emptyset$ (since v has degree more than three). Without loss of generality, assume that $\lambda(A_1) \geq \lambda(A_2)$, thus $\lambda(A_1) \geq \lambda(A)$. Hence, (A, B) is a separation of order at most k. Since (T_i, τ) is k-extendible, there exists a branch decomposition (T^*, τ^*) of M with $width(T^*, \tau^*) \leq k$ and every separation displayed

in (T_i, τ) is displayed in (T^*, τ^*). If (T^*, τ^*) displays A, we are done. Hence, we can assume that (T^*, τ^*) does not display A and we aim to construct a new branch decomposition (T', τ') where every separation of (T_{i+1}, τ) is displayed in (T', τ') and $width(T', \tau) \leq k$.

We can assume that T^* has degree-3 vertices and that $k \geq 2$, as otherwise the lemma holds trivially. If v is a vertex of T^* and e is an edge of T^*, let X_{ev} denote the set of elements of E displayed by the component of $T^* \setminus e$ in (T^*, τ^*) that does not contain v. Now, we will use a slight variation of a result by Geelen et al. [25]. We offer the proof of the claim such that the whole result is self-contained.

Claim 1. *There exists a degree-3 vertex s of (T^*, τ^*) with incident edges f_1, f_2 and f_3 such that, for each edge e of (T^*, τ^*), $\lambda(X_{es} \cap B) \leq k$, and none of $X_{f_1 s}$, $X_{f_2 s}$ and $X_{f_3 s}$ is a proper subset of A_1 or A_2.*

> **Proof:** For this proof, we will use a surjective branch decomposition of f derived from (T^*, τ^*) to find the corresponding special node s. let T^i denote the subgraph of $T^* \setminus e_i$ in (T^*, τ^*) that displays A_i and denote the end of e_i in T^i as u_i for each $i \in \{1, 2\}$. Now create $(\hat{T}, \hat{\tau})$ by $\hat{T} = T^* \setminus ((T^1 \cup T^2) \setminus \{u_1, u_2\})$ and $\hat{\tau}(e) = \tau^*(e)$ if $e \notin A_1 \cup A_2$ and $\hat{\tau}(e) = u_i$ if $e \in A_i$ for each $i \in \{1, 2\}$. Now, replace each edge of $\hat{T}$ by a directed edge (maybe two). For e, an edge of $\hat{T}$ with ends u and v, if $\lambda(X_{ev} \cap B) \leq k$, then replace e with the directed edge uv. Notice that if $\lambda(X_{ev} \cap B) \leq k$ and $\lambda(X_{eu} \cap B) \leq k$, then e is replaced with uv as well as vu. By Lemma 1, each edge of $\hat{T}$ gets replaced by a directed edge.
>
> Notice that each edge incident with a leaf in $(\hat{T}, \hat{\tau})$ is replaced by two directed edges. So, let us only consider the edges oriented away from the leaves. Also, for any other edge not incident with a leaf but has two orientations, we only consider one of the orientations. Now, $\hat{T}$ is a directed tree with each edge having only one orientation. Hence, $\hat{T}$ has more vertices than edges. By the pigeonhole principle, there is a vertex s with no outgoing edges. By the assumption of only considering edges oriented away from the leaves, s is not a leaf. Furthermore, s is also the correct node for T^* since there is an injection from the nodes of $\hat{T}$ to the nodes of T^*. Also, by the construction of $\hat{T}$, $s \in T^*$ also has the property that none of $X_{f_1 s}$, $X_{f_2 s}$ and $X_{f_3 s}$ is a proper subset of A_1 or A_2. ∎

Let s be the vertex satisfying Claim 1 and let X_i denote $X_{f_i s}$ for each $i \in \{1, 2, 3\}$. As in the proof of Claim 1, let T^i denote the subgraph of $T^* \setminus e_i$ in (T^*, τ^*) that displays A_i and denote the end of e_i in T^i as u_i for each $i \in \{1, 2\}$. Now, there exists $i_0 \in \{1, 2, 3\}$, such that $X_{i_0} \cap A_1 \neq \emptyset$. Say, $i_0 = 1$. Thus, $X_1 \cap A = A_1$, or $X_1 \cap A = A$. In either case, $\lambda(X_1 \cap A) \geq \lambda(A)$. Then by submodularity, $\lambda((X_2 \cup X_3) \cap B) = \lambda(X_1 \cup A) \leq \lambda(X_1) \leq k$. Now, we are ready to construct an injective branch decomposition $(\bar{T}, \bar{\tau})$ of width at most k that displays every separation displayed in (T_{i+1}, τ), including $(A, E \setminus A)$. First, create a copy of T^1 and T^2, including the labels (by τ^*). Next, remove all the labels (by τ^*) of A from the leaves in T^* to create $\bar{\tau}$. In addition, subdivide f_1 to create a new vertex b and make b adjacent to u_1 and u_2 from the copies of T^1 and T^2 to make $\bar{T}$. Notice that $(\bar{T}, \bar{\tau})$ can easily be transformed into a branch decomposition

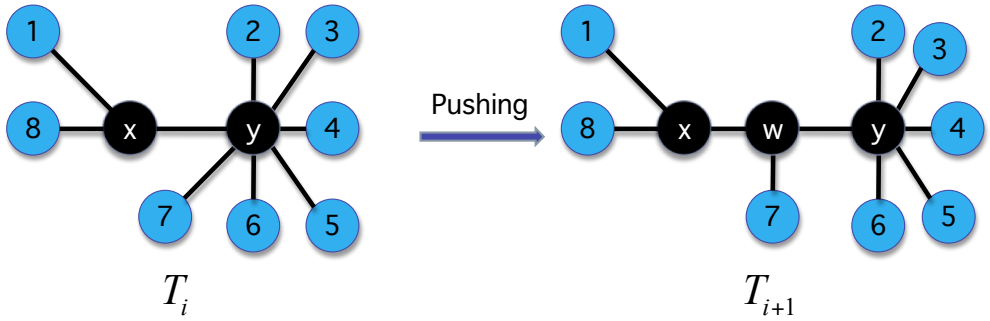

Figure 3: An Example of Applying The Pushing Lemma

(T', τ') with width most k and that displays every separation displayed in (T_{i+1}, τ) . Thus, (T_{i+1}, τ) is k-extendible. ∎

Inequality (1) is called the *pushing inequality*. In addition, the 2-split described in The Pushing Lemma is called a "push". The approach used in the proof was influenced from a proof offered by Geelen et al. [25]. Figure 3 is illustrated as an example of applying The Pushing Lemma. In Figure 3, let $D = \{xy, 2y, 3y, 4y, 5y, 6y, 7y\}$, $e_1 = xy$, and $e_2 = 7y$. Suppose $\lambda(e_1 \cup e_2) \leq \max\{\lambda(e_1), \lambda(e_2)\}$, then taking the partition $X = \{e_1, e_2\}$, $Y = D \setminus X$ of D for the 2-split of vertex v yields a new branch decomposition (T_{i+1}, τ). Thus, by The Pushing Lemma, (T_{i+1}, τ) is k-extendible if (T_i, τ) is k-extendible.

Before introducing the corollary, we will first introduce some notations and definitions to simplify later statements. Let D be the edge set incident with vertex v in partial branch decomposition (T_i, τ) which has degree greater than 3, and let $e, f \in D$ be distinct edges. The *strong closure* of any edge e in D is defined to be $Co(e) \triangleq \{f \in D \mid \lambda(e \cup f) \leq \lambda(e)\}$. Let $e_1 \in D$ with $Co(e_1) = \{f_1, f_2, \cdots f_s\}$. For $1 \leq j \leq s$, generating a new vertex x_j which is made incident with e_j and f_j; keeping $D' = D \setminus \{f_1, \cdots f_j\}$ incident with vertex v; and connecting x_j and v by a new edge e_{j+1} yields partial branch decomposition (T_{i+j}, τ). This partial branch decomposition (T_{i+j}, τ) is called *element-wise-pushed* from (T_i, τ) with respect to the strong closure of e_1.

Corollary 1. [The Strong Closure Corollary] *Let D be the edge set incident with vertex v in partial branch decomposition (T_i, τ). Given $e_1 \in D$ with $Co(e_1) = \{f_1, f_2, \cdots f_s\}$, for $1 \leq j \leq s$, element-wise-push a partial branch decomposition (T_{i+j}, τ) from (T_i, τ) with respect to the strong closure of e_1. Then, (T_{i+j}, τ) is k-extendible if (T_i, τ) is k-extendible and width$(T_{i+j}, \tau) \leq$ width(T_i, τ).*

Proof: If $|Co(e_1)|$=1, then it follows directly from The Pushing Lemma. Thus,we can assume the corollary is true for $j > 1$. For $Co(e_1) = \{f_1, f_2, \ldots f_{j+1}\}$, define $A^{(j)} \triangleq e_1 \cup \bigcup_{t=1}^{t=j} f_t$. By the submodularity, we have $\lambda(A^{(j+1)}) \leq \lambda(A^{(j)}) + \lambda(e_1 \cup f_{j+1}) - \lambda(e_1) \leq \lambda(A^{(j)})$. By The Pushing Lemma, (T_{i+j+1}, τ) is k-extendible because (T_{i+j}, τ) is k-extendible. ∎

In Figure 4, let $D = \{xy, 2y, 3y, 4y, 5y, 6y, 7y\}$ and $e_1 = xy$. Suppose $Co(e_1) = \{2y, 7y\}$, then the following operations would generate a new partial branch decomposition (T_{i+1}, τ): generate a new vertices u and w; replace xy, $2y$ and $7y$ with xu, uw, wy, $2u$ and $7w$; and keep $D' = D \setminus \{2y, 7y\}$

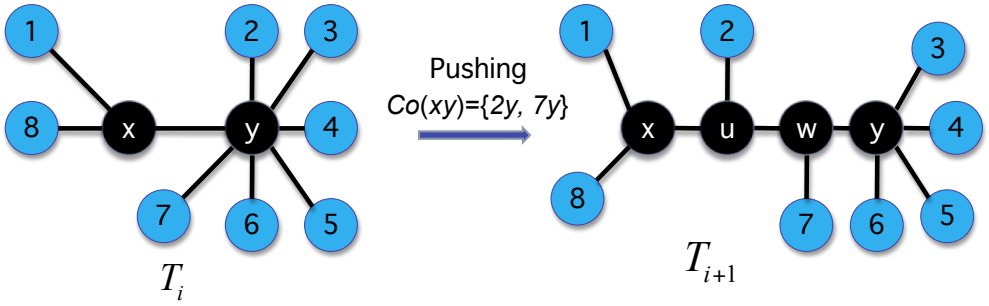

Figure 4: An Example Application of The Strong Closure Corollary

incident with vertex y. By The Strong Closure Corollary, (T_{i+1}, τ) is k-extendible if (T_i, τ) is k-extendible. Notice that $2u$ and $7w$ could be replaced with $2w$ and $7u$ and the new partial branch decomposition would still be k-extendible if (T_i, τ) is k-extendible.

4. Similarity graph

This paper focuses on the heuristics for finding near-optimal branch decompositions. In the following discussion, we will always focus on linear matroid $M = (E, \mathcal{I})$ associated with matrix $\mathbf{A} = (\mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_n) \in \mathbb{F}^{m\times n}$, where $\mathbb{F}$ is the field. Thus, the element set $E = \{1, 2, ...n\}$.

A branch decomposition can be regarded as a global arrangement of the elements in the element set E. Recall Section 2.2. The heuristics derive a branch decomposition after several iterations of incorporating separations to partial branch decompositions. As stated before, the width of the partial branch decomposition may grow after each iteration. Thus, the difficulty is how to introduce low order new separations when pushing is unavailable such that the width of the new partial branch decomposition stays small. To tackle the above difficulty, we developed three methods: the classification method, the max-flow method and the mat-flow method, all use the similarity graph in some fashion. To introduce the similarity graph, one needs to define the similarity measure first.

For an edge e of T, recall that the order of the separation (X, Y) induced by $T \setminus e$ is $\lambda(X) = r(X) + r(E \setminus X) - r(E) + 1$. Thus, a "good" partition (X, Y) always features a small similarity between the spanning space of sets displayed by X and Y. From now on, we will say "X" instead of "the columns displayed by X" and "column" instead of "the column indexed by the element" for convenience.

No matter what method is chosen to generate a separation of the branch decomposition, the above "similarity" between the spanning space of X and Y is desired to be small. Thus, there should be a way to qualitatively measure the "similarity" between the elements of the matroid. Particularly, we identify the similarity measure between two columns by the potential of the subspaces generated by a set including each column being near-parallel (i.e., the value of sine of the angle between the two columns). Thus, given an initially labeled set $B \subseteq E$, which is comprised of the set B_X of elements from X and the set B_Y of elements from Y. Intuitively, by putting the most similar columns to B_X in X, and the most similar columns to B_Y in Y, one can expect a small $\lambda(X)$.

Let $1, 2, \ldots, n$ be the nodes of the similarity graph. For vectors over the reals, we define the distance between node i and node j to be the sine of the angle between $\mathbf{A}_i$ and $\mathbf{A}_j$ (i.e.,

$dist(i, j) = \sqrt{1 - (\frac{\mathbf{A}_i \cdot \mathbf{A}_j}{\|\mathbf{A}_i\|\|\mathbf{A}_j\|})^2}$). For vectors over a Galois field, we have to devise a similar measure of sine for this vector space which we discuss in the subsequent section. Nodes i and j are adjacent in the graph if and only if i is among γ nearest neighbors of j or j is among γ nearest neighbors of i for parameter $\gamma \in \mathbb{N}$, and the distance between i and j is strictly less than 1. The parameter γ is very important for the heuristics. Small γ may cause a disconnected similarity graph although the original input graph is connected. Large γ may cause the diameter of the similarity graph or its minors to be so close to one that all the separations would have equal value.

For a sequential split in the k-th iteration, let v be the vertex to split in the iteration k, and D be the edges incident with v in T_k. For each edge e in D, delete e from T, choose the component T_e of T_k without vertex v, and denote the elements displayed by T_e in (T_k, τ) to be A_e. If $|A_e| \geq 2$, identify the nodes in A_e into one node v_e, else v_e represents A_e. An edge is placed between v_{e_i} and v_{e_j} if and only if there exists some node in A_{e_i} and some node in A_{e_j} connected to each other in the similarity graph. As a result, the constructed graph G_v can be derived from the similarity graph by node identification.

5. Galois Cosine

When working with vectors over the real numbers, there are well-established conventions for determining the "distance" between two vectors. For example, given vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$, the standard dot product $\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}|cos(\theta)$ gives a measurement of the "difference" between two vectors. Yet, when the two vectors $\vec{u}$ and $\vec{v}$ have entries over some finite field (i.e., $\vec{u}, \vec{v} \in \mathbb{F}_q^n$), such "distance" or "difference" measurements become meaningless. In this section, we define a special "cosine" function which attempts to describe the distance between two vectors in a finite field. The reader is referred to the work of Dummit and Foote [26] for unfamiliar terms discussed in this section.

Definition 1. *Fix an irreducible polynomial $\Phi(x)$ of degree n such that*

(a) $\Phi(x)$ divides $x^{q^n} - x$ in the polynomial ring $\mathbb{F}_q[x]$, and

(b) $x \bmod \Phi(x)$ is a generator for the cyclic group consisting of the non-zero elements in $GF(q^n) = \mathbb{F}_q[x]/\langle\Phi(x)\rangle$.

Recall that any vector $\vec{u} \in \mathbb{F}_q^n$ maps to an element of $GF(q^n) = \mathbb{F}_q[x]/\langle\Phi(x)\rangle$ via the following transformation:

$$\vec{u} = (u_0, \ldots, u_{n-1}) \in \mathbb{F}_q^n \quad \Longrightarrow \quad u_0 + u_1 x + u_2 x^2 + \cdots + u_{n-1} x^{n-1} \in GF(q^n) = \mathbb{F}_q[x]/\langle\Phi(x)\rangle$$

Note that we have chosen $\Phi(x)$ such that $x \bmod \Phi(x)$ is a generator for the cyclic group consisting of the non-zero elements in $GF(q^n)$. Therefore, each of the polynomials $(u_0 + u_1 x + u_2 x^2 + \cdots + u_{n-1} x^{n-1})$ and $(v_0 + v_1 x + v_2 x^2 + \cdots + v_{n-1} x^{n-1})$ has an associated integer e_u and e_v such that

$$(u_0 + u_1 x + u_2 x^2 + \cdots + u_{n-1} x^{n-1}) \equiv x^{e_u} \equiv 0 \bmod \Phi(x)\,,$$
$$(v_0 + v_1 x + v_2 x^2 + \cdots + v_{n-1} x^{n-1}) \equiv x^{e_v} \equiv 0 \bmod \Phi(x)\,.$$

Then, for each pair of nonzero vectors $\vec{u}, \vec{v} \in \mathbb{F}_q^n$, we can find an integer e such that $\Phi(x)$ divides $(u_0 + u_1x + u_2x^2 + \cdots + u_{n-1}x^{n-1}) - (v_0 + v_1x + v_2x^2 + \cdots + v_{n-1}x^{n-1})x^e$ in $\mathbb{F}_q[x]$. Finally, we define

$$\cos_\Phi(\vec{u}, \vec{v}) = \cos\left(\frac{2\pi e}{(q^n - 1)/(q - 1)}\right).$$

When defined in this way, this special $\cos_\Phi$ has several different useful properties. Since these properties are easily derived, we present them without a proof.

Proposition 1. *Let $\vec{u}$ and $\vec{v}$ be vectors in $\mathbb{F}_q^n$.*

(a) Symmetry: $\cos_\Phi(\vec{u}, \vec{v}) = \cos_\Phi(\vec{v}, \vec{u})$.

(b) Homothety: $\cos_\Phi(\alpha\vec{u}, \beta\vec{v}) = \cos_\Phi(\vec{u}, \vec{v})$ *for any non-zero scalars* $\alpha, \beta \in \mathbb{F}_q$.

(c) Nondegeneracy: $\cos_\Phi(\vec{u}, \vec{v}) = 1$ *if and only if* $\vec{v} = \alpha\vec{u}$ *for some non-zero scalar* $\alpha \in \mathbb{F}_q$.

(d) For the basis vectors $\vec{e_i}$ and $\vec{e_j}$, we have the values

$$\cos_\Phi(\vec{e_i}, \vec{e_j}) = \cos\left(\frac{2\pi(i - j)}{(q^n - 1)/(q - 1)}\right).$$

We note that the third and fourth properties are somewhat undesirable. Given a vector $\vec{u} \in \mathbb{F}_q^n$, we see by the third property that $\cos_\Phi(\vec{u}, \pm\vec{u}) = 1$. Thus, u and $-\vec{u}$ are at the same "angle". Thus, this definition does not allow for the concept of "opposite directions". Furthermore, the fourth property shows that the canonical basis vectors are not orthogonal with respect to this measure of "angle". Here is a concrete example.

Example 1. *Let $q = 3$ and $n = 2$. We must choose an irreducible polynomial $\Phi(x)$ such that $\Phi(x)$ divides $x^{q^n} - x$, and x mod $\Phi(x)$ is a generator for the cyclic group of non-zero elements in $GF(q^n) = \mathbb{F}_q[x]/\langle\Phi(x)\rangle$. We note that*

$$\frac{x^{q^n} - x}{x^q - x} = (x^2 + 1)(x^2 + x - 1)(x^2 - x - 1) .$$

Although each of these polynomials is irreducible over $\mathbb{F}_3$, when we consider x^2+1 as a candidate for $\Phi(x)$, we note that x, x^2, x^3 and x^4 are congruent to $x, 2, 2x$ and 1 mod $\Phi(x)$, respectively. Thus, the order of x in $GF(q^n) = GF(3^2) = GF(9)$ is 4, not 8. In contrast, x has order 8 when considered either mod$(x^2 + x - 1)$ or mod$(x^2 - x - 1)$. Thus, either $x^2 + x - 1$ or $x^2 - x - 1$ are reasonable candidates for $\Phi(x)$, since they are both irreducible and primitive. For this example, we choose $\Phi(x) = x^2 + x - 1$. Then we see the following:

$$\begin{aligned}
\vec{u} = (0,1) &\Longrightarrow x &&\Longrightarrow & x^1 &\equiv x && \text{mod } (x^2 + x - 1)\\
\vec{u} = (0,2) &\Longrightarrow 2x &&\Longrightarrow & x^5 &\equiv 2x && \text{mod } (x^2 + x - 1)\\
\vec{u} = (1,0) &\Longrightarrow 1 &&\Longrightarrow & x^0 &\equiv 1 && \text{mod } (x^2 + x - 1)\\
\vec{u} = (1,1) &\Longrightarrow 1 + x &&\Longrightarrow & x^7 &\equiv 1 + x && \text{mod } (x^2 + x - 1)\\
\vec{u} = (1,2) &\Longrightarrow 1 + 2x &&\Longrightarrow & x^2 &\equiv 1 + 2x && \text{mod } (x^2 + x - 1)\\
\vec{u} = (2,0) &\Longrightarrow 2 &&\Longrightarrow & x^4 &\equiv 2 && \text{mod } (x^2 + x - 1)\\
\vec{u} = (2,1) &\Longrightarrow 2 + x &&\Longrightarrow & x^6 &\equiv 2 + x && \text{mod } (x^2 + x - 1)\\
\vec{u} = (2,2) &\Longrightarrow 2 + 2x &&\Longrightarrow & x^3 &\equiv 2 + 2x && \text{mod } (x^2 + x - 1)
\end{aligned}$$

Since $(q^n - 1)/(q - 1) = 4$*, we can use the differences of the exponents above to compute the following tables of cosines:*

$\cos_{\Phi(x)}(\vec{u}, \vec{v}) = \cos\left(\frac{\pi e}{2}\right)$	$(1,0)$	$(2,0)$	$(0,1)$	$(1,1)$	$(2,1)$	$(0,2)$	$(1,2)$	$(2,2)$
$(1,0)$	1	1	0	0	−1	0	−1	0
$(2,0)$	1	1	0	0	−1	0	−1	0
$(0,1)$	0	0	1	−1	0	1	0	−1
$(1,1)$	0	0	−1	1	0	−1	0	1
$(2,1)$	−1	−1	0	0	1	0	1	0
$(0,2)$	0	0	1	−1	0	1	0	−1
$(1,2)$	−1	−1	0	0	1	0	1	0
$(2,2)$	0	0	−1	1	0	−1	0	1

□

We implemented this method in MATLAB and used the function `gfprimdf(n)` for generating the irreducible polynomial. The experimental results using this approach for binary matroids are detailed in Section 9.

6. The classification method

In this section, we are introducing the Laplacian Eigenmap techniques of data classification to our branch decomposition heuristics. The linear matroid $M = (E, \mathcal{I})$ is associated with matrix $\mathbf{A} = (\mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_n) \in \mathbb{F}^{m\times n}$ with element set $E = \{1, 2, ...n\}$. Regard $\{\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_n\}$ as the data set to be classified. Also, let $k > 0$ be some integer parameter indicating the desired number of partitions. A complete subroutine of classification can be divided into the following 5 steps.

Step 1: Construct the similarity graph

For vertex v of T with degree higher than 3, we construct G_v as described in Section 4.

Step 2: Choosing the weights

Let $\mathbf{W}$ denote the adjacency matrix of G_v.

Step 3: Computing the eigenfunctions of the Laplacian

For the initial split, assume the graph G_v is connected. Otherwise proceed with step 3 for each connected component. Compute eigenvalues and eigenvectors for the generalized eigenvector problem

$$\mathbf{Lf} = \lambda\mathbf{Df}, \tag{2}$$

where $\mathbf{D}$ is the diagonal weight matrix, and its entries are column sums of $\mathbf{W}$, $\mathbf{D}_{ii} = \sum_j \mathbf{W}_{ji}$. $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the Laplacian matrix. Laplacian is a symmetric, positive semidefinite matrix that can be thought of as an operator on functions defined on the vertices of G.

Let $\mathbf{f}_0, \cdots, \mathbf{f}_{n-1}$ be the solutions of equation (2), ordered according to the generalized eigenvalues:

$$\begin{aligned} L\mathbf{f}_0 &= \lambda_0\mathbf{Df}_0 \\ L\mathbf{f}_1 &= \lambda_1\mathbf{Df}_1 \\ &\vdots \\ L\mathbf{f}_{n-1} &= \lambda_{n-1}\mathbf{Df}_{n-1} \end{aligned} \tag{3}$$

where $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{n-1}$. We leave out the eigenvector $\mathbf{f}_0$ corresponding to eigenvalue 0 and use the p eigenvectors corresponding to the p smallest non-zero eigenvalues for embedding in p-dimensional Euclidean space.

Step 4: Building the classifier

If $k \geq 2$, pick the initially labeled set; let C_i, $i \in \{1, \cdots, k\}$, be the classes; let C_i^{lab} be the labeled elements in the i-th class; and let $\mathbf{c}_i^{lab}$ be the characteristic function of those C_i (i.e., $\mathbf{c}_i^{lab}(x) = 1$ if $x \in C_i$, and $\mathbf{c}_i^{lab}(x) = 0$ otherwise).

Given a specific class C_i, to approximate the class, one needs to minimize the error function $Err(\mathbf{a}_i) = \sum_{j=1}^{s}(\mathbf{c}_i(x_j) - \sum_{l=1}^{p} \mathbf{a}_i(l)\mathbf{f}_l(j))^2$, where s is the cardinality of the labeled set, and without loss of generality the labeled set is denoted to be $\{1, 2, ..., s\}$. The minimization is considered over the space of coefficients $\mathbf{a}_i = (\mathbf{a}_i(1), ..., \mathbf{a}_i(p))$. The solution is given by

$$\mathbf{a}_i = (\mathbf{E}_{lab}^T \mathbf{E}_{lab})^{-1} \mathbf{E}_{lab}^T \mathbf{c}_i^{lab} \tag{4}$$

where $\mathbf{c}_i^{lab} = (c_1, c_2, ..., c_s)^T$ and $\mathbf{E}_{lab}$ is an $s \times p$ matrix whose q, r entry is $\mathbf{f}_r(q)$. Also, $\mathbf{u}_i = \sum_{j=1}^{p} \mathbf{a}_i(j)\mathbf{f}_j$ for any $i \in \{1, 2, \cdots, k\}$. The vector $\mathbf{u}_i$ is called an extended label of the i-th class.

Step 5: Classifying unlabeled elements

For $k \geq 3$, assign the j-th element to the class $\text{argmax}_i \mathbf{u}_i(j)$.

Hence, the aforementioned steps form the *k-classification* subroutine; according to k, the nodes of G_v are partitioned into k classes.

6.1. Applying the classification method and pushing

In the initial split, first construct the similarity graph G of the linear matroid, then compute the diameter pairs of G. There might be multiple diameter pairs. For each diameter pair (i, j), sort all of the nodes such that their distance to i is in non-decreasing order. Let A be the $\lceil \delta |V(G)| \rceil$ first terms and B be the $\lceil \delta |V(G)| \rceil$ last terms of the sorted list of nodes where δ is some predetermined parameter. Given $A, B \subseteq V(G)$, use the 2-classification subroutine to compute a separation (X, Y) of G such that $A \subseteq X$, $B \subseteq Y$. Among the separations computed for all the diameter pairs, pick the separation that gives the minimum order and use that separation for a 2-split to construct (T_2, τ).

For a sequential split in the k-th iteration, let v be the vertex in T_k to split in the iteration k, and D be the edges incident with v in T_k, construct the corresponding minor of the similarity graph G_v. In D, there should be only one edge not incident with a leaf, and denote this edge and its corresponding separation by e and (X, Y). Without loss of generality, assume that $X = \{x_1, \cdots, x_N\}$ are the leaves adjacent to v. Then G_v should be of size $N+1$, with a node s identified from Y in G and each one of the other nodes representing a node from X. For each $node_1$ in G_v that form an eccentricity pair for s, find $node_2$ in G_v to create a pair $(node_1, node_2)$ where $node_2$ is distinct from s and either the pair $(s, node_2)$ forms an eccentricity pair for s or the pair $(node_1, node_2)$ forms an eccentricity pair for $node_1$.

For each pair $(node_1, node_2)$, let $node_1$ and the indices in Y together be the labeled elements in the first class, compute the extended label u_1 of the first class using the formula in Step 4. Let $node_2$ and Y together be the labeled elements in the second class, compute the extended label u_2 of the second class using the formula in Step 4. Compare the two extended labels and assign the element j to the class i, $i \in \{1, 2\}$ having greater value of $u_i(j)$. Thus, X is split into the part in the first class W_1 and the part in the second class W_2, and correspondingly $(X \cup W_1, W_2)$ and $(X \cup W_2, W_1)$ are two separations of the element sets. Also, extended labels u_1 for separation $(X \cup W_2, W_1)$ and u_2 for separation $(X \cup W_1, W_2)$ are stored. In T_2, connect vertex v with 3 new

vertices x_1, x_2 and x_3, keep the edges in Y incident with x_1, W_1 incident with x_2 and W_2 incident with x_3. For any $1 \leq i \leq 3$, generate a new edge $x_i v$ between x_i and v and the partial branch decomposition (T_{k+1}, τ) is formed.

After the initial split (X, Y) of the current classification method, one can compute the strong closure of the X and Y, and push the elements in the closure, then continue classification. For the sequential splits, we try applying The Strong Closure Corollary first and then use the classification method to find a split if the closures are all empty.

6.2. Justification

The Laplacian eigenmaps handle the problem of embedding the similarity graph into p-dimensional Euclidean space. The embedding is given by the $n \times p$ matrix $\mathbf{Y} = [\mathbf{y}_1 \mathbf{y}_2, \cdots, \mathbf{y}_p]$, where the i-th row provides the embedding coordinates of the i-th node. It provides a choice to minimize the following objective function,

$$\sum_{ij} \|\mathbf{y}^{(i)} - \mathbf{y}^{(j)}\|^2 \mathbf{W}_{ij} = tr(\mathbf{Y}^T \mathbf{L} \mathbf{Y}),$$

where $\mathbf{y}^{(i)} = [\mathbf{y}_1(i), \cdots, \mathbf{y}_p(i)]^T$ is the p-dimensional representation of the i-th node, under appropriate constraints. The objective function with our choice of weights $\mathbf{W}_{ij}$ incurs a heavy penalty if neighboring points $\mathbf{x}_i$ and $\mathbf{x}_j$ are far apart. Therefore, minimizing it is an attempt to ensure that if $\mathbf{x}_i$ and $\mathbf{x}_j$ are "close", then $\mathbf{y}^{(i)}$ and $\mathbf{y}^{(j)}$ are close as well. This reduces to finding

$$\underset{\substack{\mathbf{Y}^T \mathbf{D} \mathbf{Y} = \mathbf{I} \\ \mathbf{Y}^T \mathbf{D} \mathbf{1} = \mathbf{0}}}{\arg\min} tr(\mathbf{Y}^T \mathbf{L} \mathbf{Y}).$$

The constraint $\mathbf{Y}^T \mathbf{D} \mathbf{Y} = \mathbf{I}$ prevents collapse onto a subspace of dimension less than p, and by the constraint $\mathbf{Y}^T \mathbf{D} \mathbf{1} = \mathbf{0}$ we require orthogonality to the constant vector. Standard methods show that the solution is provided by the matrix of eigenvectors corresponding to the lowest non-zero eigenvalues of the generalized eigenvalue problem $\mathbf{L}\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$ [16].

6.3. Connection to the eigenvector method

It is worthwhile to mention that in Cook and Seymour's eigenvector branch decomposition heuristic for graphs, they use Laplacian eigenmaps too, though just for identifying the source and sink nodes before calling the max-flow algorithm. For this section, let us consider the classification method only on graphs. Unlike our case that the adjacency matrix of the similarity graph is the weight matrix for the line graph $L(G)$ of the input graph G, they use $\mathbf{W}_{ij} = \sum (deg(u) - 1)^{-1} : u \in V$ and $i, j \in adj(u)$ if distinct i and j are connected in $L(G)$, else $\mathbf{W}_{ij} = 0$. We compute the Laplacian only once before the initial split, and then use the same p eigenvectors of the Laplacian as basis of the maps in every split when pushing is unavailable for the entire method. In contrast, they compute the weighted Laplacian in every initial and sequential splits when pushing is unavailable and use the only eigenvector corresponding to the second smallest eigenvalue of the Laplacian every time. Of course, the way they choose their weighted matrix makes the eigenvector method inapplicable to the setting of linear matroids, in general.

7. The max-flow method

The max-flow method inherits the ingredients of the previous branch decomposition heuristics [7, 8] and tree decomposition heuristics [21] and is utilized on the similarity graph. For clarity, the max-flow method utilizes a modified version of the shortest augmenting path algorithm for the max-flow problem to compute an edge version of Menger's Theorem (see Diestel [23]) where is the object is to find the minimum edge cut that separates all u, v-paths for some pair of nodes u and v in the input graph.

Let the linear matroid $M = (E, \mathcal{I})$ be associated with matrix $\mathbf{A} = (\mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_n) \in \mathbb{F}^{m\times n}$ with the element set $E = \{1, 2, ...n\}$. A complete subroutine of the max-flow method can be divided into the following 3 steps.

7.1. Max-flow subroutine

Step 1: Construction of the similarity graph

For the initial split and sequential splits, we construct G_v as described in Section 4.

Step 2: Compute diameter and diameter pairs

Compute the diameter of the current graph G_v and the diameter pairs.

Step 3: Compute the separation using max-flow

There might be multiple diameter pairs. For each diameter pair (i, j), sort all of the nodes such that their distance to i is in non-decreasing order. Let A be the $\lceil\delta|V(G_v)|\rceil$ first terms and B be the $\lceil\delta|V(G_v)|\rceil$ last terms of the sorted list of nodes where δ is some predetermined parameter. Given $A, B \subseteq V(G_v)$, use max-flow algorithm to compute a separation (X, Y) of G_v such that $A \subseteq X$, $B \subseteq Y$ and $E(X, Y)$ i.e., the cut between X and Y, is minimized. Among the separations computed, pick the separation with the minimum order and split v using this separation.

7.2. Applying the max-flow method and pushing

For the initial split, run the 3-step max-flow subroutine. In the i-th sequential split, for splitting a vertex v, denote the edges incident with v in the current partial branch decomposition (T_{i-1}, τ) to be D, compute strong closures for all the edges in D and push according to The Strong Closure Corollary. If the strong closures are all empty sets, apply the max-flow method again to generate a new partial branch decomposition. Repeat splitting vertices until a branch decomposition is formed.

8. The mat-flow method

The mat-flow method is very similar to the max-flow method but it is unique in its own right. The mat-flow method utilizes the matroid version of Menger's Theorem, proved by Tutte [22]. This theorem states that given two disjoint subsets A and B of E, the ground set of matroid M, the maximum size of a common indepdent set in $M \setminus A/B$ and $M \setminus B/A$ is equal to the minimum value of $r(X) - r(A) + r(Y) - r(B)$, where $r(*)$ denotes the rank function, taken over all sets X with $A \subseteq X \subseteq E \setminus B$. For clarity, one needs only to solve a matroid intersection problem to derive the corresponding sets X and Y. We used the algorithm posed by Cunningham [27] for solving matroid intersection problems.

Another distinction in the mat-flow method is that with the exception of the initial partial branch decomposition, every non-leaf node has only one edge which is not incident with a leaf. Hence, all sequential splits are 3-splits with v's non leaf edge being in a partition by itself. A complete description of the mat-flow method is given below.

8.1. Mat-flow subroutine

Step 1: Construction of the similarity graph

For only the initial split, we construct G_v as described in Section 4.

Step 2: Compute distance matrix

For only the initial split, compute the all pairs distance matrix D of G_v.

Step 3: Compute the separation using max-flow

For initial and sequential splits, use D to compute pairs (i, j) with the longest distance in D where i and j are leaves of G_v, say L_v. Also without any loss of ambiguity, let H denote $E \setminus L_v$ ($H = \emptyset$ for the initial split). For each pair (i, j), sort all of the nodes such that their distance to i is in non-decreasing order. Let A be the $\lceil \delta |L_v| \rceil$ first terms and B be the $\lceil \delta |L_v| \rceil$ last terms of the sorted list of nodes where δ is some predetermined parameter. Given $A, B \subseteq E$ and H, use the matroid intersection algorithm over $M \setminus B/(A \cup H)$ and $M \setminus A/(B \cup H)$ to compute a separation (X, Y) of L_v such that $A \subseteq X$, $B \subseteq Y$. The order associated with (X, Y) is the maximum of $\lambda(X \cup H)$ and $\lambda(Y \cup H)$. Among the separations computed, pick the separation with the minimum order and split (2-split if $H = \emptyset$ and 3-split if not) v using this separation.

8.2. Applying the mat-flow method and pushing

For the initial split, run the mat-flow subroutine. In the i-th sequential split, for splitting a vertex v, denote the edges incident with v in the current partial branch decomposition (T_{i-1}, τ) to be D, compute strong closures for only the non-leaf edge in D and push according to The Strong Closure Corollary. If the strong closure is the empty set, apply the mat-flow method again to generate a new partial branch decomposition. Repeat splitting vertices until a branch decomposition is formed.

9. Computational Results

The computer codes were written in MATLAB and all computations were run on a Sun Ultra 24 Workstation (3.0Ghz). For the classification method, the key is to map the original high-dimensional elements into a lower dimensional space, a space spanned by the p most significant eigenvectors of the normalized Laplacian. Hence, more information is preserved after the Laplacian mapping with a larger value of p. In contrast, if the value of p is too large then one may include some useless noise to the Laplacian mapping. To balance the significant information and the useless noise, we use $p = 2 \times \lceil \frac{m}{500} \rceil + 4$ for all the instances in the tables, where m is the number of columns. Although the performance of the classification method does not significantly depend on the specific choices of γ and δ, we use the following parameters to be consistent with the max-flow method for all test instances: $\gamma = 4$; $\delta = \frac{1}{3}$ for the initial split; and $\delta = \frac{1}{7}$ for the sequential splits. For the max-flow method and the mat-flow method, we use $\gamma = 4$ and $\delta = \frac{1}{3}$ for instances in the tables. Note that small γ for the max-flow and mat-flow method may cause unconnected similarity graphs although the original graph is connected. Large γ may cause the diameter of the similarity graph minor to become one; the max-flow method may be inefficient in that case.

In Tables 1, 2, 3, 4, 5, and 6, each instance is named in the first column. The second column and the third column are the size of matrix associated with the linear matroid instance. The fourth column is the branchwidth of the linear matroid instance. The last six columns report the output width and computational time for the methods. Branchwidth is denoted by β in the tables

Table 1: Results of the methods for binary matroids using Galois sine

					classification	method	max-flow	method	mat-flow	method
Names	rows	columns	β	Gtime	width	time(s)	width	time(s)	width	time(s)
V8	7	12	4	.023	5	0.118	4	0.285	5	0.272
R10	5	10	4	.001	5	0.047	5	0.043	5	0.046
N23	5	10	4	.001	5	0.015	5	0.019	5	0.087
N11	4	10	4	.001	5	0.016	4	0.041	4	0.093
K5	4	10	4	.001	4	0.016	4	0.019	5	0.032
dV8	5	12	4	.001	6	0.020	6	0.034	5	0.057
dN11	6	10	4	.003	4	0.021	4	0.021	4	0.089
dK5	6	10	4	.003	5	0.018	4	0.021	4	0.296
3-grid	5	9	3	.001	4	0.015	4	0.016	3	0.091
4-grid	8	16	4	.041	6	0.048	7	0.121	5	0.310
5-grid	13	25	5	832	7	832	9	832	9	832

for simplicity. Furthermore, for the instances of the methods over graphs, an "NP" in a column means that the instance graph was non-planar and the branchwidth is unknown.

First, let us examine the performance of calculations utilizing a sine function for vectors in Galois fields, detailed in Section 5. Although this measure can be adapted to any Galois field, we only considered binary matroids (i.e. GF(2^n)). These results are given in Table 1 where the first three columns describe the matroids; β gives the branchwidth; Gtime gives the time just to compute the Galois sine between every pair of vectors; and the final columns give the results of the corresponding methods. To further explain the matroids, the first eight matroids are binary matroids in the obstruction set for matroids of branchwidth three (see [28]). Hence, they all have branchwidth equal to four. The matroid R10 as well as N23 are self-dual. The grid matroids are binary matroids derived according to the fact that the corresponding $n \times n$ grid is the matroid's fundamental graph. These matroids have branchwidth equal to n, a result of Oum [14] relating branchwidth and rankwidth in conjunction with the result of Jelinek [29] on the rankwidth of grids. As seen from Table 1, the time for computing the Galois sine grows exponentially, overshadowing the time to compute widths of the heurisitics. This is a drawback of conducting Galois computations using MATLAB. Since these methods are heuristics, for further computations for all binary matroids we just use the sine over real vectors (where each vector in the Galois field is mapped to the same identical vector in the reals) to approximate the Galois sine. Table 2 gives these results for the same binary matroids of Table 1 plus others we were not able to compute in a reasonable time frame (2 days) using the Galois sine approach.

From these results, one can garner that approximating Galois sine with real sine is just as effective and maybe even more so in some cases. In addition, the mat-flow method appears to be the clear winner in terms of optimality gap (most with a gap of one) for the binary matroids in Tables 1 and 2. For Table 1, both the mat-flow and the max-flow method had 8 instances where the heuristics provided the lowest width out of the three methods, however, the max-flow method achieved optimality 5 times compared to 4 for the mat-flow method and 2 for the classification method. For Table 2, there was only one instance the mat-flow method didn't provide one of the lowest widths. In terms of speed, the slight favorite is the max-flow method

Table 2: Results of the methods for binary matroids using real sine

				classification	method	max-flow	method	mat-flow	method
Names	rows	columns	β	width	time(s)	width	time(s)	width	time(s)
V8	7	12	4	5	.018	5	.076	5	0.06
R10	5	10	4	4	.0138	5	.058	4	0.12
N23	5	10	4	5	.014	4	.047	4	0.1
N11	4	10	4	5	.013	4	.048	4	0.08
K5	4	10	4	4	.014	4	.035	4	0.08
dV8	5	12	4	6	.017	5	.077	4	0.06
dN11	6	10	4	5	.024	4	.083	4	0.04
dK5	6	10	4	5	.282	4	.196	4	0.03
3-grid	5	9	3	4	.013	4	.013	4	0.03
4-grid	8	16	4	5	.041	5	.087	5	0.13
5-grid	13	25	5	6	.25	6	.134	6	0.22
6-grid	18	36	6	7	.21	7	.368	7	0.38
7-grid	25	49	7	8	.528	8	.398	8	0.9
8-grid	32	64	8	9	.87	9	.669	9	1.38
9-grid	41	81	9	10	1.68	11	1.88	10	3.25
10-grid	50	100	10	11	3.13	13	2.09	11	6.01
15-grid	113	225	15	16	30.1	20	14.3	16	119.14
20-grid	200	400	20	24	261	24	63.1	22	933.43
25-grid	313	625	25	29	1200	30	231	28	4215.32
30-grid	450	900	30	31	8808	32	624	34	15930

with a close second from the classification method. Unfortunately, the mat-flow method is the slowest method. The slowness of the mat-flow can be contributed to the complexity of solving a number of matroid intersection problems as opposed a number of max-flow calculations. For instance, the algorithm for the matroid intersection problems has to compute a different auxiliary graph for every augmentation conducted in the algorithm. Even the clever techniques discussed by Cunningham [27], since the matroids are linear, were not enough to reduce the speed of the method to be comparable to the other two methods.

It has been shown that if a graph has a cycle with length at least 2, then the branchwidth of the linear matroid associated with the binary node-edge incidence matrix of the graph has the same branchwidth as the branchwidth of the graph itself [11]. This observation provides another feasible way to check the performance of the heuristics because there are a number of graph instances with branchwidth known in previous literature [8, 30]. These node-edge incidence matrices can be treated as binary matroid instances to test the performance of our heuristics for graphs.

In Table 3, the binary matroid instances are all node-edge incidence matrices of planar graphs. Thus, the number of columns are about three times the number of rows for the instances (because a planar graph with n nodes can have at most $3n - 6$ edges). In Tables 4, 5, and 6, the binary matroid instances are of compiler graphs. The number of columns are about the same as the number of rows for these instances. The known branchwidth of the compiler instances are small,

Table 3: Results of the methods for planar graph node-edge incidence matrix instances

				classification	method	max-flow	method	mat-flow	method
Names	rows	columns	β	width	time(s)	width	time(s)	width	time(s)
eil51	51	150	8	8	4	8	2	9	27
eil76	76	215	10	12	21	12	12	12	100
eil101	101	290	10	12	50	14	37	13	263
bier127	127	567	14	19	88	20	63	16	184
ch130	130	377	10	11	99	14	84	12	786
ch150	150	432	12	15	177	14	99	14	533
d493	493	1467	20	27	8887	24	1871	29	48093
gil262	262	773	15	20	1157	23	328	18	4027
d198	198	571	12	19	454	14	142	17	1588
fl417	417	1179	9	18	7092	18	2479	27	91502
kroA100	100	285	9	13	52	10	24	13	138
kroA150	150	432	11	14	146	14	81	20	703
kroA200	200	586	11	17	414	14	205	22	2593
kroB100	100	284	9	12	37	15	34	14	238
kroB150	150	436	10	15	153	13	86	14	1132
kroB200	200	580	12	17	447	18	207	18	1492
kroC100	100	286	9	13	41	12	26	13	186
kroE100	100	283	8	11	44	14	25	11	348
lin105	105	292	9	13	39	12	24	13	332
a280	280	788	13	17	1099	13	268	20	4100
pr107	107	283	6	11	40	11	22	12	144
pr124	124	318	8	9	191	12	2	11	476
pr136	136	377	10	14	83	11	66	16	255
pr144	144	393	9	15	99	10	51	11	520
pr152	152	428	8	13	105	15	42	16	703
pr226	226	586	7	12	339	11	130	16	3654
pr264	264	772	13	17	802	19	305	17	2936
pr299	299	864	11	18	2351	13	399	16	5111
pr76	76	218	9	14	17	14	2	12	58
rat195	195	562	12	15	241	15	2195	15	1690
rat99	99	279	9	12	29	12	25	14	125
rat575	575	1609	17	25	15115	26	3894	30	64009
rat783	783	2322	20	32	43368	29	6813	33	205132
rd100	100	286	11	13	36	15	33	13	330
rd400	400	1183	17	26	4950	23	1136	21	12416
tsp225	225	622	12	17	509	16	292	19	2638
u159	159	431	10	17	161	19	93	19	300

ranging from 2 to 4.

Table 4: Results of the methods for compiler graph node-edge incidence matrix instances

				classification	method	max-flow	method	mat-flow	method
Graphs	rows	columns	β	width	time(s)	width	time(s)	width	time(s)
aclear	9	10	2	3	0.05	3	0.03	2	0.25
advbnd	196	242	NP	5	22.42	5	9.99	7	215.17
arret	12	15	2	3	0.05	3	0.05	3	0.28
bcndb	25	34	2	3	0.03	4	0.17	3	1.69
bcndl	25	34	2	4	0.14	4	0.14	3	1.68
bcndr	25	34	2	4	0.13	4	0.15	3	1.67
bcndt	25	34	2	4	0.13	4	0.15	3	1.64
bilan	37	49	3	4	0.39	4	0.28	3	3.01
bilsla	18	22	2	3	0.05	3	0.07	3	0.31
buts	66	82	3	4	1.12	4	1.01	4	6.85
cardeb	48	64	3	6	0.53	4	0.58	5	6.23
coeray	11	14	2	3	0.03	3	0.03	3	0.18
colbur	40	54	NP	5	0.42	5	0.43	4	3.1
cosqb1	63	74	2	3	0.75	4	0.57	3	7.72
cosqb	8	10	2	3	0.01	3	0.04	3	0.09
cosqf1	63	74	2	3	0.1	4	0.4	3	7.76
cosqf	8	10	2	3	0.01	4	0.57	3	0.09
cosqi	21	24	2	4	0.06	3	0.09	2	0.33
dcoera	11	14	2	3	0.03	3	0.06	3	0.18
debflu	108	141	3	4	3.33	4	2.38	4	80.83
debico	60	76	3	5	0.93	4	0.80	5	6.7
decomp	117	150	NP	5	6.06	4	2.93	5	64.7
denitl	15	18	2	3	0.03	3	0.06	4	0.13
denitr	15	18	2	3	0.03	3	0.05	4	0.13
denpt	80	102	2	4	1.97	4	1.36	4	20.64
dens	6	7	2	3	0.02	2	0.03	3	0.06
densx	6	7	2	3	0.01	2	0.02	3	0.05
drepvi	71	96	3	5	1.66	4	1.12	5	14.41
drigl	21	29	3	5	0.10	4	0.19	4	0.64
dyeh	27	35	3	3	0.17	3	0.19	3	1.5
ecrd	21	24	2	4	0.06	3	0.09	2	0.35
ecwr	21	24	2	4	0.06	3	0.09	2	0.35
efill	102	136	3	5	4.56	4	3.65	3	44.74
energy	74	89	2	4	1.18	4	0.89	3	10.46
error	30	36	2	3	0.14	3	0.16	3	0.97
exact	6	6	2	2	0.02	2	0.02	2	0.02
fehl	39	45	2	3	0.23	4	0.23	2	1.72
fftb	60	73	2	3	0.80	4	0.64	3	7.98

Table 5: More compiler instances results

				classification	method	max-flow	method	mat-flow	method
Graphs	rows	columns	β	width	time(s)	width	time(s)	width	time(s)
fftf	57	59	2	3	0.67	4	0.58	3	7.09
fmtset	28	55	3	4	0.0	3	0.14	4	1.17
gamgen	20	25	3	4	0.07	3	0.11	3	0.6
genb	18	24	2	4	0.07	3	0.11	3	0.9
genprb	6	7	2	3	0.01	2	0.01	3	0.05
hmoy	6	7	2	3	0.02	2	0.01	3	0.05
ibin	6	7	2	3	0.01	2	0.03	3	0.05
ihbtr	56	75	3	4	0.87	4	0.58	3	9.52
inideb	35	45	3	4	0.25	3	0.27	3	2.26
iniset	465	465	2	2	200.02	2	52.15	2	1858.18
inithx	118	151	3	4	5.81	4	3.33	5	58.72
injbat	51	69	2	3	0.74	4	0.65	4	6.44
injchk	33	44	2	4	0.27	3	0.23	3	2.89
integr	55	70	NP	5	0.71	4	0.53	4	7.18
inter	16	21	2	4	0.05	3	0.09	3	0.37
jacld	25	32	2	4	0.11	3	0.14	4	0.71
jacu	25	32	2	4	0.10	3	0.14	4	0.74
lasden	66	85	NP	5	1.14	4	1.21	5	7.72
laser	19	25	2	4	0.07	4	0.07	3	1.01
laspow	14	18	3	3	0.06	3	0.09	3	0.45
lclear	9	10	2	3	0.02	3	0.02	2	0.07
linj	27	35	NP	4	0.14	4	0.20	4	1.87
lissag	20	22	2	3	0.08	4	0.09	4	0.12
main	34	44	2	3	0.25	3	0.20	3	2.23
nprio	17	22	3	3	0.05	4	0.08	4	0.41
numb	27	32	2	4	0.10	3	0.14	3	0.77
orgpar	56	72	2	4	0.80	4	0.62	3	8.21
paroi	102	134	3	5	4.27	5	2.45	4	63.47
pintgr	90	114	3	4	2.46	4	1.60	4	30.86
prophy	48	65	2	4	0.63	4	0.63	4	6.56
putb	138	175	NP	5	8.45	5	4.34	4	99.41
putdt	53	65	2	4	0.60	4	0.49	3	7.33
radb2	130	158	NP	6	5.76	6	3.45	6	53.08
radb3	113	138	NP	6	3.98	5	2.63	5	42.57
radb4	130	158	NP	6	5.76	6	3.44	6	53.08
radb5	113	138	NP	6	3.97	5	2.63	5	42.48
radf2	130	158	NP	6	5.76	6	3.45	6	53.03

For the instances in Table 3, the max-flow method seems to be the clear winner in terms of

Table 6: More compiler instances results

				classification	method	max-flow	method	mat-flow	method
Graphs	rows	columns	β	width	gap	time(s)	width	gap	time(s)
radf3	113	138	NP	6	3.97	5	2.63	5	42.57
radf4	130	158	NP	6	5.77	6	3.44	6	53.1
radf5	113	138	NP	6	3.97	5	2.64	5	42.57
recre	27	35	NP	4	0.15	4	0.26	4	2.48
rfftb	6	7	2	3	0.02	2	0.01	3	0.05
rfftf1	55	71	2	3	0.79	4	0.71	4	7.41
rfftf	6	7	2	3	0.01	2	0.02	3	0.06
rffti	6	7	2	3	0.02	2	0.01	3	0.06
rhs	202	249	3	5	24.73	5	11.35	4	293.36
rinj	27	35	NP	4	0.15	4	0.22	4	1.83
saturr	17	23	2	4	0.08	3	0.07	3	0.41
saxpy	11	13	2	4	0.03	3	0.02	2	0.12
setb	33	40	2	3	0.20	3	0.22	3	1.53
setbv	39	48	2	4	0.29	4	0.29	3	1.63
setinj	18	23	2	4	0.29	3	0.10	3	0.74
setiv	28	36	2	4	0.15	3	0.16	4	0.82
seval	24	31	NP	4	0.12	3	0.16	4	1.07
si	23	31	3	4	0.14	4	0.15	4	0.68
sinqb	41	48	2	3	0.29	3	0.31	3	1.86
sinqf	41	48	2	3	0.29	3	0.31	3	1.88
solve	29	36	2	4	0.15	4	0.19	3	0.99
sortie	44	62	NP	6	0.55	6	0.41	4	4.59
spline	37	45	3	4	0.25	4	0.26	4	1.7
ssor	81	106	3	5	0.16	6	0.70	3	18.25
sudtbl	11	14	3	4	0.03	3	0.05	3	0.32
supp	12	15	2	3	0.03	2	0.05	3	0.21
tcomp	38	50	NP	6	0.32	5	0.34	4	5.17
tpart	14	18	2	4	0.04	3	0.06	3	.45
trans	73	95	4	5	1.49	5	1.06	4	14.89
urand	18	22	2	3	0.05	3	0.07	3	0.32
vavg	6	7	2	3	0.01	2	0.01	3	.06
verify	55	73	2	4	0.78	3	0.60	3	6.04
vgjyeh	6	7	2	3	0.01	2	0.01	3	0.07
vnewl	9	10	2	3	0.03	3	0.02	3	0.06
x21y21	6	6	2	2	0.00	2	0.02	2	0.02
yeh	50	68	3	6	0.67	5	0.64	5	6.7
zeroin	43	56	NP	4	0.46	5	0.62	4	3.86

width. For 21 of the instances, the max-flow method achieved one of the lowest widths amongst

the three methods followed by the classification method with 18 instances. Unfortunately, for these instances, the mat-flow method did not perform well. We believe the delta ($\frac{1}{3}$) was too big for these type of larger instances. On the other hand, for the instances in Tables 4, 5, and 6, the mat-flow method is the clear winner with a total of 95 instances where the method one of the lowest width or was one away from optimality. The max-flow method came in a close second on performance for these instance with a total of 84 instances where the method either achieved one of the lowest width or was one away from optimality.

These results indicate that all of the heuristics are near-optimal yet there is certainly room to improve the practical performance of the heuristics for the graph instances. This behavior for graphs (especially Table 3) can be attributed to the abstractness of the algorithms; the heuristics do not use any information of the graphs' structure in determining splits. Hence, we would recommend the heuristics of Hicks [8] when dealing with graphs. In contrast, these methods seem effective in finding near-optimal branch decompositions when dealing with the binary matroids associated with the matroids in Table 2 which we believe are not graphic. In addition, the max-flow method seems to be faster than the other methods.

In Step 3 of the k-classification subroutine, we use the eigenvectors corresponding to the generalized eigenvalue problem, which is commonly used in Eigenmaps classification. Hence, one interesting question arising would be what if the standard eigenvectors are used in the classification method (i.e., solving $\mathbf{Lf} = \lambda\mathbf{f}$ in Step 3 of k-classification subroutine, where $\mathbf{L}$ is the graph Laplacian). We investigate this by using the completely same procedure and parameters as the classification method for the linear matroid instances, except that we directly compute the standard eigenvectors of the Laplacian in Step 3 of the k-classification subroutine. We discover that this difference seems not to affect the output of the classification method much for the test instances.

Figure 5 reports the error versus the size of the element set for each planar instance (the upper image) and planar instances (the lower image). The squares in the figures are the results using the generalized eigenvectors, and the circles are the results using the standard eigenvectors. For most of the instances, the two cases produce the same results. There is only a very slight tendency that the generalized eigenvectors beats the standard eigenvectors when the size of the element set is getting larger. The mean relative error of the standard eigenvectors for planar instances is 0.40 (compared with 0.43 for the case of generalized eigenvectors) and that for the compiler instances is 0.59 (compared with 0.60 for the case of generalized eigenvectors). In summary, using standard eigenvectors for Step 3 of the k-classification subroutine is also suggested for the classification method.

10. Conclusions and Future Work

This paper presents the first near-optimal branch decomposition heuristics for linear matroids: the classification method, the max-flow method, and the mat-flow method. In the literature, there are already excellent heuristics for graphs, however, no practical branch decomposition methods for general linear matroids had been addressed. Reforming the linear matroid into a similarity graph, the first two heuristics are able to draw its connections to the classic graph theory as well as the spectral graph theory. The third method uses graph theory to find adequate subsets for separations but is more firmly grounded in matroid theory (with the side effect of a loss in term of speed). Thus, the heuristics are sufficiently supported in theory. Also, these methods can also be utilized to approximate the rankwidth of bipartite graphs. In addition, this work extends a result of Cook and Seymour [7] for any symmetric submodular function and

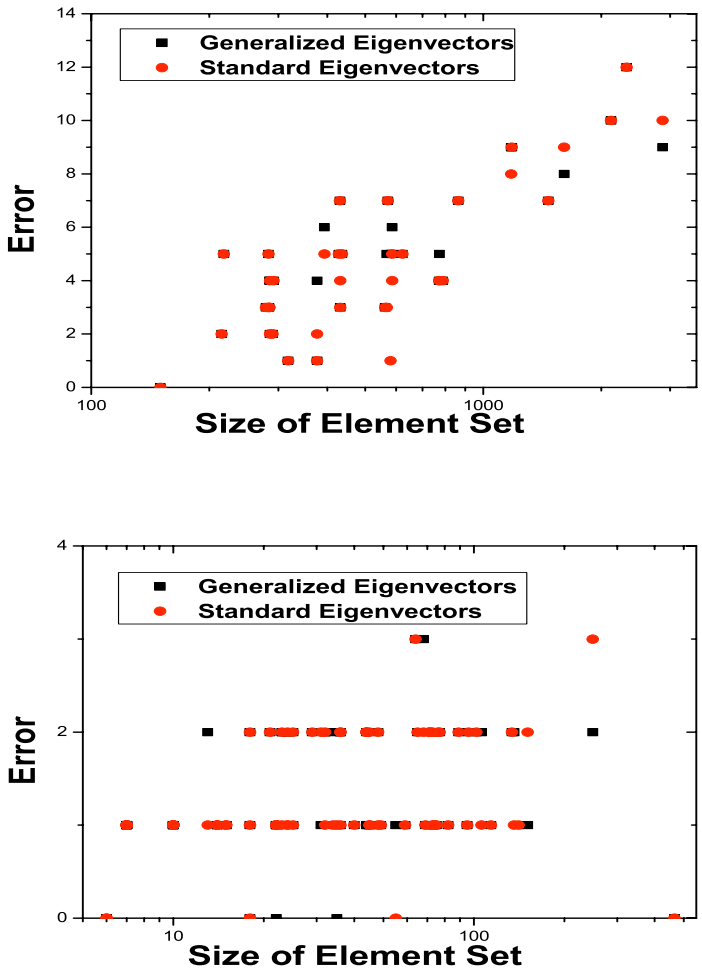

Figure 5: Generalized eigenvector and eigenvector

the result and its corresponding corollary should prove useful for future work in the area (like implementing the Cunningham-Geelen algorithm). In addition, the heuristics seem effective for finding near optimal branch decompositions for small binary matroids and binary matroids whose fundamental graphs are grid graphs. Furthermore, the computational results for the heuristics are respectable for small graphs (up to 400 edges) or graphs with small branchwidth (up to 6). Incorporating these techniques for any symmetric submodular set function is a definite direction for future work.

Acknowledgments

The authors would like to thank the beneficial remarks of the anonymous referees who helped improve the presentation of results, especially the referee who provided a more succinct proof of proving the existence of the vertex s in the proof of Claim 1. The authors were partially supported by National Science Foundation (DMS-0729251 and CMMI-0926618). Also, this work was conducted while Ma was a masters student in the Computational Applied Mathematics Department at Rice and Margulies was a post-doctoral researcher in the same department at Rice under the VIGRE Program (NSF DMS-0739420 and EMSW21-VIGRE).

References

[1] N. Robertson, P. D. Seymour, Graph minors X: Obstructions to tree-decomposition, J. Combin. Theory, Ser. B 52 (1991) 153–190.
[2] M. W. Bern, E. L. Lawler, A. L. Wong, Linear time computation of optimal subgraphs of decomposable graphs, J. Algorithms 8 (1987) 216–235.
[3] B. Courcelle, The monadic second-order logic of graphs I: Recognizable sets of finite graphs, Inform. and Comput. 85 (1990) 12–75.
[4] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, J. Algorithms 12 (1991) 308–340.
[5] R. B. Borie, R. G. Parker, C. A. Tovey, Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families, Algorithmica 7 (1992) 555–581.
[6] P. Seymour, R. Thomas, Call routing and the ratcatcher, Combinatorica 4 (1994) 217–241.
[7] W. Cook, P. D. Seymour, Tour merging via branch decomposition, INFORMS J. Comput. 15 (3) (2003) 233–248.
[8] I. V. Hicks, Branchwidth heuristics, Congressus Numerantium 159 (3) (2002) 31–50.
[9] S. Oum, P. D. Seymour, Approximating clique-width and branch-width, J. Combin. Theory, Ser. B 96 (2006) 514–528.
[10] W. Cunningham, J. Geelen, On integer programming and the branch-width of the constraint matrix, in: Proc. of the 13th Internat. IPCO Conference, Ithaca, NY, USA. June 25-27, 2007, LNCS 4513, 2007, pp. 158–166.
[11] I. V. Hicks, N. B. McMurray, The branchwidth of graphs and their cycle matroids, J. Combin. Theory, Ser. B 97 (2007) 681–692.
[12] F. Mazoit, S. Thomassé, Branchwidth of graphic matroids, in: Surveys in Combinatorics 2007, Cambridge University Press, 2007, pp. 275–286.
[13] S. Oum, Graphs of bounded rank-width, Ph.D. thesis, Princeton University (2005).
[14] S. Oum, Rank-width and vertex-minors, J. Combin. Theory, Ser. B 95 (2005) 79–100.
[15] M. Belkin, P. Niyogi, Using manifold structure for partially labelled classification, in: Adv. in Neural Inform. Processing Systems 15, MIT Press, 2003, pp. 953–960.
[16] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Comput. 15 (6) (2003) 1373–1396.
[17] A. Szlam, M. Maggioni, R. Coifman, Regularization on graphs with function-adapted diffusion processes, J. Machine Learning Res. 9 (2008) 1711–1739.
[18] N. Alon, Eigenvalues and expanders, Combinatorica 6 (2) (1986) 83–96.
[19] F. Chung, Spectral Graph Theory, AMS, Providence, RI, 1997.
[20] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Trans. on Pattern Anal. and Machine Intelligence 22 (8) (2000) 888–905.

[21] H. Bodlaender, A. Koster, Treewidth computations I. upper bounds, Information and Computation 208 (2010) 259–275.
[22] W. T. Tutte, Menger's theorem for matroids, J. Research Natl. Bur. of Stds. Sec. B. Math. and Math. Phys. 69 (1965) 49–53.
[23] R. Diestel, Graph Theory, Springer-Verlag, New York, 2006.
[24] J. G. Oxley, Matroid Theory, Oxford University Press, New York, 1992.
[25] J. Geelen, A. M. H. Gerards, N. Robertson, G. P. Whittle, On the excluded minors for the matroids of branch-width k, J. Combin. Theory, Ser. B 88 (2) (2003) 261–265.
[26] D. Dummit, R. Foote, Abstract Algebra, Wiley, 2003.
[27] W. Cunningham, Improved bounds for matroid partition and intersection algorithms, SIAM J. Comp. 15 (4) (1986) 948–957.
[28] P. Hlinený, On the excluded minors for the matroids of branch-width three, Elec. J. of Comb. 9 (1).
[29] V. Jelinek, The rank-width of a square grid, Discrete Applied Mathematics 158 (7) (2010) 841–850.
[30] I. V. Hicks, Planar branch decompsoitons I: The ratcatcher, INFORMS J. Comput. 17 (4) (2005) 402–412.