All Your bases Are Belong to Us: Listing All Bases of a Matroid by Greedy Exchanges

Arturo Merino
Department of Mathematics, TU Berlin, Germany

Torsten Mütze
Department of Computer Science, University of Warwick, Coventry, UK
Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague, Czech Republic

Aaron Williams
Department of Computer Science, Williams College, Williamstown, MA, UK

Abstract

You provide us with a matroid and an initial base. We say that a subset of the bases "belongs to us" if we can visit each one via a sequence of base exchanges starting from the initial base. It is well-known that "All your base are belong to us". We refine this classic result by showing that it can be done by a simple greedy algorithm. For example, the spanning trees of a graph can be generated by edge exchanges using the following greedy rule: Minimize the larger label of an edge that enters or exits the current spanning tree and which creates a spanning tree that is new (i.e., hasn't been visited already). Amazingly, this works for any graph, for any labeling of its edges, for any initial spanning tree, and regardless of how you choose the edge with the smaller label in each exchange. Furthermore, by maintaining a small amount of information, we can generate each successive spanning tree without storing the previous trees.

In general, for any matroid, we can greedily compute a listing of all its bases matroid such that consecutive bases differ by a base exchange. Our base exchange Gray codes apply a prefix-exchange on a prefix-minor of the matroid, and we can generate these orders using "history-free" iterative algorithms. More specifically, we store $O(m)$ bits of data, and use $O(m)$ time per base assuming $O(1)$ time independence and coindependence oracles.

Our work generalizes and extends a number of previous results. For example, the bases of the uniform matroid are combinations, and they belong to us using homogeneous transpositions via an Eades-McKay style order. Similarly, the spanning trees of fan graphs belong to us via face pivot Gray codes, which extends recent results of Cameron, Grubb, and Sawada [Pivot Gray Codes for the Spanning Trees of a Graph ft. the Fan, COCOON 2021].

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Combinatorics; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Matroids, base exchange, Gray codes, combinatorial generation, greedy algorithms, spanning trees

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.22

Supplementary Material *Software (Source Code)*: `https://gitlab.com/amerinof/greedy-matroids` archived at `swh:1:dir:40f8b84af501e2f96ef57c47d492a12d2072b99f`

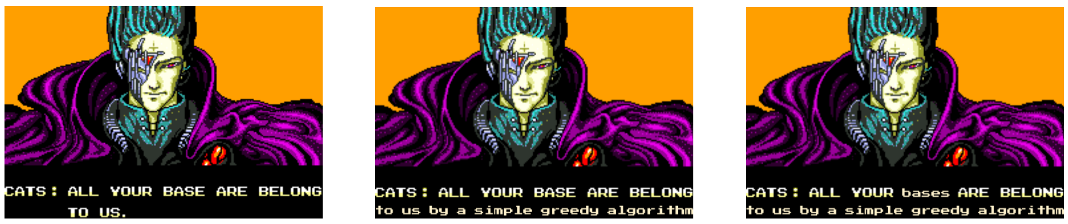
Funding *Arturo Merino*: German Science Foundation grant 413902284 and ANID Becas Chile 2019-72200522.
Torsten Mütze: German Science Foundation grant 413902284 and Czech Science Foundation grant GA 22-15272S.

Acknowledgements The authors would like to thank the anonymous referees who contributed significantly to the quality of this paper.

1 Introduction

Every graph theorist knows that a maximal spanning forest of a graph can be constructed greedily: Add an edge so long as it doesn't create a cycle with some previously added edges. Similarly, every linear algebraist knows that a base for the column space of a matrix can be constructed greedily: Add a column so long as it isn't linearly dependent with some previously added columns. In both cases, the approach works regardless of how the elements (i.e., edges or columns) are ordered. Or put another way, if there are multiple elements that can be added during the next step, then ties can be broken arbitrarily. This property motivated Whitney's [40] generalization to matroids, where bases can be constructed in a similar manner: Add an element so long as it doesn't create a circuit with some previously added elements. Again, this approach works regardless of how ties are broken. Furthermore, it can be used to generate any base.

This greedy algorithm is so powerful, it brings to mind the enemy CATS in the side-scrolling arcade shooter *Zero Wing* (1989) by the Japanese developer Toaplan [42]. In the year 2101, the alien overlord CATS breaks his peace treaty with the United Nations and attacks Earth. This backstory was expanded upon in the opening cutscene of the European port of *Zero Wing* to the SEGA Mega Drive in 1991, where CATS exclaims "`ALL YOUR BASE ARE BELONG TO US`". This poor translation [17] birthed one of the first widespread internet memes [41, 14]. Besides providing comic relief, the poor grammar can also cause confusion for game players. Is CATS saying that the player has one base, and all of that one base belongs to him? Or is CATS saying that the player has multiple bases, and all of them belong to him?[1] In this paper, we hope to create similar confusion with greedy algorithms and matroids, as shown by Figure 1.


CATS : ALL YOUR BASE ARE BELONG TO US.
CATS : ALL YOUR BASE ARE BELONG to us by a simple greedy algorithm
CATS : ALL YOUR bases ARE BELONG to us by a simple greedy algorithm

(a) Cutscene from *Zero Wing* (Mega Drive, 1991).

(b) Generating one base of a matroid.

(c) Generating all bases of a matroid.

Figure 1 An internet meme in (a), along with (b) classic and (c) new results involving greedily generating the bases of a matroid.

To fully appreciate our results, we recommend that the reader is familiar with the basic properties of spanning trees, as analogous concepts involving matroids will be introduced. For additional background reading on matroids, we suggest Oxley [23], while Mütze [21] provides a new survey on Gray codes. We also encourage the reader to (re)familiarize themselves with the *All Your Base Are Belong To Us* meme, including the video by `Bad_CRC` et al. that was originally posted as a Flash animation on Newgrounds in 2001 [2].

[1] The latter is correct according to fandom.com: "[CATS] breaks the treaty and takes over all of Japan's space colonies" [8].

1.1 New results

1.1.1 Matroid bases including spanning trees

We prove that an exhaustive listing of all the bases of a matroid can be generated greedily. Specifically, if you provide us with a matroid and an initial base, then we can list the matroid's bases one at a time as follows: From the current base perform an exchange that creates a new base and minimizes the larger element involved in the exchange. By repeating this simple greedy rule, every base will be generated exactly once. This works for any matroid, for any ordering of its elements, for any initial base, and regardless of how ties are broken to choose the second element in the exchange. In the case of graphic matroids, this greedy rule translates to the following: From the current spanning tree, exchange one edge with another edge in such a way that a new spanning tree is generated and the larger edge in the pair is minimized. We refer to the resulting order as an *edge-exchange Gray code*, or simply an *exchange Gray code*, in reference to the eponymous minimal-change order known as the binary reflected Gray code (see Section 1.3). Figure 2 provides an illustration.

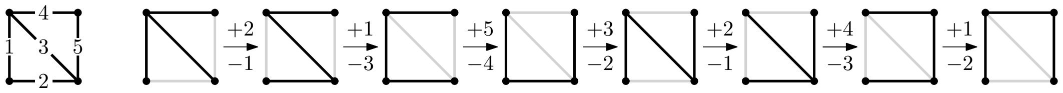


Figure 2 Listing the spanning trees of our favorite graph on the left. The list is an exchange Gray code, i.e., successive spanning trees differ in adding one edge and removing another edge. For example, the first spanning tree is transformed into the second by adding edge 2 and removing edge 1. The list is constructed greedily by always minimizing the larger edge label involved in the exchange. Ties can be broken arbitrarily; here the smaller edge in the exchange is maximized.

1.1.2 Weighted matroids including maximum spanning trees

A *weighted matroid* is a matroid with some weight associated with each element. The weight of a subset of elements is the sum of the weights of the elements in the subset. Edmonds [7] showed that the standard greedy algorithm can be modified to build a maximum weight base: Consider the elements by monotonically decreasing weight and add an element so long as it doesn't create a circuit with some previously added elements. By instead sorting the elements in monotonically increasing order, the standard greedy algorithm will instead build a minimum weight base. These and similar ideas support the well-known algorithms by Kruskal and Prim for finding optimal spanning trees.

The optimization problem on a matroid $M = (E, \mathcal{I})$ with weights $w : E \to \mathbb{R}$ can also be modeled by an unweighted matroid $M' = (E, \mathcal{I}')$: If $S \subseteq E$ is a subset of a maximum weight base M, then $S \in \mathcal{I}'$. In particular, the bases of M' are precisely the maximum weight bases of M. For this reason, our greedy algorithms also allow generating the maximum weight bases in weighted matroids. Furthermore, by replacing the weight function w with $-w$ we are also able to generate *minimum* weight bases in weighted matroid.

1.2 Efficiency and meta-algorithms

Our greedy algorithm is not efficient as stated – it requires exponential space to 'remember' the bases that have previously been created. Fortunately, we are able to completely remove this dependency, and to make our algorithm *history-free*, meaning that the history of previous objects is not stored. In other words, we can determine the next "new" base without storing the "old" bases that have been listed. Specifically, for a matroid $M = (E, \mathcal{I})$ with $m = |E|$ elements, we can generate the bases using just $O(m)$ bits of storage. Furthermore, each successive base can be determined using $O(m)$ calls to the matroid's independence and coindependence oracle (see Section 2.1.5 for a discussion on oracles). Again, these results do not depend on the matroid, the initial base, or the specific manner in which ties are broken (so long as the associated computations do not dominate the time or memory requirements).

Our new greedy algorithm should be also viewed as a *meta-algorithm* that can be specialized into specific algorithms that can be optimized in a variety of ways. For example, by specializing to certain uniform matroids, we provide a new algorithm for generating $(2n, n)$-combinations that runs in *constant amortized time* per generated combination. The algorithm produces an interesting new order of $(2n, n)$-combinations that shares similar properties to the well-known listing by Eades and McKay [6].

1.3 Relationship to previous work

1.3.1 Sublist Gray codes

The most well-known minimal-change orders, or Gray codes, involve listing the set $B(n)$ of n-bit binary strings so that successive strings differ in one bit. In other words, they trace a Hamilton path in the n-dimensional hypercube. In particular, many readers will be familiar with the eponymous *binary reflected Gray code* (*BRGC*) attributed to Frank Gray [9]. This order can be defined recursively as $C_n = 0C_{n-1}, 1\,\mathrm{rev}(C_{n-1})$, where rev denotes reversal. The $n = 4$ order is below.

$$C_4 = 0000, 0001, 0011, 0010, 0110, 0111, 0101, 1101, 1111, 1110, 1010, 1011, 1001, 1000.$$

The BRGC has been the source of many additional minimal-change orders. For example, the k-subsets of the set $[n] := \{1, 2, ..., n\}$ are often referred to as *combinations*, and their characteristic vectors form the set $B_k(n)$, which contains the n-bit binary strings with weight k (i.e., exactly k many 1-bits). If the BRGC is restricted or filtered to only include $B_k(n)$, then successive bitstrings differ by an exchange of a 0 and a 1 [33]. Such an exchange is often referred to as a *transposition*, and this particular transposition Gray code is known as the revolving door Gray code for combinations, since one element leaves the k-set and one element enters the k-set at each step. More broadly, various classes of sublist Gray codes of the BRGC have been studied in the literature [37, 27]. This is also true for the well-known Steinhaus-Johnson-Trotter or *plain change* order of permutations [15] for sublists with a continuous range of inversions [38] or avoiding certain factors or patterns [26] Similarly, the cool-lex Gray code for binary strings [30, 31] can be filtered into a prefix-shift Gray code for combinations [25] or more broadly, a shift Gray code for any bubble language [24], with similar results holding for the cool-lex order of multiset permutations [43, 45, 28]. In contrast, our algorithms are *adaptive* in the sense that they do not produce orders that are sublists of the BRGC or any other particular order. In particular, our meta-algorithm can start at any base. Due to this adaptivity, our algorithms are similar to Algorithm J, which is a greedy algorithm based on "jumps" that has recently been used to generate Gray codes for a large variety of combinatorial objects [10, 11, 12, 20, 4], based on encoding them as permutations.

1.3.2 Hamiltonicity

Previously, it was known that the bases of any matroid can be ordered by exchanges. In graph-theoretic terms, this means that the base exchange graph of a matroid – which contains one vertex per base, and an edge between bases that differ by an exchange – has a Hamilton path. In fact, Holzmann and Harary [13] showed that this graph has a Hamilton cycle including any of its edges, and excluding any of its edges. Furthermore, Naddef and

Pulleyblank [22] proved that the graph is either Hamilton-connected, i.e., it has a Hamilton path between any two end vertices, or it is a hypercube, in which case it has a Hamilton path between any two end vertices from distinct partition classes; see also [1]. The bases of a matroid can also be generated in a non-Gray code manner, without using a single exchange operation in each step. In particular, Uno [36] showed that the bases of a matroid can be generated in $O(m)$ space and the time is dependent on using $O(m)$ independence and contraction and deletion operations on the matroid. In contrast, our implementations do not rely on contracting and deleting elements from the matroid.

1.3.3 Spanning trees and column spaces

The Hamiltonicity of the base exchange graph implies that the spanning trees of a connected graph can be generated by edge exchanges. A specific order was generated efficiently by Smith [29] using $O(n^2)$ space and $O(1)$ amortized delay per tree, where n is the number of vertices of the graph; also see Knuth's detailed coverage in [15, Sec. 7.2.1.6]. Our combinatorial result provides new edge-exchange Gray codes for spanning trees, and several avenues for developing efficient algorithms by selecting specific initial spanning trees and tiebreaker rules.

The second original source for matroids is linear algebra. In this context, the Hamiltonicity of the base exchange graph implies that the bases for the column space of a matrix can be generated by column exchanges. One of our column-exchange Gray codes is illustrated in Figure 3. To our knowledge, there are no previous algorithmic results in this particular case.

$$\begin{array}{c} 1\quad 2\quad 3\quad 4\quad 5 \\ \begin{bmatrix}1&0&0&0&1\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix}\end{array} \qquad \begin{array}{c} 1\quad 2\quad 3 \\ \begin{bmatrix}1&0&0\\1&1&0\\1&0&1\end{bmatrix}\end{array} \xrightarrow[-2]{+4} \begin{array}{c} 1\quad 3\quad 4 \\ \begin{bmatrix}1&0&0\\1&0&1\\1&1&0\end{bmatrix}\end{array} \xrightarrow[-3]{+5} \begin{array}{c} 1\quad 4\quad 5 \\ \begin{bmatrix}1&0&1\\1&1&1\\1&0&0\end{bmatrix}\end{array} \xrightarrow[-1]{+3} \begin{array}{c} 3\quad 4\quad 5 \\ \begin{bmatrix}0&0&1\\0&1&1\\1&0&0\end{bmatrix}\end{array} \xrightarrow[-4]{+2} \begin{array}{c} 2\quad 3\quad 5 \\ \begin{bmatrix}0&0&1\\1&0&1\\0&1&0\end{bmatrix}\end{array} \xrightarrow[-3]{+1} \begin{array}{c} 1\quad 2\quad 5 \\ \begin{bmatrix}1&0&1\\1&1&1\\1&0&0\end{bmatrix}\end{array}$$

Figure 3 Listing the bases of the column space of the full-rank matrix over $GF(2)$ on the left. Notice that the even columns are identical, so they are not in a base together. Similarly, the odd columns are linear combinations of each other, so they do not form a base. The list is a column-exchange Gray code, i.e., successive bases differ in adding one column and removing another column. The list is constructed greedily by always minimizing the larger column label involved in the exchange. Ties can be broken arbitrarily; here the smaller column in the exchange is maximized.

1.4 Outline

Section 2 provides background information on matroids and Gray codes, and Section 3 introduces the prefix-exchange property. Our combinatorial and algorithmic results are in Section 4 and 5, respectively. We sharpen our general results for specific matroids in Section 6. Finally, we conclude with additional remarks in Section 7.

2 Preliminaries

2.1 Matroids

Matroids were introduced by Whitney in 1935 [40]. Originally conceived as a generalization of *independence* in vector spaces, they capture related notions of independence in set systems. We begin by considering the following more general notion.

▶ **Definition 1.** *An* independence system $\mathcal{F}$ *over a finite ground set* E *is a pair* $(E, \mathcal{I})$ *where* $\mathcal{I} \subseteq 2^E$ *satisfies the following two conditions:*

1. $\emptyset \in \mathcal{I}$.
2. $\mathcal{I}$ *is closed under taking subsets. Specifically, if* $Y \in \mathcal{I}$ *and* $X \subseteq Y$*, then* $X \in \mathcal{I}$.

The elements of $\mathcal{I}$ are known as the *independent sets* of $\mathcal{F}$. The subsets of E which are not in $\mathcal{I}$ will be known as *dependent sets.* We highlight that as E is finite, we usually think of it as $E = [m]$ for some m (said another way, we think of E as linearly ordered). In particular, statements like $e > f$ or $e \geq f$ make sense for $e, f \in E$. We also remark that elements of the ground set E will usually be denoted by e or f with $e > f$.

Of interest are typically the *maximal* independent sets. Thus, a *base* in an independence system $\mathcal{F} = (E, \mathcal{I})$ is an independent set which is maximal with respect to inclusion. Before defining matroids in terms of their bases, we introduce some notation: For a set A and an element a we use $A + a$ and $A - a$ as shorthands for $A \cup \{a\}$ and $A - \{a\}$ respectively. We now give the definition of a matroid.

▶ **Definition 2.** *We say that an independence system $M = (E, \mathcal{I})$ is a* matroid *if for every pair T_1, T_2 of bases in M and $e \in T_1 - T_2$ there is an element $f \in T_2 - T_1$ such that both $T_1 - e + f$ and $T_2 + e - f$ are bases.*

This condition is known as the (strong) base exchange property, as one is always allowed to exchange elements from bases while keeping the base property. It is not hard to see that the base exchange property implies that all bases have the same size. Thus, if T is a base, instead of writing "$T + e - f$ is a base" or "$T - e + f$ is a base" we can write the equivalent "$T \Delta \{e, f\}$ is a base" as it is clear that one element must be swapped in and the other out.

If B is a base of $M = (E, \mathcal{I})$ we say that its complement $E - B$ is a *cobase.* Furthermore, if $S \subseteq E$ is contained in *some* cobase, then we will say that S is *coindependent.* Note that, by definition, a cobase is an inclusion-wise maximal coindependent set.[2]

2.1.1 Examples

In this subsection we formally define the example matroids mentioned in Section 1.

Let $n, k \in \mathbb{N}$ such that $n \geq k$. The *uniform matroid* $U(n, k)$ is the matroid with ground set $[n]$ and a subset $S \subseteq [n]$ is independent if and only if its size is at most k. The bases of $U(n, k)$ are the subsets of $[n]$ of size exactly k, its cobases are the subsets of $[n]$ of size exactly $n - k$ and its coindependent sets are the subsets of $[n]$ of size at most $n - k$. If $k = n$ this is also known as the *free matroid* of rank n.

Let $G = (V, E)$ be a graph. The *graphic matroid* $M(G)$, is the matroid with ground set E and a subset $S \subseteq E$ is independent if and only if it is a forest in G. The bases of $M(G)$ are the spanning forests of G, and its coindependent sets are subsets of edges that do not form cuts in G. If G is connected, then the bases of $M(G)$ are the spanning trees of G.

Consider a field $\mathbb{F}$ and a matrix $A \in \mathbb{F}^{m \times n}$. The *column matroid* $M_{\mathbb{F}}(A)$ is the matroid with the columns of A as the ground set and a subset of the columns of A is independent if and only if it is linearly independent over $\mathbb{F}$. The bases of $M_{\mathbb{F}}(A)$ are linearly independent subsets of the columns of A of size $rank(A)$.

Consider a matroid $M = (E, \mathcal{I})$ and a weight function $w : E \to \mathbb{R}$ on the ground set. We say that a base B of M is *w-optimal* if it maximizes $\sum_{e \in B} w(e)$ among all bases of M. The *optimality matroid* M_w is the matroid with E as a ground set and a subset $S \subseteq E$ is independent if and only if it is contained in some w-optimal base. The bases of M_w are exactly the w-optimal bases of M.

[2] It is interesting to note that the coindependent sets form another matroid called the dual matroid.

2.1.2 Base exchange graph

We consider the base exchange graph as originally defined by Maurer [18, 19].

▶ **Definition 3.** *Let $M = (E, \mathcal{I})$ be a matroid. The* base exchange graph *of M is a graph $\mathcal{G}(M)$ with the bases of M as vertices and two bases T, T' are connected by an edge if and only if there exist $e, f \in E$ such that $T' = T\Delta\{e, f\}$. We refer to the edges in the graph $\mathcal{G}(M)$ as* exchanges.

Note that for $e > f$ and the edge $(T, T\Delta\{e, f\})$ there are two possibilities for the directions in which the elements are exchanged; either (1) $e \in T$ and $f \notin T$ in which case the edge is $(T, T - e + f)$ or (2) $e \notin T$ and $f \in T$ in which case the edge is $(T, T + e - f)$.

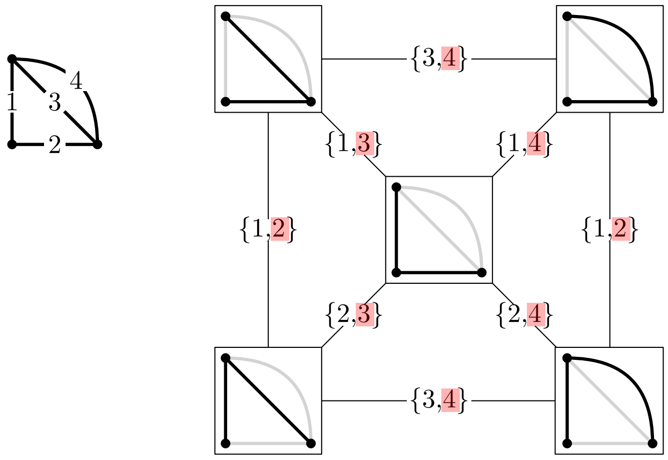


Figure 4 (Left) A graph H with $n = 3$ vertices and $m = 4$ edges. (Right) The base exchange graph $\mathcal{G}(M(H))$. Recall that that the maximum size independent sets are the bases of $M(H)$, which are the spanning trees of H. These spanning trees are the vertices of $\mathcal{G}(M)$, and its edges join spanning trees that differ by an exchange of two edges.

For brevity, we use the term *exchange graph* to refer to $\mathcal{G}(M)$. Observe that the edges of the exchange graph have the form $(T, T\Delta S)$ for some non-empty set $S \subseteq E$. In Figure 4, each edge is labeled with its corresponding set S, and its largest element is highlighted.

As mentioned in Section 1.3, it is known that $\mathcal{G}(M)$ has a Hamilton cycle for all M with at least three bases, and in fact much stronger Hamiltonicity properties are known.

2.1.3 Circuits, cocircuits, loops and coloops

A *circuit* is a minimal dependent set. Unlike bases, which all have the same size, the size of circuit can vary. In particular, an element that forms a circuit by itself is a *loop*. Similarly, a *cocircuit* is a minimal set of elements that intersect with every base. In particular, an element that forms a cocircuit by itself is a *coloop*.

Note that the circuits of $U(n, k)$ are the sets of size $k + 1$, similarly the cocircuits of $U(n, k)$ are the sets of size $n - k + 1$. For a graph G, the circuits of $M(G)$ are exactly the cycles of G. On the other hand, the cocircuits of $M(G)$ are minimal cuts of G and, consequently, the coloops of $M(G)$ are the *bridges* of G.

The following remarks describe the circuits and cocircuits that are formed by adding or removing an element to a base, respectively. Most readers will be familiar with these remarks in the special case of graphic matroids. More specifically, Remark 4 implies that adding an edge to a spanning tree creates a unique cycle, whereas Remark 5 implies that removing an edge from a spanning tree creates a unique minimal cut.

▶ Remark 4. If $M = (E, \mathcal{I})$ is a matroid and T is a base with $e \in T$, then $T + e$ contains a unique circuit $C(T, e)$ called the *fundamental circuit of* $T + e$. Furthermore, $T\Delta\{e, f\}$ is a base, if and only if, f is in this unique circuit.

▶ Remark 5. If $M = (E, \mathcal{I})$ is a matroid and T is a base with $e \notin T$, then $T - e$ contains a unique cocircuit $C^*(T, e)$ called the *fundamental cocircuit of* $T - e$. Furthermore, $T\Delta\{e, f\}$ is a base, if and only if, f is in this unique cocircuit.

2.1.4 Operations

We now define minor operation on matroids (cf. minor operations on graphs) which will be useful to do induction on matroids.

▶ **Definition 6.** *Let $M = (E, \mathcal{I})$ be a matroid and $X \subseteq E$. We define the* deletion of X *as a new matroid $M - X = (E - X, \mathcal{I}')$ where*

$$\mathcal{I}' = \{I \subseteq E - X : I \in \mathcal{I}\}.$$

We now define a dual operation to deletion. Let T be a maximal independent set contained in X. We define the contraction of X *as a new matroid $M/X = (E - X, \mathcal{I}'')$ where*

$$\mathcal{I}'' = \{I \subseteq E - X : I \cup T \in \mathcal{I}\}.$$

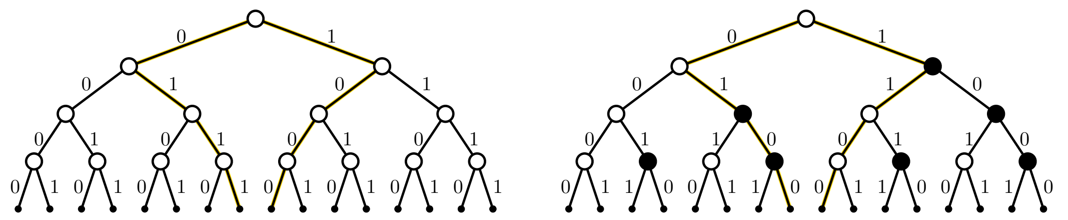
It is well known that the contraction and deletion of X are matroids and the contraction of X does not depend on the choice of maximal independent set $T \subseteq X$. The matroids which can be obtained by a sequence of contractions and deletions are known as the *minors* of M. It is also known that the operations of contraction and deletion are associative and commutative, thus every minor of M can be uniquely written as $M - A/B$ for $A, B \subseteq E$ disjoint.

2.1.5 Algorithmic considerations for matroids

When dealing with computational aspects for arbitrary matroids, it is important to discuss how they are given as an input to algorithms. A naive approach to the input problem is to encode the matroid M by a list of all the elements in E and all the bases of M. This approach is typically avoided because the number of bases can be incredibly large. Furthermore, it does not make much sense for generation algorithms to already have access to all the bases of the matroid, since the problem of generating all of the bases would have already been solved. Thus, we assume only oracle-based access to the matroids, that is, given a set $S \subseteq E$ we can test whether it is independent, coindependent or a base with an independence, coindependence and base oracle respectively. Furthermore, in our algorithmic results we always make explicit which type of oracle is needed and how many calls to them we need.

2.2 Lexico and colexico trees

Figure 5a illustrates a binary tree whose leaves are the binary strings of length $n = 4$ in lexicographic order. More specifically, the root has two children and the branches to these children are labeled 0 and 1. In turn, the branches at the second level fix the second bit to 0 or 1, and so on. Then each labeled path from the root to a leaf spells out a different member of $B(4)$ from the first bit to the last bit. The same approach can be applied for any n, and the resulting tree is the *lexicographic tree* for $B(n)$. By reversing the children of every second node on each level of a lexicographic tree, we obtain a *twisted lexico tree.* This is precisely how the binary reflected Gray code is created, as illustrated in Figure 5b.



(a) Lexicographic tree. The tree orders the strings from 0000 to 1111 in lexicographic order.

(b) Twisted lexico tree. The tree orders the strings from 0000 to 1000 in binary reflected Gray code.

Figure 5 The strings in $B(4)$ ordered according to (a) the lexicographic tree, and (b) the twisted lexico tree. The large white nodes have two children with their branches labeled 0 and 1; the large black nodes reverse the labels to 1 and 0. The consecutive pair of highlighted paths differ in all bits in (a) and only one bit in (b).

More generally, we can construct lexicographic and twisted lexico trees for any subset of $B(n)$, although some internal nodes will have only one child. For example, consider the following set of binary strings, written in lexicographic order,

$$B_f = \{01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010\}. \tag{1}$$

These strings are the incidence vectors of spanning tree edges in our favorite graph from Figure 2. In particular, 10110 corresponds to the edge set $\{1, 3, 4\}$, which is the first spanning tree in 2. The lexicographic tree for B_f is in Figure 6a, and it does not provide an exchange Gray code. More specifically, consecutive binary strings do not always differ by a transposition of a 0 and 1, and so the corresponding spanning trees do not differ by an edge exchange.

It is often more convenient to work with *colexicographic order*, which orders strings from right-to-left instead of left-to-right. For example, B_f is written below in this order,

$$B_f = \{11010, 10110, 01110, 11001, 10101, 01101, 10011, 01011\}. \tag{2}$$

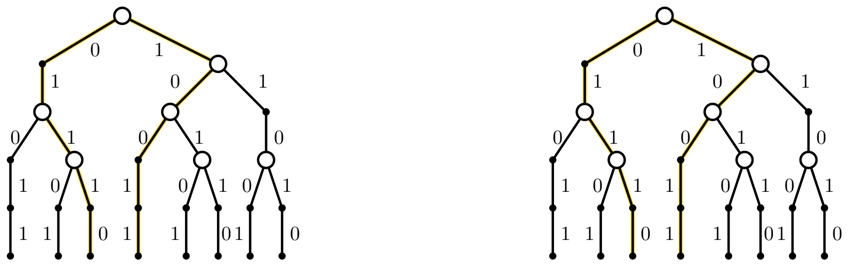
The advantage of colexicographic order comes from the fact that *colexicographic trees* fix the value of the largest element at the top of the tree. For example, the colexicographic tree for B_f appears in Figure 6b, and the branches from the root determine the fifth bit, or edge number 5, in the corresponding spanning tree. This leaves[3] the first four bits, or the edges numbered 1–4, to be determined, which makes for cleaner inductive results.

While twisted lexico trees have been used to generate a variety of Gray codes – see [32] where the lexico terminology originates – this singular approach does not create minimal change orders for matroid bases. Instead we'll need to consider the more broad definition of a *lexico tree* from [32], or more precisely a *colexico tree*, in which *some* of the nodes have reversed children[4]. For example, our exchange Gray code in Figure 2 is backed by a colexico tree, as seen in Figure 7a. Furthermore, if we change the tiebreaker rule from closest to furthest, then another colexico tree is obtained, as seen in Figure 7b. In fact, Remark 7 asserts that something much more general is true.

▶ Remark 7. Every order generated by our generic greedy algorithms has a colexico tree.

Remark 7 should come as somewhat of a surprise. For example, note that the root node in a colexico tree has only two branches. As a result, Remark 7 implies that the largest element changes only once during all of our orders. While our greedy rule always tries to

[3] No pun intended!
[4] Swapping the first and last child of some nodes has also been considered for larger alphabets (see [16]).



(a) Lexicographic tree. The leaves are ordered lexicographically from 01011 to 11010 as in (1).

(b) Colexicographic tree. The leaves are ordered colexicographically from 11010 to 01011 as in (2).

Figure 6 Lexicographic and colexicographic trees for B_f, which contains the edge incidence vectors of the spanning trees of our favorite labeled graph from Figure 2. The trees in (a) and (b) are structurally the same due to the fact that $b_1b_2b_3b_4b_5 \in B_f \iff b_5b_4b_3b_2b_1 \in B_f$, which in turn is due to the labeling used. However, the root-to-leaf paths encode reversed strings with respect to each other. In particular, (a) begins with 01011 which corresponds to the spanning tree $\{2,4,5\}$, while (b) begins with 11010 which corresponds to the spanning tree $\{1,2,4\}$. The pairs of highlighted paths show that these orders are not exchange Gray codes.

minimize the larger element involved in an exchange, there is no immediate reason why it will be involved in only one exchange throughout the order. We'll see that Remark 7 follows from the inductive structure provided in the proof of Theorem 10.

2.2.1 Relabeling

Our main result holds regardless of how the elements are labeled. What does this mean? Before our algorithm is run, the elements of the matroid are given a labeling that has a total order, and then the algorithm creates an order of the bases that has a colexico tree with respect to this labeling. As mentioned earlier, the use of colexicographic order tends to give cleaner inductive results, but otherwise it is not "special" in any particular way.

3 Prefix minors and the prefix exchange property

In this section, we introduce the matroid property that is central to the proper functioning of our greedy algorithm. We also discuss its implications in terms of colexico trees. The property is based on the notion of a prefix minor, which is defined next.

3.1 Prefix minors

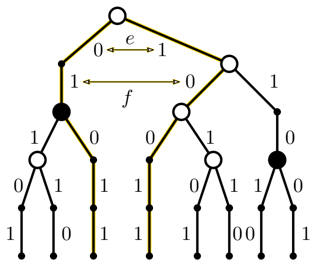
When the ground set of a matroid has a total order, we can consider minors in which some number of the largest elements are deleted or contracted, or equivalently, some number of the smallest elements remain. This leads to the following definition.

▶ **Definition 8.** *$M' = (E', \mathcal{I}')$ is an e-*prefix minor* of $M = (E, \mathcal{I})$ if it is a minor of M and $E' = \{e \in E : f \leq e\}$.*

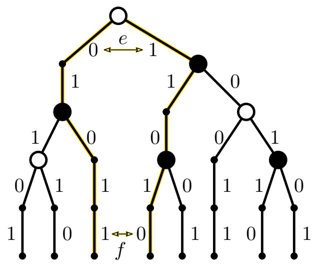
We say that M' is a *prefix minor of M* if M' is an e-prefix minor of M for some $e \in E$.

3.2 Prefix exchange property for bases

The following lemma is a specialization of the standard base exchange property of matroids.



(a) Colexico tree arising from Algorithm $\texttt{G}_{\texttt{close}}$. The tree orders the strings from 10110 to 10011 in the same exchange Gray code seen in Figures 2 and 9. In particular, the spanning trees corresponding to the highlighted paths are $T = \{1, 2, 5\}$ and $T' = T\Delta\{e, f\} = \{1, 2, 4\}$ for $e = 5$ and $f = 4$.



(b) Colexico tree arising from Algorithm $\texttt{G}_{\texttt{far}}$. The tree orders the strings from 10110 to 10101 in another exchange Gray code from the same initial base. In particular, the spanning trees corresponding to the highlighted paths are $T = \{1, 2, 4\}$ and $T' = T\Delta\{e, f\} = \{2, 4, 5\}$ for $e = 5$ and $f = 1$.

Figure 7 The incidence vectors of edges in the spanning trees of our favorite graph are found in (1) (and also (2)). These binary strings are ordered above by two different colexico trees. The branches at the roots determine the inclusion or exclusion of edge 5 (not edge 1). The large nodes are as in Figure 5, and the remaining internal nodes have one child. The consecutive pair of highlighted paths differ in only two bits are changed and the larger is $e = 5$. The algorithm generating (a) selects the smaller change $f = 4$ to be as high up the tree as possible, whereas the algorithm generating (b) selects $f = 1$ to be as low in the tree as possible.

▶ **Lemma 9** (Prefix exchange property for matroid bases). *Let M' be an e-prefix minor of M such that e is not a loop or coloop in M'. For every base T of M' there exists $f < e$ such that $T\Delta\{e, f\}$ is a base of M'.*

Proof. Let $M' = (E', \mathcal{I})$ be an e-prefix minor of $M = (E, \mathcal{I})$, and suppose that e is not a loop or coloop in M'. We proceed in two cases.

- Case 1: $T \subseteq E'$ is a base of M' with $e \in T$. Since e is not a coloop in M', there is a base $T' \subseteq E'$ of M' with $e \notin T'$. Since $e \in T - T'$, the base exchange property implies that there exists an $f \in T' - T$ such that $T\Delta\{e, f\}$ is a base of M'. Since e is the largest element in E', we know that $f < e$. Hence, the result holds.
- Case 2: $T \subseteq E'$ is a base of M' with $e \notin T$. A similar argument proves this case. ◀

We now illustrate Lemma 9 for two matroids. In the case of a uniform matroid every prefix minor is also a uniform matroid. Consider a specific uniform matroid $M' = U(8, 5)$, which is an 8-prefix minor of $M = U(14, 7)$. Let $T = \{1, 2, 4, 5, 8\}$ be a base of M'. Since $e = 8$, we can finish the exchange by adding any $f \in [8] - T = \{3, 6, 7\}$. On the other hand, if $T = \{1, 2, 4, 6, 7\}$ and $e = 8$ as before, we can finish the exchange by removing any $f \in T$. For a graphic matroid, once again every prefix minor is also a graphic matroid. So, if M' is the graphic matroid of some graph and e is the largest labelled edge, then if $e \notin T$ the exchange can be completed by removing a smaller labelled edge f from the unique cycle in $T + e$, and if $e \in T$ the exchange can be completed by removing a smaller labelled edge f from the unique minimal cut in $T - e$; such an f exists since e is neither a loop nor coloop.

3.3 Tree interpretation

It is instructive to consider the implications of the prefix exchange property in terms of colexico trees. The condition that e is not a loop or coloop in M' implies that there are two subtrees under the node corresponding to M' in any colexico tree of the bases of the matroid M. We don't know which of these two children is first or second in the colexico tree, but we do know that both subtrees contain at least one leaf (which corresponds to a base in M).

The rest of the property implies the following: For every leaf in either subtree, there is a base exchange that results in some leaf of the other subtree. This turns out to be a very strong and helpful property in the context of our greedy algorithm. As our algorithm is building a colexico tree, there will always be an exchange that moves from the last leaf within the first subtree to some leaf within the second subtree; the specific details of which subtree is first or second, and which leaf is last within the first subtree simply do not matter. In other words, our algorithm can never get stuck; this will be formalized in Theorem 10.

This interpretation of the prefix exchange property brings to mind the ZIG ship in *Zero Wing*. Regardless of how "`SOMEBODY SET UP US THE BOMB`" [17], there is always a ZIG that can escape and continue the mission to catch CATS. This connection is illustrated in Figure 8.

4 Combinatorial result: Hamilton paths

We now describe our main result: the following simple greedy algorithm generates all the bases of a matroid in a Gray code order.

Algorithm `G` *(Greedy bases).* This algorithm attempts to greedily generate the bases of a matroid M starting from an initial base T_0.

G1. [Initialize] Visit the initial base T_0.

G2. [Greedy] Generate an unvisited base of M by performing *any* exchange whose larger element is as small as possible. If no such exchange exists, then terminate. Otherwise visit the resulting base and repeat G2.

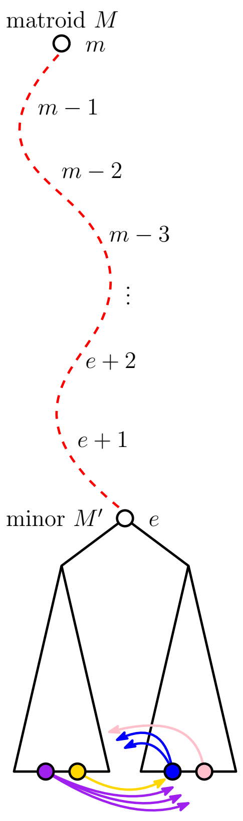
We add the following four notes on Algorithm $\mathtt{G}(M, T_0)$.

1. *Efficiency.* It is *not* efficient as presented. This is because the previously visited bases must be maintained. In Section 5, we present several history-free implementations.
2. *Tiebreakers.* It can proceed in many different ways depending on the choices made for "any exchange". This is discussed in more detail in Section 4.1.
3. *Restricted operations.* In many cases, it is possible to prove that the algorithm operates using only a small subset of the exchanges that it would use in general. Several examples of this appear in Section 6.
4. *Starting base.* Even though the algorithm succeeds *independently* of the initial base, choosing a particular initial base may lead to nicer and/or stronger properties for the listings produced. Examples of this appear in Section 6.

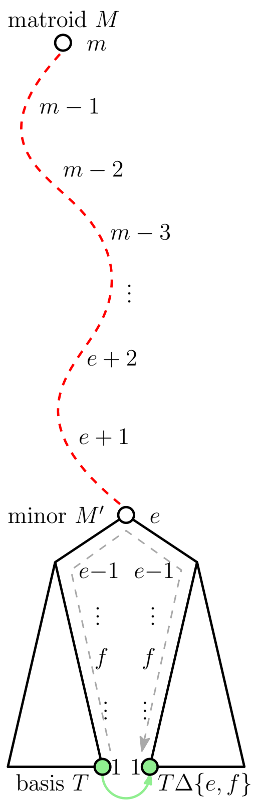
Now we prove that the algorithm works as intended.

▶ **Theorem 10.** *If $M = (E, \mathcal{I})$ is a matroid and T is a base of M, then $\mathtt{G}(M, T)$ always provides a Hamilton path of $\mathcal{G}(M)$. In other words, Algorithm `G` creates an exchange Gray code for the bases of M starting from any initial base T, regardless of how ties are broken.*

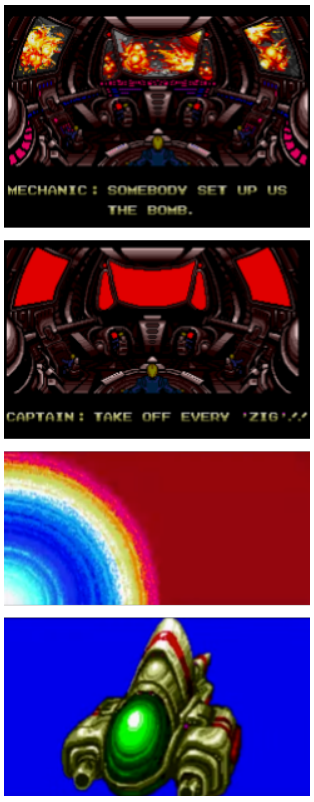
Proof. We prove the statement of the theorem by induction on the number of elements, $|E|$. If $|E| = 1$, then there is at most one base, and so the base case holds. Otherwise, suppose that the statement holds for all matroids with k elements and all initial bases. Now consider a matroid $M = (E, \mathcal{I})$ with $k + 1 = |E|$ elements, and an initial base T. Without loss of generality, label the elements in M as $E = [k + 1]$. Let $M_1 = M/e$ and $M_2 = M - e$ be the two prefix minors with respect to the largest element $e = k + 1 \in E$. Finally, let m, m_1, and m_2 be the number of bases in M, M_1, and M_2, respectively. Note that m_1 and m_2 are the number of bases of M that do and do not contain e, respectively. We proceed in two cases.



(a) The prefix-exchange property ensures that there is at least one exchange from each leaf in either subtree to a leaf in the other subtree. For example, the blue arrows are meant to indicate that there are two different exchanges from the blue leaf in the second subtree to some leaves in the first subtree. Similarly, there is one exchange from the gold leaf that moves from the first subtree to the second subtree.



(b) Algorithm `G` escapes the first subtree via an exchange on the last leaf in the first subtree, which is labeled as base T. Visually, `G` goes up the colexico tree to e, which is the lowest node with an unvisited branch. Then it switches branches and goes down the same branches it went up, except for switching at f to complete the exchange. This gives a new base $T' = T\Delta\{e,f\}$, which becomes the first leaf in the second subtree.


MECHANIC: SOMEBODY SET UP US THE BOMB.

CAPTAIN: TAKE OFF EVERY 'ZIG'!!

(c) No matter how many times *Zero Wing* is played, there will always be a ZIG that escapes the bomb. This ZIG continues the pursuit of CATS. Thus, the meme never ends – just as Algorithm `G` can never get stuck.

Figure 8 The prefix exchange property for a matroid $M = (E, \mathcal{I})$ with $E = [m]$. The red path illustrates a sequence of contractions or deletions of $m, m-1, \ldots, e+1$ that result in an e-prefix minor M' in which e is neither a loop nor a co-loop. Hence, there are two non-empty subtrees below the node labeled e: one subtree for bases that include e, and the other subtree for bases that exclude e. An exchange can always "escape" the first subtree, just as a ZIG can always escape the exploding ship in *Zero Wing*.

- Case 1: $e \in T$. Observe that $T_1 = T - e$ is a base of M_1. Since M_1 has only k elements, we know by induction that $\mathtt{G}(M_1, T_1)$ creates an exchange Gray code for all m_1 bases of M_1 starting at T_1, regardless of how ties are broken. This also implies that $\mathtt{G}(M, T)$ will begin by generating an exchange Gray code for its bases that include e. More precisely, $\mathtt{G}(M, T)$ can begin by generating an ordered list of m_1 bases of M starting from T, if and only if, $\mathtt{G}(M_1, T_1)$ generates the same list but with e removed from each base. This is due to the fact that algorithm G always minimizes the maximum element involved in an exchange, and hence, $\mathtt{G}(M, T)$ will not use an exchange involving e until it has no other options (and by induction this happens only after every base including e has been generated). If $m_2 = 0$, then $\mathtt{G}(M, T)$ has successfully generated every one of its bases, and the induction is complete. For the remainder of the proof, we assume that $m_2 > 0$, and this implies that e is neither a loop nor a co-loop in M.
  Let T_1' be an arbitrary last base generated by $\mathtt{G}(M_1, T_1)$. Hence, $T' = T_1' + e$ is an arbitrary last base of M that contains element e. By the prefix exchange property, there is a base of M that differs from T' by an exchange involving e and a smaller element f. That is, there exists $T_2 = T' \Delta \{e, f\}$ with $f < e$. Therefore, algorithm $\mathtt{G}(M, T)$ will be able to continue generating bases by making an exchange to some such base T_2.
  Since T_2 is a base of M that does not contain element e, we know that it is a base of M_2. Hence, by induction, $\mathtt{G}(M_2, T_2)$ creates an exchange Gray code for all m_2 bases of M_2 starting at T_2, regardless of how ties are broken. This also implies that $\mathtt{G}(M, T)$ will end by generating an exchange Gray code for its bases that do not include e. More precisely, $\mathtt{G}(M, T)$ can end by generating an ordered list of m_2 bases of M starting from T', if and only if, $\mathtt{G}(M_2, T_2)$ generates the same list of bases. Again, this is due to G minimizing the maximum element involved in an exchange.
  The previous three paragraphs have shown that $\mathtt{G}(M, T)$ generates an exchange Gray code for all of its m_1 bases including e, followed by an exchange to some base T' that does not include e, followed by an exchange Gray code for all of its m_2 bases that do not include e. Hence, the inductive statement is true.
- Case 2: $e \notin T$. This case is nearly identical to the previous case by dual arguments. ◀

4.1 Tiebreaker rules

Theorem 10 proves that Algorithms G is always successful in generating minimal change orders starting from any base. However, the specific orders that they generate will depend on how ties are broken. These tiebreaker choices can affect how efficiently the resulting order can be generated or ranked and unranked, and even the specific types of operations that are used. For example, in Section 6.1 we'll see that one tiebreaker rule causes Algorithm G to use homogeneous transpositions, while another causes it to use consistent transpositions.

While tiebreaker rules can be quite creative, they must adhere to some basic principles. For example, if the larger element e involved in an exchange is entering (exiting) the base, then the smaller element f must be exiting (entering) to satisfy Theorem 10. This observation is further refined below according to Remarks 4–5.

▶ Remark 11. If Algorithm $\mathtt{G}(M, T)$ performs an exchange on base T' and e is the larger element in the exchange, then its smaller element f must satisfy the following:
- If $e \notin T'$, then f is in the fundamental circuit of $T + e$.
- If $e \in T'$, then f is in the fundamental cocircuit of $T - e$.

The existence of such an f is guaranteed by Lemma 9.

This leads us to the following pair of simple and fundamental tiebreaker rules in Sections 4.1.1–4.1.2. See Figures 7a–7b for an example of how they differ.

4.1.1 Maximum/closest tiebreaker rule

The following tiebreaker rule is *generic* in the sense that it can be applied to any matroid without any additional information. When applied to matroid bases, it simply maximizes the smaller element involved in an exchange.

Algorithm $\texttt{G}_{\texttt{close}}$ *(Greedy bases with maximum tiebreaker).* This algorithm greedily generates the bases of a matroid M starting from an initial base T_0.

C1. [Initialize] Visit the initial base T_0.

C2. [Greedy] Generate an unvisited base of M by performing the exchange whose larger element is minimized, and then breaking ties by maximizing the smaller element. If no such exchange exists, then terminate. Otherwise visit the resulting independent set and repeat C2.

We note that the term "maximizing the smaller element" is in reference to the set S in an edge of the form $(T, T\Delta S)$ in the underlying exchange graph. More specifically, the larger element of such an edge is $\max(S)$, and the smaller element is $\min(S)$. Seen another way, the tiebreaker rule chooses the smaller element to be as close as possible to the larger element, hence the terminology *maximum/closest tiebreaker rule*.

4.1.2 Minimum/furthest tiebreaker rule

The following tiebreaker rule is opposite to the previous, in the sense that it minimizes the smaller element involved in an exchange. Since it doesn't depend on the particular matroid, we also refer to it as a generic tiebreaker rule.

Algorithm $\texttt{G}_{\texttt{far}}$ *(Greedy bases with minimum tiebreaker).* This algorithm is identical to Algorithm $\texttt{G}_{\texttt{close}}$, but with *minimizing the smaller element.*

This tiebreaker rule chooses the smaller element to be as far away as possible from the larger element, hence the terminology *minimum/furthest tiebreaker rule.*

5 Algorithmic results

In this section, we provide history-free specializations of the meta-algorithm from Section 4. In other words, we can remove the "which hasn't appeared earlier" part of Algorithm $\texttt{G}$ by maintaining additional data structures. More specifically, we give history-free implementations of Algorithms $\texttt{G}_{\texttt{close}}$ and $\texttt{G}_{\texttt{far}}$. The implementations works for any matroid and initial base.

5.1 History-free implementation

We now present history-free iterative implementations of our generic greedy algorithm. To do so, the implementations maintain an additional data structure, like an array or a stack, that allows them to implicitly navigate the colexico tree, as in Figure 9. In particular, these algorithms do not store minors of the matroid (cf. [36]).

5.1.1 Active array

Let $M = ([m], \mathcal{I})$ be a matroid and T_0 an initial base of M. In this subsection we show how to implement $\texttt{G}_{\texttt{close}}(M, T_0)$ with an active array. Let T be a base of M and recall that T is a leaf on the colexico tree of $\texttt{G}_{\texttt{close}}(M, T_0)$. We also introduce the notation $P(T)$ for the path between T and the root of the colexico tree.

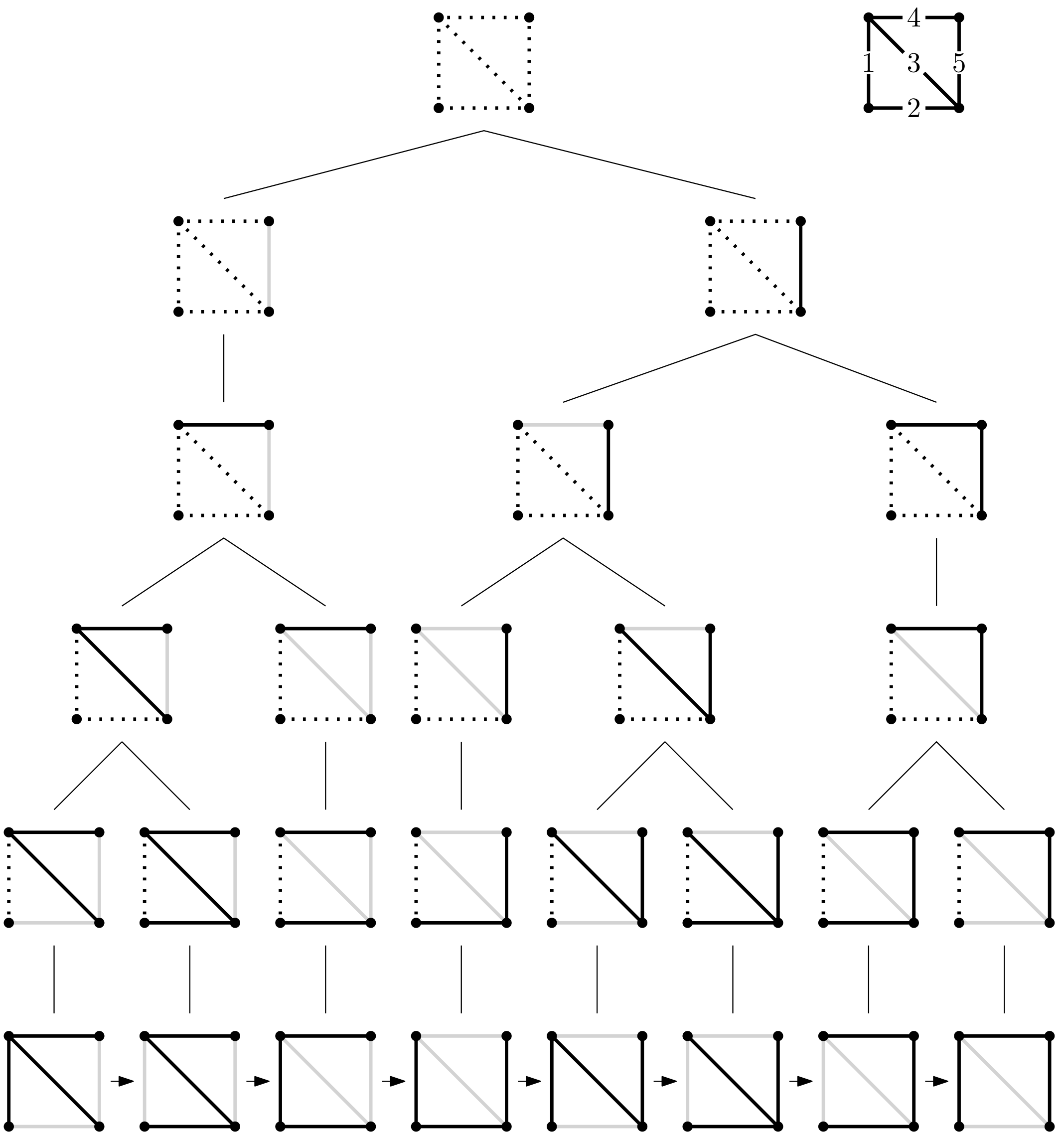


Figure 9 A colexico tree of our favorite graph. More specifically, it is the colexico tree generated by algorithm $\mathtt{G}_{\mathtt{close}}$, which is the specialization of greedy algorithm $\mathtt{G}$ using the maximum tiebreaker rule. Observe that the bottom row provides the exchange Gray code for the spanning trees of our favorite graph seen in Figure 2. The same tree with incidence vectors is in Figure 7a.

The main idea behind the active array approach is that whenever we generate T, we also store $A(T)$ which contains $i \in [m]$ if and only if the i-th edge on $P(T)$ (counting from T towards the root) was the leftmost (i.e. first) branch on the colexico tree. Said another way, $A(T)$ stores the potential directions we could follow when traversing the lexico tree. Note that initially $A(T_0) = [m]$ as every edge on $P(T_0)$ was a first branch. Furthermore, if we are given the state $(T, A(T))$ we can compute the next state $(T', A(T'))$ as follows:

1. Computing T'. Let $e \in A(T)$ be such that there exists $f \leq e$ and $T\Delta\{e, f\}$ is a base. Moreover, if there are multiple choices for f we pick the maximum possible ones due to the tie-breaker rule. Note that $T\Delta\{e, f\}$ is the next base as e is the minimum larger exchange possible. This can be computed by checking whether $T\Delta\{e, f\}$ is a basis for all pairs of possible e's and f's.

2. Computing $A(T')$. Note that the path only changed for elements smaller than e. We observe that e is now in its second branch, so it should not appear on $A(T')$. Furthermore, as we just moved to the second branch labelled by e, this implies that everything smaller than e is in its first branch, so it should appear in $A(T')$. This implies that

$$A(T') = [e-1] \cup \{g \in A(T) : g > e\}.$$

These observations translate into the pseudocode in Algorithm 1.

Algorithm 1 $\texttt{G}_{\texttt{close}}(M, T_0)$.

```
    Input: A matroid $M = ([m], \mathcal{I})$ and an initial base $T_0$.
 1: $T \leftarrow T_0$
 2: $A \leftarrow [m]$
 3: Visit $T$
 4: repeat
 5:   visited ← False
 6:   for $e = 2, \ldots, m$ do
 7:     if $e \in A$ then
 8:       for $f = e-1, \ldots, 1$ do
 9:         if $T\Delta\{e,f\}$ is a base and not visited then
10:           $T \leftarrow T\Delta\{e,f\}$
11:           $A \leftarrow [e-1] \cup \{a \in A : a > e\}$
12:           Visit $T$
13:           visited ← True
14: until not visited
```

We make the following observations on the pseudocode implementation.

- The tiebreaker rule only comes into play in Line 8; this for loop is done in a descending way as to prioritize the f which is closest to e. In particular, if one would like to implement the furthest tiebreak, one only needs to do this for loop in an ascending manner. In fact, any (computable) tiebreaker rule can be implemented by following a similar scheme.
- This implementation of $\texttt{G}_{\texttt{close}}$ requires $O(m^2)$ calls to the base oracle. Additionally, we can replace every call to the base oracle for a set S to checking if $|S| = |T_0|$ and S is independent. Thus, this implementation of $\texttt{G}_{\texttt{close}}$ requires $O(m^2)$ calls to the independence oracle. Moreover, we only need to keep track of the state $(T, A(T))$ and the counters, obtaining an algorithm which uses $O(m)$ space.

Further implementation details can be seen in the Appendix.

5.1.2 Stack

In this subsection we provide a stack implementation of $\texttt{G}_{\texttt{far}}(M, T)$. The main idea behind the stack approach is very similar to the active vector one: Whenever we generate T, we store alongside $S(T)$ which contains $i \in [m]$ if and only if the edge labelled i on $P(T)$ has an unvisited second branch the colexico tree. Said another way $S(T)$ stores the unvisited directions when traversing the lexico tree.

We now characterize when a node has two children in the colexico tree. To this end, we define the following: (1) T^* is the complement of the base T; (2) for $e \in [m]$ the partial base $\partial_e T$ and cobase $\partial_e T^*$ are the “suffix” of size $m - e$ of T and T^* respectively. More formally,

$$\partial_e T := T \cap \{e+1, \ldots, m\}, \qquad \partial_e T^* = T^* \cap \{e+1, \ldots, m\}.$$

Recall that every node on the colexico tree has a prefix minor associated to it. Furthermore, if M' is an e-prefix minor whose node associated lies on $P(T)$ we have that $M' = M - \partial_e T^* / \partial_e T$. The key observation here, is that the node associated to M' has two children if and only if $\partial_e T + e$ is independent and $\partial_e T^* + e$ is coindependent. Since M' has two children if and only if both taking element e and not taking e lead to bases, said another way $\partial_e T + e$ can be extended to a base and $\partial_e T^* + e$ can be extended to a cobase.

With this observation, we can compute the state $(T', S(T'))$ that follows $(T, S(T))$.

1. Computing T'. Let $e \in S(T)$ be the smallest element in the stack. We now consider the set $T\Delta\{e\}$, this has either a fundamental circuit or cocircuit. In particular, any $f \leq e$ which lies on the fundamental circuit or cocircuit will form a possible base (this exists as there was a branching on this node of the colexico tree). Moreover, if there are multiple choices for f we pick the minimum possible one due to the tie-breaker rule. Note that $T\Delta\{e, f\}$ is the next base as e is the minimum larger exchange possible.
2. Computing $S(T')$. Note that the path only changed for elements smaller that e. Thus, the only elements which can enter the stack are smaller than e. In view of this, we update the stack by checking if for every element $f \leq e$ the partial bases and cobases $\partial_f T'$ and $\partial_f T'^*$ are independent and coindependent respectively.
3. Delay. Note that if $\partial_f T'$ is *not* independent, f is forced to be out of the next base in order to actually reach a leaf. Similarly, whenever $\partial_f T'$ is *not* coindependent, f is forced to be inside of the next base in order to actually reach a leaf. Perhaps surprisingly, whenever we are not forced we do not take a choice and delay the decision to complete the exchange at a later point (See 5.1.3 for details on why this works).

These observations translate into the pseudocode in Algorithm 2.

Algorithm 2 $\texttt{G}_{\texttt{far}}(M, T_0)$.

```
Input: A matroid $M = ([m], \mathcal{I})$ and an initial base $T_0$.
1: $T \leftarrow T^0$
2: $S \leftarrow \emptyset$
3: for $e = m, \ldots, 1$ do
4:   if $\partial_e T + e \in \mathcal{I}$ and $\partial_e T^* + e \in \mathcal{I}^*$ then
5:     $S$.push($e$)
6: Visit $T$
7: while $S \neq \emptyset$ do
8:   $e \leftarrow S$.pop()
9:   for $f = e - 1, \ldots, 1$ do
10:    if $\partial_f T + f$ is independent and $\partial_f T^* + f$ is coindependent then
11:      $S$.push($f$)
12:    else if $\partial_f T + f$ is independent and $f \notin T$ then
13:      $T \leftarrow T\Delta\{f\}$
14:      $T^* \leftarrow T^*\Delta\{f\}$
15:    else if $\partial_f T^* + f$ is coindependent and $f \in T$ then
16:      $T \leftarrow T\Delta\{f\}$
17:      $T^* \leftarrow T^*\Delta\{f\}$
18:  Visit $T$
```

We make the following observations on the pseudocode implementation.

- The tiebreaker of $\texttt{G}_{\texttt{far}}$ comes into play with the delay strategy. More specifically, we always delay the decision of picking f until it gets forced on us. Note that it is not hard to replace this tiebreaker with another by adding some conditions to select f.

Figure 10 The implementation of the minimum/furthest tiebreaker rule seems dangerous: Is it safe to delay choosing f until we are forced to do so? Don't worry, "`WE KNOW WHAT WE DOING`".

- This implementation of $\texttt{G}_{\texttt{far}}$ requires $O(m)$ calls to the independence and coindependence oracles. Moreover, we only need to keep track of the state $(T, S(T))$ and the counters, obtaining an algorithm which uses $O(m)$ space.

Further implementation details can be seen in the Appendix.

5.1.3 Delay and the furthest tiebreaker rule

Recall that the furthest tiebreaker rule delays switching branches until it is forced to do so. For example, the smaller change occurs at the very bottom of the colexico tree in Figure 7b.

At first, this approach to implementing the furthest tiebreaker rule may seem to be dangerous. *How do we know that there will be a smaller element f that is forced? How do we know that the remaining smaller values can be chosen without any additional changes?* This approach works due to Remarks 4–5. More specifically, the minimum smaller element f is precisely the smallest element on the fundamental circuit $C(T, e)$ or cocircut $C^*(T, e)$ created by adding or removing the larger element e. If we have not changed the other elements on this circuit or cocircuit, then this f will be a loop or coloop, respectively, and so the algorithm will be forced into choosing it to be the smaller element in the exchange. In the case of Figure 7b, the first highlighted base is the spanning tree with edges $\{1, 2, 4\}$ in the graph found in Figure 2; adding edge $f = 5$ at the top of the colexicotree creates the fundamental cycle 1234 whose smallest element is $e = 1$ at the bottom of the colexico tree. The spirit behind this technique is illustrated in Figure 10.

5.1.4 Example

We illustrate the execution of the iterative algorithms using the *Vámos matroid.* First described in an unpublished manuscript of Peter Vámos [39], the Vámos matroid $\mathcal{V} = ([8], \mathcal{I})$ has $[8]$ as a ground set and its bases are all subsets of size four *except* $\{1, 2, 3, 4\}$, $\{1, 2, 5, 6\}, \{1, 2, 7, 8\}, \{3, 4, 5, 6\}$ and $\{3, 4, 7, 8\}$ which are marked as faces in Figure 11. Thus, it has $\binom{8}{4} - 5 = 65$ bases. The remaining independent sets are all subsets of size at most three. The Vámos matroid is a very interesting example to run our algorithms, as it is not a column matroid for any field. We show the run of $\texttt{G}_{\texttt{close}}(\mathcal{V}, \{1, 2, 3, 5\})$ on Table 1.

5.2 Analysis

In this subsection we provide running times for specific matroids, without referencing oracle calls. We are particularly interested in the *delay* of the algorithm, that is, the time it takes to visit the next base.

We highlight that the algorithms only need to check for *incremental independence* and *incremental coindependence.* Specifically, if we are given an independent (resp. coindependent) set $S \subseteq E$, we must be able to check whether $S + s$ is independent (resp. coindependent) for

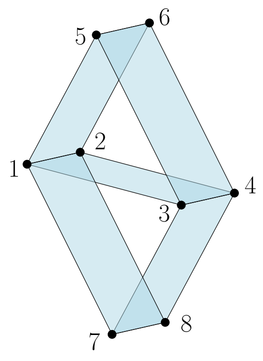


Figure 11 A representation of the five circuits of size four in the Vámos Matroid. Each of $\{1,2,3,4\}$, $\{1,2,5,6\}$, $\{1,2,7,8\}$, $\{3,4,5,6\}$, $\{3,4,7,8\}$ appears as a face in the figure.

Table 1 Vámos matroid bases with variable traces from the array and stack implementations.

#	base T	Active array $A(T)$	Stack $S(T)$
1	1, 2, 3, 5	1, 2, 3, 4, 5, 6, 7, 8	8, 7, 6, 4
2	1, 2, 4, 5	1, 2, 3, 5, 6, 7, 8	8, 7, 6, 3
3	1, 3, 4, 5	1, 2, 5, 6, 7, 8	8, 7, 6, 2
4	2, 3, 4, 5	1, 5, 6, 7, 8	8, 7, 6
5	2, 3, 4, 6	1, 2, 3, 4, 5, 7, 8	8, 7, 5, 4, 3, 2
6	1, 3, 4, 6	1, 3, 4, 5, 7, 8	8, 7, 5, 4, 3
7	1, 2, 4, 6	1, 2, 4, 5, 7, 8	8, 7, 5, 4
8	1, 2, 3, 6	1, 2, 3, 5, 7, 8	8, 7, 5
9	1, 3, 5, 6	1, 2, 3, 4, 7, 8	8, 7, 4, 2
10	2, 3, 5, 6	1, 3, 4, 7, 8	8, 7, 4
11	2, 4, 5, 6	1, 2, 3, 7, 8	8, 7, 2
12	1, 4, 5, 6	1, 3, 7, 8	8, 7
13	1, 4, 5, 7	1, 2, 3, 4, 5, 6, 8	8, 6, 5, 4, 3, 2
14	2, 4, 5, 7	1, 3, 4, 5, 6, 8	8, 6, 5, 4, 3
15	3, 4, 5, 7	1, 2, 4, 5, 6, 8	8, 6, 5, 4
16	2, 3, 5, 7	1, 2, 3, 5, 6, 8	8, 6, 5, 3, 2
17	1, 3, 5, 7	1, 3, 5, 6, 8	8, 6, 5, 3
18	1, 2, 5, 7	1, 2, 5, 6, 8	8, 6, 5
19	1, 2, 4, 7	1, 2, 3, 4, 6, 8	8, 6, 4, 3
20	1, 3, 4, 7	1, 2, 4, 6, 8	8, 6, 4, 2
21	2, 3, 4, 7	1, 4, 6, 8	8, 6, 4
22	1, 2, 3, 7	1, 2, 3, 6, 8	8, 6
23	1, 2, 6, 7	1, 2, 3, 4, 5, 8	8, 5, 4, 3
24	1, 3, 6, 7	1, 2, 4, 5, 8	8, 5, 4, 2
25	2, 3, 6, 7	1, 4, 5, 8	8, 5, 4
26	2, 4, 6, 7	1, 2, 3, 5, 8	8, 5, 3, 2
27	1, 4, 6, 7	1, 3, 5, 8	8, 5, 3
28	3, 4, 6, 7	1, 2, 5, 8	8, 5
29	3, 5, 6, 7	1, 2, 3, 4, 8	8, 4, 3
30	2, 5, 6, 7	1, 2, 4, 8	8, 4, 2
31	1, 5, 6, 7	1, 4, 8	8, 4
32	4, 5, 6, 7	1, 2, 3, 8	8
33	4, 5, 6, 8	1, 2, 3, 4, 5, 6, 7	7, 6, 5, 4
34	3, 5, 6, 8	1, 7	7, 6, 5, 3
35	2, 5, 6, 8	1, 2, 5, 6, 7	7, 6, 5, 2
36	1, 5, 6, 8	1, 5, 6, 7	7, 6, 5
37	1, 4, 6, 8	1, 2, 3, 4, 6, 7	7, 6, 4, 3, 2
38	2, 4, 6, 8	1, 3, 4, 6, 7	7, 6, 4, 3
39	3, 4, 6, 8	1, 2, 4, 6, 7	7, 6, 4
40	2, 3, 6, 8	1, 2, 3, 6, 7	7, 6, 3, 2
41	1, 3, 6, 8	1, 3, 6, 7	7, 6, 3
42	1, 2, 6, 8	1, 2, 6, 7	7, 6
43	1, 2, 5, 8	1, 2, 3, 4, 5, 7	7, 5, 4, 3
44	1, 3, 5, 8	1, 2, 4, 5, 7	7, 5, 4, 2
45	2, 3, 5, 8	1, 4, 5, 7	7, 5, 4
46	2, 4, 5, 8	1, 2, 3, 5, 7	7, 5, 3, 2
47	1, 4, 5, 8	1, 3, 5, 7	7, 5, 3
48	3, 4, 5, 8	1, 2, 5, 7	7, 5
49	2, 3, 4, 8	1, 2, 3, 4, 7	7, 4, 3, 2
50	1, 3, 4, 8	1, 3, 4, 7	7, 4, 3
51	1, 2, 4, 8	1, 2, 4, 7	7, 4
52	1, 2, 3, 8	1, 2, 3, 7	7
53	1, 3, 7, 8	1, 2, 3, 4, 5, 6	6, 5, 4, 2
54	2, 3, 7, 8	1, 3, 4, 5, 6	6, 5, 4
55	2, 4, 7, 8	1, 2, 3, 5, 6	6, 5, 2
56	1, 4, 7, 8	1, 3, 5, 6	6, 5
57	1, 5, 7, 8	1, 2, 3, 4, 6	6, 4, 3, 2
58	2, 5, 7, 8	1, 3, 4, 6	6, 4, 3
59	3, 5, 7, 8	1, 2, 4, 6	6, 4
60	4, 5, 7, 8	1, 2, 3, 6	6
61	4, 6, 7, 8	1, 2, 3, 4, 5	5, 4
62	3, 6, 7, 8	1, 2, 3, 5	5, 3
63	2, 6, 7, 8	1, 2, 5	5, 2
64	1, 6, 7, 8	1, 5	5
65	5, 6, 7, 8	1, 2, 3, 4	

$s \notin S$. This is enough, as ∂T is independent and ∂T^* is coindependent, which are the only sets checked by the oracles. Furthermore, if incremental independence can be checked in time t_1 and incremental coindependence in time t_2, then this gives an $O(|E|(t_1 + t_2))$-delay algorithm for generating bases.

Note that the space requirements for bases algorithms are the sum of the space it takes to store one base and one cobase and any extra data structures we would maintain for testing incremental independence/coindependence.

1. Uniform matroid: For the uniform matroid $U(n, k)$ let $S \subseteq [n]$ and $s \notin S$, in addition to S we also store its size. We can check incremental independence by checking if $|S|+1 \leq k$. Similarly, the coindependence oracle can be implemented by checking if $|S| + 1 \leq n - k$. These checks and updates take $O(1)$ time, thus we obtain a running time of $O(n)$ per generated base.

   An interesting phenomenon occurs when we consider a fixed $\lambda > 1$ and $n = \lambda k$. In this case, the implementation of `Gclose` is such that the larger element e, on average, is bounded by a constant (which depends on λ, but not on n). Since, the time actually needed to generate a new base is $O(e)$, we obtain a bound of $O_\lambda(1)$ time on average. Details on the exact bound are provided in Section 6.1.2.

2. Graphic matroid: Given a graph $G = (V, E)$ such that $|V| = n$ and $|E| = m$, we consider now the graphic matroid $M(G)$. The problem of incremental independence/coindependence has been well studied on graphic matroids, as in particular, these appear as subproblems on algorithms like Kruskal and Reverse-Kruskal. Tarjan [34] showed that incremental independence can be checked in time $O(\alpha(m, n))$ where α is the inverse Ackermann function. Thorup [35] showed that incremental coindependence can be checked in time $O(\log n(\log \log n)^3)$. Thus, we obtain an $O(m \log n(\log \log n)^3)$-delay algorithm for the spanning trees of a graph.

3. Column matroid: Let A be an n by m matrix and consider its column matroid. Here, we note that we can check independence/coindependence by solving an $n \times n$ system of linear equations in time $O(n^3)$. Thus, we obtain a time of $O(mn^3)$ per generated base.

We summarize these results in Table 2 and have the following theorem as bookkeeping.

▶ **Theorem 12.** *Run-times and memory usage for algorithms in this section are in Table 2.*

Table 2 The running times and memory usage by the modified greedy algorithms.

Matroid	Preprocessing	Space	Delay
Generic Matroid	$O(\lvert E\rvert)$	$O(\lvert E\rvert)$	$O(\lvert E\rvert)$ calls to the independence and coindependence oracle.
Graphic $M(G)$	$O(m+n)$	$O(m+n)$	$O(m \log n(\log \log n)^3)$
Uniform $U(n,k)$	$O(n)$	$O(n)$	$O(n)$
Uniform $U(\lambda k, k)$	$O(k)$	$O(k)$	$O_\lambda(1)$ amortized
Column $M_\mathbb{F}(A)$	$O(mn^2)$	$O(mn)$	$O(mn^3)$

6 Special cases

In this section, we specialize our results to a couple of interesting matroids. The goal is not to cover all of the interesting cases that can be derived from our meta-algorithms, nor is it to obtain the sharpest results in any specific case. Instead, the goal is to exemplify that our algorithms are very versatile and can be refined in various ways. Collectively, the examples in this section illustrate the four notes provided in Section 4.

- *Efficiency.* We include an algorithm that runs in constant amortized time.
- *Tiebreakers.* We illustrate that the different tiebreaker rules result in different orders.
- *Restricted operations.* We include orders that use restricted types of exchanges, including homogeneous transpositions and edge pivots.
- *Starting base.* We include orders that make use of particular starting bases and obtain stronger properties like cyclicity or homogeneity.

6.1 All your combinations: uniform matroid bases

In this subsection we consider the particular case when M is the uniform matroid $U(n,k)$. Recall that the bases in this case are all subsets of $[n]$ of size k. Furthermore, in this Section we will usually think of these base in terms of their *characteristic vector*. The characteristic vector of a set S of $[n]$ is simply a bitstring $x \in \{0,1\}^n$ which has ones in the positions given by S. More formally,

$$x_i = \begin{cases} 1 & \text{if } i \in S, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the characteristic vectors of bases of $U(n,k)$ are exactly the bitstrings $x \in \{0,1\}^n$ with exactly k ones. We call such vectors (n,k)-combinations. In this context, a base exchange will mean simply the transposition of a 1 and a 0 in the vector. Finally, we remark that we will usually make no distinction between the subsets and their characteristic vectors.

The generation of (n,k)-combinations has been well studied and is covered in Section 7.2.1.3 of The Art of Computer Programming [15]. It is not hard to see that restriction of the binary reflected Gray code for n to (n,k)-combinations gives a *transposition* Gray code, but nevertheless, stronger constraints on the type of transpositions are also possible. We mention two which are relevant for the section. Given (n,k)-combinations x, y, we say that y is obtained by a *homogeneous transposition* from x if it is obtained by transposing elements which have only zeros in between, i.e. x_i and x_j are transposed with $i < j$ and $x_{i+1} \dots x_{j-1} = 0^{j-i-1}$. Due to a result by Eades and McKay [6] homogeneous transpositions Gray codes are known to exist and moreover, they are obtained by the following recursion for $n, k > 0$

$$EM_{n,k} = EM_{n-1,k}0,\ \mathrm{rev}(EM_{n-2,k-1})01,\ EM_{n-2,k-2}11, \qquad EM_{n,0} = 0^n,\ EM_{n,n} = 1^n$$

with the convention that $EM_{n,k} = \emptyset$ for n or k negative. Note that with a homogeneous transposition Gray code, we can play all k-note chords on an n-note piano by moving exactly one finger at a time. We also consider the weaker notion of a *consistent transposition* which only requires that the bits in between the transposed elements are the same, i.e. if x_i and x_j were transposed with $i < j$, then $x_{i+1} \dots x_{j-1}$ is either 0^{j-i-1} or 1^{j-i-1}. In the piano interpretation, a consistent transposition Gray code allows us to play all k-note chords on an n-note piano by either shifting a block of contiguous fingers one note or moving exactly one finger (across possibly many notes).

6.1.1 A closer look into the closest tiebreaker

We already know that the greedy Algorithm allows us to generate combinations by transpositions, but much more can be obtained by closer examination of the listings provided by Algorithm $\mathtt{G_{close}}$. A first interesting observation is that, if x_i and x_j are transposed for $i < j$ then the bits in between are either all zeros or all ones. Otherwise, there would have been a closer transposition with the same largest element j. This directly implies that the listings obtained by Algorithm $\mathtt{G_{close}}$ are *consistent transposition Gray codes.* We analyze this further by obtaining a recursive description of the listing obtained by Algorithm $\mathtt{G_{close}}$. To this end, we introduce the *prefix function* $p : \{0,1\}^n \to \{0,1\}^{n-1}$ as a function which maps $x_1 \dots x_n$ to its longest proper prefix $x_1 \dots x_{n-1}$. A simple inductive argument gives the following recursive definition.

▶ **Lemma 13.** *Let x be an (n, k)-combination and $L(n, k, x)$ the Gray code computed by $\mathtt{G_{close}}(U(n, k), x)$. The following recursive formulas hold for $n > k \geq 1$.*

- *If $x_n = 1$: $L(n, k, x) = L(n-1, k-1, p(x))1, L(n-1, k, 1^{k-1}0^{n-k-1}1)0$.*
- *If $x_n = 0$: $L(n, k, x) = L(n-1, k, p(x))0, L(n-1, k-1, 0^{n-k-1}1^{k-1}0)1$.*

The following remark is a direct consequence of Lemma 13

▶ Remark 14. There are only two possible final bitstrings in the execution of Algorithm $\mathtt{G_{close}}$ for (n, k)-combinations: 1^k0^{n-k} or $0^{n-k}1^k$. In particular, note that only the final bit of the starting bitstring decides this: if it is a 1 the final (n, k)-combination will be 1^k0^{n-k} and if it is a 0 the final one will be $0^{n-k}1^k$.

By Remark 14, we know that starting Algorithm $\mathtt{G_{close}}$ with $1^{k-1}0^{n-k}1$ will end up with a *cyclic* Gray code. Furthermore, this will be a consistent transposition Gray code. We summarize the results of this subsection in the following corollary.

▶ **Corollary 15.** *Running Algorithm $\mathtt{G_{close}}$ on $U(n, k)$ starting from any (n, k)-combination gives a consistent transposition Gray code. Moreover, this Gray code is cyclic (in the consistent transposition sense) if and only if the starting bitstring is either $1^{k-1}0^{n-k}1$ or $0^{n-k-1}1^k0$.*

6.1.2 Amortized analysis: Average larger label

In this subsection we state the result anticipated in Section 5.2. The proof is omitted due to space considerations.

▶ **Lemma 16.** *Let x be an (n, k)-combination. We denote by $\operatorname{avg}(n, k, x)$ the average larger label involved in a transposition in the listing produced by $\mathtt{G}(U(n, k), x)$. The following holds:*

$$\operatorname{avg}(n, k, x) \leq \frac{5n^2}{k(n-k)} + O(1).$$

In particular, if $\lambda > 1$ and $n = \lambda k$, then

$$\operatorname{avg}(\lambda k, k, x) \leq \frac{5\lambda^2}{(\lambda - 1)} + O(1) = O_\lambda(1).$$

6.1.3 Homogeneous transpositions

The reader may wonder whether Algorithm $\mathtt{G_{close}}$ gives a *homogeneous* transposition Gray code. This is not true as transpositions over contiguous blocks of ones are utilized (see Figure 12). Interestingly, we can fulfill this stronger requirement just by choosing a suitable tiebreaker for Algorithm $\mathtt{G}$. Furthermore, the tiebreaker rule is very natural as it just adds the *homogeneous* requirement to the furthest tiebreaker rule as follows.

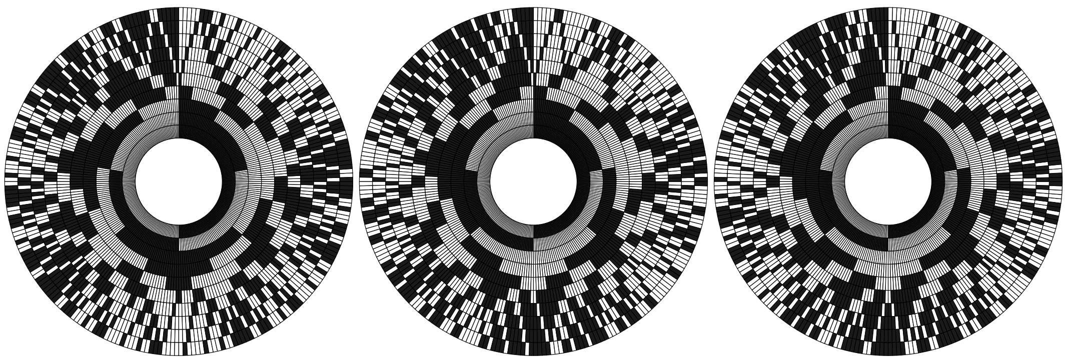


Figure 12 Illustration of three gray codes for $(10,5)$-combinations. The strings appear in clockwise order, starting at 12 o'clock. The inner track is the leftmost bit and the outer track the rightmost bit, where 1-bits are drawn white and 0-bits are drawn black. (a) The homogeneous transposition Gray code for $(10,5)$-combinations produced by the Eades-McKay algorithm (b) The homogeneous transposition Gray code for $(10,5)$-combinations produced by $\mathtt{G}_{\mathtt{hom}}(10,5,1111100000)$ (c) The consistent transposition Gray code for $(10,5)$-combinations produced by $\mathtt{G}_{\mathtt{close}}(10,5,1111100000)$.

Algorithm $\mathtt{G}_{\mathtt{hom}}$ *(Greedy combinations w/ homogeneous transpositions).* This algorithm attempts to generate all (n,k)-combinations by homogeneous transpositions starting from x^0.

H1. [Initialize] Visit the initial (n,k)-combination x^0.

H2. [Greedy] Generate an unvisited (n,k)-combination by performing a homogeneous transposition that minimizes the larger element, and then breaks ties by minimizing the smaller element. If no such transposition exists, then terminate. Otherwise visit the resulting (n,k)-combination and repeat H2.

This is not the first greedy approach for homogeneous transposition Gray codes, as such an approach was given by Williams in 2013 [44]. It is interesting to note that the greedy approach is different to both the one by Eades-McKay (see Figure 12 for a sample comparison with Eades-McKay's approach). On the other hand, even though the approach by Williams has a different description, it is not hard to see that it generates the same listings as $\mathtt{G}_{\mathtt{hom}}$.

We note that it is not immediate from Theorem 10 that Algorithm $\mathtt{G}_{\mathtt{hom}}$ generates *all* (n,k)-combinations, as we have restricted ourselves to only homogeneous transpositions (instead of all possible transpositions or base exchanges). In particular, it is not true that Algorithm $\mathtt{G}_{\mathtt{hom}}$ generates all (n,k)-combinations independently of the starting combination, as it fails to do so for some of them. Interestingly enough, we can ensure that Algorithm $\mathtt{G}_{\mathtt{hom}}$ works, whenever the starting bitstring has all of its ones consecutively. More formally, for $a \in \{0,\dots,n-k\}$ define $s_{n,k,a}$ as the (n,k)-combination which has all of its ones consecutively and begins with a zeroes, that is $0^a 1^k 0^{n-k-a}$. We have the following theorem.

▶ **Theorem 17.** *For every $a \in \{0,\dots,n-k\}$, $\mathtt{G}_{\mathtt{hom}}(n,k,s_{n,k,a})$ generates all (n,k)-combinations.*

The proof is by induction and it is omitted due to space considerations.

6.2 Your biggest fan: graphic matroid bases and pivot Gray codes

As mentioned earlier, the bases of a graphic matroid are the spanning trees of the associated graph. Our results provide *edge-exchange Gray codes*, meaning that one edge is added and another one is deleted to obtain the next spanning tree. Recently, a new type of spanning tree Gray code was introduced by Cameron, Grubb, and Sawada [3]. A *pivot Gray code* refines an exchange as follows: The two edges involved in the exchange must share a common vertex u. In other words, the operation is *u-pivot* (or simply a *pivot*), meaning that an edge changes its other endpoint by pivoting around u.

Besides introducing this new concept, the authors of [3] provide a specific pivot Gray code for the family of graphs known as fans. The *fan graph* F_n contains n vertices and $(n-2)+(n-1)=2n-3$ edges that are organized into a *path* of length $n-1$, along with an edge from each of these path vertices to an additional vertex known as the *handle*. The name of these graphs comes from their natural planar embedding (i.e., with every vertex on the outer face) that resembles an unfolded fan.

The Gray code in [3] is also constructed greedily, but in a vertex-centric manner. They label the vertices of the path $2,3,\ldots,n$ and the handle as ∞, and then they aim to minimize the vertex that supports a pivot to a new spanning tree. More specifically, their greedy rule is stated as follows:

Prioritize the pivots u from smallest to largest and then for each pivot, prioritize the edges uv that can be removed from the current tree in increasing order of the label on v, and for each such v, prioritize the edges uw that can be added to the current tree in increasing order of the label on w.

By starting at a specific spanning tree – the path together with edge 2∞ – an understandable recursive structure is obtained. From this recursive structure, the authors of [3] are able to obtain algorithmic results including efficient generation and ranking / unranking.

At first glance, the results in [3] appear to have little in common with the general results presented in this paper. After all, we consider exchange operations not pivots, and we prioritize edges rather than vertices. However, that initial judgement proves to be incorrect.

By carefully setting our initial conditions, we are able to recreate the combinatorial result in [3]. In other words, we can also create pivot Gray codes for the fan graph. In fact, we strengthen the combinatorial result in two ways.

1. There is a *face pivot Gray code* of F_n's spanning trees. This means that the pivot operation is further refined as follows: The added and deleted edges are consecutive in one of the faces of F_n's natural planar embedding[5].
2. There is a face pivot Gray code starting from any spanning tree of F_n.

To obtain our results, we label the edges in increasing order from left-to-right as shown in Figure 13. Then Algorithm $\texttt{G}_{\texttt{close}}$ will generate a face pivot Gray code, regardless of the starting spanning tree, with one such order illustrated in Figure 13.

▶ **Theorem 18.** *Algorithm $\texttt{G}_{\texttt{close}}$ generates a face pivot Gray code for the spanning trees of the fan graph F_n, regardless of the initial spanning tree, so long as it uses the left-to-right labeling.*

[5] This stronger property is not directly mentioned in [3], but it might hold. In particular, the example order F_6 in Figure 4 only uses face pivots.

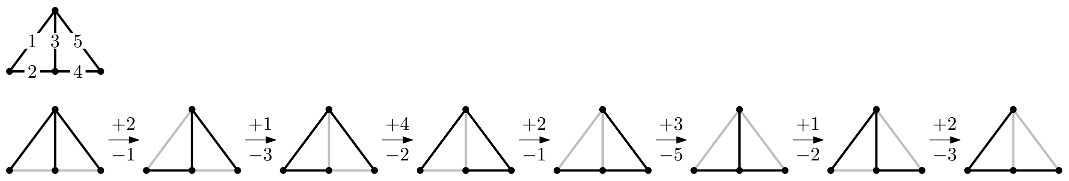


Figure 13 A face pivot Gray code of the fan graph F_4.

Proof. By Theorem 10, we know that Algorithm $\mathtt{G}_{\mathtt{close}}$ generates exchange Gray codes under these conditions. Consider one such Gray code, and let T be an arbitrary spanning tree in this order, so long as it is not the last one. Let $T' = T\Delta\{e, f\}$ be the next spanning tree in the order with $f < e$. To complete the proof, we simply need to show that this exchange is a face pivot. We proceed in two cases.

- Case 1: $e \notin T$. Let us consider the fundamental cycle C in $T + e$. Observe that C must include exactly two handle edges, say g and h, and some positive number of path edges, say $p_1, p_2, \dots, p_k$. Without loss of generality, we can assume $g < p_1 < p_2 < \cdots < p_k < h$ due to the left-to-right labeling. More specifically, we have the following equalities:

$$g = p_1 - 1 \qquad p_1 = p_2 - 2 \qquad p_2 = p_3 - 2 \qquad \dots \qquad p_{k-1} = p_k - 2 \qquad p_k = h - 1 \tag{3}$$

  We now consider the possibilities for e in turn.
  - Note that $e \neq g$, since otherwise e would be a loop in the e-prefix minor, and hence, Algorithm $\mathtt{G}_{\mathtt{close}}$ would not use it in an exchange.
  - Consider $e = h$. In this case, the closest tiebreaker used in Algorithm $\mathtt{G}_{\mathtt{close}}$ will select $f = p_k$ due to the rightmost equality in (3). Since $e = h$ and $f = p_k$ are on the same triangle face, this exchange is a face pivot.
  - Consider $e = p_1$. In this case, the closest tiebreaker used in Algorithm $\mathtt{G}_{\mathtt{close}}$ will select $f = g$ due to the leftmost equality in (3). Since $e = p_1$ and $f = g$ are on the same triangle face, this exchange is a face pivot.
  - Consider $e = p_i$ for some $i > 1$. Note that the handle edge $p_i - 1 \notin T$, since otherwise there would be a cycle in T. Therefore, Algorithm $\mathtt{G}_{\mathtt{close}}$ cannot select $f = p_i - 1$. The next smallest edge is $f = p_i - 2 = p_{i-1}$ due to the non-edge cases of (3), so Algorithm $\mathtt{G}_{\mathtt{close}}$ will select this by using the closest tiebreaker. Since $e = p_i$ and $f = p_{i-1}$ are on the outer face, this exchange is a face pivot.
- Case 2: $e \in T$. This case is similar to the previous case. ◀

7 Final remarks

In this paper, we proved that greedy algorithms are fundamental to matroids in a new way. More specifically, we showed that a simple meta-algorithm can be used to generate exhaustive lists of bases using exchanges. We then provided iterative implementations for generating these lists. Finally, we specialized our results for specific matroids and tiebreaker rules. The authors have observed that similar results can be established for the independent sets of a matroid, and we look forward to establishing these results in the full version of this paper. Furthermore, we also plan to provide efficient implementations for specific matroids in the Combinatorial Object Server [5].

References

1 B. Alspach and G. Liu. Paths and cycles in matroid base graphs. *Graphs Combin.*, 5(3):207–211, 1989.

2 Bad_CRC. all yor base r blong 2 us. `www.newgrounds.com/portal/view/11940`, 2001.

3 Ben Cameron, Aaron Grubb, and Joe Sawada. A pivot gray code listing for the spanning trees of the fan graph. In *International Computing and Combinatorics Conference*, pages 49–60. Springer, 2021.

4 Jean Cardinal, Arturo Merino, and Torsten Mütze. Efficient generation of elimination trees and hamilton paths on graph associahedra. In *Proceedings of the Thirty-third ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*. SIAM, Philadelphia, PA, 2022.

5 The Combinatorial Object Server: `http://www.combos.org/`.

6 P. Eades and B. McKay. An algorithm for generating subsets of fixed size with a strong minimal change property. *Inform. Process. Lett.*, 19(3):131–133, 1984.

7 Jack Edmonds. Matroids and the greedy algorithm. *Mathematical programming*, 1(1):127–136, 1971.

8 Fandom contributors. CATS - Villains Fandom. `villains.fandom.com/wiki/CATS`, 2022.

9 Frank Gray. Pulse code communication, 1953.

10 E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1214–1225. SIAM, 2020.

11 E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Transactions of the American Mathematical Society*, 2022.

12 Hung P. Hoang and Torsten Mütze. Combinatorial generation via permutation languages. II. Lattice congruences. *Israel J. Math.*, 244:359–417, 2021.

13 C. A. Holzmann and F. Harary. On the tree graph of a matroid. *SIAM J. Appl. Math.*, 22:187–193, 1972.

14 Know your meme contributors. All your base are belong to us – Know your meme. `knowyourmeme.com/memes/all-your-base-are-belong-to-us`, 2022.

15 D. E. Knuth. *The art of computer programming. Vol. 4A. Combinatorial algorithms. Part 1.* Addison-Wesley, Upper Saddle River, NJ, 2011.

16 Yue Li and Joe Sawada. Gray codes for reflectable languages. *Inf. Proc. Letters*, 109(5):296–300, 2009.

17 Mandelin, Clyde. How Zero Wing's "all your base" translation compares with the Japanese script. `https://legendsoflocalization.com/lets-take-a-peek-at-zero-wings-all-your-base-translation/`, 2014.

18 Stephen B. Maurer. Matroid basis graphs. I. *J. Comb. Theory Ser. B*, 14:216–240, 1973.

19 Stephen B. Maurer. Matroid basis graphs. II. *J. Comb. Theory Ser. B*, 15:121–145, 1973.

20 Arturo I. Merino and Torsten Mütze. Efficient generation of rectangulations via permutation languages. In *Proceedings of the 37th International Symposium on Computational Geometry (SoCG 2021), Buffalo, NY, USA, June 7–11 (Virtual Conference)*, volume 189 of *LIPIcs*, pages Art. No. 54, 18 pp. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2021.

21 Torsten Mütze. Combinatorial Gray codes – an updated survey. *arXiv preprint 2202.01280*, 2022.

22 D. J. Naddef and W. R. Pulleyblank. Hamiltonicity and combinatorial polyhedra. *J. Combin. Theory Ser. B*, 31(3):297–312, 1981.

23 J. G. Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

24 Frank Ruskey, Joe Sawada, and Aaron Williams. Binary bubble languages and cool-lex order. *J. Comb. Theory, Series A*, 119(1):155–169, 2012.

25 Frank Ruskey and Aaron Williams. The coolest way to generate combinations. *Discrete Mathematics*, 309(17):5305–5320, 2009.

26 Ahmad Sabri. *Gray codes and efficient exhaustive generation for several classes of restricted words.* PhD thesis, Université de Bourgogne, 2015.

27 J. Sawada, A. Williams, and D. Wong. Inside the binary reflected gray code: Flip-swap languages in 2-gray code order. In *International Conference on Combinatorics on Words*, pages 172–184. Springer, 2021.

28 Joe Sawada and Aaron Williams. A universal cycle for strings with fixed-content (which are also known as multiset permutations). In *Workshop on Algorithms and Data Structures*, pages 599–612. Springer, 2021.

29 Malcolm J. Smith. Generating spanning trees. Master's thesis, University of Victoria, 1997.

30 Brett Stevens and Aaron Williams. The coolest order of binary strings. In *International Conference on Fun with Algorithms*, pages 322–333. Springer, 2012.

31 Brett Stevens and Aaron Williams. The coolest way to generate binary strings. *Theory of Computing Systems*, 54(4):551–577, 2014.

32 Tadao Takaoka. O (1) time algorithms for combinatorial generation by tree traversal. *The Computer Journal*, 42(5):400–408, 1999.

33 D. T. Tang and C. N. Liu. Distance-2 cyclic chaining of constant-weight codes. *IEEE Trans. Computers*, C-22:176–180, 1973.

34 Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.

35 M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 343–350, New York, NY, USA, 2000. Association for Computing Machinery.

36 Takeaki Uno. A new approach for speeding up enumeration algorithms and its application for matroid bases. In *Computing and combinatorics (Tokyo, 1999)*, volume 1627 of *Lecture Notes in Comput. Sci.*, pages 349–359. Springer, Berlin, 1999.

37 Vincent Vajnovszki. Gray code order for Lyndon words. *Discrete Math. Theor. Comput. Sci.*, 9(2):145–151, 2007.

38 Vincent Vajnovszki and Rémi Vernay. Restricted compositions and permutations: from old to new gray codes. *Information Processing Letters*, 111(13):650–655, 2011.

39 P. Vámos. On the representation of independence structures. Unpublished manuscript., 1968.

40 Hassler Whitney. On the abstract properties of linear dependence. *Amer. J. Math.*, 57(3):509–533, 1935.

41 Wikipedia contributors. All your base are belong to us – Wikipedia, the free encyclopedia. `en.wikipedia.org/wiki/All_your_base_are_belong_to_us`, 2022.

42 Wikipedia contributors. Zero wing – Wikipedia, the free encyclopedia. `en.wikipedia.org/wiki/Zero_Wing`, 2022.

43 Aaron Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 987–996. SIAM, 2009.

44 Aaron Williams. The greedy Gray code algorithm. In *Algorithms and data structures*, volume 8037 of *Lecture Notes in Comput. Sci.*, pages 525–536. Springer, Heidelberg, 2013.

45 Aaron M. Williams. *Shift Gray codes.* PhD thesis, University of Victoria, 2009.