Neil White's Conjecture: Combinatorial and Algebraic Approaches

Ellen Emilia Anna Zscheile (Chemnitz University of Technology)
Advisor: Christoph Helmberg
Advisor: Kurt Klement Gottwald

April 15, 2025, Chemnitz

Contents

1. Introduction

1.1. Acknowledgements

I am grateful to Locria Cyber ("iacore") in helping with attempting to formalize this in Lean4. I'm also grateful to my supervisors Kurt Klement Gottwald and Christoph Helmberg, who found a few errors in earlier and later drafts of this thesis and pointed me towards potential fixes. I'm grateful to "42triangles" Geißler for proofreading parts of the pre-final draft version of this thesis.

Regards to Sebastian Debus for providing me with a reference for 3.2.7, which is a special case of a Lemma noted in the lecture notes for his lecture on representation theory (winter semester 2024).

Dedicated to the memory of Hans-Dieter Zscheile († 30.08.2024).

1.2. Overview

This thesis tries to approach the question what exactly is difficult about the Neil White Conjecture(s) (for the different variants, see Definition 2.2.3), and what isn't. This deeply relies on the existing results by Michał Lasoń and Mateusz Michałek [LM14; Las21]. It introduces an algorithm to split some problem instances of the Neil White Conjecture into smaller ones (Algorithm 1, Theorem 3.3.1), including an implementation in Rust; and as a consequence of Lemma 3.3.4, an interesting decomposition of the toric ideal of a matroid in chapter 4, that restricts the generators of the ideal for the single-element shift case (shifts in general are discussed in [Las16]).

Historically, the Neil White Conjecture(s) were initially introduced 1980 in [Whi80], which had an additional uniqueness restriction that later didn't appear to receive much attention, but the main problem stayed.

> "In 1980 White conjectured that every element of the toric ideal of a matroid is generated by quadratic binomials corresponding to symmetric exchanges." [Las21, abstract]

2. Basics / Preliminaries

Definition 2.0.1. An undirected **graph** is a pair (V, E) where V is an arbitrary set of nodes and $E \subseteq \binom{V}{2}$ (set of edges). ■

Definition 2.0.2. A **directed graph (digraph)** is a pair (V, A) where V is an arbitrary set of nodes and $A \subseteq V^2$ (set of arcs, sometimes without the diagonal Δ_V). ■

Definition 2.0.3. Let $t \in \mathbb{N}$. A **strong t-cycle** in a digraph (V, A) is a tuple $(n_i)_{i=1}^t \in V^t$ such that

- it is a trail along arcs: $(n_i, n_{i+1}) \in A$ for each $i \in [t-1]$ and $(n_t, n_1) \in A$,
- $n_i \neq n_j$ for each $\{i, j\} \in \binom{t}{2}$

Every *strong t-cycle* is a **strong cycle**. ■

Regarding graphs, this work tries to adhere to the notations from [Die17] where possible/suitable.

The following definitions were extracted from lecture notes about "Introduction to discrete mathematics", a lecture held by Christoph Helmberg, partially translated from [Aig06] (except for the definition of ranks) and influenced [Whi86].

Definition 2.0.4. An **independence system** is a pair $(E, \mathcal{F})$, where E (ground set) is an arbitrary **finite** set (at least in the cases we consider), $\mathcal{F} \subseteq 2^E = \{F : F \subseteq E\}$ (independent sets), and such that the following holds:

1. $\emptyset \in \mathcal{F}$ and
2. $B \subseteq A \in \mathcal{F} \implies B \in \mathcal{F}$.

Cardinality-maximal independent subsets are called bases. The set of all bases of $(E, \mathcal{F})$ is called a **basis system**. ■

Definition 2.0.5. Let $(E, \mathcal{F})$ be an independence system, and $X \subseteq E$. Then the **rank** $r(X)$ of X is defined by

$$r(X) := \max\{|Y| : Y \subseteq X \wedge Y \in \mathcal{F}\}. \tag{2.1}$$

■

Definition 2.0.6. A **matroid** is an independence system such that additionally, the following holds:

$$A, B \in \mathcal{F}, |B| = |A| + 1 \implies \exists v \in B \setminus A \text{ with } A \cup \{v\} \in \mathcal{F}$$

All bases have equal cardinality. ■

Regarding matroids, this work tries to adhere to [Aig06] and [Whi86].

2.1. Matroid base exchange properties

Theorem 2.1.1 (Symmetric exchange axiom (Proposition 4 in [Bry73, p. 335]))**.** *If $\mathcal{B}$ is a basis system [of a matroid] with bases $X, Y \in \mathcal{B}$, then for each $x \in X$, there exists $y \in Y$ such that $(X \setminus \{x\}) \cup \{y\}$ and $(Y \setminus \{y\}) \cup \{x\}$ are both bases. We say x can be exchanged with y.*

Lemma 2.1.2 (Multiple symmetric exchange [Gre73])**.** *Let X and Y be bases of a [matroid] G. Then for any subset $A \subseteq X$, there exists a subset $B \subseteq Y$ with the property that $(X \setminus A) \cup B$ and $(Y \setminus B) \cup A$ are both bases of G.*

Definition 2.1.3 (Base-orderable matroids (Definition 1.1 in [BS16, p. 397]))**.** Let M be a matroid.

M is **base-orderable** if, for any two bases X, Y of M, there is a bijection $\sigma : X \to Y$ such that for every $x \in X$, both $(X \setminus \{x\}) \cup \{\sigma(x)\}$ and $(Y \setminus \{\sigma(x)\}) \cup x$ are bases.

M is **strongly base-orderable** if, for any two bases X, Y of M, there is a bijection $\sigma : X \to Y$ such that for every $X' \subseteq X$,

1. $(X \setminus X') \cup \sigma(X')$ is a basis, and
2. $(Y \setminus \sigma(X')) \cup X'$ is a basis.

■

In regard to the indirectly cited sources, note that $-$ is often used instead of $\setminus$.

There are plenty of matroids which are either strongly base-orderable or there already exists proof that they fulfill one of the Neil White conjecture(s), see [Las21, p. 3]. For matroids which aren't (strongly) base-orderable, see [BS16, sections 8, 9],

Lemma 2.1.4. *Let M be a strongly base-orderable matroid, X, Y be bases of M, and $A \subseteq X$. By the strongly base-orderable property there exists a bijection $\sigma : X \to Y$, such that A can be exchanged via a multiple symmetric exchange (2.1.2) with $B = \sigma(A)$. Then $\sigma(X \cap Y) = X \cap Y$ and this multiple symmetric exchange can be expressed as a set of symmetric exchanges (2.1.1) (meaning their order doesn't matter), which, when all of these symmetric exchanges get combined, correspond to the multiple symmetric exchange.*

Proof. Let $x \in X \cup Y$. We know that $\sigma(x) = x \iff x \in X \cap Y$, because otherwise, if

$x \in X \cap Y$ then $(X \setminus \{x\} \cup \{\sigma(x)\} \subsetneq X$, which isn't a basis because it has strictly lower rank than X.

$x \in X \triangle Y$ $(X \triangle Y = (X \cup Y) \setminus (X \cap Y))$, then $\sigma(x) \neq x$ because $x \in X$ implies $x \notin Y$.

and both cases (which are the only possible ones) are contradictions.

Let $\mathcal{B}_M$ be the set of all bases of M. The set of symmetric exchanges is the following:

$$S := \{f_{a,\sigma(a)} : a \in A \setminus (X \cap Y)\} \text{ with } a, b \in X \triangle Y, a \neq b \implies$$
$$f_{c,d} : \mathcal{B}_M^2 \to \mathcal{B}_M^2$$
$$f_{c,d}(C, D) = (C \setminus \{c\} \cup \{d\}, D \setminus \{d\} \cup \{c\})$$

As required, we have for each bijection $\pi : [|S|] \to S$, that

$$\bigcirc_{i=1}^{|S|} \pi(i) = (A, B) \mapsto ((X \setminus A) \cup \sigma(A), (Y \setminus \sigma(A)) \cup A).$$

□

2.2. The Neil White conjecture(s)

The Neil White conjecture is a central topic of this work, it was originally introduced in [Whi80], and a good summary can be found in [Las21, pp. 1–2] by Michał Lasoń (and see also previous paper [LM14]). The latest paper we encountered about this is [HMW25] (preprint). These aren't really compact enough to be easy to quote directly here. We paraphrase them here instead.

Definition 2.2.1 (Compatibility)**.** Let E be an arbitrary finite set (called the "ground set"). We call two multisets $X, Y : 2^E \to \mathbb{N}_0$ in 2^E **compatible** (denoted by $X \sim_0 Y$) if their multiset union is equal, i.e.

$$\sum_{e \in V \subseteq E} X(V) = \sum_{e \in V \subseteq E} Y(V) \quad \forall e \in E \tag{2.2}$$

and they have an equal amount of members per size, i.e.

$$\sum_{V \in \binom{E}{n}} X(V) = \sum_{V \in \binom{E}{n}} Y(V) \quad \forall n \in \mathbb{N} \tag{2.3}$$

Let $t \in \mathbb{N}$. We call two t-tuples $(X_i)_{i=1}^t, (Y_i)_{i=1}^t \in (2^E)^t$ compatible ($X \sim_0 Y$) if their corresponding multisets (by forgetting the order of their elements), i.e. $X \sim_0 Y$ for $X = L_t((X_i)_{i=1}^t), Y = L_t((Y_i)_{i=1}^t)$ defined as:

$$L_t : (2^E)^t \to (2^E \to \mathbb{N}_0), V \mapsto \sum_{i=1}^{t} \mathbb{1}_{\{V_i\}} \tag{2.4}$$

(and analogously Y) are compatible, and we have that $|X_i| = |Y_i|$ for each $i \in [t]$ (this automatically ensures that (2.3) holds).

Let D be the set of either all multisets or t-tuples and let $D' \subseteq D$. We call $T : D' \to D'$ a **transformation** if for each $X \in D'$, X is compatible to $T(X)$. ■

Lemma 2.2.2 (Properties of compatibility as an relation)**.** *Compatibility ($\sim_0$) is an equivalence relation between multisets and between tuples.*

Proof. Compatibility is an equality relation on the image of the multiset union.

Let E be a finite set.

Reflexivity: $X : 2^E \to \mathbb{N}_0$ is compatible to itself ($X \sim_0 X$) because

$$\sum_{e \in V \subseteq E} X(V) = \sum_{e \in V \subseteq E} X(V) \quad \forall e \in E, \qquad \sum_{V \in \binom{E}{n}} X(V) = \sum_{V \in \binom{E}{n}} X(V) \quad \forall n \in \mathbb{N}.$$

Let $X, Y : 2^E \to \mathbb{N}_0$ be arbitrary multisets on 2^E.
Symmetry: If $X \sim_0 Y$, then $Y \sim_0 X$ because

$$\sum_{e \in V \subseteq E} X(V) = \sum_{e \in V \subseteq E} Y(V) = \sum_{e \in V \subseteq E} X(V) \qquad \forall e \in E,$$
$$\sum_{V \in \binom{E}{n}} X(V) = \sum_{V \in \binom{E}{n}} Y(V) = \sum_{V \in \binom{E}{n}} X(V) \qquad \forall n \in \mathbb{N}.$$

Transitivity: Let $Z : 2^E \to \mathbb{N}_0$. If $X \sim_0 Y$ and $Y \sim_0 Z$, then it holds that $X \sim_0 Z$ because

$$\sum_{e \in V \subseteq E} X(V) = \sum_{e \in V \subseteq E} Y(V) = \sum_{e \in V \subseteq E} Z(V) \qquad \forall e \in E,$$
$$\sum_{V \in \binom{E}{n}} X(V) = \sum_{V \in \binom{E}{n}} Y(V) = \sum_{V \in \binom{E}{n}} Z(V) \qquad \forall n \in \mathbb{N}.$$

and for tuples, we first consider the map L_t (line (2.4)) which maps tuples to multisets (similar to the map above from $(2^E \to \mathbb{N}_0) \to (E \to \mathbb{N}_0)$, which was implicitly given in the equation (2.2)), for which reflexivity, symmetry, and transitivity hold analogously (by the same argument). Then, for "$|X_i| = |Y_i|$ for each $i \in [t]$", we have $|X_i| = |X_i|$ (reflexivity), $|X_i| = |Y_i| = |X_i|$ (symmetry) and $|X_i| = |Y_i| \wedge |Y_i| = |Z_i| \implies |X_i| = |Z_i|$ (transitivity). □

The overall goal of all "Neil White conjectures" is to transform one t-tuple or multiset into another one, under the condition that they're compatible, and all their elements are bases of a given matroid M, and transform them by means of symmetric (sometimes: multiple) matroid basis-exchanges.

Originally, in [Whi80, p. 82], we also have the additional restriction that the exchange should be unique, in the sense that for a given choice of exchanging subset $A \subseteq X_i$ with $B \subseteq X_j$, there should only exist a single B such that the result $(X_i' \leftarrow (X_i \setminus A) \cup B$, $X_j' \leftarrow (X_j \setminus A) \cup B)$ is still a tuple of bases.

Definition 2.2.3. For the Neil White conjecture, there are various formulations that vary in how strict they are. We define the following equivalence relations:

Two compatible t-tuples are considered equivalent (under transitive closure of the following) iff they are the result of:

- $\sim_1$ a symmetric exchange (see Theorem 2.1.1) between neighboring bases;
- $\sim_4$ a symmetric exchange between bases in the same tuple, but without reordering;
- $\sim_2$ a symmetric exchange between bases in the same tuple, but also allowing reordering;
- $\sim_3$ a multiple symmetric exchange (see Lemma 2.1.2) between bases in the same tuple.

(here ordered by increasing flexibility) ■

We have the following relation between the different levels of strictness:

$$\sim_1 \subseteq \sim_4 \subseteq \sim_2 \subseteq \sim_3 \subseteq \sim_0$$

And the main question, which we want to approach in this thesis, is: **"for which matroids are which of these $\subseteq$ actually $=$?"** The conjecture [LM14, Conjecture 1] is saying that at least the last three of them are equal ($\sim_2$=$\sim_0$), (but it is still open if that holds or not, otherwise it wouldn't still be a conjecture).

Remark 2.2.4. Most of the equivalence relations defined in 2.2.3 are also present in [LM14], namely $\sim_1$, $\sim_2$ and $\sim_3$. Note that $\sim_4$ doesn't have a corresponding name in their paper.

That paper also introduces $TE(i)$ for $i = 1, 2, 3$ to represent the classes of matroids for which $\sim_0$ is equal to $\sim_i$ for all tuples of bases, i.e.

$$TE(i) = \{\text{matroid } (E, \mathcal{F}) \text{ with basis system } \mathcal{B} : \\ \forall t \in \mathbb{N} : \forall X, Y \in \mathcal{B}^t : X \sim_0 Y \iff X \sim_i Y\}$$

■

Lemma 2.2.5 ([LM14, Proposition 13])**.** *For any matroid M the following conditions are equivalent (TE is defined in 2.2.4):*

1. $M \in TE(1)$,
2. $M \oplus M \in TE(1)$,
3. $M \oplus M \in TE(2)$.

Remark 2.2.6. Case $\sim_1$ is already sufficiently handled in [LM14, Lemma 12, Proposition 13] (the latter is cited above in 2.2.5). ■

2.3. Directed closed trails

In order to be able to make working with deltas/differences between tuples easier, we introduce directed closed trails, which represent exactly those differences The *full* directed closed trails even allow reconstructing the original tuples.

Definition 2.3.1. Let (V, A) be a directed graph. A **directed closed trail** (shortened to **DCT**) is a set of arcs $T \subseteq A$, such that per node in V, the amount of incoming and outgoing arcs in T are equal, i.e.

$$|T \cap (\{x\} \times V)| = |T \cap (V \times \{x\})| \quad \forall x \in V \tag{2.5}$$

A **non-full** DCT isn't allowed to contain strong cycles of length 2:

$$\forall x, y \in V : (x, y) \in T \implies (y, x) \notin T$$

A **full** DCT is allowed to contain strong cycles of length 2.

A **sub-DCT** A' of a DCT A is a DCT such that $A' \subseteq A$. ■

Definition 2.3.2 (associated full DCT from tuple pair)**.** Let $t \in \mathbb{N}$, let E be an arbitrary set and $(X_i)_{i=1}^t, (Y_i)_{i=1}^t \in (2^E)^t$ such that $(X_i)_{i=1}^t \sim_0 (Y_i)_{i=1}^t$. Assume without loss of generality (w.l.o.g.) that $[t] \cap E = \emptyset$ (otherwise, rename elements of E).

Then we define the **associated** *full directed closed trail* T to $((X_i)_{i=1}^t, (Y_i)_{i=1}^t)$ as:

$$\begin{aligned} V &:= [t] \dot\cup E \\ A := T &:= \{(i,v) : i \in [t], v \in X_i\} \dot\cup \{(v,i) : i \in [t], v \in Y_i\} \\ T' &:= T \setminus \{(i,v),(v,i) : i \in [t], v \in X_i \cap Y_i\} \end{aligned}$$

The associated non-full DCT is then T'. ■

Proof. We need to verify that the result fulfills the conditions required for T to be a DCT. Note that if T is a DCT, then T' is a DCT because the elements we remove are paired properly.

The condition $(X_i)_{i=1}^t \sim_0 (Y_i)_{i=1}^t$ ensures that the equation 2.5 holds for the result:

$$\begin{aligned} |X_i| = |Y_i| &\implies |\{(i,v) : v \in X_i\}| = |\{(v,i) : v \in Y_i\}| \forall i \in [t] \\ 2.2 \wedge 2.3 &\implies \sum_{e \in V \subseteq E} \left(\sum_{i=1}^t \mathbb{1}_{\{X_i\}}(V) \right) = \sum_{e \in V \subseteq E} \left(\sum_{i=1}^t \mathbb{1}_{\{Y_i\}}(V) \right) \quad \forall e \in E \\ &\implies \sum_{i=1}^t \mathbb{1}_{X_i}(e) = \sum_{i=1}^t \mathbb{1}_{Y_i}(e) \quad \forall e \in E \\ &\implies |\{(i,e) : i \in [t] \text{ with } e \in X_i\}| = |\{(e,i) : i \in [t] \text{ with } e \in Y_i\}| \forall e \in E \end{aligned}$$

□

In Definition 2.3.2, we have arcs from $i \in [t]$ to $v \in E$ if v should be removed from the tuple component i, and we have arcs from $v \in E$ to $i \in [t]$ if v should be inserted into the tuple component i. Note that (V, A) is a bipartite graph.

Definition 2.3.3 (tuple pair from full DCT)**.** Let $t \in \mathbb{N}$, let E be an arbitrary set, and T be a full DCT on the digraph $([t] \dot\cup E, A)$ with $T \subseteq A$. Then we define the **associated** tuple pair (X, Y) with $X, Y \in (2^E)^t$ as follows: For each $i \in [t]$,

$$X_i := \{v \in E : (i,v) \in T\}; \qquad Y_i := \{v \in E : (v,i) \in T\}$$

■

Lemma 2.3.4. *Fix $t \in \mathbb{N}$, and fix E to be an arbitrary set. Then 2.3.2 and 2.3.3 are both bijective and inverse to each other.*

Proof. Bijectivity follows from the inverse relation between the two maps.

"$\to$, roundtrip starting from tuple pair": Let $(X_i)_{i=1}^t, (Y_i)_{i=1}^t \in (2^E)^t$ such that $|X_i| = |Y_i|$ for each $i \in [t]$. W.l.o.g. $[t] \cap E = \emptyset$. Then

$$T := \{(i,v) : i \in [t], v \in X_i\} \dot\cup \{(v,i) : i \in [t], v \in Y_i\}$$

Furthermore, the associated tuple pair (W, Z) to T is then:

$$W_i := \{v \in E : (i, v) \in T\} = X_i$$
$$Z_i := \{v \in E : (v, i) \in T\} = Y_i$$

"$\leftarrow$, roundtrip starting from full DCT": Let T be a full DCT on the digraph $([t] \dot\cup E, A)$ with $T \subseteq A$. Then the associated tuple pair (X, Y) is:

$$X_i := \{v \in E : (i, v) \in T\}; \qquad Y_i := \{v \in E : (v, i) \in T\}$$

Due to the equations 2.5 (which hold because T is a DCT), we have that $|X_i| = |Y_i|$.

Then, the associated full DCT T'' to (X, Y) is:

$$\begin{aligned} T'' &:= \{(i, v) : i \in [t], v \in X_i\} \dot\cup \{(v, i) : i \in [t], v \in Y_i\} \\ &= \{(i, v) : i \in [t], (i, v) \in T\} \dot\cup \{(v, i) : i \in [t], (v, i) \in T\} \\ &= T \qquad \text{(because } T \text{ is bipartite)} \end{aligned}$$

□

Definition 2.3.5 (Matrix representation of tuple)**.** Let $t \in \mathbb{N}$, let E be an arbitrary set and $(X_i)_{i=1}^t \in (2^E)^t$. Then we define the **associated matrix** to $(X_i)_{i=1}^t$ as:

$$\mathrm{tm}(X) := \left(\fatsemi_{r=1}^{t} (\mathbb{1}_{X_r}(c))_{c \in E} \right)$$

(r iterates over the rows, and c iterates over the columns) ■

Definition 2.3.6 (Matrix representaton of DCT)**.** Let T be a directed closed trail on bipartite digraph $([t] \dot\cup E, A)$ $(T \subseteq A)$.

Then the **associated matrix** $\mathrm{dctm}(T) : \{-1, 0, 1\}^{[t] \times E}$ to T is defined as:

$$\mathrm{dctm}(T) := \left(\fatsemi_{r=1}^{t} \left(\underbrace{\mathbb{1}_T((c, r))}_{1 \text{ if to be inserted}} - \underbrace{\mathbb{1}_T((r, c))}_{1 \text{ if to be removed}} \right)_{c \in E} \right) \tag{2.6}$$

An entry is the matrix is zero either if both indicator function expressions are zero or one (getting one for both means that T is a full DCT). ■

Lemma 2.3.7. *Let T be a directed closed trail on bipartite digraph $([t] \dot\cup E, A)$ $(T \subseteq A)$. Let $\mathrm{dctm}(T)$ be the associated matrix to T according to Definition 2.3.6. Then $\mathrm{dctm}(T)$ has row-sums and column-sums equal to zero.*

Proof. Column-sums: For each $c \in E$, it holds that:

$$\sum_{r=1}^{t} \mathbb{1}_T((c, r)) - \mathbb{1}_T((r, c)) = |T \cap (\{c\} \times [t])| - |T \cap ([t] \times \{c\})| \overset{\text{Eq 2.5}}{=} 0$$

Row-sums: For each $r \in [t]$, it holds that:

$$\sum_{c \in E} \mathbb{1}_T((c, r)) - \mathbb{1}_T((r, c)) = |T \cap (E \times \{r\})| - |T \cap (\{r\} \times E)| \overset{\text{Eq 2.5}}{=} 0$$

□

Thus, a "DCT" can be represented in multiple equivalent ways:

1. As a subset of arcs
2. As an induced directed graph (V, T)
3. If (V, A) is a bipartite graph $(V_1 \dot\cup V_2, A)$ with $A \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$, then the DCT can be represented as a matrix $\in \{-1, 0, 1\}^{V_1 \times V_2}$ (see Definition 2.3.6), where each row sum and column sum is zero.
4. As a "Neil White conjecture tuple pair", i.e. see compatible tuples in Definition 2.2.1.
5. As an element of an appropriately chosen ideal of a polynomial ring (see also chapter 4).

For an example, see section B.6.
A generalization of DCTs will be introduced in appendix section A.1.

3. Results

Throughout this chapter, fix a matroid $M = (E_M, \mathcal{F}_M)$ with rank $r(M)$ and basis system $\mathcal{B}_M$. Assume without loss of generality that $E_M \cap \mathbb{N} = \emptyset$. If that wasn't already the case, we can achieve that using the following method: Let E' be a set with $E' \cap \mathbb{N} = \emptyset$ and let $\pi : E_M \to E'$ be a bijection, rename elements of E_M using π by setting M to $(E', \{\pi(F) : F \in \mathcal{F}_M\})$, which is still a matroid because $\pi(F)$ is a bijection, and $\pi(F) = \{\pi(f) : f \in F\}, |\pi(F)| = |F|$, it preserves all matroid axioms.

This chapter first discusses some low-level aspects of the Neil White conjecture(s), here that means some simple restrictions about which elements are always or never present (section 3.1), and how permutations interact with the problem (section 3.2), which is relevant for figuring out how Definition 2.2.3's $\sim_1, \sim_2, \sim_3$ differ from eatch other. It turns out that $\sim_2$ is somewhat weird because it effectively allows the smallest possible exchange between two bases ((single-element) symmetric exchange, see Theorem 2.1.1), and the largest possible exchange between two bases (swapping two bases, because it allows permutations, and thus transpositions).

Then we continue to inspect some ways to break down "problem instances" of the Neil White conjecture (compatible tuple pairs, such that components are at least constrained to be members of $\mathcal{F}_M$) into smaller ones (smaller compatible tuple pairs with the same constraint), first by trying to approach this directly (section 3.3), and after that by trying to extract the approach used in [LM14, Proof of Theorem 2] (section 3.4).

3.1. Loops and Coloops

Definition 3.1.1 ([Whi86, p. 130])**.** If $e \in E_M$ forms a circuit $\{e\}$ of M, then it's called a **loop** and belongs to no basis. If $e \in E_M$ belongs to no circuit, then $e \in \bigcap_{B \in \mathcal{B}_M} B$ and it's called an **isthmus** or **coloop** (because it is a *loop* in the dual matroid M^* to M). ■

A tuple consisting of bases has the property that every loop of M is not present in any tuple component, and that every coloop is present in every tuple component.

This starts to matter when looking at the generalization where we deal with tuples in which every element is an independent set instead of specifically a basis (maximally independent set), but complicates matters enough that we limit our attention here to matroids which contain neither loops nor coloops. A side effect of this is that the problem of transforming between compatible tuples of independent sets stays relatively hard unless one allows permutations, even in a matroid where the ground set E is the only basis (which allows arbitrary exchanges).

3.2. Permutations

Now, we want to prove that the settings which "allow reordering" and "disallow reordering" are equivalent if we allow multiple symmetric exchanges instead of just exchanges of single elements from E_M.

Lemma 3.2.1. *Given a pair $(X, Y) \in \mathcal{B}_M^2$ of bases of M, then either $X = Y$ or it can be transformed into (Y, X) by a multiple symmetric exchange.*

Proof. Let $X \neq Y$ and $A = X \setminus Y$. By the "multiple symmetric exchange property" 2.1.2, there exists $B \subseteq Y$, such that $(X \setminus A) \cup B$ and $(Y \setminus B) \cup A$ are bases of M. Note that $X \setminus (X \setminus Y) = X \cap Y$ and $|A| = |B|$ because:

$$r(X) = r((X \setminus A) \cup B) \qquad\qquad r(Y) = r((Y \setminus B) \cup A).$$

Let

$$Z = X \cap B \subseteq X \cap Y = X \setminus (X \setminus Y).$$

Assume that Z would be non-empty. Then $e \in Z$ implies that

$$(X \cap Y) \cup B = ((X \cap Y) \cup (B \setminus \{e\})).$$

Taking the rank of each side of the equation reveals that $r((X \cap Y) \cup B) < r(X)$, so it isn't a basis, which is a contradiction. Thus Z is empty.

Because $|B| = |A| = |X \setminus Y| = |Y \setminus X|$ holds and $X \cap B = \emptyset$, this implies $B = Y \setminus X$. The rest follows by

$$X = (X \setminus Y) \dot{\cup} (X \cap Y) \qquad\qquad Y = (Y \setminus X) \dot{\cup} (X \cap Y)$$

□

Lemma 3.2.2. *Let $t \in \mathbb{N}$. Given a t-tuple $(A_i)_{i=1}^t \in \mathcal{B}_M^t$ of bases of M and a permutation $\pi : [t] \to [t]$, the tuple $(A_i)_{i=1}^t$ can be transformed into $(A_{\pi(i)})_{i=1}^t$ by a sequence of multiple symmetric exchanges.*

Proof. Decompose π into an (not necessarily unique) ordered composition of transpositions. We limit our considerations to $\pi = (ij)$ and generalize this to all permutations by iteration on transpositions. The case of a single transposition is proved by Lemma 3.2.1. □

Corollary 3.2.3. *If one replaces "symmetric exchanges" with "multiple symmetric exchanges" in 2.2.3, then $\sim_1$, $\sim_2$, $\sim_3$ and $\sim_4$ are equal.*

Proof. By Lemma 3.2.2 and noting that [LM14, Lemma 12] also holds for the multiple symmetric exchange case, this follows immediately. □

Remark 3.2.4. For matroids, we can't in general assume that a multiple symmetric exchange can be represented via a sequence of symmetric exchanges. In strongly base-orderable matroids, this is possible: see Definition 2.1.3, [LM14]. ■

Lemma 3.2.5. *Every transformation of a tuple into another compatible tuple of independent sets of M can be represented by a permutation of the entries of the associated matrix $\in \{0,1\}^{[t]\times E_M}$.*

Proof. The amount of entries which are equal to 1, and the amount of those which are equal to 0 stays the same, thus this follows immediately. □

Lemma 3.2.6. *Let $(G,\cdot)$ be a group with group operation $\cdot$ and neutral element 1, and let $\pi \in G$. Let $A, B \subseteq G$ be subgroups of G with $A \cap B = \{1\}$. Then π admits at most one decomposition into $\pi = \pi_A \cdot \pi_B$ with $\pi_A \in A$ and $\pi_B \in B$.*

Proof. This is based upon the lecture notes for a lecture (winter semester 2024) by Sebastian Debus on representation theory.

If there doesn't exist any such decomposition, we are done. Otherwise assume $\pi = \pi_{A'} \cdot \pi_{B'}$ with $\pi_{A'} \in A$ and $\pi_{B'} \in B$ is a different decomposition of the same π. Then,

$$\begin{aligned} & \pi_A \cdot \pi_B = \pi = \pi_{A'} \cdot \pi_{B'} \\ \iff \quad & B \ni \pi_B = \pi_A^{-1} \cdot \pi_{A'} \cdot \pi_{B'} \\ \iff \quad & B \ni \pi_B \cdot \pi_{B'}^{-1} = \pi_A^{-1} \cdot \pi_{A'} \in A \\ \iff \quad & e = \pi_B \cdot \pi_{B'}^{-1} = \pi_A^{-1} \cdot \pi_{A'} \end{aligned}$$

This implies that $\pi_A = \pi_{A'}$ and $\pi_B = \pi_{B'}$, meaning $\pi = \pi_A \cdot \pi_B$ is the only possible decomposition. □

Corollary 3.2.7. *Let C and R be arbitrary sets (subsequently called "columns" and "rows"). A permutation of entries $\pi : R \times C \to R \times C$ admits at most one decomposition into $\pi = \pi_R \circ \pi_C$:*

$$\pi_R \in \{\textit{bijective } (\pi : R \times C \to R \times C) : \forall (r,c) \in R \times C : \exists s \in R : \pi(r,c) = (s,c)\}$$
$$\pi_C \in \{\textit{bijective } (\pi : R \times C \to R \times C) : \forall (r,c) \in R \times C : \exists d \in C : \pi(r,c) = (r,d)\}$$

where π_R doesn't change the C (column) of each entry, and π_C doesn't change the R (row) of each entry.

Proof. We use an argumentation similar to the handling of row and column subgroups (of a symmetric group) induced by a Young tableaux in [Eti+11, p. 113]. First note that the sets of all possible π_R (and resp. π_C) are a subgroup $\hat{R}$ (resp. $\hat{C}$) of the symmetric group $S_{R\times C}$, and that the only permutation that is contained in both subgroups is the identity. Use Lemma 3.2.6 with $G = S_{R\times C}, A = \hat{R}, B = \hat{C}, \pi_A = \pi_R, \pi_B = \pi_C$. □

3.3. Directed closed trail extraction

Lets limit our attention to the case $\sim_4$ in Definition 2.2.3.

First we want to highlight the following trick:

Theorem 3.3.1. *Let $t \in \mathbb{N}$. Let $(A_i)_{i=1}^t, (B_i)_{i=1}^t \in \mathcal{F}_M^t$ with $(A_i)_{i=1}^t \sim_0 (B_i)_{i=1}^t$ (compatible t-tuples of independent sets), and let $I \subseteq [t]$ such that $(A_i)_{i\in I} \sim_0 (B_i)_{i\in I}$ (which are then also compatible t-tuples of independent sets). We say that I induces a sub-transformation on $((A_i)_{i=1}^t, (B_i)_{i=1}^t)$.*

Then $[t]\setminus I$ also induces a sub-transformation on $((A_i)_{i=1}^t, (B_i)_{i=1}^t)$, i.e. $(A_i)_{i\in([t]\setminus I)} \sim_0 (B_i)_{i\in([t]\setminus I)}$.

Proof. We have to prove that $(A_i)_{i\in([t]\setminus I)} \sim_0 (B_i)_{i\in([t]\setminus I)}$. First, $|A_i| = |B_i|$ for each $i \in ([t] \setminus I)$ follows directly from $(A_i)_{i=1}^t \sim_0 (B_i)_{i=1}^t$.

L_t is defined in line (2.4) as:

$$L_t : (2^E)^t \to (2^E \to \mathbb{N}_0), V \mapsto \sum_{i=1}^{t} \mathbb{1}_{\{V_i\}}.$$

Define:

$$\begin{aligned} A &:= L_t((A_i)_{i=1}^t); & B &:= L_t((B_i)_{i=1}^t) \\ C' &:= L_{|I|}((A_i)_{i\in I}); & D' &:= L_{|I|}((B_i)_{i\in I}) \\ C &:= L_{t-|I|}((A_i)_{i\in([t]\setminus I)}); & D &:= L_{t-|I|}((B_i)_{i\in([t]\setminus I)}). \end{aligned}$$

We know that $C' \sim_0 D'$. It remains to prove that $C \sim_0 D$.

We know due to $|A_i| = |B_i|$ (see above) that they have an equal amount of members per size (equation (2.3)).

It remains to prove that their multiset union is equal (equation (2.2)). Let $e \in E$ be arbitrary.

Define:

$$\hat{Z} := \sum_{e\in V\subseteq E} Z(V) \quad \forall Z \in \{A, B, C, C', D, D'\}.$$

Using $[t] = I \dot\cup ([t] \setminus I)$, it then follows that:

$$\begin{aligned} \hat{A} = \hat{C}' + \hat{C} = \hat{D}' + \hat{D} = \hat{B} \\ \hat{C}' = \hat{D}' \implies 0 = \hat{A} - \hat{C}' - (\hat{B} - \hat{D}') = \hat{C} - \hat{D} \implies \hat{C} = \hat{D} \end{aligned}$$

Because $e \in E$ was arbitrary, this (equal multiset union) holds for all $e \in E$. □

This result is surprisingly hard to see when using DCTs (we have to treat nodes in $[t]$ and nodes in E fundamentally differently).

In the setting of Theorem 3.3.1, it should be noted that we *don't* require that nodes in $\bigcup_{i\in I}(A_i \triangle B_i)$ to not be present in $A_i \triangle B_i$ outside of I, which is what differentiates this result from the following one, making it stronger than the following, more obvious result of "the graph components of DCTs are themselves DCTs".

Corollary 3.3.2. *Let $t \in \mathbb{N}$. Let $V := [t] \dot\cup E_M$. Let T be a full DCT on the digraph (V, T). Let $T' := \{(x, y) \in V^2 : (x, y) \in T \wedge (y, x) \in T\}$. Then each graph component of $(V, T \setminus T')$ induces a non-full sub-DCT (by using all the arcs from the graph component as arcs, and V as the nodes).*

Proof. Let C be a graph component of $(V, T \setminus T')$, given as a list of nodes of V. If $C \cap [t]$ is empty, the component consists of a single isolated node in E_M, and corresponds to an empty/trivial DCT. Otherwise set $I := C \cap [t]$. Note that this might be just a single isolated node in $[t]$, which would also correspond to a trivial DCT. Let $((A_i)_{i=1}^t, (B_i)_{i=1}^t) \in (\mathcal{F}_M^t)^2$ be the corresponding tuple pair to T, as defined in 2.3.3. Then $(A_i)_{i=1}^t \sim_0 (B_i)_{i=1}^t$. We have that $(A_i)_{i\in I} \sim_0 (B_i)_{i\in I}$ because C is a graph component. We can now plug $t, (A_i)_{i=1}^t, (B_i)_{i=1}^t, I$ into Theorem 3.3.1. □

Remark 3.3.3. When using the matrix representation (see Definition 2.3.6), Corollary 3.3.2 means that each block, when interpreting the matrix $\mathrm{dctm}(T)$ as a block-diagonal matrix, is itself a matrix representation of a DCT (according to Definition 2.3.6). But note that the order of columns and rows is arbitrary, so optimally, one should choose an order which maximizes (primarily) the amount of blocks and (secondarily) the amount of zero-entries in the matrix. ■

It remains to find a way to partition $[t]$ for a given tuple pair according to Theorem 3.3.1 ($(I_i)_{i=1}^l$ with $l \in \mathbb{N}$ such that $(A_j)j \in I_i \sim_0 (B_j)j \in I_i$ for each $i \in [l]$), such that one arrives at the finest possible partition, i.e. maximum amount of partition components. It might be possible to approach this by inspecting the circuits of the associated matroid (given by sets of the independent columns) of $\mathrm{dctm}(T)^T$.

An alternative approach to this is "exploring" / walking along arcs in the directed closed trail: The algorithm 1 decomposes a DCT into graph components, and then can use a trick (Lemma 3.3.4, using Theorem 3.3.1) to possibly decompose those even further. A full implementation would require enumeration which is very computationally expensive, because in order to arrive at an optimal decomposition (maximizing the amount of parts into which a DCT is decomposed), all the "arbitrary" choices ("Select any") in the algorithm would have to be replaced by such an enumeration, resulting in a combinatorial explosion. For simplicity, the implementation in Rust just selects the first item that matches the necessary conditions, but includes the useful biases noted in the algorithm description.

Lemma 3.3.4 (Single-shift trick)**.** *In the context of algorithm 1 (and 2), we can ignore updates to $F(x)$ (line 2.25) iff $x \in X$ wasn't reused, or x isn't contained in the selection (line 1.10).*

Proof. If nodes $x \in X$ are re-used, then we can't ignore updates to $F(x)$, because some nodes inside the selection use that, which should get split and pushed into ret, and we can't easily rollback those changes (except when dealing with a strongly base-orderable matroid). If no nodes $\in X$ are re-used, then we can ignore the updates to $F(x)$ for $x \in (\mathrm{cur}_{j;1})_{j=1}^{f-1}$, because nothing inside the selected DCT is affected by them. □

This problem can be demonstrated via the following possible execution sequence, with arbitrary intermediate steps from algorithm 1 treated as opaque: Let X, Y be as given as "Input" to algorithm 1. Let $i, j, k, l \in \mathbb{N}$ with $i < j \leq k < l$. Let $x \in X$, $y_i, y_{i+1}, y_j, y_k, y_{k+1}, y_l \in Y$, and F, vy as used in algorithm 1 to denote the current independent sets.

Algorithm 1 Directed closed trail extraction (Part 1)

Input: A diagonal-free bipartite directed graph $\Gamma = (X \dot\cup Y, A)$ with $Y \subseteq E_M$ which is also a non-full directed closed trail (e.g. constructed using Definition 2.3.2 on tuples of independent sets of M, and made non-full by removing all 2-cycles, but those 2-cycles should be remembered to construct F'_x for $x \in X$ next, because w.l.o.g. $F'_x = \{y \in Y : (x,y) \in A' \wedge (y,x) \in A'\}$ if A' denotes the full DCT ($A \subseteq A'$)), but might have multiple components. For each node $x \in X$, there is a list $F'_x \in \mathcal{F}_M$. All the neighbors per incoming arcs per node in X (and resp. outgoing arcs per node in X) must be independent sets in M, disjunct to F'_x, and still be independent sets when taking the set union with F'_x

Output: (rseq) A sequence of directed closed trails of that graph, each of which having exactly one component, and futhermore, biased towards being the shortest sub-DCTs of A, which could e.g. be reliably achieved by interpreting the algorithm below as "non-deterministic" (the evaluation gets forked/branched at each "Select any" instruction) and after computing all possible results, taking one of the results which has the longest sequence of DCTs and those DCTs being the shortest sub-DCTs of A; this differs from the usual interpretation of non-determinisitc to mean "the only non-diverging result, if any".

1: ret $\leftarrow$ () tuple with elements in $X \times \underbrace{Y}_{y_+} \times \underbrace{Y}_{y_-}, r \leftarrow 1$
▷ The current independent sets per $x \in X$
2: Build a map $F : X \to 2^Y, x \mapsto F'_x \dot\cup \{y \in Y : (y,x) \in A\}$
3: **while** the graph $(X \cup Y, A)$ has any node $\in Y$ with positive degree **do**
▷ Let vy be the set of already visited nodes in Y
4: vy $\leftarrow \emptyset \subset Y$
5: $i \leftarrow 1$, cur $\leftarrow$ ()
6: Select any $y_1 \in Y$ with positive degree
7: **while** true **do**
8: **if** $y_i \in \text{vy}(y_i)$ **then**
9: $f \leftarrow \min_{i \in [n]} \text{cur}_{i;2} = y_i$
10: **if** $(\text{cur}_{j;1})_{j=1}^{f-1} \cap (\text{cur}_{j;1})_{j=f}^{i-1} = \emptyset$ **then**
▷ only finish r on re-encounter $\in Y$ if $x \in X$ are disjunct.
11: $\text{ret}_r \leftarrow (\text{cur}_j)_{j=f}^{i-1}$
12: Truncate cur to $f-1$
13: $r \leftarrow r+1, i \leftarrow f$
14: **if** $f = 1$ **then**
15: **Break** out of infinite loop
16: **end if**
17: vy $= \{y \in Y : \exists j \in [f-1]$ such that $\text{cur}_{j;2} = y\}$
18: **end if**
19: **end if**

Algorithm 2 Directed closed trail extraction (Part 2)

Now the output just needs to be transformed into sets of arcs again.

```
20:          vy ← vy ∪ {y_i}
             ▷ Select any outgoing arc from y_i (preferably towards not yet visited x ∈ X)
21:          Select any x ∈ X such that (y_i, x) ∈ A
22:          F'' ← F(x) ∪̇ {y_i}
                                          ▷ (preferably bias towards y_{i+1} ∈ vy)
23:          Choose any y_{i+1} ∈ F(x) such that (x, y_{i+1}) ∈ A and F'' \ {y_{i+1}} ∈ F_M
24:          cur ← cur append (x, y_i, y_{i+1})                ▷ write into position i
25:          F(x) ← (F(x) ∪̇ {y_i}) \ {y_{i+1}}
26:          A ← A \ {(y_i, x), (x, y_{i+1})}
27:          i ← i + 1
28:      end while
29:  end while
30:  rseq ← ()
31:  for r' = 1, ..., r do
32:      cur = ret_{r'}; rseq_{r'} ← ∅
33:      for i = 1, ..., |cur| do
34:          rseq_{r'} ← rseq_{r'} ∪̇ {(cur_{i;2}, cur_{i;1}), (cur_{i;1}, cur_{i;3})}
35:      end for
36:  end for
```

1. Let $y_j \notin \text{vy}$,
2. Line 2.25: $F(x) \leftarrow (F(x) \dot\cup \{y_i\}) \setminus \{y_{i+1}\}$,
3. ... intermediate steps from algorithm 1, 2 ... ,
4. Line 2.20: vy gets updated to include y_j,
5. ... intermediate steps from algorithm 1, 2 ... ,
6. Line 2.25: $F(x) \leftarrow (F(x) \dot\cup \{y_k\}) \setminus \{y_{k+1}\}$,
7. ... intermediate steps from algorithm 1, 2 ... ,
8. Line 1.8: vy includes y_j, and we have $y_j = y_l$.

The path taken by the algorithm along the DCT is illustrated in figure 3.1b. The problematic part is that after the last step in this enumeration, we want to isolate the DCT around $y_j = y_k$ (the instructions between 4 and 8), but we would have to verify that the instructions before and after 4 commute with each other (including the checks in line 2.23, which do also check that the new set is independent), and this is hard in general because of the mutation to $F(x)$ (unless M is a strongly base-orderable matroid, and we adapt the algorithm to use π's from 2.1.3 to choose the next $y \in Y$, by $y_{i+1} = \pi(y_i)$ for $i \in \mathbb{N}$ with appropriately chosen bijective π (fulfilling 2.1.3 and selecting the bases

X, Y appropriately)). But if we forbid that some modifications of $F(x)$ (for any $x \in X$) happen outside (before 4) of the DCT to be isolated (the isolation would happen inside the block 1.10), and some modifications happen inside of the DCT to be isolated (after 4) by checking that those $x \in X$ used are disjunct (in the condition check 1.10), then we don't encounter that case.

Theorem 3.3.5. *The algorithm 1 (including part 2: algorithm 2) works as expected, i.e.*

- *the algorithm always terminates, and does not stall (= emit errors),*
- *the resulting DCTs (*rseq*) are valid (intermediate tuples are tuples of bases which are compatible with the source tuple) in sequence, including after each DCT,*
- *the resulting DCTs have exactly one component*

Proof. It should first be noted that the resulting DCTs each have exactly one component because the generation algorithm inner loop only walks along directed edges, meaning it can never jump to another component during construction of one entry in ret.

The algorithm terminates because each iteration of the infinite inner loop removes two edges from the graph, and it never aborts with an error at 2.21 because each node $y \in Y$ has the same number of incoming and outgoing edges.

Of particular interest is algorithm line 2.23. There always exists such a $y_{i+1} \in F(x)$ with $(x, y_{i+1}) \in A$ and $F'' \setminus \{y_{i+1}\} \in \mathcal{F}_M$ because this is a matroid exchange (see Thm. 2.1.1) with

$$B \leftarrow F(x); \qquad B' \leftarrow F'_x \dot\cup \{y \in Y : (x, y) \in \text{Input } A\}$$

with rank $|B| = |B'|$.

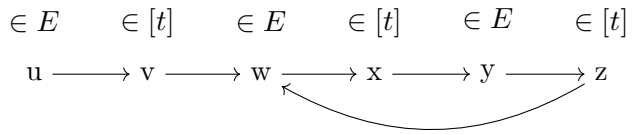
The block 1.10 is interesting, because it reveals an important edge case, which is explained by 3.3.4. □

Something closely related to this is the multiple cyclic exchange introduced in [Las16] by Michał Lasoń.

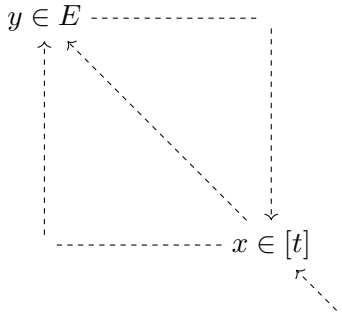
One trick used here is a conditional shortcut on $\in Y$ in case that entries $\in X$ are disjunct (between the "before" and "after" cases), which is not obviously present in the multiple cyclic exchange, because it deals with multiple occurences $\in Y$ by introducing additional parallel elements in the matroid under discussion.

3.4. Inspecting the method used for strongly base-orderable matroids

Let $t \in \mathbb{N}$ be the length of the compatible tuples $(B_i)_{i=1}^t, (D_i)_{i=1}^t \in \mathcal{B}_M^t$ under consideration.



(a) DCT can encounter the same $w \in E$ twice



(b) DCT can encounter the same $y \in E$ and $x \in [t]$ twice, leading to difficulties

Definition 3.4.1 (Overlap function, [LM14, Proof of Theorem 2])**.**

$$d((B_i)_{i=1}^t, (D_i)_{i=1}^t) := \max_{\pi \in S_t} \sum_{i=1}^{t} |B_i \cap D_{\pi(i)}| \tag{3.1}$$

■

In [LM14, Proof of Theorem 2], essentially the following method is used:

1. Use the maximum of an overlap function ($\sum |\bullet|$) to find the optimal permutation to use (see also Lemma 3.2.2, which means that in possible multiple symmetric exchanges, permutations are already included).

2. Use decreasing induction on the maximum of overlap. The induction base case is that both tuples only differ by a permutation. The maximum of overlap is bounded both from above ($r(M)t$) and below (0)[1].

3. The induction step proceeds by picking two positions $i \neq j$ inside the tuples $i, j \in [t]$, such that there exists some $e \in (B_i \setminus D_i) \cap (D_j \setminus B_j)$.

4. Then we perform some computation that is specific to strongly base-orderable matroids in order to bring (B_i, B_j) and (D_i, D_j) closer together via multiple symmetric exchanges on both tuples.

5. For strongly base-orderable matroids, the multiple symmetric exchanges can be decomposed into sequences of symmetric exchanges (see Remark 3.2.4, Definition 2.1.3).

6. Then we use the induction hypothesis to deal with the gap that remains (which is smaller than originally) after applying those symmetric exchanges.

[1] a better bound from below might be possible, but is not relevant here

We've left out some details which won't be interesting for what we want to focus on.

In step 3, such an e must exist due to $(B_i)_{i=1}^t, (D_i)_{i=1}^t$ being compatible (see Definition 2.2.1), in particular equation 2.2 applied to $e \in (B_i \setminus D_i)$, which exists because otherwise we would be in the base case of the induction.

An particularly interesting part is the step 4, and therefore we want to inspect all possible choices at that point without assuming M to be strongly base-orderable.

Definition 3.4.2. We define the set of all possible multiple symmetric exchanges $E(A, B)$ between two independent sets $A, B \in \mathcal{F}_M$:

$$E(A,B) := \left\{ (A\triangle C, B\triangle C) \in \mathcal{F}_M^2 : \begin{array}{r} C \subseteq A \cup B \\ \wedge\, |A| = |A\triangle C| \\ \wedge\, |B| = |B\triangle C| \end{array} \right\} \tag{3.2}$$

■

Lemma 3.4.3. *Let $A, B \in \mathcal{F}_M$. Then for each $(A', B') \in E(A, B)$ (as defined in 3.4.2), it holds that there exists a $C \subseteq A\triangle B$ such that $A' = A\triangle C$ and $B' = B\triangle C$.*

Proof. Let $A, B \in \mathcal{F}_M$ and $(A', B') \in E(A, B)$. By definition, there exists a $C \subseteq A \cup B$, such that $A' = A\triangle C$ and $B' = B\triangle C$.

Let's assume $|A\cap B| = |(A\cap B)\setminus C| = |A\cap B| - |A\cap B\cap C|$, which implies $A\cap B\cap C = \emptyset$, and thus $C \subseteq A\triangle B$, which we wanted to prove.

So it reamins to show what we just assumed:

$$|A\triangle C| = |A \setminus C| + |C \setminus A|; \qquad |B\triangle C| = |B \setminus C| + |C \setminus B|$$

$$\begin{aligned} |A| + |B| &= |A \setminus B| + |B \setminus A| + 2|A \cap B| &&= |A\triangle C| + |B\triangle C| \\ 0 &= |A \setminus B| + |B \setminus A| + 2|A \cap B| &&- |A \setminus C| - |C \setminus A| - |B \setminus C| - |C \setminus B| \end{aligned}$$

$$\begin{aligned} 2|A \cap B| &= |A \setminus C| - |A \setminus B| + |\underbrace{C \setminus B}_{\subseteq A}| && + |B \setminus C| - |B \setminus A| + |\underbrace{C \setminus A}_{\subseteq B}| \\ &= |A \setminus C| - |A \setminus (B \cup (C \setminus B))| && + |B \setminus C| - |B \setminus (A \cup (C \setminus A))| \\ &= |\underbrace{A \setminus C}_{\text{split along } B}| - |A \setminus (B \cup C)| && + |\underbrace{B \setminus C}_{\text{split along } A}| - |B \setminus (A \cup C)| \\ &= |(A \cap B) \setminus C| + 0|A \setminus (B \cup C)| && + |(A \cap B) \setminus C| + 0|B \setminus (A \cup C)| \\ &= 2|(A \cap B) \setminus C| \end{aligned}$$

□

Lemma 3.4.4. *Let $A, B \in \mathcal{F}_M$. Then for each $(A', B') \in E(A, B)$ (as defined in 3.4.2), it holds that $(A, B) \sim_0 (A', B')$ and if additionally $(A, B) \in \mathcal{B}_M$, then $(A, B) \sim_4 (A', B')$.*

Proof. Let $A, B \in \mathcal{F}_M, C \subseteq A \triangle B$ (due to Lemma 3.4.3) and $(A', B') = (A \triangle C, B \triangle C) \in E(A, B)$.

Assume that $A = B$. Then $C \subseteq A$ and $|A| = |A \triangle C|$. Thus $C = \emptyset$ and the rest follows because $\sim_0$ and $\sim_4$ are reflexive.

Otherwise ($A \neq B$):

$\sim_0$: Call the corresponding multiset X for (A, B) and Y for (A', B'); $X, Y : 2^E \to \mathbb{N}_0$.

$$X(V) = \begin{cases} 1 & \text{if } V = A \vee V = B \\ 0 & \text{otherwise} \end{cases}; \qquad Y(V) = \begin{cases} 1 & \text{if } V = A' \vee V = B' \\ 0 & \text{otherwise} \end{cases}$$

For each $e \in E$, the following holds:

$$\sum_{e \in V \subseteq E} X(V) = \mathbb{1}_A(e) + \mathbb{1}_B(e); \qquad \sum_{e \in V \subseteq E} Y(V) = \mathbb{1}_{A'}(e) + \mathbb{1}_{B'}(e)$$

If $e \in C \cap A$ (and analogously for $e \in C \cap B$), then

$$\mathbb{1}_A(e) + \mathbb{1}_B(e) \overset{e \notin B}{=} \mathbb{1}_A(e) \overset{e \in C}{=} \mathbb{1}_{B'}(e) \overset{e \notin A'}{=} \mathbb{1}_{A'}(e) + \mathbb{1}_{B'}(e)$$

$\sim_4$: Here we have additionally that $A, B \in \mathcal{B}_M$, and due to having maximal rank, also $A', B' \in \mathcal{B}_M$.

$$\begin{aligned} \mathcal{B}_M \ni A' &= A \triangle C = (A \setminus C) \cup (C \setminus A) \\ &= (A \setminus (A \cap C)) \cup (C \cap B) && \text{(due to Lemma 3.4.3)} \\ \mathcal{B}_M \ni B' &= B \triangle C = (B \setminus C) \cup (C \setminus B) \\ &= (B \setminus (B \cap C)) \cup (C \cap A) \end{aligned}$$

The last equation for each A', B' thus has the required format (per Lemma 2.1.2) and we are done. □

A general (valid for arbitrary matroids, e.g. M) formulation of step 4. is then the following:

Let $(A, B), (C, D) \in \mathcal{F}_M^2$ (not necessarily compatible), such that $|(A \setminus B) \cap (D \setminus C)| \geq 1 \wedge |A| = |C| \wedge |B| = |D|$. Then we consider possible improvements by maximizing overlap:

$$\underset{\substack{(A', B') \in E(A, B) \\ (C', D') \in E(C, D)}}{\operatorname{argmax}} (|A' \cap C'| + |B' \cap D'|)$$

A special case would be when no improvement is possible (the new overlap being equal to $|A \cap C| + |B \cap D|$), but I wasn't able to find a compact description of that case (or what properties M needs to fulfill to attain that state).

4. Translation into algebra

In papers about the Neil White Conjecture(s), there are apparently two main branches: Combinatorics and Algebra. Both can be used somewhat interchangably (with an exception of 2.2.3' $\sim_4$, at least on first sight).

DCTs / NWC's compatible tuple pairs get represented in the algebraic framework via differences of two monomials taken from polynomial rings (whose set of variables depends on the matroid under discussion).

Definition 4.0.1. A **toric ideal** is an ideal generated by binomials (= differences of two monomials). (see e.g. [Stu96, Lemma 4.1]) ■

For an introduction to usual notations and definitions used in algebra, see [BWK93]. A good description of this is given in the [Las21, p. 1] (quoting almost verbatim):

> "Let M be a matroid on the ground set E with the set of bases $\mathfrak{B}$ and the rank function $r : \underbrace{\mathcal{P}(E)}_{[=2^E]} \to \mathbb{N}_0$. We denote the rank of M, that is $r(E)$, simply by r.
>
> For a fixed field $\mathbb{K}$ consider a $\mathbb{K}$-homomorphism φ_M between polynomial rings:
>
> $$\varphi_M : S_M = \mathbb{K}[y_B : B \in \mathfrak{B}] \ni y_B \to \prod_{e \in B} x_e \in \mathbb{K}[x_e : e \in E] \qquad (4.1)$$
>
> The *toric ideal of a matroid* M, denoted by I_M, is the kernel of the map φ_M." [1]

The specification of φ_M is given per variable y_B, and extended linearly and multiplicatively to the whole polynomial ring.

Remark 4.0.2 (Relation between the algebraic setting and NWC strictness)**.** 2.2.3.$\sim_1$ is realized by working in a non-commutative polynomial ring. 2.2.3.$\sim_2$ is realized by working in a commutative polynomial ring instead. ■

From 3.3.1 it follows that when ground set elements only "move around" inside a subset of bases in the multisets corresponding to a { homogenous difference between two monomials } contained in the toric ideal of a matroid M, then we can isolate that subset of bases, and the corresponding difference of monomials is still contained in the toric ideal of M.

[1] The referenced paper used $\mathbb{N}$ instead of $\mathbb{N}_0 = \{0, 1, 2, ...\}$, this was adjusted here to fit the convention of the rest of this thesis

Lemma 4.0.3. *Let $t \in \mathbb{N}$. Let $(B_i)_{i=1}^t, (D_i)_{i=1}^t \in \mathcal{B}_M^t$ be compatible ($(B_i)_{i=1}^t \sim_0 (D_i)_{i=1}^t$) tuple pairs of bases of M, such that $|B_i \triangle D_i| \leq 2$[2] for each $i \in [t]$. Then we can decompose the transformation (from $(B_i)_{i=1}^t$ into $(D_i)_{i=1}^t$) into transformations which only move each ground set element $e \in E_M$ at most once, i.e. such that there is at most one $i \in [t]$ with $e \in B_i \setminus D_i$, and at most one $j \in [t]$ with $e \in D_i \setminus B_i$.*

Proof. Let $(B_i)_{i=1}^t, (D_i)_{i=1}^t \in \mathcal{B}_M^t$ with $(B_i)_{i=1}^t \sim_0 (D_i)_{i=1}^t$.

Then consider the associated non-full DCT (see Definition 2.3.2). Every strong cycle (that is, arcs are only followed along their direction, not in the reverse direction) in that DCT is a valid sub-DCT, because we if an tuple index $i \in [t]$ is present in the cycle, then either it has at most one incoming and one outgoing arc. Either both of these arcs are present in the cycle, or none.

Collect all the tuple indices $i \in [t]$, for which both arcs are present in the strong cycle, into $I \subseteq [t]$. Then apply Theorem 3.3.1 to $t, (B_i)_{i=1}^t, (D_i)_{i=1}^t, I$. Both I and $[t] \setminus I$ thus sub-transformations, and $((A_i)_{i\in I}, (B_i)_{i\in I})$ is one transformation which only move each ground set element at most once.

Continue with decomposing the remaining $[t] \setminus I$. □

Corollary 4.0.4. *Let $t \in \mathbb{N}$. Then the ideal generated by summation over*

$$\left(\prod_{i=1}^t y_{B_i} - \prod_{i=1}^t y_{D_i}\right) \mathbb{K}[y_B : B \in \mathcal{B}_M]$$

*with transformations given via $((B_i)_{i=1}^t, (D_i)_{i=1}^t)$ compatible tuple pairs of bases of M with $|B_i \triangle D_i| \leq 2$ is **also already generated** by transformations with $((B_i)_{i=1}^t, (D_i)_{i=1}^t)$ compatible tuple pairs of bases of M with $|B_i \triangle D_i| \leq 2$, which only move each ground set element at most once.*

This conclusion is something that [Las16] doesn't provide.

[2] $|B_i \triangle D_i|$ is always a multiple of 2 due to compatibility

5. Conclusions and further work

The main result of this thesis is basically Theorem 3.3.1.

It would be interesting to find an efficient implementation of the algorithm which does always arrive at an optimal decomposition (here, we don't even have an implementation that always arrives at an optimal decomposition). It remains to formalize the main results of this in Lean4 (which is difficult due to the lack of abstractions of bipartite graphs in *mathlib*) or alternatively in Agda. It might be useful to write an extension for *polymake* implementing that algorithm, or wrapping the Rust implementation, in order to make it easier to use for a wider audience.

Regarding 3.4, a compact description of the "no improvements possible" case remains to be found.

Appendices

A. Generalizations

We want to note that we don't use most properties of matroids in this thesis, in many cases it suffices to have objects which just have the exchange properties from section 2.1.

A.1. Margin sizes of functions

For algebraic constructs, we assume the definitions used in [BWK93].

Lets fix some Abelian, partially ordered group $(G, \leq)$ and denote the group operation with $+$. Let $m \in \mathbb{N} = \{1, 2, ...\}$. Let $\{E_i\}_{i=1,...,m}$ be a finite family of arbitrary measurable sets (in the discrete case, we assume that the measure is such that the integral over E_i is equal to the sum over E_i). Define $\mathcal{E} := \prod_{i=1}^{m} E_i$.

Definition A.1.1. Let $f : \mathcal{E} \to G$ be a measurable function. We call the following (linear in f) functions the **margin sizes of** f (integration over the i'th component).

$$m_{f,i} : \mathcal{E}/_{E_i} \to G, (x_j)_j \mapsto \int_{E_i} f(x_1, ..., x_{i-1}, y, x_{i+1}, ..., x_m)\, \mathrm{d}y \tag{A.1}$$

(where $E_i/_{E_i} = \{\bullet\}$, a set containing a single element) and we require that all these $m_{f,i}$ are well-defined (the result is indeed contained in G).

We call the function f **margin-size neutral** iff $m_{f,i} \equiv 0 \, \forall i = 1, ..., m$. The group operation $+ : G \times G \to G$ gives rise to a pointwise addition on such measurable functions $: \mathcal{E} \to G$.

We say that two functions f, g have **equal margin sizes** $\iff (f - g) = f + (-g)$ is margin-size neutral. ■

Margin-size neutral functions are of interest because the matrix representation (see Definition 2.3.6) **directed closed trails** on bipartite graphs can be considered margin-size neutral functions.

Definition A.1.2. Let $f, g : \mathcal{E} \to G$ be measurable functions. We call g with given $\mathcal{E}' \subset \mathcal{E}$ with $|\mathcal{E}'| < 2^{m-1}$ a **subfunction** of f if the following hold $\forall x \in \mathcal{E} \setminus \mathcal{E}'$:

1. $0 = f(x) \implies 0 = g(x)$
2. $0 < f(x) \implies 0 \leq g(x) \leq f(x)$
3. $0 > f(x) \implies 0 \geq g(x) \geq f(x)$

We call $f - g$ the **application** of g to f. ■

Now lets additionally assume that E_i's are **discrete**, and that:

$$\int_{E_i} f(y)\,\mathrm{d}y := \sum_{y \in E_i} f(y)$$

Definition A.1.3. We call a margin-size neutral function f **cubical** if $|\mathcal{E} \setminus (\operatorname{Ker} f)| = 2^m$ (has exactly 2^m non-zero entries). It always has the following representation: There exists $x, y \in \mathcal{E}$ with $\forall i = 1, \dots, m : x_i \neq y_i$, and there exists $g \in G$, such that:

$$\forall v \in 2^{\{1,\dots,m\}} : f\left(\bigtimes_{i=1}^{m} \begin{cases} x_i & \text{if } i \notin v, \\ y_i & \text{if } i \in v \end{cases}\right) = \begin{cases} g & \text{if } |v| \equiv 0 \mod 2 \\ -g & \text{if } |v| \equiv 1 \mod 2 \end{cases} \tag{A.2}$$

and we call f the **cubical associated to** (x, y, g). ■

Lemma A.1.4. *If $m \equiv 0 \mod 2$, then the cubical associated to (x, y, g) is equal to the cubical associated to (y, x, g).*

If $m \equiv 1 \mod 2$, then the cubical associated to (x, y, g) is equal to the cubical associated to $(y, x, -g)$.

B. Implementation

An implementation in Rust of the Algorithm 1 follows. Note that this is a deterministic implementation, it doesn't enumerate all possible cases which would result from an indeterministic implementation (so it usually won't find the best possible decomposition of graph components).

B.1. src/lib.rs

```rust
use core::{cmp, fmt};
use std::collections::{BTreeMap, BTreeSet};
use tracing::trace;

mod bp;
pub use bp::{BasePair, TriState};

mod bxchg;
pub use bxchg::{find_base_exchange_by_ins, BaseExchange};

mod graph;
pub use graph::Graph;

#[derive(Clone, Debug)]
pub enum Error<E> {
    TupleItemSizeMismatch(usize),
    ElementMarginMismatch(E, usize, usize),
    BaseSizeMismatch(usize),

    NonIndepInSource(usize),
    NonIndepInTarget(usize),

    DeadEndAtX(usize),
}

/// @return `elems_count?`
///
/// Complexity: `O(|X| |Y| log |Y|)`
pub fn chkeq_margin_sizes<E>(tuple: &[BasePair<E>]) -> Result<usize, Error<E>>
where
    E: Clone + cmp::Ord + cmp::PartialEq,
{
    let mut margins = BTreeMap::new();

    for (i, idata) in tuple.iter().enumerate() {
        if idata.dublen() == 0 {
```

```
            return Err(Error::TupleItemSizeMismatch(i));
        }

        for (k, v) in idata.data() {
            let ent = margins.entry(k.clone()).or_insert((0, 0));
            match v {
                TriState::Source => {
                    ent.0 += 1;
                }
                TriState::Target => {
                    ent.1 += 1;
                }
                TriState::Both => {
                    ent.0 += 1;
                    ent.1 += 1;
                }
            }
        }
    }

    let elems_count = margins.len();

    for (k, v) in margins {
        if v.0 != v.1 {
            return Err(Error::ElementMarginMismatch(k, v.0, v.1));
        }
    }

    Ok(elems_count)
}

/// Check if two base exchange sequences are disjunct in `[t]`
///
/// Complexity: `O(|lhs| log |lhs| + |rhs| log |rhs|)`
fn check_disjunct_t<E>(lhs: &[BaseExchange<E>], rhs: &[BaseExchange<E>])
    -> bool
{
    if lhs.is_empty() || rhs.is_empty() {
        return true;
    }
    let lhs_tid: BTreeSet<_> = lhs.iter().map(|i| i.tuple_item_id).collect();
    let rhs_tid: BTreeSet<_> = rhs.iter().map(|i| i.tuple_item_id).collect();
    lhs_tid.intersection(&rhs_tid).next().is_some()
}

/// Count how many base exchanges there are
/// until some `\in E` gets inserted
///
/// Complexity: `O(|cur|)`
fn count_until_e_ins<E>(cur: &[BaseExchange<E>], cur_ify: &E) -> usize
where
    E: cmp::PartialEq,
{
    cur
```

```
        .iter()
        .take_while(|i| &i.ins != cur_ify)
        .count()
}

/// Tries to find a series of tuple shifts to transform one relation
/// into another. Allows arbitrary independent sets as tuple base pairs.
/// It should even be able to handle some cases where the bases don't
/// form matroids...
///
/// complexity (`Arcs` is the set of all arcs; `|Ax|`, `|Ay|`
///   are maximum outgoing arcs from any X and Y, resp.):
/// * invariant checks:
///    `O(|X| |Y| (log |X| + log |Y|) + |X| oracle)`
/// * main loop:
///    `O(|Arcs| (|Ay| log |X| + |Ax| log |Ax| log |Y| + oracle |Ax|))`
pub fn base_tuple_shifts<E, Isf>(
    is_suitable: Isf,
    tuple: &[BasePair<E>],
) -> Result<Vec<Vec<BaseExchange<E>>>, Error<E>>
where
    E: Clone + cmp::Ord + cmp::PartialEq + fmt::Debug,
    Isf: Fn(&BTreeSet<E>) -> bool,
{
    // we internally represent the graph as follows:
    // for $x_i$, we use an array to a map to bool (non-usable outgoing
    //   edge (first, false) vs usable outgoing edge (second, true))
    // for $y_i$, we use a map to a set (usable outgoing edges)

    // the construction of DWBP already ensures that we don't have to
    // worry too much about the intersection between R and R_g

    let mut g = Graph::new();

    // check invariant of equal margin sizes and that tuple contains bases
    // oracle calls: 2 |X|
    // complexity of rest: O(|X| |Y| (log |X| + log |Y|))
    {
        chkeq_margin_sizes(tuple)?;
        g.x.resize_with(tuple.len(), BTreeMap::new);

        // already do initialization here
        // complexity: O(|X| |Y| (log |X| + log |Y|))
        for (i, idata) in tuple.iter().map(|i| i.data()).enumerate() {
            // complexity per iteration:
            // - 2 oracle calls
            // - |Y| log |Y| for set build
            // - |Y| (log |Y| + log |X|) for graph build
            let mut elems_source = BTreeSet::new();
            let mut elems_target = BTreeSet::new();
            for (k, &v) in idata {
                assert!(g.insert(i, k.clone(), v));
                match v {
                    TriState::Source => {
```

```
                    elems_source.insert(k.clone());
                }
                TriState::Target => {
                    elems_target.insert(k.clone());
                }
                TriState::Both => {
                    elems_source.insert(k.clone());
                    elems_target.insert(k.clone());
                }
            }
        }
        if !is_suitable(&elems_source) {
            return Err(Error::NonIndepInSource(i));
        }
        if !is_suitable(&elems_target) {
            return Err(Error::NonIndepInTarget(i));
        }
    }
}

let mut ret = Vec::<Vec<BaseExchange<E>>>::new();

// main part
// complexity:
//   (outer * inner) iterations: |Arcs| = |symmdiff(R, R_g)|
//   we call the maximum amount of outgoing arcs
//       from some x and y: Ax, Ay
//   complexity of inner iteration:
//     O(log |Y| + log |X|
//       + |Ay| log |X| "set diff towards not yet visited x"
//       + |Ax| (log |Ax|) (log |Y|) "sort next y's")
//     oracle calls: O(|Ax|)
//  overall complexity:
//     O(|Arcs| (|Ay| log |X| + |Ax| log |Ax| log |Y| + oracle |Ax|))

// temporary variables (e.g. also for backtracking)
let mut cur = Vec::<BaseExchange<E>>::new();

while !g.y.is_empty() {
    trace!("x = {:?}; y = {:?}", g.x, g.y);
    let mut visited_yis = BTreeSet::<E>::new();
    let mut visited_tplits = BTreeSet::<usize>::new();
    let mut cur_ify: E = if cur.is_empty() {
        // entries in y only exist if they have outgoing arcs
        g.y.iter().next().unwrap().0.clone() // O(1)
    } else {
        for i in &cur {
            visited_yis.insert(i.ins.clone());
            visited_tplits.insert(i.tuple_item_id);
        }
        cur.last().unwrap().del.clone()
    };

    let mut to_prune = 0usize;
```

```
        while {
            if visited_yis.insert(cur_ify.clone()) {
                true
            } else {
                // Only abort on re-encountered $y \in E$
                // if $x \in [t]$ are disjunct.
                // complexity: O(|RetCyc|)
                //    (insignificant compared to producer below)
                to_prune = count_until_e_ins(&cur[..], &cur_ify);
                let (lhs, rhs) = cur.split_at(to_prune);
                // the following is true is they're not disjunct
                !check_disjunct_t(lhs, rhs)
            }
        } {
            trace!(
                "cur_ify = {:?}, visited_yis = {:?}",
                cur_ify,
                visited_yis,
            );

            // select any outgoing arc from $y_i$
            let ydat =
                g.y.range(
                    (cur_ify.clone(), 0)
                    ..=(cur_ify.clone(), usize::MAX)
                )
                    .map(|(_, ent)| *ent)
                    .collect::<BTreeSet<usize>>();
            trace!("ydat = {:?}", ydat);

            // bias towards selecting not yet visited $x$
            let tuple_item_id = *ydat
                .difference(&visited_tplits)
                .next()
                .or_else(|| ydat.first())
                .expect("unvisited y_i should have outgoing arcs");
            trace!("tuple_item_id = {}", tuple_item_id);
            visited_tplits.insert(tuple_item_id);

            // collect base and excludable elements
            let (mut next_base, mut excludable)
                = g.base_and_excludable(tuple_item_id);
            if !next_base.insert(cur_ify.clone()) {
                unreachable!(
                    "encountered base which already \
contains element to be inserted"
                );
            }

            // bias towards already visited ys (false < true)
            excludable.sort_by_key(|i| !visited_yis.contains(i));
            let (next_base, excludable) = (next_base, excludable);
```

```
            // iterate over possible $y_{i+1}'s$ to find new base
            match find_base_exchange_by_ins(
                &is_suitable,
                next_base,
                excludable.into_iter())
            {
                None => return Err(Error::DeadEndAtX(tuple_item_id)),
                Some(next_ify) => {
                    assert!(next_ify != cur_ify);
                    // I'd try to handle $(a,d) \in R \cap R_g$ here,
                    // but it is too difficult to get right...
                    let bxchg = BaseExchange {
                        tuple_item_id,
                        ins: cur_ify,
                        del: next_ify.clone(),
                    };
                    trace!("got: {:?}", bxchg);
                    assert!(g.apply(&bxchg));
                    cur.push(bxchg);
                    cur_ify = next_ify;
                }
            }
        }

        // the following is a backtracking operation on `cur`.
        // we might need to prune elements from the start of `cur`
        // use `to_prune` from above.
        let to_ret = if to_prune != 0 {
            trace!("backtracked to {} bxchgs", to_prune);
            cur.drain(to_prune..).collect()
        } else {
            core::mem::replace(&mut cur, Vec::new())
        };

        if to_ret.is_empty() {
            unreachable!("found start again in graph,\
 but not in bxchg list");
        }
        ret.push(to_ret);
    }

    Ok(ret)
}
```

B.2. Cargo.toml

```
[package]
name = "nwc-dct-extract"
version = "0.1.0"
edition = "2021"

[dependencies]
tracing = "0.1.40"
```

```
[dev-dependencies]
insta = "1.39"

[dev-dependencies.test-log]
version = "0.2.16"
default-features = false
features = ["trace"]
```

B.3. src/bp.rs – base pairs

```rust
use core::cmp;
use std::collections::BTreeMap;

#[derive(Clone, Copy, Debug, PartialEq, Eq, Hash, PartialOrd, Ord)]
pub enum TriState {
    Source,
    Target,
    Both,
}

impl TriState {
    pub fn flip(self) -> Self {
        match self {
            TriState::Source => TriState::Target,
            TriState::Target => TriState::Source,
            TriState::Both => TriState::Both,
        }
    }
}

#[derive(Clone, Debug, PartialEq, Eq)]
pub struct BasePair<E> {
    data: BTreeMap<E, TriState>,
    dublen: usize,
}

impl<E: Clone + cmp::PartialEq + cmp::Ord> BasePair<E> {
    #[inline]
    pub fn new() -> Self {
        Self {
            data: BTreeMap::new(),
            dublen: 0,
        }
    }

    #[inline(always)]
    pub fn data(&self) -> &BTreeMap<E, TriState> {
        &self.data
    }

    #[inline(always)]
    pub fn dublen(&self) -> usize {
```

```
    self.dublen
}

pub fn push(&mut self, source: E, target: E) -> bool {
    if source == target {
        if self.data.contains_key(&source) {
            return false;
        }
        self.data.insert(source, TriState::Both);
    } else {
        let l1 = !self.data.contains_key(&source);
        let r1 = !self.data.contains_key(&target);
        let l2 = l1 ||
            self.data.get(&source) == Some(&TriState::Target);
        let r2 = r1 ||
            self.data.get(&target) == Some(&TriState::Source);

        if (!l2) || (!r2) {
            return false;
        }

        self.data.insert(source,
            if l1 { TriState::Source } else { TriState::Both });

        self.data.insert(target,
            if r1 { TriState::Target } else { TriState::Both });
    }
    self.dublen += 1;
    true
}

pub fn run_base_exchange(&mut self, ins: &E, del: &E) -> bool {
    let (ins_is_target, del_is_target) =
        match (self.data.get(ins), self.data.get(del))
    {
        (Some(&TriState::Source) | Some(&TriState::Both), _) => return false,
        (_, Some(&TriState::Target) | None) => return false,
        (a, Some(b)) => (a.is_some(), b == &TriState::Both),
    };
    self.data.insert(
        ins.clone(),
        if ins_is_target {
            TriState::Both
        } else {
            TriState::Source
        },
    );
    if del_is_target {
        self.data.insert(del.clone(), TriState::Target);
    } else {
        self.data.remove(del);
    }
    true
}
```

```
}
```

B.4. src/bxchg.rs – base exchanges

```rust
use super::BasePair;
use core::cmp;
use std::collections::BTreeSet;
use tracing::trace;

#[derive(Clone, Copy, Debug, PartialEq, Eq, Hash, PartialOrd, Ord)]
pub struct BaseExchange<E> {
    pub tuple_item_id: usize,
    pub ins: E,
    pub del: E,
}

impl<E> core::ops::Neg for BaseExchange<E> {
    type Output = Self;
    fn neg(self) -> Self {
        let Self {
            tuple_item_id,
            ins,
            del,
        } = self;
        Self {
            tuple_item_id,
            ins: del,
            del: ins,
        }
    }
}

impl<E: Clone + cmp::Ord + cmp::PartialEq> BaseExchange<E> {
    pub fn apply_to_tuple(&self, tuple: &mut [BasePair<E>]) -> bool {
        let xdat = match tuple.get_mut(self.tuple_item_id) {
            None => return false,
            Some(xdat) => xdat,
        };
        xdat.run_base_exchange(&self.ins, &self.del)
    }
}

/// Given an element to insert into a base (as a set `non_base`
///   containing the base merged with that elem),
/// find the suitable (`it`) element to remove
///
/// Complexity: `O(|non_base| log |non_base| + oracle |non_base|)`
pub fn find_base_exchange_by_ins<E, Isf, Iter>(
    is_suitable: Isf,
    non_base: BTreeSet<E>,
    it: Iter,
) -> Option<E>
where
```

```
    E: core::fmt::Debug + cmp::Ord + cmp::PartialEq,
    Isf: Fn(&BTreeSet<E>) -> bool,
    Iter: Iterator<Item = E>,
{
    let mut next_base_try: BTreeSet<_> = non_base;
    for k in it {
        trace!("try using removal of k = {:?}", k);
        assert!(next_base_try.remove(&k));
        if is_suitable(&next_base_try) {
            return Some(k);
        }
        next_base_try.insert(k);
    }
    None
}
```

B.5. src/graph.rs – support graph

```
use core::cmp;
use std::collections::btree_map::Entry;
use std::collections::{BTreeMap, BTreeSet};

use super::{BaseExchange, TriState};

// Note:
// there is an asymmetry between x ad y to speed up searching,
// but it makes verification of the algorithms here a bit difficult.
#[derive(Clone)]
pub struct Graph<E> {
    pub x: Vec<BTreeMap<E, bool>>,
    pub y: BTreeSet<(E, usize)>,
}

impl<E: Clone + cmp::Ord + cmp::PartialEq> Graph<E> {
    #[inline(always)]
    pub fn new() -> Self {
        Self {
            x: Vec::new(),
            y: BTreeSet::new(),
        }
    }

    /// Complexizy: `O(log |Y| + log |X|)`
    pub fn insert(&mut self, x: usize, y: E, typ: TriState) -> bool {
        let xi_dat = &mut self.x[x];
        match typ {
            TriState::Source => match xi_dat.entry(y.clone()) {
                Entry::Vacant(vac) => {
                    let had_target = self.y.remove(&(y, x));
                    vac.insert(!had_target);
                    true
                }
                Entry::Occupied(_) => false,
```

```
            },
            TriState::Target => {
                if let Some(xdt) = xi_dat.get_mut(&y) {
                    // was source -> now both
                    return core::mem::replace(xdt, false);
                }
                self.y.insert((y, x))
            }
            TriState::Both => match xi_dat.entry(y) {
                Entry::Vacant(vac) => {
                    vac.insert(false);
                    true
                }
                Entry::Occupied(_) => false,
            },
        }
    }

    pub fn apply(&mut self, bxchg: &BaseExchange<E>) -> bool {
        // verify preconditions:
        // x[tid] contains del, and not ins
        //
        // (tid, del) \in {Source, Both}
        // (tid, ins) \in {Absent, Target}
        let BaseExchange {
            tuple_item_id: tid,
            ins,
            del,
        } = bxchg;
        if ins == del {
            return true;
        }
        let tid = *tid;
        let xi_dat = &mut self.x[tid];
        if xi_dat.contains_key(ins) {
            return false;
        }
        let ins_was_target;

        // and perform exchange
        match xi_dat.entry(del.clone()) {
            Entry::Vacant(_) => return false,
            Entry::Occupied(occ) => {
                ins_was_target = self.y.remove(&(ins.clone(), tid));
                if !*occ.get() {
                    // del was both => now target
                    assert!(self.y.insert((del.clone(), tid)));
                }
                occ.remove();
            }
        }

        // (x,ins) was absent => now source
        // (x,ins) was target => now both
```

```
        xi_dat.insert(ins.clone(), !ins_was_target);
        true
    }

    pub(crate) fn base_and_excludable(&self, x: usize)
        -> (BTreeSet<E>, Vec<E>)
    {
        let xdat = &self.x[x];

        let mut base = BTreeSet::new();
        // no need to shrink later, its size shouldn't matter that much
        let mut excludable = Vec::with_capacity(xdat.len());

        for (k, &v) in xdat {
            base.insert(k.clone());
            if v {
                excludable.push(k.clone());
            }
        }

        (base, excludable)
    }
}
```

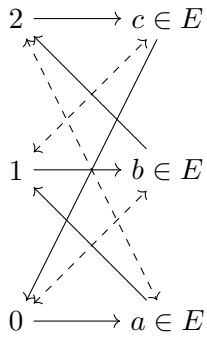
B.6. Usage example: tests/naive_counterexamples.rs

This uses a test case which we call "`stalling3`" here.

It uses the following matroid:

$$(E, \mathcal{F}) = (\{a, b, c\}, \{\{a, b\}, \{a, c\}, \{b, c\}, \{a\}, \{b\}, \{c\}\})$$

In graph notation (see also 2.3.1; dotted lines are those bidirectional arrows which are contained only in the full DCT):



In matrix notation (rows are entries $\in [2]$ and columns are entries $\in E$):

$$\begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix}$$

As tuples:

$$\begin{aligned} &(\{a, b\}, \{b, c\}, \{a, c\}) \\ \to &(\{b, c\}, \{a, c\}, \{a, b\}) \end{aligned}$$

Note in particular that this example can also be solved using the permutation $(2 \quad 1 \quad 0)$.

```
use nwc_dct_extract::*;
use insta::assert_debug_snapshot;
use std::{cmp, collections::BTreeSet};
```

```rust
use test_log::test;

fn mk_basepair<E: Clone + cmp::Ord + cmp::PartialEq>(s: &[E], t: &[E])
    -> BasePair<E>
{
    assert!(s.len() == t.len());
    let mut x = BasePair::new();
    for (s, t) in s.iter().cloned().zip(t.iter().cloned()) {
        assert!(x.push(s, t));
    }
    x
}

#[test]
fn stalling3() {
    let mut bases = BTreeSet::<BTreeSet<u8>>::new();
    for i in [[0, 1], [0, 2], [1, 2]] {
        bases.insert(i.into_iter().collect());
    }
    let is_suitable = |mb_indep: &BTreeSet<u8>| bases.contains(&mb_indep);

    let basepairs: Vec<_> =
        [([0, 1], [2, 1]), ([1, 2], [0, 2]), ([0, 2], [0, 1])]
        .into_iter()
        .map(|(s, t)| mk_basepair(&s[..], &t[..]))
        .collect();

    assert_debug_snapshot!(base_tuple_shifts(&is_suitable, &basepairs[..])
        .unwrap());
}
```

B.6.1. Test metadata: tests/snapshots/naive_counterexamples__stalling3.snap

This is the expected result of the test case (utilized by the crate `insta` for snapshot testing).

```
---
source: tests/naive_counterexamples.rs
expression: "base_tuple_shifts(&is_suitable, &basepairs[..]).unwrap()"
---
[
    [
        BaseExchange {
            tuple_item_id: 1,
            ins: 0,
            del: 1,
        },
        BaseExchange {
            tuple_item_id: 2,
            ins: 1,
            del: 2,
        },
        BaseExchange {
            tuple_item_id: 0,
```

```
            ins: 2,
            del: 0,
        },
    ],
]
```

C. Notation

$\mathbb{N}$ natural numbers $= \{1, 2, 3, \dots\}$.

$\mathbb{N}_0 = \mathbb{N} \cup \{0\}$

$[t] = \{1, \dots, t\}$

$\triangleright$ Comments in algorithms get marked with this symbol.

$,$ Commas are often used to separate tuple components. The special form $\fatsemi$ is defined as follows: Let X be an arbitrary set, let $t \in \mathbb{N}_0, f : [t] \to X$. Then

$$\left(\overset{t}{\underset{i=1}{\fatsemi}} f(i)\right) := (f(i))_{i=1}^{t}$$

This is used to emphasize the tuple construction.

$\bigcirc$ Let $t \in \mathbb{N}_0$, let X be an arbitrary set, and $f_i : X \to X$ for each $i \in [t]$, then $\bigcirc_{i=1}^{t} f_i := f_t \circ \dots \circ f_1 = f_t(\dots f_1(\bullet) \dots)$ (mass-composition of functions $(f_i)_{i=1}^{t}$).

$\dot\cup$ disjunct (multi-) set union, meaning

$$X \cap Y = \emptyset \implies X \mathbin{\dot\cup} Y = X \cup Y$$

$\triangle$ symmetric difference between two (multi-) sets:

$$X \triangle Y = (X \setminus Y) \mathbin{\dot\cup} (Y \setminus X)$$

$(i \quad j)$ A transposition of i and j.

$$(ij) : x \mapsto \begin{cases} i & \text{if } x = j, \\ j & \text{if } x = i, \\ x & \text{otherwise.} \end{cases}$$

Δ The **diagonal** Δ_X of a set X is defined by $\Delta_X := \{(e, e) : e \in X\}$.

$\mathbb{1}$ Let $A \subseteq X$ be an arbitrary set. The function $\mathbb{1}_A$, is called **indicator function** of the set A, and defined as follows:

$$\mathbb{1}_A : X \to \{0, 1\}, x \mapsto \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise.} \end{cases}$$

$\binom{E}{n}$ where $n \in \mathbb{N}_0$. If E is a number, it is just the normal binomial coefficient. If E is a set, then $\binom{E}{n} := \{V \subseteq E : |V| = n\}$.

argmax gives the set of parameters which yield the maximum of the given expression, i.e. Let S be a set, V be a totally ordered set, and $f : S \to V$ be the function that gets maximized.

$$\operatorname*{argmax}_{s \in S} f(s) := \begin{cases} \{s \in S : f(s) = v\} & \text{if } v := \max_{s \in S} f(s) \text{ is well-defined and finite} \\ \emptyset & \text{otherwise.} \end{cases}$$

Bibliography

[Bry73] Thomas H. Brylawski. "Some properties of basic families of subsets". In: *Discrete Mathematics* 6.4 (1973), pp. 333–341. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(73)90064-2. URL: https://www.sciencedirect.com/science/article/pii/0012365X73900642.

[Gre73] Curtis Greene. "A multiple exchange property for bases". In: *Proceedings of the American Mathematical Society* 39.1 (Jan. 1973), pp. 45–50. ISSN: 0002-9939. DOI: 10.1090/s0002-9939-1973-0311494-8.

[Whi80] Neil L. White. "A unique exchange property for bases". In: *Linear Algebra and its Applications* 31 (1980), pp. 81–91. ISSN: 0024-3795. DOI: https://doi.org/10.1016/0024-3795(80)90209-8. URL: https://www.sciencedirect.com/science/article/pii/0024379580902098.

[Whi86] Neil L. White. *Theory of Matroids*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1986.

[BWK93] Thomas Becker, Volker Weispfenning, and Heinz Kredel. *Gröbner Bases. A Computational Approach to Commutative Algebra*. Springer, 1993. ISBN: 0387979719, 3540979719.

[Stu96] Bernd Sturmfels. *Gröbner bases and convex polytopes*. English. Vol. 8. Univ. Lect. Ser. Providece, RI: AMS, American Mathematical Society, 1996. ISBN: 0-8218-0487-1.

[Aig06] Martin Aigner. *Diskrete Mathematik*. German. Ed. by Martin Aigner. 6., korrigierte Auflage. vieweg studium; Aufbaukurs Mathematik. Wiesbaden: Vieweg, 2006. ISBN: 9783834890399. URL: https://zbmath.org/?q=an:1109.05001.

[Eti+11] Pavel I Etingof et al. *Introduction to representation theory*. Vol. 59. American Mathematical Society, 2011. ISBN: 9780821853511. URL: https://math.mit.edu/~etingof/reprbook.pdf.

[LM14] Michał Lasoń and Mateusz Michałek. "On the toric ideal of a matroid". In: *Advances in Mathematics* 259 (July 2014), pp. 1–12. ISSN: 0001-8708. DOI: 10.1016/j.aim.2014.03.004.

[BS16] Joseph E. Bonin and Thomas J. Savitsky. "An infinite family of excluded minors for strong base-orderability". In: *Linear Algebra and its Applications* 488 (2016), pp. 396–429. ISSN: 0024-3795. DOI: https://doi.org/10.1016/j.laa.2015.09.055. URL: https://www.sciencedirect.com/science/article/pii/S0024379515005935.

[Las16] Michał Lasoń. *Matroid multiple cyclic exchange property*. 2016. arXiv: 1605.09749 [math.CO]. URL: https://arxiv.org/abs/1605.09749.

[Die17] Reinhard Diestel. *Graph theory*. English. Fifth edition. Graduate texts in mathematics. Berlin; Heidelberg: Springer, 2017. ISBN: 3662536218. URL: https://zbmath.org/?q=an:1375.05002.

[Las21] Michał Lasoń. "On the toric ideals of matroids of a fixed rank". In: *Selecta Mathematica* 27.2 (Mar. 2021). ISSN: 1420-9020. DOI: 10.1007/s00029-021-00633-6.

[HMW25] Kangjin Han, Mateusz Michałek, and Julian Weigert. *White's conjecture for matroids and inner projections*. 2025. arXiv: 2501.17738 [math.CO]. URL: https://arxiv.org/abs/2501.17738.

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Zentrales Prüfungsamt
Selbstständigkeitserklärung

Name: Zscheile Vorname: Ellen Emilia Anna geb. am: 03.08.2000 Matr.-Nr.: 643023	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 14.04.2025 Unterschrift: Ellen Emilia Anna Zscheile

---

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.