Obstructions to convexity in neural codes

Caitlin Lienkaemper[*], Anne Shiu[†], and Zev Woodstock[‡]

September 10, 2015

Abstract

How does the brain encode spatial structure? One way is through hippocampal neurons called place cells, which become associated to convex regions of space known as their receptive fields: each place cell fires at a high rate precisely when the animal is in the receptive field. The firing patterns of multiple place cells form what is known as a convex neural code. How can we tell when a neural code is convex? To address this question, Giusti and Itskov identified a local obstruction, defined via the topology of a code's simplicial complex, and proved that convex neural codes have no local obstructions. Curto *et al.* proved the converse for all neural codes on at most four neurons. Via a counterexample on five neurons, we show that this converse is false in general. Additionally, we classify all codes on five neurons with no local obstructions. This classification is enabled by our enumeration of connected simplicial complexes on 5 vertices up to isomorphism. Finally, we examine how local obstructions are related to maximal codewords (maximal sets of neurons that co-fire). Curto *et al.* proved that a code has no local obstructions if and only if it contains certain "mandatory" intersections of maximal codewords. We give a new criterion for an intersection of maximal codewords to be non-mandatory, and prove that it classifies all such non-mandatory codewords for codes on up to 5 neurons.

Keywords: neural code, place cell, convex, good cover, simplicial complex, homology

1 Introduction

The brain's ability to navigate within and represent the physical world is fundamental to our everyday experience and ability to function. How does the brain accomplish this? For their work shedding light on this question, neuroscientists John O'Keefe, May Britt Moser, and Edvard Moser won the 2014 Nobel Prize in Physiology and Medicine. Their work led to the discovery of place cells, grid cells, and head direction cells, all of which take part in rodents' and other animals' mechanisms for representing, navigating through, and forming memories of their environments.

This paper focuses on place cells, which are hippocampal neurons which become associated to regions of the environment known as their receptive fields or place fields. When an animal is located in a place cell's receptive field, the place cell fires at a higher rate than when the animal is outside the place field. The firing patterns of a collection of place cells describe an animal's position within its environment. These receptive fields have been experimentally observed to be approximately convex regions of space. *Convex codes* are those neural codes (firing patterns) that can arise from the activity of place cells with convex receptive fields.

[*]Corresponding author; Department of Mathematics, Harvey Mudd College, 301 Platt Boulevard, Claremont CA 91711-5901, USA; `clienkaemper@g.hmc.edu`

[†]Department of Mathematics, Texas A&M University, Mailstop 3368, College Station, Texas 77843–3368, USA; `annejls@math.tamu.edu`

[‡]Department of Mathematics and Statistics, James Madison University, Roop Hall 305, MSC 1911, Harrisonburg, Virginia 22807, USA; `woodstzc@dukes.jmu.edu`

Which neural codes are convex? What are signatures of convexity or non-convexity? Curto *et al.* [2, 3] and Giusti and Itskov [4] addressed these questions using combinatorial topology and commutative algebra, and gave complete answers for codes on up to 4 neurons. Curto *et al.* achieved this classification by organizing neural codes according to their simplicial complexes, and, additionally, by focusing on *local obstructions* to convexity. Earlier, Giusti and Itskov had introduced this concept and proved that codes with local obstructions are necessarily non-convex. Curto *et al.* proved that local obstructions have the following interpretation: for each simplicial complex Δ, there is a set of "mandatory" codewords whose presence in a code (whose simplicial complex is Δ) is required to avoid local obstructions [2]. Therefore, a code must contain all its mandatory codewords to be convex. Moreover, the mandatory codewords are necessarily intersections of maximal codewords. This motivates the following questions:

Question 1.1. Is every code which has no local obstructions convex?

Question 1.2. Is every intersection of maximal codewords a mandatory codeword?

Question 1.3. For codes on 5 neurons, which have local obstructions? Which are convex?

Our work addresses all 3 questions.

In a preliminary version of [2], the answer to Question 1.1 was conjectured to be "yes", and this was verified for codes on up to 4 neurons. Moreover, this was the main open problem in this subject. Here we demonstrate that even for codes on 5 neurons, the answer is in fact "no": Theorem 3.1 gives the first example of a non-convex code with no local obstructions.

For Question 1.2, again the first negative answer appears in codes on 5 neurons [2]. Here we give a sufficient criterion for an intersection of maximal codewords to be non-mandatory (Theorem 5.2). Furthermore, our criterion classifies all such non-mandatory codewords for codes on 5 neurons. In other words, our result shows that codes with no local obstructions on at most 5 neurons are precisely those codes that contain all intersections of maximal codewords together with those codes that satisfy our new criterion.

Finally, we completely answer the first part of Question 1.3 by first enumerating the 157 connected simplicial complexes on 5 vertices, and then determining for each simplicial complex which codewords are mandatory. Here we recall that a code has a local obstruction if and only if it is missing a mandatory codeword. Our enumeration is therefore an important step toward answering the second part of the Question 1.3, which we leave for future work.

2 Background

In this section, we introduce our assumptions, definitions, and notation. We approximate neural activity as binary: under this model, neurons are either firing or they are not. We encode the combinatorial data generated by the firing patterns of place cells as a neural code (Definition 2.1). We index the neurons with the integers $\{1, \ldots, n\} =: [n]$.

2.1 Neural codes

Biologically, a codeword corresponds to a set of neurons which fire together while no other neurons fire, and a neural code describes which groups of neurons are observed firing together:

Definition 2.1. A *neural code* $\mathcal{C}$ on n neurons is a set of subsets of $[n]$ (called *codewords*), i.e. $\mathcal{C} \subseteq 2^{[n]}$. A *maximal* codeword in $\mathcal{C}$ is a codeword that is not properly contained in any other codeword in $\mathcal{C}$.

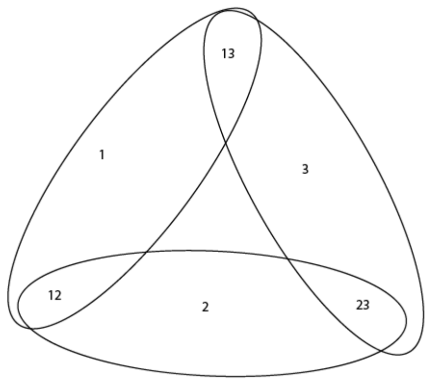
Definition 2.2. For a neural code $\mathcal{C}$ on n neurons, a collection $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$ of subsets of a set X *realizes* $\mathcal{C}$ if a codeword σ is in $\mathcal{C}$ if and only if $\left(\bigcap_{i\in\sigma} U_i\right) \setminus \bigcup_{i\notin\sigma} U_i$ is nonempty[1].

[1]We use the usual convention: the empty intersection is X, and the empty union is the empty set. Also, in this paper we make the simplifying assumption that $X \supsetneq \bigcup_{i\in[n]} U_i$, i.e. the empty set is a codeword in every code.

Definition 2.3. A neural code is:

1. *intersection-complete* if it is closed under taking intersections.
2. *max-intersection-complete* if it is closed under taking intersections of maximal codewords.
3. a *good-cover* code if it can be realized by a good cover $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$ of some set $X \subseteq \mathbb{R}^d$. (Recall that $\mathcal{U}$ is a *good cover* of X if the U_i's are contractible open sets that cover X and each intersection $U_{i_1} \cap U_{i_2} \cap \cdots \cap U_{i_k}$ is contractible or empty.)
4. *convex* if it can be realized by a collection of convex open sets $U_1, U_2, \ldots, U_n \subseteq \mathbb{R}^d$. The *minimal embedding dimension* of a code is the smallest value of d for which this is possible.

Example 2.4. Consider the code $\mathcal{C} = \{\{1,2\}, \{1,3\}, \{2,3\}, \{1\}, \{2\}, \{3\}, \emptyset\}$. We will generally write this as $\mathcal{C} = \{\mathbf{12}, \mathbf{13}, \mathbf{23}, 1, 2, 3\}$, where the maximal codewords are marked in bold. We interpret this code to mean that each pair of neurons fires together and each neuron fires alone, but all three neurons never fire at the same time. This code is intersection-complete, max-intersection-complete, a good-cover code, and convex, as realized here:



Example 2.5. The simplest example of a neural code that is not convex is the three-neuron code $\mathcal{C} = \{\mathbf{12}, \mathbf{13}, \emptyset\}$. To see this, suppose $\mathcal{C}$ were convex; then there would exist convex open sets U_1, U_2, and U_3 such that $U_1 = U_2 \cup U_3$, and $U_2 \cap U_3 = \emptyset$. We see that U_2 and U_3 would form a disconnection of the open set U_1, thus U_1, U_2, and U_3 cannot all be convex open sets. In fact, since we did not use convexity in this argument, but only connectedness, we have shown that $\mathcal{C}$ is not a good-cover code either.

Remark 2.6. Intersection patterns of convex (and other types of) sets is a well-established subject; for instance, see [6, 7, 9, 12] and the references therein. Nevertheless, the related questions we consider here—which focus on *all* regions cut out by the convex sets, in addition to which sets intersect—have only recently received attention.

2.2 Simplicial complexes

An abstract *simplicial complex* on n vertices is a set of subsets (*faces*) of $[n]$ that is closed under taking subsets[2]. That is, if Δ is a simplicial complex, then $\sigma \in \Delta$ and $\tau \subset \sigma$ implies $\tau \in \Delta$. *Facets* are the faces of a simplicial complex that are maximal with respect to inclusion.

For a code $\mathcal{C}$ on n neurons, $\Delta(\mathcal{C})$ is the smallest simplicial complex on $[n]$ that contains $\mathcal{C}$:

$$\Delta(\mathcal{C}) \ := \ \{\omega \subseteq [n] \mid \omega \subseteq \sigma \text{ for some } \sigma \in \mathcal{C}\} \ .$$

[2]Thus, the empty set is in every simplicial complex. However, for simplicity, we omit the empty set when listing the faces of simplicial complexes arising in examples.

Note that two codes on n neurons have the same simplicial complex Δ if and only if they have the same maximal codewords (which are the facets of Δ). For a face $\sigma \in \Delta$, the *link of σ in Δ* is the simplicial complex

$$\mathrm{Lk}_{\Delta}(\sigma) \ := \ \{\omega \in \Delta \mid \sigma \cap \omega = \emptyset, \ \ \sigma \cup \omega \in \Delta\} \ .$$

Next, the *restriction* of Δ to σ is the simplicial complex

$$\Delta|_{\sigma} \ := \{\omega \in \Delta \mid \omega \subseteq \sigma\} \ .$$

Finally, a simplicial complex is *contractible* if its geometric realization is contractible.

2.3 Local obstructions

Here we introduce local obstructions, which prevent a code from being convex, and furthermore prevent a code from being a good-cover code (Proposition 2.11).

Definition 2.7. Let $\mathcal{C}$ be a code on n neurons, let $\Delta = \Delta(\mathcal{C})$, and let $\mathcal{U} = \{U_1, U_2, \dots, U_n\}$ be any collection of open sets that realizes $\mathcal{C}$. The code $\mathcal{C}$ has a *local obstruction* if there exist disjoint, nonempty sets $\sigma, \tau \subseteq [n]$ such that:

1. $(\cap_{i\in\sigma} U_i) \cap U_j$ is nonempty for all $j \in \tau$,
2. $(\cap_{i\in\sigma} U_i) \ \subseteq \ (\cup_{j\in\tau} U_j)$, and
3. $\mathrm{Lk}_{\Delta|_{\sigma\cup\tau}}(\sigma)$ is not contractible.

It is important to note that the definition of local obstruction does not depend on the choice of realization $\mathcal{U}$. Indeed, this can be seen from the following characterization of codes with local obstructions, due to Curto *et al.* [2, Theorem 1.6]:

Proposition 2.8 (Characterization of codes with local obstructions via maximal codewords)**.** *A neural code $\mathcal{C}$ has a* local obstruction *if and only if some nonempty intersection of maximal codewords is not in $\mathcal{C}$ and has a non-contractible link. More precisely, if $\mathcal{M} = \{M_1, ..., M_m\}$ is the set of maximal codewords of $\mathcal{C}$, we say that $\mathcal{C}$ has a* local obstruction *if for some $I \subseteq [m]$,*

1. *$\sigma := \bigcap_{i\in I} M_i$ is nonempty,*
2. *$\sigma \notin \mathcal{C}$, and*
3. *$\mathrm{Lk}_{\Delta(\mathcal{C})}(\sigma)$ is not contractible.*

Two observations follow immediately from Proposition 2.8. First, each simplicial complex defines a set of *mandatory* codewords, those nonempty intersections of facets for which the link is non-contractible:

Definition 2.9. A face σ of a simplicial complex Δ is a *mandatory codeword* of Δ if it is the nonempty intersection of a set of facets of Δ such that $\mathrm{Lk}_{\Delta}(\sigma)$ is non-contractible.

Proposition 2.8 then states that *a code $\mathcal{C}$ has no local obstructions if and only if*

$$\{\text{mandatory codewords of } \Delta(\mathcal{C})\} \ \subseteq \ \mathcal{C} \ .$$

The second observation is that Proposition 2.8 gives a method for determining which codes on n neurons have local obstructions. Namely, first enumerate all simplicial complexes on n vertices, and then for each simplicial complex, determine the set of mandatory codewords. Curto *et al.* completed this analysis for $n \leq 4$ (and additionally proved that all such codes without local obstructions are convex) [2], and we will complete the $n = 5$ case in Section 4.

Example 2.10. We revisit the non-convex three-neuron code $\mathcal{C} = \{\mathbf{12}, \mathbf{13}, \emptyset\}$ from Example 2.5. Its simplicial complex is $\Delta(\mathcal{C}) = \{\mathbf{12}, \mathbf{13}, 1, 2, 3, \emptyset\}$, a path of length 2. Thus, the codeword 1 is an intersection of maximal codewords. However, $\mathrm{Lk}_{\Delta(\mathcal{C})}(1) = \{\mathbf{2}, \mathbf{3}, \emptyset\}$, which is a simplicial complex consisting of two disconnected points. Thus, 1 is a mandatory codeword which is not in $\mathcal{C}$, so $\mathcal{C}$ has a local obstruction (by Proposition 2.8).

The following result summarizes prior results about the relationships among properties of codes; part 1 is due to Giusti *et al.* [5], part 2 follows from the fact that every convex open cover is a good cover, part 3 is due to Giusti and Itskov [4] (see also [2, Lemma 1.5]), and part 4 follows immediately from Proposition 2.8.

Proposition 2.11. *Let $\mathcal{C}$ be a neural code.*

1. *If $\mathcal{C}$ is intersection-complete, then $\mathcal{C}$ is convex.*
2. *If $\mathcal{C}$ is convex, then $\mathcal{C}$ is a good-cover code.*
3. *If $\mathcal{C}$ is a good-cover code, then $\mathcal{C}$ has no local obstructions.*
4. *If $\mathcal{C}$ is max-intersection-complete, then $\mathcal{C}$ has no local obstructions.*

Curto *et al.* showed that for all neural codes on up to 4 neurons, the converses of parts 2–4 of Proposition 2.11 hold [2, §4]:

Proposition 2.12. *Let $\mathcal{C}$ be a neural code on at most* 4 *neurons. The following are equivalent:*

1. *$\mathcal{C}$ is convex.*
2. *$\mathcal{C}$ is a good-cover code.*
3. *$\mathcal{C}$ has no local obstructions.*
4. *$\mathcal{C}$ is max-intersection-complete.*

A preliminary version of [2] conjectured that the equivalence of parts 1-3 in Proposition 2.12 generalizes to codes on more than 4 neurons. In the next section, we give the first counterexample to that conjecture, which shows that parts 1 and 2 are not equivalent (Theorem 3.1). Furthermore, our counterexample uses only 5 neurons. It is still unknown whether parts 2 and 3 are equivalent.

Part 4 of Proposition 2.11 also can not be generalized to codes with more than 4 neurons [2]. Section 5 focuses on this gap: the intersections of facets that are *not* mandatory. More precisely, we identify a criterion that guarantees that a max-intersection-incomplete code has no local obstructions, and we prove that our criterion classifies all such codes on up to 5 neurons.

Example 2.13. Consider the following code:

$$\mathcal{C} = \{\mathbf{2345},\ \mathbf{123},\ \mathbf{134},\ \mathbf{145},\ 13,\ 14,\ 23,\ 34,\ 45,\ 3,\ 4,\ \emptyset\}\ .$$

The nonempty intersections of maximal codewords are 13, 14, 23, 34, 45, 1, 3, and 4. Of these, only 1 is missing from the code. We find that

$$\mathrm{Lk}_{\Delta(\mathcal{C})}(1) = \{\mathbf{23}, \mathbf{34}, \mathbf{45}, 2, 3, 4, 5, \emptyset\}\ ,$$

which is a path of length 3 (thus, contractible). Hence, $\mathcal{C}$ has no local obstructions (Proposition 2.8), and, moreover, is a max-intersection-*incomplete* code with no local obstructions. In the next section, we show that despite having no local obstructions, $\mathcal{C}$ is not convex.

3 A non-convex code with no local obstructions

Here we show that the code in Example 2.13 is non-convex despite having no local obstructions (recall from Proposition 2.11 that good-cover codes have no local obstructions):

Theorem 3.1. *The following code on* 5 *neurons:*

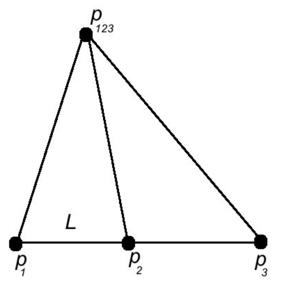
$$\mathcal{C} \quad = \quad \{\mathbf{2345},\ \mathbf{123},\ \mathbf{134},\ \mathbf{145},\ 13,\ 14,\ 23,\ 34,\ 45,\ 3,\ 4,\ \emptyset\} \tag{1}$$

is a non-convex, good-cover code.

The proof of Theorem 3.1 requires the following lemma, which has proven useful in other contexts as well (for instance, in distinguishing between minimal embedding dimension 2 vs. 3 among convex codes arising from the simplicial complex labeled by L24 in [2]).

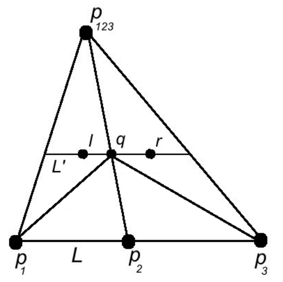
Lemma 3.2. *Let W_1, W_2, and W_3 be convex open sets in $\mathbb{R}^n$ such that their intersection is nonempty and is equal to all pairwise intersections: $W_1 \cap W_2 \cap W_3 = W_i \cap W_j$ for all $1 \le i < j \le 3$. Then, any line that intersects each of the W_i's must intersect $W_1 \cap W_2 \cap W_3$.*

Proof. Let $W_{123} := W_1 \cap W_2 \cap W_3$. Assume for contradiction that there exists a line L that intersects each $W_i \setminus W_{123}$ (for $i = 1, 2, 3$) but *not* W_{123}. For $i = 1, 2, 3$, let p_i be such an intersection point on W_i, i.e. $p_i \in L \cap (W_i \setminus W_{123})$. By relabeling if necessary, we may assume that p_2 lies between p_1 and p_3 on the line L. Now let $p_{123} \in W_{123}$; then p_{123} is *not* on L by hypothesis. So, our points have a triangular structure:



Removing the convex open set W_{123} from the line segment $\overline{p_2 p_{123}}$ yields a smaller, closed line segment (nonempty because it contains p_2), namely, the line segment $\overline{p_2 q} := \overline{p_2 p_{123}} \setminus W_{123}$ for some point q. We claim that the interior of the triangle $\triangle p_1 p_{123} q$ is contained in W_1. Indeed, any point in the interior resides on a line segment between p_1 and a point ("above q") on the half-open line segment $\overline{p_{123} q} \setminus \{q\}$, both of which are in the convex set W_1. Similarly, the interior of $\triangle p_3 p_{123} q$ is contained in W_3.

Let L' be the line parallel to L that passes through q. Note that on one side of q (the left side in the figure), L' passes through the triangle $\triangle p_1 p_{123} q$, and on the other (right) side, L' passes through $\triangle p_3 p_{123} q$. Now q is in the open set W_2, so there is an open neighborhood N of q that is entirely in W_2. Intersect N with L' and pick from the "left" side a point $l \in N \cap L'$ which is therefore in $W_1 \cap W_2 = W_{123}$. Similarly, pick a "right" point $r \in N \cap L'$ which is in W_{123}.



Now, q is on the line segment $\overline{lr}$ by construction, so $q \in W_{123}$ by convexity, which contradicts the construction of q. Thus we are done. $\square$

Proof of Theorem 3.1. Figure 1 demonstrates that the code (1) is a good-cover code.

Thus, it remains only to show that the code (1) is not convex. Assume for contradiction that there exist convex, open sets $U_1, U_2, \ldots, U_5$ in $\mathbb{R}^d$ which realize the code $\mathcal{C}$. Our strategy is to derive a contradiction with Lemma 3.2.

For $\sigma \subseteq [5]$, we define $U_\sigma := \cap_{i \in \sigma} U_i$. Let $p_{123} \in U_{123}$, $p_{145} \in U_{145}$, and $p_{2345} \in U_{2345}$; these three points exist and are distinct, because 123, 134, and 2345 are maximal codewords of $\mathcal{C}$ (so, U_{123} does not intersect U_{145}, and so on). Also, we may assume that these three points are not collinear: if they were, since the U_σ's are open sets, one of the points can be perturbed slightly.

We claim that the line segment $L = \overline{p_{123} p_{145}}$ intersects U_{134}. Indeed, by convexity, L is contained in U_1, which is covered by U_3 and U_4, so L is too. The only way for L (a connected

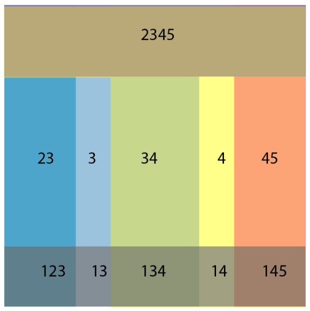
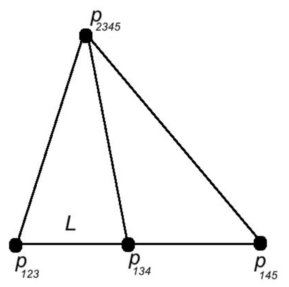


Figure 1: A good-cover realization of the code (1). More precisely, U_i is the open set formed by the union of all regions in the figure that are labeled by a codeword that contains i. For instance, U_5 is the union of the rectangular regions 2345, 45, and 145. It is straightforward to check that $\{U_i\}$ forms a good cover.

set) to be covered by two open sets is if the sets overlap and this overlap intersects L, i.e. $L \cap U_3 \cap U_4 \neq \emptyset$. Note that $L \cap U_3 \cap U_4 \subseteq U_{134}$, so we are done. Let p_{134} be a point in that intersection; thus, p_{134} is in U_{134}. So far, our points have the following configuration:



With Lemma 3.2 in mind, we define

$$W_1 := U_2 \cap U_3 \ , \qquad W_2 := U_3 \cap U_4 \ , \qquad W_3 := U_4 \cap U_5 \ .$$

First, note that the line extending the line segment $L = \overline{p_{123}p_{145}}$ contains $p_{123} \in W_1$, $p_{134} \in W_2$, and $p_{145} \in W_3$. Also, we claim that this line does not intersect the triple-intersection. Indeed, if the line contains a point r in the triple-intersection $W_1 \cap W_2 \cap W_3 = U_{2345}$, then r can not be on the line segment $L = \overline{p_{123}p_{145}}$, because L is in U_1 and 2345 is a maximal codeword of $\mathcal{C}$. Also, r can not lie to the "left" of p_{123}, because then $p_{123} \in \overline{p_{134}r} \subseteq U_5$, which is a contradiction (123 is a maximal codeword of $\mathcal{C}$). Similarly, r can not lie to the "right" of p_{134}.

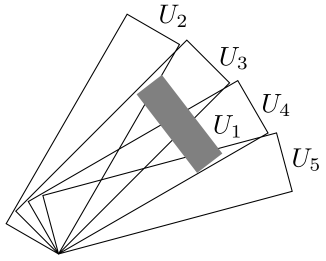
So, to reach a contradiction with Lemma 3.2, we need only check that the sets W_i satisfy the hypotheses of Lemma 3.2. First, as intersections of convex open sets, the W_i's are convex open sets in $\mathbb{R}^d$. Next, the triple-intersection $W_1 \cap W_2 \cap W_3 = U_{2345}$ contains the point p_{2345}, so is nonempty. Finally, to show that the double-intersections coincide with the triple-intersection, it suffices to show that $W_i \cap W_j \subseteq U_{2345}$ for all $1 \leq i < j \leq 3$ ($\supseteq$ holds by construction). First, $W_1 \cap W_3 = U_{2345}$ by construction. Next, $W_1 \cap W_2 = U_2 \cap U_3 \cap U_4 \subseteq U_{2345}$, because 2345 is the only codeword in $\mathcal{C}$ that contains 234. Analogously, $W_2 \cap W_3 \subseteq U_{2345}$, so we are done. $\square$

The proof of Theorem 3.1 shows that there is a "second-level" obstruction: for codes having the same simplicial complex as the counterexample code (1), if the codeword 1 is not in the code, then both 234 and 345 must be in the code (for the code to be convex). Here we see that adding these 2 codewords does make the code convex:

Proposition 3.3. *The following neural code (obtained by adding the codewords 234 and 345 to the counterexample code* (1)*) is convex, and thus a good-cover code:*

$$\mathcal{C} \quad = \quad \{\mathbf{2345},\ \mathbf{123},\ \mathbf{134},\ \mathbf{145},\ 234,\ 345,\ 13,\ 14,\ 23,\ 34,\ 45,\ 3,\ 4,\ \emptyset\}\ .$$

Proof. First note that the following is a convex realization for the code obtained by adding codewords 234, 345, 2, and 5 to the code (1):



Now it is straightforward to see that if U_2 is replaced by $U_2 \cap U_3$ and U_5 is replaced by $U_4 \cap U_5$, then the resulting sets U_i would be a convex realization that verifies our claim. □

Another way to make the counterexample code (1) convex, is simply to add the codeword 1, which makes the code max-intersection-complete (Proposition 3.4 below). Then, from the proof of Theorem 3.1, a line from p_{123} to p_{145} no longer must pass through a point $p_{134} \in U_{134}$. Thus, we do not have the same forced structure which we used to show that $\mathcal{C}$ is not convex.

Proposition 3.4. *The following neural code (obtained by adding the codeword* 1 *to the counterexample code* (1)*) is convex, and thus a good-cover code:*

$$\mathcal{C} \quad = \quad \{\mathbf{2345},\ \mathbf{123},\ \mathbf{134},\ \mathbf{145},\ 13,\ 14,\ 23,\ 34,\ 45,\ 1,\ 3,\ 4,\ \emptyset\}\ .$$

Proof. Consider the following construction. Let U_{2345} be an open cube in $\mathbb{R}^3$, centered at the origin with sides parallel to the x, y, and z axes. Then let U_{23} be a rectangular prism created by extending the cube in the positive x direction, let U_{34} be a rectangular prism created by extending the cube in the positive y direction, and let U_{45} be a rectangular prism created by extending the cube in the positive z direction. Then let U_3 be the convex hull of U_{23} and U_{34}, and let U_4 be the convex hull of U_{34} and U_{45}. Thus, we have the codewords 2345, 23, 34, 45, 3, and 4. Now, pick points $p_{23} \in U_{23}$, $p_{34} \in U_{34}$, and $p_{45} \in U_{45}$, and let U_1 be an open ϵ-neighborhood of the convex hull of p_{23}, p_{34}, and p_{45}. This creates regions corresponding to the codewords 1, 123, 134, and 145. This also must create regions corresponding to the codewords 13 and 14, since a line from 123 to 134 must pass through 13 and a line from 134 to 145 must pass through 14. It is straightforward to check that for ϵ sufficiently small, all the above codewords remain, and no new ones are created. Therefore, this is a convex realization of the code. □

4 Enumerating and classifying codes on 5 neurons

Previous classification of neural codes on up to 4 neurons as having local obstructions or not was done by hand [2, 3]. We automated this process for codes on 5 neurons using `SageMath` [11]: we first enumerated all simplicial complexes on 5 vertices, up to symmetry, and then computed for each simplicial complex the list of mandatory codewords. We describe these procedures in more detail below. Source code and the list of simplical complexes on five vertices and their mandatory codewords can be found in the appendices.

For our list of simplicial complexes, we first used `Nauty` [8] to generate a list of all connected simplicial complexes on up to 5 vertices, up to isomorphism. We worked only with connected simplicial complexes because any disconnected simplicial complex on five neurons can be expressed as the disjoint union of simplicial complexes on fewer than five vertices, and has thus been dealt with in previous work. We found that there is 1 connected simplicial complex on each of 1 and 2 vertices, 3 connected simplicial complexes on 3 vertices, 14 connected simplicial complexes on 4 four vertices, and 157 connected simplicial complexes on 5 vertices. The simplicial complexes on up to 4 vertices appears in [2, Figure 4]. Including disconnected simplicial complexes brings these counts to 1, 2, 5, 20, and 180, respectively, which agrees with the corresponding sequence in the on-line encyclopedia of integer sequences [10, A261005]. The next term in this sequence tells us that there are 16,143 simplicial complexes on six vertices, up to isomorphism. We do not produce a list of all simplicial complexes on six vertices, since we view this as too large a data set to be useful at the moment.

Our algorithm for computing the mandatory codewords of a simplicial complex follows closely the characterization local obstructions via maximal codewords (Proposition 2.8).

Algorithm 4.1 (Algorithm for computing mandatory codewords[3])**.**
Input: a simplicial complex Δ
Output: the list of mandatory codewords of Δ
Initialize: MANDATORY:= $\emptyset$
Steps:

```
1. List all nonempty intersections of facets of $\Delta$.
2. Compute the link of each nonempty intersection of facets.
3. Compute the reduced homology groups of each link.
4. For each reduced homology group which is nontrivial, add the corresponding intersection
   of facets to MANDATORY.
5. Return MANDATORY.
```

5 The tree criterion for max-intersection-incomplete codes

Recall that mandatory codewords of a simplicial complex necessarily are intersections of facets, but not vice-versa. In this section, we present a new criterion for an intersection of facets to be non-mandatory (Theorem 5.2), and then show that this criterion characterizes all intersection-*incomplete* codes without local obstructions for codes on at most 5 neurons (Theorem 5.7). However, this criterion is insufficient for codes on 6 or more neurons (Example 5.9). We end the section by showing that certain codes satisfying our new criterion are in fact convex with minimal embedding dimension 1 (Proposition 5.12).

We begin with several definitions. First, recall that for a finite collection $\mathcal{W} = \{W_1, W_2, \ldots, W_n\}$ of subsets of a set X, the *nerve* of $\mathcal{W}$ is the simplicial complex that records the intersection patterns among the sets:

$$\mathcal{N}(\mathcal{W}) \ := \ \{I \subseteq [n] \mid \cap_{i \in I} W_i \text{ is nonempty}\} \ .$$

Next, for a face σ of a simplicial complex Δ, we let $\mathcal{M}_\Delta(\sigma)$ denote the set of all facets (maximal faces) of Δ that contain σ. Thus, if $\Delta = \Delta(\mathcal{C})$ for some code $\mathcal{C}$, then $\mathcal{M}_\Delta(\sigma)$ is the set of all maximal codewords of $\mathcal{C}$ that contain the codeword σ. Finally, we let $\mathcal{L}_\Delta(\sigma)$ denote the set of facets of $\mathrm{Lk}_\Delta(\sigma)$; it is straightforward to see that these facets are obtained by removing σ from the facets of Δ that contain σ:

$$\mathcal{L}_\Delta(\sigma) \ = \ \{(M \setminus \sigma) \mid M \in \mathcal{M}_\Delta(\sigma)\} \ .$$

We will need the following version of the nerve lemma [1, Theorem 6 and Remark 7]:

Lemma 5.1. *Let $\mathcal{D} = (\Delta_i)_{i \in I}$ be a family of sub-complexes of a connected simplicial complex Δ for which:*

(1) $\Delta = \bigcup_{i \in I} \Delta_i$, *and*

[3]Eventually, Algorithm 4.1 will fail (for some simplicial complexes), because having trivial homology groups does not imply contractibility in general, but does for simplicial complexes on few vertices.

(2) every finite nonempty intersection $\Delta_{i_1} \cap \Delta_{i_2} \cap \cdots \cap \Delta_{i_k}$ is contractible.

Then Δ is homotopy-equivalent to the nerve of the Δ_i's: $\Delta \simeq \mathcal{N}(\mathcal{D})$.

We apply Lemma 5.1 in the following setting: the simplicial complex is a link $\text{Lk}_\Delta(\sigma)$ and $\mathcal{D}$ is $\mathcal{L}_\Delta(\sigma)$, the facets of the link. Clearly the union of the facets is equal to the link, so condition (1) of the lemma holds. Also, nonempty intersections of any faces of a simplicial complex are themselves faces, which are contractible, so (2) holds. Therefore, the lemma implies that:

$$\text{Lk}_\Delta(\sigma) \ \simeq \ \mathcal{N}\left(\mathcal{L}_\Delta(\sigma)\right) \ . \tag{2}$$

Theorem 5.2 (Tree criterion for mandatory codewords)**.** *Let Δ be a simplicial complex, and let $\sigma \in \Delta$ be a nonempty intersection of facets of Δ. If the nerve $\mathcal{N}\left(\mathcal{L}_\Delta(\sigma)\right)$ is a tree graph, then σ is not a mandatory codeword of Δ, i.e.* $\text{Lk}_\Delta(\sigma)$ *is contractible.*

Proof. The theorem follows directly from the definition of mandatory (Definition 2.9), the homotopy-equivalence (2), and the fact that tree graphs are contractible. □

Remark 5.3. The nerve $\mathcal{N}\left(\mathcal{L}_\Delta(\sigma)\right)$ is a tree graph if and only if all triple-wise intersections among facets of Δ that contain σ (i.e. elements of $\mathcal{M}_\Delta(\sigma)$) are equal to σ.

What Theorem 5.2 says is that in light of the characterization of local obstructions in terms of intersections of facets (Proposition 2.8), those codewords that satisfy the tree criterion do not generate local obstructions. We rephrase this in Corollary 5.5 below via the following definition:

Definition 5.4. A code $\mathcal{C}$ on n neurons *satisfies the tree criterion* if for every codeword $\sigma \subseteq [n]$ that is not in $\mathcal{C}$ and is a nonempty intersection of maximal codewords of $\mathcal{C}$, the nerve $\mathcal{N}\left(\mathcal{L}_\Delta(\sigma)\right)$ is a tree graph.

Corollary 5.5. *If a neural code $\mathcal{C}$ satisfies the tree criterion, then $\mathcal{C}$ has no local obstructions.*

Example 5.6. Returning again to our counterexample code (1), recall from Example 2.13 that 1 is the only nonempty intersection of maximal codewords that is not in the code, and that its link $\text{Lk}_{\Delta(\mathcal{C})}(1) = \{\mathbf{23}, \mathbf{34}, \mathbf{45}, 2, 3, 4, 5\}$ is a path of length 3. Therefore, the set of facets of the link is $\mathcal{L}_\Delta(\sigma) = \{23, 34, 45\}$, and hence the nerve $\mathcal{N}\left(\mathcal{L}_\Delta(\sigma)\right)$ is a path of length 2. We conclude that the code satisfies the tree criterion. (We already knew that it has no local obstructions.)

5.1 Using the tree criterion to classify codes with no local obstructions

Max-intersection-complete codes (Definition 2.3) vacuously satisfy the tree criterion (by Proposition 2.8). Also, recall that max-intersection-complete codes on at most 4 neurons are precisely the codes with no local obstructions; in fact, they are convex (by Proposition 2.12). Thus, the converse of Corollary 5.5 is true for up to 4 neurons. In fact, our next result extends this converse to 5 neurons. However, this converse is false for codes on six or more neurons (Example 5.9).

Theorem 5.7 (Tree criterion characterization of codes with no local obstructions on up to 5 neurons)**.** *For a code $\mathcal{C}$ on at most 5 neurons, $\mathcal{C}$ has no local obstructions if and only if $\mathcal{C}$ satisfies the tree criterion.*

Proof of Theorem 5.7. As explained above, the $\Leftarrow$ direction of Theorem 5.7 is Corollary 5.5. Our proof of the $\Rightarrow$ direction requires the next lemma, due to Curto *et al.* [2, Lemma 2.11 and subsequent discussion]. Recall that a simplicial complex Δ on $[n]$ is a *cone* if there exists $i \in [n]$ (called a *cone point*) such that every nonempty facet of Δ contains i. For instance, the simplicial complex $\{\{123\}, \{34\}\}$ is a cone with cone point 3, whereas the simplicial complex $\{\{123\}, \{34\}, \{45\}\}$ is not a cone, since there is no point contained in every facet.

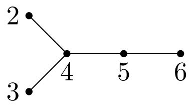
Lemma 5.8. *A simplicial complex Γ is not a cone if and only if there exists a simplicial complex Δ and a face σ of Δ such that (1) σ is an intersection of facets of Δ and (2)* $\text{Lk}_\Delta(\sigma) = \Gamma$.

Recall that the $\Leftarrow$ direction is Corollary 5.5. For the $\Rightarrow$ direction, let $\mathcal{C}$ be a code on n neurons, where $n \leq 5$, that has no local obstructions. Let $\sigma \subseteq [n]$ be a codeword that is not in $\mathcal{C}$ and is a nonempty intersection of maximal codewords of $\mathcal{C}$. Then $\mathrm{Lk}_{\Delta(\mathcal{C})}(\sigma)$ is (1) a simplicial complex on at most 4 vertices (because σ is nonempty), (2) contractible (by Proposition 2.8: $\mathcal{C}$ has no local obstructions) and thus nonempty, and (3) not a cone (by Lemma 5.8). Among the 28 simplicial complexes on up to 4 vertices (depicted in [2, Figure 4]), only one is a contractible non-cone: P_3, the path of length 3. Thus, $\mathrm{Lk}_{\Delta(\mathcal{C})}(\sigma) \cong P_2$ (the path of length 2), which is a tree graph. Hence, by definition, $\mathcal{C}$ satisfies the tree criterion. $\square$

Example 5.9 (A convex code that the tree criterion misses)**.** Let

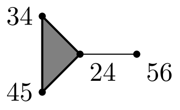
$$\mathcal{C} \ = \ \{\mathbf{124},\ \mathbf{134},\ \mathbf{145},\ \mathbf{156},\ 14,\ 15,\ \emptyset\}\ ,$$

so $\Delta(\mathcal{C})$ is the cone, with cone point 1, over the following graph:



We claim that $\mathcal{C}$ has no local obstructions. Indeed, the codeword 1 is the unique nonempty intersection of maximal codewords of $\mathcal{C}$ that is not in $\mathcal{C}$, and $\mathrm{Lk}_{\Delta(\mathcal{C})}(1)$ is the above tree graph, which is contractible.

To complete the proof, we now show that $\mathcal{C}$ does not satisfy the tree criterion. To see this, note that $\sigma = \{1\}$ is not in $\mathcal{C}$ and is the nonempty intersection of all maximal codewords of $\mathcal{C}$. However, the nerve $\mathcal{N}\left(\mathcal{L}_{\Delta(\mathcal{C})}(\sigma)\right)$ is the following non-tree simplicial complex:



This shows that $\mathcal{C}$ fails to satisfy the tree criterion.

5.2 Proving convexity for a special case when the nerve is a path

Here we show that a certain family of codes that satisfy the tree criterion is in fact convex with minimal embedding dimension 1. The corresponding simplicial complexes only contain one non-mandatory codeword, and the corresponding nerve is the simplest type of tree: a path. These simplicial complexes include cones over a path:

Definition 5.10. Let $\mathcal{M}$ denote the set of facets of a simplicial complex Δ. We say that Δ is a *coned path* if:

1. the intersection of all facets is nonempty: $\sigma := \bigcap_{M \in \mathcal{M}} M \neq \emptyset$, and
2. the nerve $\mathcal{N}(\mathcal{L}_\Delta(\sigma))$ is a path graph (of length at least 1).

Next we will show that for each simplicial complex, there is a unique minimal code with no local obstructions, and then for the case of coned paths, prove that these minimal codes are convex with minimal embedding 1 (Proposition 5.12). To introduce minimal codes, recall from Proposition 2.8 that for a given simplicial complex Δ, a code $\mathcal{C}$ with $\Delta(\mathcal{C}) = \Delta$ has no local obstructions if and only if it contains the following code:

$$\mathcal{C}_{\min}(\Delta) \ := \ \{\text{mandatory codewords of } \Delta(\mathcal{C})\} \ \cup \ \{\text{facets of } \Delta\} \ \cup \ \{\emptyset\}\ .$$

In other words, $\mathcal{C}_{\min}(\Delta)$ is the minimal code with simplicial complex Δ that has no local obstructions. For instance, our counterexample code (1) is the minimal code for its simplicial complex. Now we determine this minimal code for the case of a coned path:

Lemma 5.11. *Let σ be the (nonempty) intersection of all the facets of a coned path Δ. Then:*

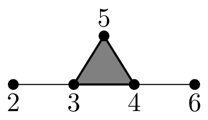
1. *there exists an ordering of the facets of* $\mathrm{Lk}_\Delta(\sigma)$*:*
$$\mathcal{L}_\Delta(\sigma) \;=\; \{N_1, N_2, \ldots, N_m\}$$
such that $N_i \cap N_{i+1} =: \tau_i$ is nonempty and all other pairwise intersections are empty: $N_i \cap N_j = \emptyset$ if $|i-j| \geq 2$, and
2. $\{\text{mandatory codewords of } \Delta\} = \{\sigma \cup \tau_i \mid 1 \leq i \leq m-1\}$.

Proof. Part 1 follows immediately from the fact that the nerve $\mathcal{N}(\mathcal{L}(\sigma))$ is a path graph. For part 2, we begin by noting that $\mathcal{M} = \{\sigma \cup N_i \mid 1 \leq i \leq m\}$ is the set of facets of Δ. By part 1, the only nonempty intersections of facets are σ and the $\sigma \cup \tau_i$'s, and σ is non-mandatory by the tree criterion (Theorem 5.2). So, we need only show that $\mathrm{Lk}_\Delta(\sigma \cup \tau_i)$ is non-contractible (by Proposition 2.8). Indeed, we will see that this link is disconnected. By part 1, the only facets that contain $\sigma \cup \tau_i$ are $\sigma \cup N_i$ and $\sigma \cup N_{i+1}$, and by definition neither facet contains the other. Thus $\mathrm{Lk}_\Delta(\sigma \cup \tau_i)$ is the disjoint union of two full simplices: one on $N_i \setminus \tau_i$ and one on $N_{i+1} \setminus \tau_i$., and thus is disconnected. □

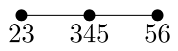
Proposition 5.12. *If Δ is a coned path, then the minimal code $\mathcal{C}_{\min}(\Delta)$ is convex with minimal embedding dimension 1.*

Proof. Let N_i be as in Lemma 5.11, so that $\mathcal{M} = \{\sigma \cup N_1, \sigma \cup N_2, \ldots, \sigma \cup N_m\}$ is the set of facets of Δ. Also, let $\tau_i = N_i \cap N_{i+1}$ for $1 \leq i \leq m-1$. Then, Lemma 5.11 implies that $\mathcal{C}_{\min}(\Delta) = \{\sigma \cup \tau_i \mid 1 \leq i \leq m-1\} \cup \mathcal{M} \cup \{\emptyset\}$. We show that this code is convex with minimal embedding dimension 1 via a convex realization in $\mathbb{R}$ so that the region for $\sigma \cup N_1$ is the interval $(0,1)$, the $\sigma \cup \tau_1$ region is $[1,2]$, the $\sigma \cup N_2$ region is $(2,3)$, the region for $\sigma \cup \tau_2$ is $[3,4]$, and so on. More precisely, the receptive fields U_j for each neuron j are:
$$U_j \;=\; \begin{cases} (0,\ 2m-1) & \text{if } j \in \sigma \\ (2i-2,\ 2i+1) & \text{if } j \in \tau_i, \text{ for some } 1 \leq i \leq m-1 \\ (2i-2,\ 2i-1) & \text{if } j \in \left(N_i \setminus \left(\bigcup_{k=1}^{m-1} \tau_k\right)\right), \text{ for some } 1 \leq i \leq m\ . \end{cases}$$
It is straightforward to check that this code has the regions described above. □

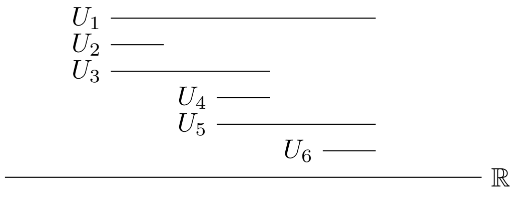
Example 5.13. Let Δ be the cone, with cone point 1, over the following simplicial complex:



Then Δ is a coned path; indeed, the intersection of all facets of Δ is $\{1\}$, and the nerve $\mathcal{N}(\mathcal{L}_\Delta(\sigma))$ is the following path of length 2:



Following the proof of Proposition 5.12, the minimal code is $\mathcal{C}_{\min}(\Delta) = \{\mathbf{123},\ \mathbf{1345},\ \mathbf{156},\ 13,\ 15,\ \emptyset\}$, and a 1-dimensional convex realization is as follows, where the open intervals U_i are depicted above the real line for clarity:



From left to right, the nonempty codewords are 123, 13, 1345, 15, and 156.

6 Discussion

We resolved the question of whether all neural codes with no local obstructions are convex. Nonetheless, several related questions remain open. Are all neural codes with no local obstructions good-cover codes? Are all max-intersection-complete codes (i.e. closed under maximal intersection) convex? Even for codes on 5 neurons, these questions are unresolved. Our enumeration of codes on 5 neurons without local obstructions is a step toward attacking these problems.

Additional questions arise from our counterexample code. First, is there another code with the same simplicial complex as the counterexample code that also is non-convex despite having no local obstructions? We generalize this question as follows. We defined the minimal code on a simplicial complex Δ to be the smallest neural code $\mathcal{C}$ with no local obstructions such that $\Delta(\mathcal{C}) = \Delta$. (We could also consider a minimal convex code with a given simplicial complex, although this is not, in general, unique: recall the codes discussed at the end of Section 3.) If the minimal code on a simplicial complex is convex, are all codes on the same simplicial complex that contain the minimal code convex? If true, this would reduce the classification of convex codes to the determination of the minimal convex codes on each simplicial complex.

Finally, we pose some questions that arise from our counterexample code. First, our proof that this code is non-convex hinged upon a forced two-dimensional structure. Similar techniques have helped us to draw receptive fields for other five-neuron codes which permitted exclusion of an intersection of maximal codewords. Can we use these techniques to draw place fields in general, or to determine bounds on minimal embedding dimension?

Additionally, can we characterize a new obstruction such that neural codes are convex if and only if they do not have this new obstruction? Can we give an algebraic signature for such an obstruction (see [2])? Whatever the signature of our new obstruction is, it is clear that it cannot have as simple an interpretation as the local obstruction. While a local obstruction specifies a set of missing codewords, all of which must be added to the code to resolve the obstruction, the obstruction to convexity present in our counterexample code has a more complicated structure: adding *either* the codeword 1 or both the codewords 234 and 345 makes the code convex. One form this new obstruction could take is the requirement that convex codes either be max-intersection-complete or satisfy some other condition, yet to be determined.

Acknowledgments

CL and ZW conducted this research as part of the NSF-funded REU in the Department of Mathematics at Texas A&M University (DMS-1460766), in which AS served as mentor. The authors benefited from guidance from Lauren Grimley and Jacob White, and from discussions with Carina Curto, Bryan Félix, Chad Giusti, Elizabeth Gross, Vladimir Itskov, William Kronholm, Sean Owen, and Nora Youngs. AS was supported by the NSF (DMS-1312473).

References

[1] Anders Björner. Nerves, fibers and homotopy groups. *J. Comb. Theory A*, 102:88–93, 2003.

[2] Carina Curto, Elizabeth Gross, Jack Jeffries, Katie Morrison, Mohamed Omar, Zvi Rosen, Anne Shiu, and Nora Youngs. What makes a neural code convex? Available at `arXiv:1508.00150`.

[3] Carina Curto, Vladimir Itskov, Alan Veliz-Cuba, and Nora Youngs. The neural ring: an algebraic tool for analyzing the intrinsic structure of neural codes. *Bull. Math. Biol.*, 75(9):1571–1611, 2013.

[4] Chad Giusti and Vladimir Itskov. A no-go theorem for one-layer feedforward networks. *Neural Comput.*, 26(11):2527–2540, 2014.

[5] Chad Giusti, Vladimir Itskov, and William Kronholm. On convex codes and intersection violators. Preprint.

[6] Eduard Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresber. Dtsch. Math.-Ver.*, 32:175–176, 1923.

[7] Gil Kalai. Intersection patterns of convex sets. *Israel J. Math.*, 48(2-3):161–174, 1984.

[8] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014.

[9] T. McKee and F. McMorris. *Topics in Intersection Graph Theory.* Society for Industrial and Applied Mathematics, 1999.

[10] N. J. A. Sloane. The on-line encyclopedia of integer sequences. `http://oeis.org`.

[11] W. A. Stein et al. *Sage Mathematics Software (Version 6.7).* The Sage Development Team, 2015. `http://www.sagemath.org`.

[12] Martin Tancer. Intersection patterns of convex sets via simplicial complexes: a survey. In *Thirty essays on geometric graph theory*, pages 521–540. Springer, New York, 2013.

Appendix 1: Source code for enumerating simplicial complexes and checking for local obstructions

Our code is available on `SageMathCloud`[4].

6.1 Enumerating simplicial complexes

Here are the functions we used to generate the set of simplicial complexes on five vertices, up to isomorphism. `Nauty` generates lists of hypergraphs, rather than of simplicial complexes, so we needed to sort through the output to find those hypergraphs which contained only facets.

```
#input: two ordered tuples of integers, the smaller one listed first
#output: boolean, whether the smaller is contained in the larger
def containment_check(smaller, larger):
    counter_s = 0
    counter_l = 0
    while (counter_l < len(larger)):
        #print "counters: S-"+str(counter_s)+ "   L-" +str(counter_l)
        if (counter_s >= len(smaller)):
            return true
        elif (smaller[counter_s] == larger[counter_l]):
            counter_s = counter_s + 1
            counter_l = counter_l + 1
            if counter_s==len(smaller):
                return true
        elif (smaller[counter_s] < larger[counter_l]):
            #print "first false"
            return false
        elif (smaller[counter_s] > larger[counter_l]):
            counter_l = counter_l + 1
    #print "second"
    return false

#We used nauty() to enumerate hypergraphs up to isomorphism
#nauty(number_of_sets, number_of_vertices, vertex_min_degree=1,connected=True)
#to enumerate all simplicial complexes on n vertices, we let number_of_sets range from 1 to (n choose 2)/2, since a
#  simplicial complex on n vertices cannot have more than (n choose 2)/2 facets


#input: tuple of tuples of integers. tuple of tuple is ordered by length; tuples of integers are in ascending order
#output: whether there are any containments in that tuple
def max_only_check(code):
    for i in range(0,len(code)):
        for j in range((i+1),len(code)):
            if (len(code[j])>len(code[i])):
                if (containment_check(code[i], code[j]) == true):
                    return false
    else:
        return true

#input: a list of hypergraphs, presented as list of tuples of tuples, ordered as specified in max_only_check
#output: a tuple containing (a list of the simplicial complexes described in terms of maximal faces, "bad:",
# a list of hypergaphs that contain non-maximal faces )
def check_list_for_maximality(test):
```

[4] `https://cloud.sagemath.com/projects/8fdd3fd5-5b65-4059-8e3f-95e02b104e84/files/obstructions_to_convexity_in_neural_codes.sagews`

```
    good_complexes = []
    bad_complexes = []
    for code in test:
        if max_only_check(code):
            good_complexes.append(code)
        else:
            bad_complexes.append(code)
    return (good_complexes, "bad: ", bad_complexes)

#input: a list of hypergraphs, presented as list of tuples of tuples, ordered as specified in max_only_check
#output: the number of simplicial complexes, specified in terms of maximal elements, in that list
def numerical_check_list_for_maximality(test):
    num_simplicial_complexes = 0
    for code in test:
        if max_only_check(code):
            num_simplicial_complexes= num_simplicial_complexes+1
    return num_simplicial_complexes
```

6.2 Checking for (homological) local obstructions and enumerating mandatory codewords

Below are the functions we used to classify neural codes on five neurons.

```
#given a set of sets, returns the intersection of all sets in that set.
def intersect_all_c(setList):
    first = setList[0]
    length = (setList).cardinality()
    for i in range (0, length):
        first = first.intersection(setList[i])
    return first

#given a set of sets, generates all intersections of collections of those sets, and returns those which are not already in the code
#if fed a set of facets, will generate all intersections of facets
def intersections_c(setList):
    result= []
    subsets = Subsets(setList)
    for i in range(1, (subsets).cardinality()):
        result.append(intersect_all_c(subsets[i]))
    return Set(result)-setList

#given a homology, returns true if the homology group is trivial in every dimension
def is_trivial_c(homology):
    length = len(homology.viewvalues())
    trivial_complex = SimplicialComplex([range(0, length)])
    return homology == trivial_complex.homology()

#the function link_checker takes a neural code as an input and outputs the links of all intersections of elements in the code that
# are not in the code
def link_checker_c(code):
    simplicial_complex = SimplicialComplex(code)
    maximal = simplicial_complex.facets()
    links_to_check = intersections_c(code)
    bad_links = []
    homologies = []
    for i in range(0,(links_to_check).cardinality()):
        current_link = simplicial_complex.link(Simplex(links_to_check[i]))
        if (is_trivial_c(current_link.homology())==false):
            bad_links.append(current_link)
            homologies.append(current_link.homology())
    return (links_to_check, bad_links, homologies)

#this function takes a neural code as input and outputs true if has no local obstructions and false otherwise
def bool_link_checker(code):
    simplicial_complex = SimplicialComplex(code)
    maximal = simplicial_complex.facets()
    if maximal.cardinality()==1:
        return true
    else:
        links_to_check = intersections_c(code)
        for i in range(0,(links_to_check).cardinality()):
            current_link = simplicial_complex.link(Simplex(links_to_check[i]))
            if (is_trivial_c(current_link.homology())==false):
                return false
        return true

#Given a neural code, returns detailed information about it. Will not give the right result if the code does not contain the empty set.
def detailed_link_checker(code):
    simplicial_complex = SimplicialComplex(code)
    maximal = simplicial_complex.facets()
    n_max = maximal.cardinality()
    if n_max == 1:
        print "The neural code " + str(code) + " is convex because it only has one maximal face."
        return true
    else:
        links_to_check = intersections_c(code)
        missing_codewords = []
        for i in range(0,(links_to_check).cardinality()):
            current_link = simplicial_complex.link(Simplex(links_to_check[i]))
            if (is_trivial_c(current_link.homology())==false):
                missing_codewords.append(links_to_check[i])
        if (missing_codewords == []):
            print "The neural code " + str(code) + " has no local obstructions, and has maximal faces " + str(maximal) + " ."
            return true
        else:
            print "The neural code " + str(code) + " has a local obstruction. This could be remedied if the following codewords were added:
             " + str(missing_codewords)

#input: a neural code
#output: a list of mandatory codewords (required codewords for the neural code to not have local obstructions)
def another_link_checker(code):
    simplicial_complex = SimplicialComplex(code)
    maximal = simplicial_complex.facets()
    n_max = maximal.cardinality()
    links_to_check = intersections_c(code)
    missing_codewords = []
    for i in range(0,(links_to_check).cardinality()):
        current_link = simplicial_complex.link(Simplex(links_to_check[i]))
        if (is_trivial_c(current_link.homology())==false):
            missing_codewords.append(links_to_check[i])
    return missing_codewords

#input: a list of neural codes
#output: a list of lists of mandatory codewords
def required_codewords(codes):
    result = []
    for code in codes:
        result.append(another_link_checker(code))
    return result

#input: a neural code
#output: a tuple containing (missing mandatory codewords, missing intersections of maximal codewords which are not mandatory,
# number of faces of the simplicial complex of the code which are neither maximal nor mandatory codewords)
#this is most useful to run on a neural code described only in terms of maximal codewords--you then get the mandatory codewords,
# the non-mandatory intersections of maximal codewords, and the number of optional facets of the simplicial complex--this lets you
# find the number of neural codes on the simplex with no local obstructions
def optional_max_inter(code):
    simplicial_complex = SimplicialComplex(code)
    maximal = simplicial_complex.facets()
    links_to_check = intersections_c(code)
    missing_codewords = []
    for i in range(0,(links_to_check).cardinality()):
            current_link = simplicial_complex.link(Simplex(links_to_check[i]))
            if (is_trivial_c(current_link.homology())==false):
                missing_codewords.append(links_to_check[i])
    num_optional_faces = len([f for f in simplicial_complex.face_iterator()])-len(simplicial_complex.facets())-len(missing_codewords)
    return (missing_codewords, links_to_check.difference(Set(missing_codewords)), num_optional_faces)


#input: list of tuples of tuples
#output: list of sets of sets, each set of sets containing the empty set
#converts output from nauty() into a format which works well with the neural code functions
def tuples_to_sets(the_list):
    outputlist = []
    for big_tuple in the_list:
        outerset = Set([])
        for small_tuple in big_tuple:
            innerset = Set(small_tuple)
            outerset = outerset.union(Set([innerset]))
            outerset = outerset.union(Set([Set([])]))
        outputlist.append(outerset)
    return outputlist

#input: list of tuples of tuples, each tuple of tuple giving the facets of a simplicial complex
#output: a latex table whose rows are, from left to right:
#facets of simplicial complex, mandatory codewords, non-mandatory intersections of facets, number of non-facet faces
def make_table(complex_list):
    complex_set = tuples_to_sets(complex_list)
    missing_codeword_list = []
    optional_codeword_list = []
    num_optional_list = []
    for current_complex in complex_set:
        current_tuple = optional_max_inter(current_complex)
        missing_codeword_list.append(current_tuple[0])
        optional_codeword_list.append(current_tuple[1])
        num_optional_list.append(current_tuple[2])
    return latex(table(columns=[complex_set, missing_codeword_list, optional_codeword_list, num_optional_list]))


#input: a list of neural codes as tuples of tuples
#output: a list of intersections of facets whose links are convex
def find_optional_max_inter(code_list):
    result = []
    for code in code_list:
        result.append(optional_max_inter(code))
    return result
```

Appendix 2: Classification of codes on 5 neurons with no local obstructions

Below we list (up to isomorphism) all 157 connected simplicial complexes on 5 vertices; here the simplicial complexes are listed by their facets, and the vertex set is $\{0, 1, 2, 3, 4\}$. For each simplicial complex, we list the mandatory codes, the non-mandatory intersections of facets, the number of non-mandatory codewords, and the number of codes with no local obstructions.

Simplicial complex (only facets listed)	Mandatory codewords	Non-mandatory intersections of facets	Number of non-mandatory codewords	Number of codes with no local obstructions
{{0, 1, 2, 3, 4}}	[]	{}	30	1073741824
{ {0, 2, 3, 4}, {0, 1}}	[{0}]	{}	14	16384
{{0, 1, 2}, {0, 3, 4}}	[{0}]	{}	10	1024
{{0, 1, 3, 4}, {0, 1, 2}}	[{0, 1}]	{}	16	65536
{{0, 1, 2, 3}, {0, 1, 2, 4}}	[{0, 1, 2}]	{}	20	1048576
{ {0, 3, 4}, {0, 2}, {0, 1}}	[{0}]	{}	7	128
{ {1, 3, 4}, {0, 2}, {0, 1}}	[{0}, {1}]	{}	6	64
{{1, 2, 3, 4}, {0, 2}, {0, 1}}	[{2}, {0}, {1}]	{}	12	4096
{{2, 3}, {0, 2, 4}, {0, 1}}	[{2}, {0}]	{}	6	64
{{0, 2, 3}, {0, 2, 4}, {0, 1}}	[{0, 2}, {0}]	{}	8	256
{{0, 2, 3}, {0, 1}, {1, 2, 4}}	[{2}, {0}, {1}]	{}	8	256
{{0, 2, 3}, {2, 3, 4}, {0, 1}}	[{2, 3}, {0}]	{}	8	256
{{1, 2, 3, 4}, {0, 2, 3}, {0, 1}}	[{2, 3}, {0}, {1}]	{}	14	16384
{{1, 2, 3, 4}, {0, 2, 3, 4}, {0, 1}}	[{2, 3, 4}, {0}, {1}]	{}	18	262144
{{0, 1, 4}, {0, 1, 2}, {0, 1, 3}}	[{0, 1}]	{}	11	2048
{{0, 1, 2}, {0, 2, 3, 4}, {0, 1, 3}}	[{0, 2}, {0, 3}, {0}, {0, 1}]	{}	14	16384
{{0, 1, 2}, {0, 1, 3}, {0, 3, 4}}	[{0, 3}, {0, 1}]	{{0}}	10	1024
{ {0, 1, 2}, {1, 2, 3}, {0, 3, 4}}	[{1, 2}, {3}, {0}]	{}	10	1024
{{1, 2, 3, 4}, {0, 1, 2}, {0, 3, 4}}	[{3, 4}, {0}, {1, 2}]	{}	16	65536
{{0, 1, 3, 4}, {0, 1, 2}, {0, 2, 3, 4}}	[{0, 3, 4}, {0, 2}, {0}, {0, 1}]	{}	18	262144
{{0, 1, 2, 3}, {0, 1, 3, 4}, {0, 1, 2, 4}}	[{0, 1, 4}, {0, 1, 2}, {0, 1, 3}, {0, 1}]	{}	20	1048576
{{1, 2}, {0, 3, 4}, {0, 2}, {0, 1}}	[{2}, {0}, {1}]	{}	5	32
{ {0, 4}, {0, 2}, {0, 3}, {0, 1}}	[{0}]	{}	4	16
{ {1, 4}, {0, 2}, {0, 3}, {0, 1}}	[{0}, {1}]	{}	3	8
{ {1, 2, 4}, {0, 2}, {0, 3}, {0, 1}}	[{2}, {0}, {1}]	{}	5	32
{{1, 2, 3, 4}, {0, 2}, {0, 3}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	11	2048
{{2, 4}, {1, 3}, {0, 2}, {0, 1}}	[{2}, {0}, {1}]	{}	2	4
{{1, 3}, {1, 2, 4}, {0, 2}, {0, 1}}	[{2}, {0}, {1}]	{}	5	32
{{1, 3}, {0, 2}, {2, 3, 4}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	4	16
{{3, 4}, {1, 2, 3}, {0, 2}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	4	16
{ {1, 2, 4}, {1, 2, 3}, {0, 2}, {0, 1}}	[{1, 2}, {2}, {0}, {1}]	{}	6	64
{{1, 2, 3}, {0, 3, 4}, {0, 2}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	7	128
{ {1, 3, 4}, {1, 2, 3}, {0, 2}, {0, 1}}	[{1, 3}, {2}, {0}, {1}]	{}	6	64
{{1, 2, 3, 4}, {0, 3, 4}, {0, 2}, {0, 1}}	[{3, 4}, {2}, {0}, {1}]	{}	13	8192
{ {1, 3, 4}, {0, 3, 4}, {0, 2}, {0, 1}}	[{3, 4}, {0}, {1}]	{}	7	128
{ {1, 3, 4}, {0, 2}, {2, 3, 4}, {0, 1}}	[{3, 4}, {2}, {0}, {1}]	{}	6	64
{{1, 2, 4}, {2, 3}, {0, 2, 4}, {0, 1}}	[{2, 4}, {2}, {0}, {1}]	{}	6	64
{{1, 3, 4}, {2, 3}, {0, 2, 4}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	6	64
{{0, 2, 3}, {1, 2, 3}, {0, 2, 4}, {0, 1}}	[{0, 2}, {2, 3}, {0}, {1}]	{{2}}	8	256
{{0, 2, 3}, {1, 2, 3}, {2, 3, 4}, {0, 1}}	[{2, 3}, {0}, {1}]	{}	9	512
{{0, 2, 3}, {0, 3, 4}, {0, 2, 4}, {0, 1}}	[{0, 4}, {0, 2}, {0, 3}, {0}]	{}	7	128
{{0, 2, 3}, {1, 3, 4}, {0, 2, 4}, {0, 1}}	[{4}, {0, 2}, {3}, {0}, {1}]	{}	8	256
{{0, 2, 3}, {2, 3, 4}, {0, 2, 4}, {0, 1}}	[{2, 4}, {0, 2}, {2, 3}, {2}, {0}]	{}	6	64
{{1, 2, 3, 4}, {0, 2, 3}, {0, 2, 4}, {0, 1}}	[{2, 4}, {2, 3}, {0, 2}, {0}, {1}, {2}]	{}	12	4096
{{0, 2, 3}, {1, 2, 4}, {2, 3, 4}, {0, 1}}	[{2, 4}, {2, 3}, {0}, {1}]	{{2}}	8	256

{{1, 2, 3, 4}, {0, 2, 3}, {0, 1, 2}, {0, 1, 3}}	[{3}, {0}, {1, 2}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	8	256
{{0, 2, 3}, {0, 1, 2}, {0, 1, 3}, {0, 1, 4}}	[{0, 2}, {0, 3}, {0}, {0, 1}]	{}	9	512
{{0, 1, 4}, {0, 1, 2}, {0, 2, 3, 4}, {0, 1, 3}}	[{0, 4}, {0, 2}, {0, 3}, {0}, {0, 1}]	{}	14	16384
{{1, 2, 3, 4}, {0, 1, 2}, {0, 2, 3, 4}, {0, 1, 3}}	[{3}, {0}, {1, 2}, {2, 3, 4}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	12	4096
{{0, 1, 2}, {0, 3, 4}, {0, 2, 4}, {0, 1, 3}}	[{0, 4}, {0, 2}, {0, 3}, {0}, {0, 1}]	{}	8	256
{{1, 2, 3, 4}, {0, 1, 2}, {0, 3, 4}, {0, 1, 3}}	[{3}, {1, 2}, {3, 4}, {1}, {1, 3}, {0, 3}, {0, 1}]	{{0}}	12	4096
{{0, 1, 2}, {1, 2, 3}, {0, 3, 4}, {0, 1, 3}}	[{1}, {1, 3}, {1, 2}, {0, 3}, {0, 1}]	{{3}, {0}}	8	256
{{0, 1, 4}, {1, 2, 3}, {0, 3, 4}, {0, 1, 2}}	[{3}, {1, 2}, {0, 4}, {0, 1}]	{{0}, {1}}	10	1024
{{0, 1, 2}, {1, 2, 3}, {0, 3, 4}, {1, 2, 4}}	[{1, 2}, {4}, {3}, {0}]	{}	11	2048
{{1, 2, 3, 4}, {0, 1, 3, 4}, {0, 1, 2}, {0, 2, 3, 4}}	[{0, 3, 4}, {0}, {1, 2}, {2, 3, 4}, {3, 4}, {1, 3, 4}, {0, 2}, {1}, {2}, {0, 1}]	{}	14	16384
{ {0, 1, 3, 4}, {0, 1, 2, 3}, {0, 2, 3, 4}, {0, 1, 2, 4}}	[{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {0}, {0, 4}, {0, 2, 4}, {0, 2}, {0, 1, 4}, {0, 1, 2}, {0, 3}, {0, 1}]	{}	14	16384
{{1, 3, 4}, {0, 3, 4}, {0, 2}, {1, 2}, {0, 1}}	[{3, 4}, {2}, {0}, {1}]	{}	6	64
{{0, 2}, {1, 2}, {1, 4}, {0, 3}, {0, 1}}	[{2}, {0}, {1}]	{}	2	4
{{1, 3, 4}, {0, 2}, {1, 2}, {0, 3}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	4	16
{{0, 2}, {1, 2}, {0, 4}, {0, 3}, {0, 1}}	[{2}, {0}, {1}]	{}	2	4
{{0, 2}, {0, 4}, {1, 2, 3}, {0, 3}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	4	16
{{0, 2}, {1, 2, 3, 4}, {0, 4}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	10	1024
{{0, 2}, {0, 1}, {1, 2, 3}, {0, 3}, {1, 2, 4}}	[{1, 2}, {2}, {3}, {0}, {1}]	{}	5	32
{{1, 3, 4}, {0, 2}, {0, 1}, {0, 3}, {1, 2, 4}}	[{1, 4}, {2}, {3}, {0}, {1}]	{}	5	32
{{2, 3}, {0, 2}, {1, 3}, {0, 4}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	1	2
{{2, 3}, {0, 2}, {1, 3}, {0, 1}, {1, 2, 4}}	[{2}, {3}, {0}, {1}]	{}	4	16
{{2, 4}, {0, 2}, {1, 3}, {0, 4}, {0, 1}}	[{4}, {2}, {0}, {1}]	{}	1	2
{{0, 2}, {1, 3}, {0, 4}, {0, 1}, {1, 2, 4}}	[{4}, {2}, {0}, {1}]	{}	4	16
{{0, 2}, {1, 3}, {0, 4}, {2, 3, 4}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{0, 3, 4}, {0, 2}, {1, 3}, {0, 1}, {1, 2, 4}}	[{4}, {2}, {3}, {0}, {1}]	{}	6	64
{{0, 2}, {1, 3}, {0, 1}, {2, 3, 4}, {1, 2, 4}}	[{2, 4}, {2}, {3}, {0}, {1}]	{}	5	32
{{2, 4}, {3, 4}, {0, 2}, {1, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{3, 4}, {0, 2}, {1, 3}, {0, 1}, {1, 2, 4}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{3, 4}, {0, 2}, {1, 2, 3}, {0, 1}, {1, 2, 4}}	[{4}, {3}, {0}, {1}, {1, 2}, {2}]	{}	4	16
{{0, 3, 4}, {0, 2}, {1, 2, 3}, {0, 1}, {1, 2, 4}}	[{4}, {3}, {0}, {1}, {1, 2}, {2}]	{}	7	128
{{1, 3, 4}, {0, 2}, {1, 2, 3}, {0, 1}, {1, 2, 4}}	[{0}, {1}, {1, 3}, {1, 2}, {1, 4}, {2}]	{}	5	32
{{1, 3, 4}, {0, 3, 4}, {0, 2}, {1, 2, 3}, {0, 1}}	[{3, 4}, {0}, {1}, {1, 3}, {2}]	{{3}}	7	128
{{1, 3, 4}, {0, 2}, {1, 2, 3}, {2, 3, 4}, {0, 1}}	[{3, 4}, {2, 3}, {3}, {0}, {1}, {1, 3}, {2}]	{}	4	16
{{1, 3, 4}, {0, 3, 4}, {0, 2}, {2, 3, 4}, {0, 1}}	[{3, 4}, {2}, {0}, {1}]	{}	8	256
{{0, 3, 4}, {2, 3}, {0, 1}, {0, 2, 4}, {1, 2, 4}}	[{2, 4}, {3}, {0}, {1}, {0, 4}, {2}]	{{4}}	6	64
{{0, 2, 3}, {0, 1}, {1, 2, 3}, {0, 2, 4}, {1, 2, 4}}	[{2, 4}, {2, 3}, {0, 2}, {0}, {1}, {1, 2}, {2}]	{}	6	64

{{0, 2, 3}, {0, 3, 4}, {1, 2, 3}, {0, 2, 4}, {0, 1}}	[{0}, {0, 4}, {2, 3}, {0, 2}, {1}, {0, 3}]	{{2}, {3}}	7	128
{{0, 2, 3}, {1, 3, 4}, {1, 2, 3}, {0, 2, 4}, {0, 1}}	[{0}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}]	{{2}, {3}}	8	256
{{0, 2, 3}, {0, 3, 4}, {2, 3, 4}, {0, 2, 4}, {0, 1}}	[{2, 4}, {3}, {0}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {2}, {0, 3}]	{}	1	2
{{0, 2, 3}, {0, 3, 4}, {1, 2, 3, 4}, {0, 2, 4}, {0, 1}}	[{2, 4}, {3}, {0}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {2}, {0, 3}]	{}	7	128
{{0, 2, 3}, {1, 3, 4}, {2, 3, 4}, {0, 2, 4}, {0, 1}}	[{2, 4}, {0}, {3, 4}, {2, 3}, {0, 2}, {1}, {2}]	{{4}, {3}}	6	64
{{0, 2, 3}, {1, 2, 3}, {2, 3, 4}, {0, 2, 4}, {0, 1}}	[{2, 4}, {2, 3}, {0, 2}, {0}, {1}, {2}]	{}	7	128
{{0, 2, 3}, {0, 1, 3}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{3}, {0}, {1, 2}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	3	8
{{0, 2, 3}, {0, 1, 3}, {0, 1, 4}, {0, 1, 2}, {2, 3, 4}}	[{0}, {4}, {2, 3}, {0, 2}, {0, 3}, {0, 1}]	{{2}, {3}}	9	512
{{0, 2, 3}, {0, 1, 3}, {1, 2, 3, 4}, {0, 1, 4}, {0, 1, 2}}	[{3}, {0}, {1, 2}, {1, 4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	8	256
{{0, 1, 3}, {1, 2, 3, 4}, {0, 1, 4}, {0, 1, 2}, {0, 2, 3, 4}}	[{3}, {0}, {1, 2}, {0, 4}, {1, 4}, {2, 3, 4}, {4}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	9	512
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {0, 1, 2}, {0, 2, 4}}	[{0, 4}, {0, 2}, {0, 3}, {0}, {0, 1}]	{}	9	512
{{0, 3, 4}, {0, 1, 3}, {0, 1, 2}, {1, 2, 3}, {0, 2, 4}}	[{0}, {1, 2}, {0, 4}, {0, 2}, {1}, {1, 3}, {0, 3}, {0, 1}]	{{2}, {3}}	6	64
{{0, 3, 4}, {0, 1, 3}, {1, 2, 3, 4}, {0, 1, 2}, {0, 2, 4}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {3, 4}, {4}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	6	64
{{0, 3, 4}, {0, 1, 3}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{0}, {1, 2}, {0, 4}, {1}, {1, 3}, {0, 3}, {0, 1}]	{{3}}	7	128
{{0, 3, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}, {2, 3, 4}}	[{1, 2}, {0, 4}, {3, 4}, {2, 3}, {0, 1}]	{{4}, {2}, {3}, {0}, {1}}	10	1024
{{0, 1, 3, 4}, {1, 2, 3, 4}, {0, 1, 2, 3}, {0, 2, 3, 4}, {0, 1, 2, 4}}	[{0, 2, 3}, {4}, {1, 2}, {0, 4}, {1, 2, 4}, {1, 3, 4}, {1, 3}, {0, 3}, {2, 4}, {0, 3, 4}, {3}, {0}, {1, 2, 3}, {1, 4}, {2, 3, 4}, {0, 2, 4}, {3, 4}, {2, 3}, {0, 2}, {0, 1, 3}, {1}, {0, 1, 4}, {0, 1, 2}, {2}, {0, 1}]	{}	0	1
{{1, 3, 4}, {0, 3, 4}, {0, 2}, {1, 2}, {2, 3, 4}, {0, 1}}	[{3, 4}, {2}, {0}, {1}]	{}	8	256
{{0, 2}, {1, 3}, {1, 2}, {0, 1}, {2, 3, 4}, {0, 3}}	[{2}, {3}, {0}, {1}]	{}	4	16
{{1, 3, 4}, {0, 2}, {1, 2}, {0, 1}, {2, 3, 4}, {0, 3}}	[{3, 4}, {2}, {3}, {0}, {1}]	{}	5	32
{{0, 2}, {1, 3}, {1, 2}, {0, 4}, {0, 3}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	1	2
{{3, 4}, {0, 2}, {1, 2}, {0, 4}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{1, 3, 4}, {0, 2}, {1, 2}, {0, 4}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{0, 2}, {0, 4}, {0, 1}, {1, 2, 3}, {0, 3}, {1, 2, 4}}	[{4}, {3}, {0}, {1}, {1, 2}, {2}]	{}	4	16
{{1, 3, 4}, {0, 2}, {0, 1}, {1, 2, 3}, {0, 3}, {1, 2, 4}}	[{3}, {0}, {1}, {1, 3}, {1, 2}, {1, 4}, {2}]	{}	4	16
{{1, 3, 4}, {0, 2}, {0, 1}, {2, 3, 4}, {0, 3}, {1, 2, 4}}	[{2, 4}, {3}, {0}, {1, 4}, {3, 4}, {4}, {1}, {2}]	{}	3	8

{{0, 3, 4}, {0, 2}, {1, 3}, {2, 3}, {0, 1}, {1, 2, 4}}	[{4}, {2}, {3}, {0}, {1}]	{}	6	64
{{2, 3}, {0, 2}, {1, 3}, {1, 2}, {0, 4}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	1	2
{{3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 4}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{2, 3}, {0, 2}, {1, 3}, {0, 4}, {0, 1}, {1, 2, 4}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{0, 2}, {1, 3}, {0, 4}, {0, 1}, {2, 3, 4}, {1, 2, 4}}	[{2, 4}, {4}, {3}, {0}, {1}, {2}]	{}	4	16
{{0, 3, 4}, {0, 2}, {1, 3}, {0, 1}, {2, 3, 4}, {1, 2, 4}}	[{2, 4}, {3, 4}, {3}, {0}, {1}, {2}]	{{4}}	6	64
{{2, 4}, {3, 4}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{3, 4}, {0, 2}, {1, 3}, {0, 1}, {0, 3}, {1, 2, 4}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{1, 3, 4}, {0, 3, 4}, {0, 2}, {1, 2, 3}, {0, 1}, {1, 2, 4}}	[{0}, {1, 2}, {1, 4}, {3, 4}, {1}, {1, 3}, {2}]	{{4}, {3}}	6	64
{{1, 3, 4}, {0, 3, 4}, {0, 2}, {1, 2, 3}, {2, 3, 4}, {0, 1}}	[{3, 4}, {2, 3}, {3}, {0}, {1}, {1, 3}, {2}]	{}	6	64
{{1, 3, 4}, {0, 2}, {0, 1}, {1, 2, 3}, {2, 3, 4}, {1, 2, 4}}	[{2, 4}, {3}, {0}, {1, 2}, {1, 4}, {3, 4}, {4}, {2, 3}, {1}, {1, 3}, {2}]	{}	0	1
{{1, 3, 4}, {0, 3, 4}, {2, 3}, {0, 1}, {0, 2, 4}, {1, 2, 4}}	[{2, 4}, {3}, {0}, {0, 4}, {1, 4}, {3, 4}, {4}, {1}, {2}]	{}	4	16
{{0, 2, 3}, {0, 1}, {1, 2, 3}, {2, 3, 4}, {0, 2, 4}, {1, 2, 4}}	[{2, 4}, {2, 3}, {0, 2}, {0}, {1}, {1, 2}, {2}]	{}	7	128
{{0, 2, 3}, {0, 3, 4}, {0, 1}, {1, 2, 3}, {0, 2, 4}, {1, 2, 4}}	[{2, 4}, {0}, {1, 2}, {0, 4}, {2, 3}, {0, 2}, {1}, {2}, {0, 3}]	{{4}, {3}}	5	32
{{0, 2, 3}, {0, 3, 4}, {1, 2, 3}, {2, 3, 4}, {0, 2, 4}, {0, 1}}	[{2, 4}, {3}, {0}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {2}, {0, 3}]	{}	2	4
{{0, 2, 3}, {1, 3, 4}, {1, 2, 3}, {2, 3, 4}, {0, 2, 4}, {0, 1}}	[{2, 4}, {3}, {0}, {3, 4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}]	{{4}}	5	32
{{0, 2, 3}, {0, 1, 3}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}, {2, 3, 4}}	[{3}, {0}, {1, 2}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	4	16
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {0, 1, 4}, {0, 1, 2}, {0, 2, 4}}	[{0, 4}, {0, 2}, {0, 3}, {0}, {0, 1}]	{}	10	1024
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {1, 2, 3, 4}, {0, 1, 2}, {0, 2, 4}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	5	32
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {0, 1, 2}, {1, 2, 3}, {0, 2, 4}}	[{3}, {0}, {1, 2}, {0, 4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	3	8
{{0, 3, 4}, {0, 1, 3}, {0, 1, 2}, {1, 2, 3}, {2, 3, 4}, {0, 2, 4}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	0	1
{{0, 3, 4}, {0, 1, 3}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}, {0, 2, 4}}	[{0}, {1, 2}, {0, 4}, {0, 2}, {1}, {1, 3}, {0, 3}, {0, 1}]	{{2}, {3}}	7	128
{{0, 3, 4}, {0, 1, 3}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}, {2, 3, 4}}	[{3}, {0}, {1, 2}, {0, 4}, {3, 4}, {2, 3}, {1}, {1, 3}, {0, 3}, {0, 1}]	{{4}, {2}}	5	32
{{1, 2}, {0, 4}, {2, 3, 4}, {1, 3, 4}, {0, 2}, {0, 3}, {0, 1}}	[{3, 4}, {4}, {3}, {0}, {1}, {2}]	{}	4	16
{{0, 4}, {1, 2, 4}, {1, 3, 4}, {0, 2}, {1, 2, 3}, {0, 3}, {0, 1}}	[{3}, {0}, {1, 2}, {1, 4}, {4}, {1}, {1, 3}, {2}]	{}	3	8

{ {2, 3, 4}, {1, 2, 4}, {1, 3, 4}, {0, 2}, {1, 2, 3}, {0, 3}, {0, 1}}	[{2, 4}, {3}, {0}, {1, 2}, {1, 4}, {3, 4}, {4}, {2, 3}, {1}, {1, 3}, {2}]	{}	0	1
{{1, 2}, {0, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{0, 4}, {1, 2, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{1, 2}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{2}, {3}, {0}, {1}]	{}	1	2
{{0, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{1, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{ {1, 2, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{0, 3, 4}, {2, 3, 4}, {1, 2, 4}, {1, 3, 4}, {0, 2}, {1, 2, 3}, {0, 1}}	[{2, 4}, {3}, {0}, {1, 2}, {1, 4}, {3, 4}, {4}, {2, 3}, {1}, {1, 3}, {2}]	{}	2	4
{{0, 2, 3}, {0, 3, 4}, {0, 2, 4}, {1, 2, 4}, {1, 3, 4}, {1, 2, 3}, {0, 1}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {1, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}]	{}	0	1
{{0, 2, 3}, {0, 3, 4}, {2, 3, 4}, {0, 2, 4}, {1, 2, 4}, {1, 2, 3}, {0, 1}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {2}, {0, 3}]	{}	2	4
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {0, 2, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{3}, {0}, {1, 2}, {0, 4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	4	16
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {1, 2, 3, 4}, {0, 2, 4}, {0, 1, 4}, {0, 1, 2}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {1, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	4	16
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {2, 3, 4}, {0, 2, 4}, {0, 1, 2}, {1, 2, 3}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	0	1
{{0, 3, 4}, {0, 1, 3}, {0, 2, 4}, {1, 2, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{2, 4}, {0}, {1, 2}, {0, 4}, {1, 4}, {4}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{{3}}	3	8
{{0, 3, 4}, {0, 1, 3}, {2, 3, 4}, {0, 2, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	1	2
{{2, 4}, {1, 2}, {0, 4}, {3, 4}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{0, 4}, {2, 3, 4}, {1, 2, 4}, {1, 3, 4}, {0, 2}, {1, 2, 3}, {0, 3}, {0, 1}}	[{2, 4}, {3}, {0}, {1, 2}, {1, 4}, {3, 4}, {4}, {2, 3}, {1}, {1, 3}, {2}]	{}	0	1
{{1, 2}, {0, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{0, 4}, {1, 2, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	3	8
{{0, 2, 3}, {0, 3, 4}, {2, 3, 4}, {0, 2, 4}, {1, 2, 4}, {1, 3, 4}, {1, 2, 3}, {0, 1}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {1, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}]	{}	0	1
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {0, 2, 4}, {1, 2, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {1, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	1	2
{{0, 3, 4}, {0, 1, 3}, {2, 3, 4}, {0, 2, 4}, {1, 2, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {1, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	0	1

{{1, 2}, {0, 4}, {1, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {2, 3, 4}, {0, 2, 4}, {1, 2, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {1, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	0	1
{{2, 4}, {1, 2}, {0, 4}, {1, 4}, {3, 4}, {2, 3}, {0, 2}, {1, 3}, {0, 3}, {0, 1}}	[{4}, {2}, {3}, {0}, {1}]	{}	0	1
{{0, 2, 3}, {0, 3, 4}, {0, 1, 3}, {2, 3, 4}, {0, 2, 4}, {1, 2, 4}, {1, 3, 4}, {0, 1, 4}, {0, 1, 2}, {1, 2, 3}}	[{2, 4}, {3}, {0}, {1, 2}, {0, 4}, {1, 4}, {3, 4}, {4}, {2, 3}, {0, 2}, {1}, {1, 3}, {2}, {0, 3}, {0, 1}]	{}	0	1