Experimenting with the Gram-Schmidt Walk

Gaëtan Bossy
Supervision by Pr. Adam Marcus
Chair of Combinatorial Analysis
EPFL

June 28, 2022

Abstract

In this paper, we first discuss vector balancing and discrepancy theory. We then introduce the Gram-Schmidt Walk as well as why it was invented. We discuss its properties, and a cubic time implementation, when the trivial implementation runs in quadratic time. In the second half, we perform several experiments to see how the algorithm behaves and how and why we could modify it. Finally, we discuss a new problem and how we can use the Gram-Schmidt Walk to solve it.

1 Introduction

Imagine that you want to divide a group of soccer players into two teams such that the match between them will be fair. Or, that you want to divide several quantities of several types of sweets between your two kids who each have their own tastes. In this paper, we will discuss an algorithm performing vector balancing, that is, an algorithm that could solve these problems. The basic goal is, given $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^n$, to find $\mathbf{z} \in \{-1, 1\}^n$ such that $\sum_{i=1}^n \mathbf{z}(i)\mathbf{v}_i$ is small.

Let $(X, \mathcal{S})$ be a finite set system, with $X = \{1, 2, \dots, d\}$ and $\mathcal{S} = \{S_1, S_2, \dots S_n\}$ a collection of subsets of X. Given a coloring function $x : X \to \{-1, 1\}$, the discrepancy of a set S is defined as $x(S) = |\sum_{i \in S} x(i)|$. It is a measure of how balanced S is with respect to the coloring x. The discrepancy of the set system $(X, \mathcal{S})$ is defined as

$$\operatorname{disc}(X, \mathcal{S}) = \min_{x:X \to \{-1,1\}} \max_{S \in \mathcal{S}} x(S)$$

This can be seen as a specific case of a vector balancing problem. If we consider the incidence matrix $\mathbf{A} \in \mathbb{R}^{d \times n}$ of the set system $(X, \mathcal{S})$, we can write

$$\operatorname{disc}(\mathbf{A}) = \min_{\mathbf{x} \in \{-1,1\}^n} \|\mathbf{A}\mathbf{x}\|_\infty$$

But we could also balance vectors with coordinates different from 0 and 1, so if we have a set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^d$ we can see its discrepancy as the minimum infinity norm of the **vector of imbalances** (that is, $\mathbf{Ax}$) among all colorings $\mathbf{x} \in \{-1, 1\}^n$.

In 1997, polish mathematician Wojciech Banaszczyk published *Balancing vectors and Gaussian measures of n-dimensional convex bodies* [1] in which we can read the following theorem:

Theorem 1 (Banaszczyk's Theorem from [1])**.** *For all convex body $K \subseteq \mathbb{R}^m$, with Gaussian measure $\gamma_m(K) \geq 1/2$, and given $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n \in \mathbb{R}^m$, $\|\boldsymbol{v}_i\|_2 \leq 1$ for all i, then there exists $\boldsymbol{z} \in \{-1, 1\}^n$ such that*

$$\sum_{i=1}^{n} \boldsymbol{z}(i)\boldsymbol{v}_i \in 5 \cdot K$$

The Gaussian measure of a body is defined as

$$\gamma_m(S) = \mathbb{P}[\mathbf{g} \in S] = \int_{\mathbf{y} \in S} \frac{1}{(2\pi)^{m/2}} e^{-||\mathbf{y}||^2/2} d\mathbf{y}$$

where $\mathbf{g}$ is a standard Gaussian random vector in $\mathbb{R}^m$, i.e. $\mathbf{g} \sim \mathcal{N}(0, I_m)$.

Thus the idea of the theorem is that for any subset of the space big enough and set of vectors, one can find an assignment such that the sum of the vectors in the positive group minus the sum of the vectors in the negative group is in a dilatation of the big subset of the space. As the conditions are easily satisfied, that is a very powerful statement. It gave several bounds for various discrepancy theory problems.

It is easy to see that for any bound smaller than 1/2 we could find a counterexample, but it isn't easy to see that this is true for the bound of 1/2. The original proof of this theorem was non-constructive though, so until recently, no algorithm was known to find the colorings whose existences are proved by the theorem.

Nearly 20 years later, Dadush, Garg, Lovett and Nikolov showed that in order to get a constructive algorithm, all that is needed is a subgaussian distribution sampler on $\{\sum_{i=1}^n \mathbf{z}(i)\mathbf{v}_i : \mathbf{z} \in \{-1, 1\}^n\}$.

Definition 2. *A random vector $\boldsymbol{Y} \in \mathbb{R}^m$ is said to be subgaussian with parameter σ (or σ-subgaussian) if for all $\boldsymbol{\theta} \in \mathbb{R}^m$:*

$$\mathbb{E}\left[e^{\langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle}\right] \leq e^{\sigma^2 \|\boldsymbol{\theta}\|_2^2/2}.$$

Subgaussianity is discussed more in details in section 3.5.

Here is their result:

Theorem 3 ([4])**.** *Banaszczyk's Theorem (up to the constant) is equivalent to the following statement:*
Let $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n \in \mathbb{R}^d$ of ℓ_2 norm at most 1, with $\boldsymbol{B} = (\boldsymbol{v}_1, \dots, \boldsymbol{v}_n) \in \mathbb{R}^{d \times n}$ and $K \subseteq \mathbb{R}^d$ a convex body of Gaussian measure at least 1/2. Then, there exists $\boldsymbol{z}_0 \in [-1, 1]^n$ and there exists a distribution D on $\{-1, 1\}^n$, such that:

1. $\boldsymbol{B}\boldsymbol{z}_0 = \sum_{i=1}^{n} \boldsymbol{z}_0(i)\boldsymbol{v}_i \in K$

2. *for* $\boldsymbol{z} \sim D$, $\boldsymbol{B}(\boldsymbol{z} - \boldsymbol{z}_0) = \sum_{i=1}^{n}(\boldsymbol{z}(i) - \boldsymbol{z}_0(i))\boldsymbol{v}_i$ *is* σ*-subgaussian, for some* $\sigma > 0$ *and if* $\boldsymbol{z}_0(i) \in \{-1, 1\}$, $\mathbb{P}[\boldsymbol{z}(i) = \boldsymbol{z}_0(i)] = 1, \forall i$.

3. $\mathbb{P}_{\boldsymbol{z}\sim D}[\sum_{i=1}^{n}(\boldsymbol{z}(i) - \boldsymbol{z}_0(i))\boldsymbol{v}_i \in c(\sigma)K] \geq 1/2$ *for some* $c(\sigma)$.

In this last theorem, one can just pick $\mathbf{z}_0 = \mathbf{0}$ because the gaussian measure bound forces it to be inside K, but finding the subgaussian distribution is far from trivial. The proof uses the minimax theorem from Von Neumann [7], turning the choice of the distribution in a game, and a result from Talagrand [8] on the norm of subgaussian vectors. In addition to all that, there is a complicated recentering procedure to show that the result still holds for non-symmetric bodies.

A couple years later, a team including three of the previous authors, Dadush, Garg and Lovett, joined by Bansal this time, actually found and analysed an algorithm to successfully sample colorings that allows us to construct the colorings described by Banaszczyk in his work from 1998. They call this algorithm the Gram-Schmidt walk, as it uses a procedure that can remind of the Gram-Schmidt orthogonalization process. They proved that the vector of imbalances produced by the assignments of the Gram-Schmidt Walk are subgaussian with constant $\sigma^2 = 40$.

While the Gram-Schmidt walk was originally described as a way to get a constructive algorithm for Theorem 1 and thus produced constructive algorithms for many other discrepancy theory problems, it was later also used for experiment design by Spielman, Harshaw, Zhang and Sävje in [6]. In their setting, they're looking to separate experiment units into a control group and a treatment group. They use the balancing power of the algorithm to explicitly navigate the tradeoff between experiment unit covariate balance, that is having two groups that are similar, and robustness of the design, that is being random enough to be on average well-balanced among unobserved covariates. They improved the constant in the subgaussianity result and proved several other results which let them document the expected behavior of their design.

In this work, we will try to understand the algorithm, think about how to modify it to achieve different goals, and what other things we could use it for.

All tests and simulation found in this document were run in Jupyter notebooks through a self-written Python implementation of the Gram-Schmidt walk and other needed functions, available at [3].

2 Notation

There are a few notations that will appear several times in this document:

- $[m]$ refers to the set $\{1, \ldots, m\}$.

- Every $\mathbf{v} \in \mathbb{R}^d$ for some dimension d is seen as a column vector of $\mathbb{R}^{d\times 1}$ when it appears in a product, and for $i \in [d]$, $\mathbf{v}(i)$ is the ith coordinate of $\mathbf{v}$.
- $\mathbf{e}_k \in \mathbb{R}^n$ is the vector with only 0's except for a 1 at position k for some $k \in [n]$. Note that in particular for $\mathbf{M} \in \mathbb{R}^{n\times n}$, $\mathbf{e}_k^T\mathbf{M}$ is the kth row of $\mathbf{M}$ and $\mathbf{M}\mathbf{e}_k$ is its kth column.
- Given a condition c, $\mathbb{1}_c = \begin{cases} 1 \text{ if } c \text{ is true} \\ 0 \text{ if } c \text{ is false} \end{cases}$.
- Vectors and matrices are **bolded**.

3 The Algorithm

3.1 General idea

The idea of the algorithm is that it walks step by step in the hypercube of dimension n, $[-1, 1]^n$. We start at some initial fractional coloring $\mathbf{z}_0 \in [-1, 1]^n$. At each step, the coloring moves in the hypercube until at least one additional coordinate becomes 1 or -1, that is it moves to a facet of the hypercube it wasn't on yet. Once a coordinate is set to -1 or 1, it won't move for the rest of the algorithm, thus after step i our fractional coloring is in the intersection of at least i different facets. From all this, we can see that the algorithm reaches a non fractional coloring $\mathbf{z}_t \in \{-1, 1\}^n$ in at most n steps.

The question now becomes "how to choose the facet to move to ?". To do so, the algorithm finds an update direction $\mathbf{u}_t \in \mathbb{R}^n$ such that $\sum_{i=1}^n \mathbf{u}_t(i)\mathbf{v}_i$ is small, and updates $\mathbf{z}_t$ into $\mathbf{z}_{t+1} = \mathbf{z}_t + \delta_t\mathbf{u}_t$. $\delta_t \in \mathbb{R}$ is chosen so that $\mathbf{z}_{t+1} \in [-1, 1]^n$ and at least one additional coordinate now has absolute value 1. Moreover, δ_t is chosen randomly among the two potential candidates that fulfill the previously mentioned condition, and the distribution between the two candidates is engineered so that its expectation is 0 and no direction is favored. Thus before the algorithm starts, ending at $\mathbf{z}_t \in \{-1, 1\}^n$ or at $-\mathbf{z}_t \in \{-1, 1\}^n$ is equally likely.

3.2 Pseudocode

The algorithm takes as input $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^d$, and an initial coloring $\mathbf{z}_0 \in [-1, 1]^n$. A run of the algorithm consists of at most n time steps. At the end of each time step t, the algorithm obtains a fractional coloring $\mathbf{z}_t \in [-1, 1]^n$. An element $i \in [n]$ is said to be *alive* at time t if $|\mathbf{z}_{t-1}(i)| < 1$, and *fixed* otherwise. Let $A_t = \{i \in [n] : |\mathbf{z}_{t-1}(i)| < 1\}$. The *pivot* $p(t)$ is an element that is alive at time t, which can for example be chosen randomly or as the largest indexed element, among the elements that are still alive. We define the set V_t as $\operatorname{span}\{\mathbf{v}_i : i \in A_t, i \neq p(t)\}$. We denote by $\Pi_{V_t^\perp}$ the projection operator on $V_t^\perp$. Finally, we need $\mathbf{v}^\perp(t) = \Pi_{V_t^\perp}(\mathbf{v}_{p(t)})$ as the projection of the pivot vector over all alive vectors. We are now ready to discover the actual pseudocode of the algorithm.

It produces an assignment $\mathbf{z}$ such that the corresponding vector of imbalances, also referred to as the output sum of groups, has a small norm. Another way to describe it is that it divides the vectors in two groups such that the sum of the vectors in each group are pretty close.

Algorithm 1: Gram-Schmidt Walk [2]

```
Input  : $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^d$, an initial coloring $\mathbf{z}_0 \in [-1,1]^n$
Output: a coloring $\mathbf{z}_n \in \{-1,1\}^n$
$A_1 = \{i \in [n] : |\mathbf{z}_0(i)| < 1\}$, $p(1) = \max\{i \in A_1\}$ and $t = 1$.
while $A_t \neq \emptyset$ do
    Compute $\mathbf{u}_t \in \mathbb{R}^n$ such that $\begin{cases} \mathbf{u}_t(p(t)) = 1 \\ \mathbf{u}_t(i) = 0 \text{ if } i \notin A_t \\ \mathbf{v}^{\perp}(t) = \mathbf{v}_{p(t)} + \sum_{i \in A_t \setminus \{p(t)\}} \mathbf{u}_t(i)\mathbf{v}_i \end{cases}$
    $\Delta = \{\delta : \mathbf{z}_{t-1} + \delta \mathbf{u}_t \in [-1,1]^n\}$, let $\begin{cases} \delta_t^+ = \max \Delta \\ \delta_t^- = \min \Delta \end{cases}$ then $\delta_t = \begin{cases} \delta_t^+ \text{ w.p. } \frac{-\delta_t^-}{(\delta_t^+ - \delta_t^-)} \\ \delta_t^- \text{ w.p. } \frac{\delta_t^+}{(\delta_t^+ - \delta_t^-)} \end{cases}$
    $\mathbf{z}_t = \mathbf{z}_{t-1} + \delta_t \mathbf{u}_t$, $t \leftarrow t+1$, $A_t = \{i \in [n] : |\mathbf{z}_{t-1}(i)| < 1\}$, $p(t) = \max\{i \in A_t\}$.
end
Output $\mathbf{z}_t \in \{-1,1\}^n$.
```

Throughout the text, we will refer to this algorithm by its initials as the **GSW**.

3.3 Observations

Most if not all of these observations are already available in [2]. We include them here to help the reader understand the algorithm through these specific properties.

According to the choice of δ_t, it is clear that at least one element gets frozen at each time step as δ_t is maximal such that $\mathbf{z}_t + \delta_t \mathbf{u}_t \in [-1,1]^n$. Thus, as fixed elements stay fixed until the end of the algorithm once they are fixed, the GSW runs in at most n iterations. If we chose δ_t according to some more complicated distribution, it wouldn't be possible to guarantee a coloration per time step and the algorithm would lose most of its appeal and simplicity. Additionally, the update direction depends on the elements that are alive. Thus if they did not change between two time steps, the update direction would not change either and we would just keep moving along the same line until we hit a border, which is why choosing one of the two maximal valid lengths, δ_t^+ or δ_t^-, is the best method to choose the length of the move.

We can see that

$$\mathbb{E}[\delta_t \mid \delta_t^-, \delta_t^+] = \delta_t^+ \frac{-\delta_t^-}{\delta_t^+ - \delta_t^-} + \delta_t^- \frac{\delta_t^+}{\delta_t^+ - \delta_t^-} = 0$$

so the choice of delta and thus, as mentioned previously, the sequence of fractional coloring produced by the algorithm before its termination is indeed a martingale. This gives a sort of unbiasedness to the algorithm which is desirable when dividing elements into groups, as it means that when the algorithm starts from $\mathbf{z}_0 = \mathbf{0}$, each vector has an equal probability of ending in each group. It also means that we can tune the final outcome through the starting coloring $\mathbf{z}_0$. Indeed, if $\mathbf{z}_0(i) = 2\mathbf{p}(i) - 1$, we can see thanks to the martingale property that $\mathbb{P}(\mathbf{z}_n(i) = 1) = \mathbf{p}(i)$.

It is important to notice that $\mathbf{v}^{\perp}(t)$ depends on t and not just on $\mathbf{v}_{p(t)}$, as A_t can change while the pivot $p(t)$ stays the same.

The update direction $\mathbf{u}_t$ satisfying our conditions always exists because

$$\mathbf{v}^{\perp}(t) = \mathbf{v}_{p(t)} + \sum_{i \in A_t \setminus \{p(t)\}} \mathbf{u}_t(i)\mathbf{v}_i$$

$$\Leftrightarrow \sum_{i \in A_t \setminus \{p(t)\}} \mathbf{u}_t(i)\mathbf{v}_i = -\left(\mathbf{v}_{p(t)} - \mathbf{v}^{\perp}(t)\right) \in V_t$$

The vector $\mathbf{u}_t$ is defined with $\mathbf{u}_t(i) = 0$ if i is frozen, $\mathbf{u}_t(p(t)) = 1$ and for the rest of the indices we have that :

$$\begin{aligned}
\mathbf{v}^{\perp}(t) &= \mathbf{v}_{p(t)} + \sum_{i \in A_t \setminus \{p(t)\}} \mathbf{u}_t(i)\mathbf{v}_i \\
&= 1 \cdot \mathbf{v}_{p(t)} + \sum_{i \in A_t \setminus \{p(t)\}} \mathbf{u}_t(i)\mathbf{v}_i + \sum_{i \notin A_t} 0 \cdot \mathbf{v}_i \\
&= \sum_{i=1}^{n} \mathbf{u}_t(i)\mathbf{v}_i
\end{aligned}$$

Thus at each step the vector of imbalances moves by $\delta_t \mathbf{v}^{\perp}(t)$.

$\mathbf{u}_t = \arg\min_{\mathbf{u} \in U} \|\mathbf{B}\mathbf{u}\|$ where U is the set of vectors in $\mathbb{R}^n$ such that $\mathbf{u}(i) = 0\ \forall i \notin A_t$ and $\mathbf{u}(p(t)) = 1$. This means that one can completely forget about $\mathbf{v}^{\perp}(t)$ and system solving and just use least squares at each step to find the coordinates of the update directions that are alive but not the pivot through the following formula

$$\mathbf{u}_t(A_t \setminus \{p(t)\}) = \arg\min_{\mathbf{u}_t} \|\mathbf{v}_{p(t)} + \sum_{i \in A_t \setminus \{p(t)\}} \mathbf{u}_t(i)\mathbf{v}_i\|^2 = \left(\mathbf{B}_t^T \mathbf{B}_t\right)^{-1} \mathbf{B}_t^T \mathbf{v}_{p(t)}$$

where $\mathbf{B}_t \in \mathbb{R}^{d \times (|A_t|-1)}$ is the matrix containing all vectors that are alive except the pivot vector as columns. In some cases when $\mathbf{v}^{\perp}(t) = \mathbf{0}$, there are infinitely many solutions that minimize our objective function, so it would be interesting to add some conditions or do some quadratic programming to try to get a $\mathbf{u}_t$ that has some desirable properties. That is investigated a bit in section 6.4.3.

Thanks to the previous paragraph, we can see that $\mathbf{v}^{\perp}(t)$ corresponds to the direction that the vector of imbalances will move toward during time step t, that is if the matrix $\mathbf{B}$ contains the input vectors as columns, $\mathbf{v}^{\perp}(t) = \mathbf{B}\mathbf{u}_t$. So the algorithm is greedily optimal in the sense that the update direction is trying not to move away from $\mathbf{0}$ as much as possible while moving towards a vertex of the hypercube.

Another interesting point is that one could choose each new pivot randomly among alive elements. This is equivalent to shuffling the input vectors before inputing them to the walk. When doing multiple runs with the same input vector set, we will want to shuffle the vectors unless we have reasons to prefer a specific pivot ordering, this way we ensure a greater variability of vector of imbalances. One thing that can be interesting though, is seeing how the choice of the pivot affects the behavior of the algorithm, as the choice of the pivot doesn't need to follow a specific rule except for one small result about confidence intervals from [6]. Thus, trying to optimize the algorithm for our needs by tweaking the choice of the pivot could be an interesting direction to explore and is explored several times in section 6.

3.3.1 Connection to the Gram-Schmidt orthogonalization process

One can see that $\mathbf{v}^{\perp}(t)$ is the last vector of the Gram-Schmidt orthogonalization of the ordered sequence of vectors $(\mathbf{v}_i)_{i \in A_t}$. This is where the name of the algorithm comes from. The authors of [6] detail this connection more in depth in their first appendix by defining a permutation σ that indicates us the *reverse order* of when a vector was decided in the algorithm, that is when it was chosen as the pivot or colored without being the pivot. Thus $\sigma(n)$ is the very first pivot, and if that pivot is colored in the first step, $\sigma(n-1)$ is the second pivot. If that isn't the case, $\sigma(n-1)$ is the unit colored in the first step.

From σ, they define the Gram-Schmidt orthogonalization of the input vectors in the order $\sigma(1), \sigma(2), \ldots, \sigma(n)$ as usual:

$$\mathbf{w}_{\sigma(1)} = \frac{\mathbf{v}_{\sigma(1)}}{\|\mathbf{v}_{\sigma(1)}\|}, \quad \mathbf{w}_{\sigma(r)} = \frac{\mathbf{v}_{\sigma(r)} - \sum_{i=1}^{r-1} \langle \mathbf{w}_{\sigma(i)}, \mathbf{v}_{\sigma(r)} \rangle \mathbf{w}_{\sigma(i)}}{\|\mathbf{v}_{\sigma(r)} - \sum_{i=1}^{r-1} \langle \mathbf{w}_{\sigma(i)}, \mathbf{v}_{\sigma(r)} \rangle \mathbf{w}_{\sigma(i)}\|} \text{ for } r = 2 \ldots, n.$$

Let $S_k = \{i \in [n] : p(i) = k\}$ be the pivot phase of i, that is the set of time steps during which $\mathbf{v}_i$ is the pivot. Then, defining l_t and g_t as the least and greatest re-ordering positions that were decided during the current pivot phase before the third line of the GSW at iteration t, they can express $\mathbf{v}^{\perp}(t)$ as the projection of $\mathbf{v}_{p(t)}$ on $\mathbf{w}_{\sigma(l_t)}, \ldots, \mathbf{w}_{\sigma(g_t)}$, that is we have

$$\mathbf{v}^{\perp}(t) = \mathbf{B}\mathbf{u}_t = \sum_{r=l_t}^{g_t} \langle \mathbf{w}_{\sigma(r)}, \mathbf{v}_{p(t)} \rangle \mathbf{w}_{\sigma(r)}.$$

This characterization is, among other things, what allows them to improve the subgaussianity result from [2].

3.4 Basic variants

One pretty close variant would always choose the delta with the smallest absolute value at the end of line 4, instead of δ_t being a martingale. We will denote the algorithm with this modification as the Deterministic Gram-Schmidt Walk, or DGSW. In case of equality between the two absolute value, the algorithm would still pick randomly. The results from section 3.5 do not apply to this variant, but in practice it is pretty close to the classical GSW, except it is expectedly returning slightly shorter vector of imbalances as we always pick the shorter move. Notice that there is still randomness in the order of the pivots, which can be done either by shuffling the vectors before the run or by choosing the pivot randomly among alive elements when needed.

Another really different variant which we will compare a bit with the GSW in this document is the following algorithm:

Algorithm 2: Naive Walk

```
Input  : $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^d$
Output: a coloring $\mathbf{a} \in \{-1, 1\}^n$
$\mathbf{s} = \mathbf{0} \in \mathbb{R}^d$
$\mathbf{a} = \mathbf{0} \in \mathbb{R}^n$
for $i \in [n]$ do
    if $\|\boldsymbol{s} + \boldsymbol{v}_i\| < \|\boldsymbol{s} - \boldsymbol{v}_i\|$ then
        $\mathbf{a}(i) = 1$; $\mathbf{s} = \mathbf{s} + \mathbf{v}_i$
    else
        $\mathbf{a}(i) = -1$; $\mathbf{s} = \mathbf{s} - \mathbf{v}_i$
    end
end
Output $\mathbf{a} \in \{-1, 1\}^n$.
```

This simple algorithm that we call the naive walk can have a random component through a pre-input shuffling of the vectors, similarly to the DGSW. It doesn' actually compare badly to the GSW in term of norm of the resulting vectors of imbalances, as can be seen via the experiment in section 6.1.1.

Another way to do it would be to choose the sign according to a distribution that is inversely proportional to the norm of the potential updated $\mathbf{s}$ vector, or to greedily try every assignment of every vector at each step and add the assignment that results in the smallest norm, or do a distribution based on how big the norm would be if an assignment was added. In the end, I wanted to compare with a simple algorithm and chose this one.

3.5 Subgaussianity result

We will first define subgaussianity, which is a way to quantify how centered a random variable or vector is and state a result showing that the vector of imbalances produced by a GSW run is actually 1-subgaussian.

Definition 4. *A random variable $X \in \mathbb{R}$ is said to be subgaussian with parameter σ (or σ-subgaussian) if $\forall \lambda \in \mathbb{R}$:*

$$\mathbb{E}\left[e^{\lambda(X - \mathbb{E}[X])}\right] \leq e^{\sigma^2 \lambda^2 / 2}$$

For example, any σ^2-subgaussian random variable X has $\mathbb{V}[X] \leq \sigma^2$ with equality if it is gaussian. The above definition is equivalent to the following definition based on concentration, which tells us that a centered σ^2-subgaussian random variable is at least as centered as a gaussian variable with variance σ^2:

Definition 5 (Equivalent to definition 4)**.** *A random variable X is said to be σ-subgaussian if $\forall t > 0$, $\max\{\mathbb{P}(X \geq t), \mathbb{P}(X \leq -t)\} \leq exp\left(\frac{-t^2}{2\sigma^2}\right)$.*

This definition is derived from the first one via Chernov's bound. It lets us derive the same kind of useful inequalities about subgaussian random variables as about gaussian variables, which, given that gaussian variables are well-behaved and well-studied, is a desirable feature.

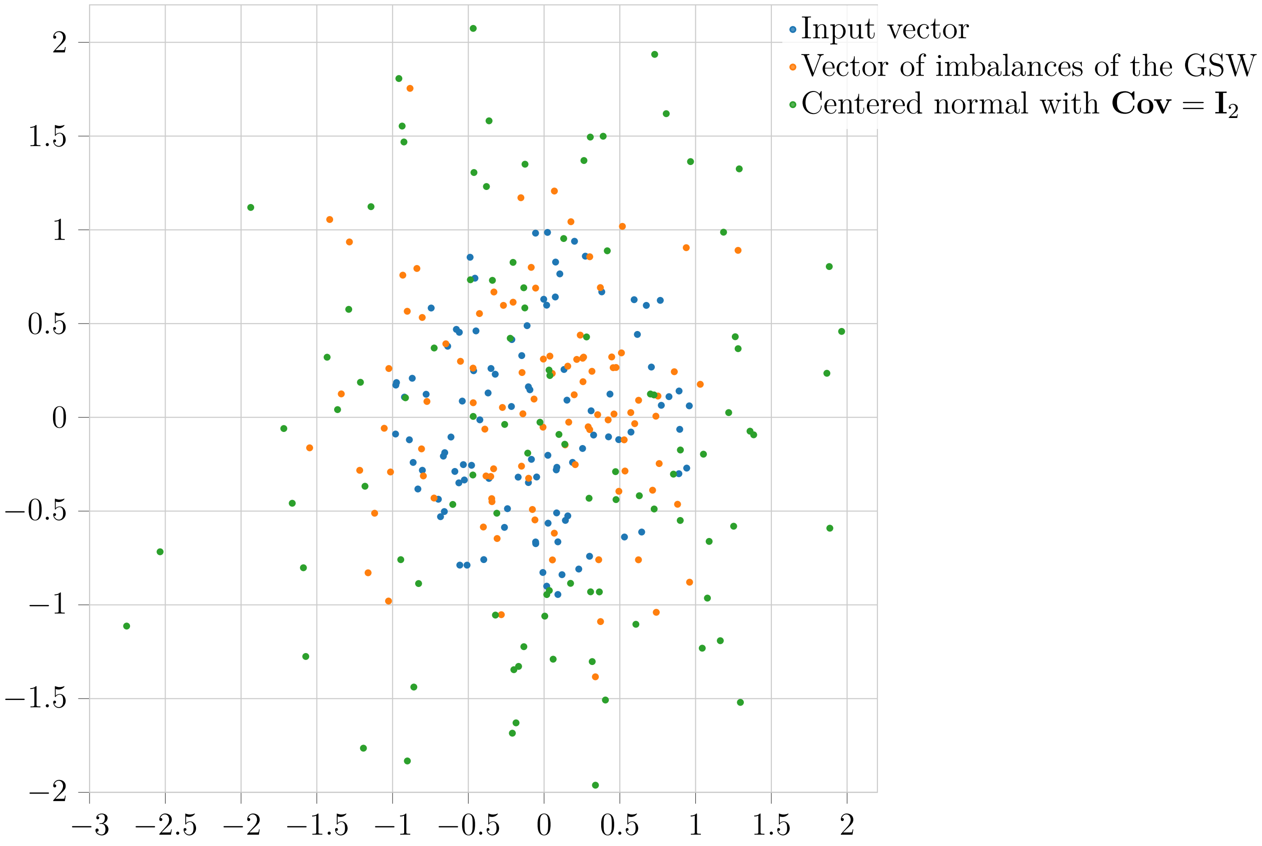


Figure 1: Plot of vectors of imbalances of group assignments from the GSW, its input vectors, and samples from a centered normal with $\mathbf{Cov} = \mathbf{I}_2$ to illustrate subgaussianity. There are 100 input vectors and 100 of each kinds of point, everything is in dimension 2. The same plot with vectors of imbalances from random assignments is visible in figure 2 for comparison.

There exists several other equivalent definitions, but the first one is the one-dimensional version of the definition we will use for subgaussianity for random vectors, and the second one helps us understand how subgaussian random variables are related to gaussian variables.

Now we can adapt the notion of subgaussianity to random vectors:

Definition 6 (same as definition 2)**.** *A random vector* $\boldsymbol{Y} \in \mathbb{R}^m$ *is said to be subgaussian with parameter* σ *(or* σ*-subgaussian) if for all* $\boldsymbol{\theta} \in \mathbb{R}^m$*:*

$$\mathbb{E}\left[e^{\langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle}\right] \leq e^{\sigma^2 \|\boldsymbol{\theta}\|_2^2/2}.$$

This definition will help us give a concrete result on the balancing power of the GSW. A random vector being 1-subaussian means that it is less spread that a gaussian with covariance matrix being the identity. We can see that for the vectors of imbalances of the GSW, on figure 1, it seems to indeed be the case. This was proved, first for $\sigma^2 = 40$ in [2] and then for $\sigma^2 = 1$ in [6].

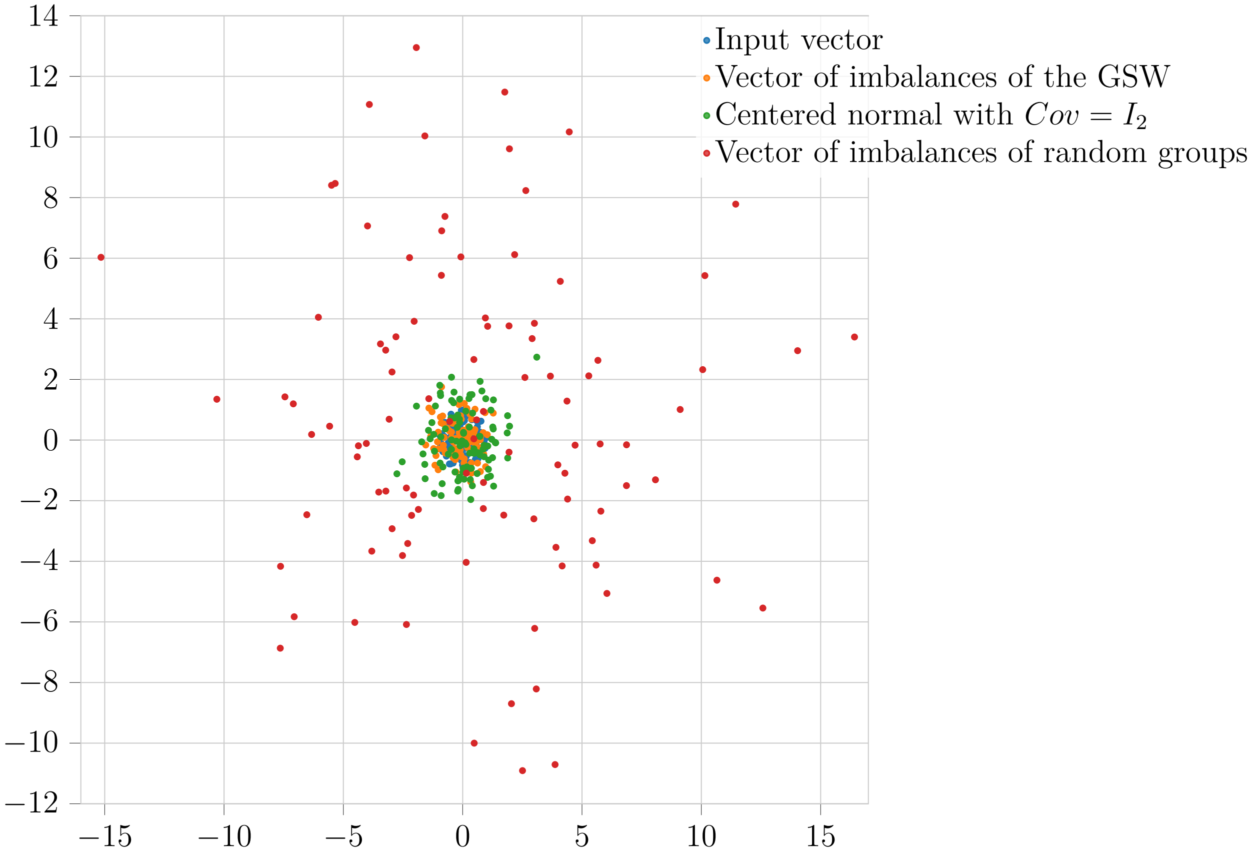


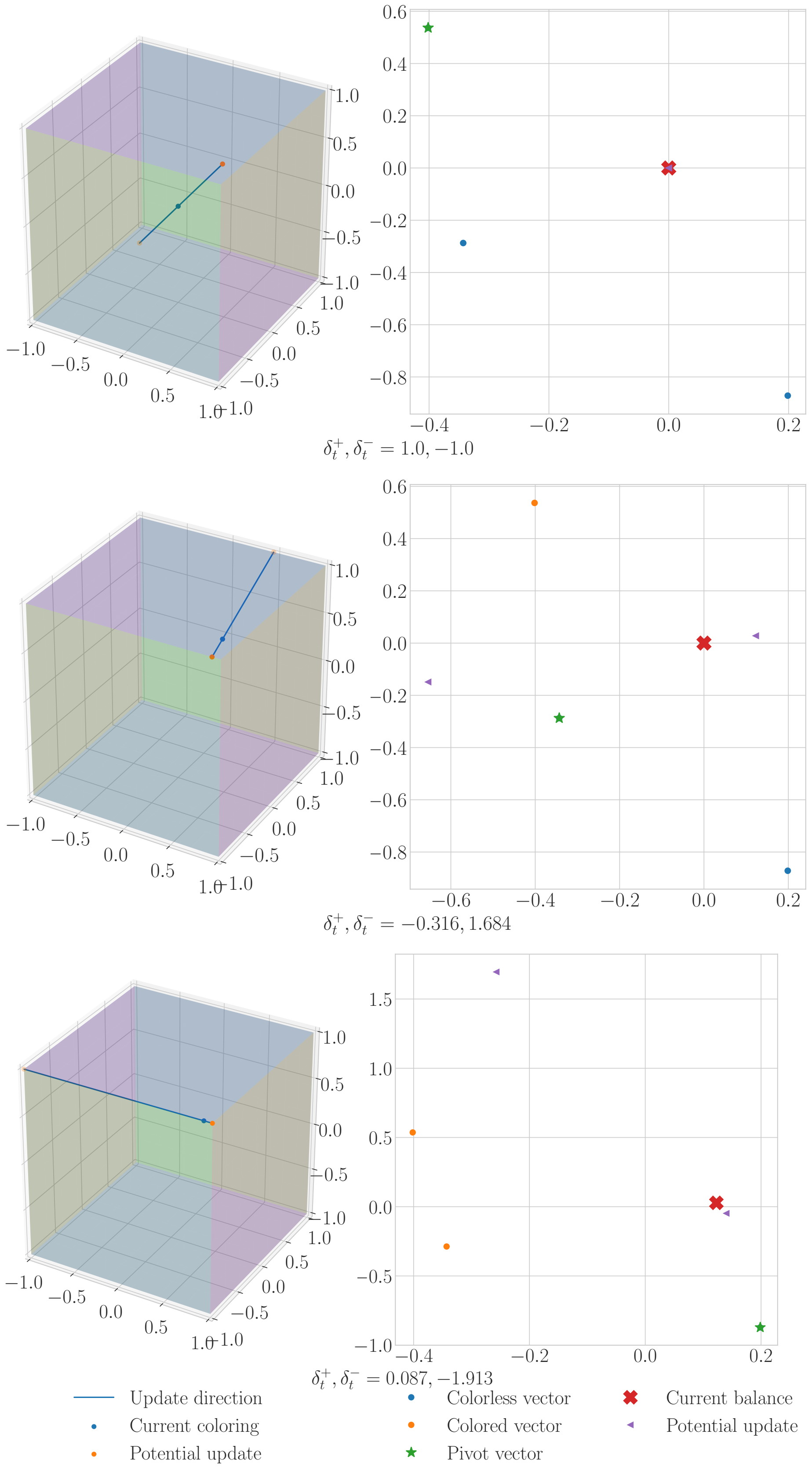
Figure 2: Plot of vectors of imbalances of group assignments from the GSW, its input vectors, and samples from a centered normal with $\mathbf{Cov} = \mathbf{I}_2$ similarly to figure 1, except we added the vectors of imbalances of random group assignments for comparison. There are 100 input vectors and 100 of each kinds of point, everything is in dimension 2.

Theorem 7 ([6])**.** *For* $\boldsymbol{z}$ *sampled via the Gram-Schmidt walk with input* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^d$ *each with* ℓ_2 *norm at most 1, we have that* $\boldsymbol{Bz}$ *is subgaussian with parameter* $\sigma^2 = 1$*.*

This result is necessary to prove that the GSW indeed gives a constructive algorithm for Theorem 1, by using Theorem 3.

4 Fast Computations

Thanks to an original idea from professor Marcus, one can make the GSW run faster than the trivial implementation which takes time $O(n^3 d)$ down to $O(n^2 d + n^3)$, similarly to what was done in [6] using Cholesky factorizations, except the proofs are in our opinion easier to understand. The general idea is similar: at the beginning of a GSW run, we compute some matrices and then during each step of the algorithm, we use only rank one updates to maintain



$\delta_t^+, \delta_t^- = 1.0, -1.0$

$\delta_t^+, \delta_t^- = -0.316, 1.684$

$\delta_t^+, \delta_t^- = 0.087, -1.913$

Figure 3: Example of a Gram-Schmidt walk run with 3 vectors in 2 dimension. The left part shows the cube where the coloring is living, and the right part shows the vector of imbalances and the input vectors.

them and derive the update direction from them at each iteration. This results in the removal of an n factor from the asymptotic running time since rank-one update can be computed in quadratic time rather than the cubic time necessary to compute matrix multiplications and inverses.

We first remind the definition of the matrix pseudoinverse:

Definition 8. *The pseudoinverse $\boldsymbol{M}^\dagger \in \mathbb{R}^{n\times m}$ of a matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ is the unique matrix that verifies:*

1. $\boldsymbol{M}\boldsymbol{M}^\dagger \boldsymbol{M} = \boldsymbol{M}$
2. $\boldsymbol{M}^\dagger \boldsymbol{M}\boldsymbol{M}^\dagger = \boldsymbol{M}^\dagger$
3. $(\boldsymbol{M}\boldsymbol{M}^\dagger)^T = \boldsymbol{M}\boldsymbol{M}^\dagger$
4. $(\boldsymbol{M}^\dagger \boldsymbol{M})^T = \boldsymbol{M}^\dagger \boldsymbol{M}$

The pseudoinverse is basically a generalization of the inverse for singular and non-square matrices. In particular $\mathbf{P} = \mathbf{M}\mathbf{M}^\dagger$ is the orthogonal projector onto the column space of $\mathbf{M}$.

For some set $S \subset \{1, \dots, n\}$, let $\mathbf{I}_S \in \mathbb{R}^{n\times n}$ be the diagonal $n \times n$ matrix with $\mathbf{I}_S(i,i) = 1$ if $i \in S$ and 0 otherwise, and $\mathbf{B}_S \in \mathbb{R}^{d\times n}$ be $\mathbf{B}\mathbf{I}_S$ where $\mathbf{B}$ is as previously the matrix containing the input vectors as columns. We define $\mathbf{C}_S \in \mathbb{R}^{n\times d}$ to be such that $\mathbf{C}_S\mathbf{B}_S = \mathbf{I}_S$ and if $i \notin S$, the ith row of $\mathbf{C}_S$ is $\mathbf{0}$. That is, the rows of $\mathbf{C}_S$ whose index are in S are the pseudo-inverse of the corresponding columns in $\mathbf{B}_S$. Finally, $\mathbf{D}_S \in \mathbb{R}^{n\times n}$ is $\mathbf{C}_S\mathbf{C}_S^T$.

This technique necessitates that the dimension of the space spanned by the input vectors is at most the dimension d of the input vectors, as otherwise $\mathbf{C}_S$ doesn't exist. One way to adapt it to the other cases is to add an identity matrix multiplied by some small factor at the bottom of the $\mathbf{B}_S$ matrix. That way, we can reduce it to the case where the space spanned by the vectors is large enough, and the results differ only very slightly from those we'd obtain via conventional computational methods. This technique does introduce some numerical issues for big n and d though, and it seems preferable to use a more conventional implementation of the GSW in this case. Another way would be to use a parameter and basically apply the GSW design from [6] with a very small ϕ, which would be similar to the previous technique outside of the scale.

Here are the rank one updates that let us compute our matrices for some $S' = S \setminus \{k\}$.

Lemma 9. *For $k \in S$, let*

- $\boldsymbol{c}_k \in \mathbb{R}^n$ *be the kth row of* $\boldsymbol{C}_S$
- $\boldsymbol{d}_k \in \mathbb{R}^d$ *be the kth row of* $\boldsymbol{D}_S$
- $S' = S \setminus \{k\}$

Then $\boldsymbol{C}_{S'} = \boldsymbol{C}_S - \frac{1}{\boldsymbol{d}_k(k)}\boldsymbol{d}_k\boldsymbol{c}_k^T$ *and* $\boldsymbol{D}_{S'} = \boldsymbol{D}_S - \frac{1}{\boldsymbol{d}_k(k)}\boldsymbol{d}_k\boldsymbol{d}_k^T$.

Proof. For the first part, we have that

$$\begin{aligned}\left(\mathbf{C}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\right)\mathbf{B}_{S'} &= \left(\mathbf{C}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\right)\mathbf{B}_S\mathbf{I}_{S'}\\ &= \mathbf{C}_S\mathbf{B}_S\mathbf{I}_{S'} - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\mathbf{B}_S\mathbf{I}_{S'}\\ &= \mathbf{I}_{S'} - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{e}_k^T\mathbf{I}_{S'}\\ &= \mathbf{I}_{S'} - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{0}\\ &= \mathbf{I}_{S'}\end{aligned}$$

Thus we just need to show that the rows of $\mathbf{C}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T$ not in S' are $\mathbf{0}$. As $\frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k(k) = 1$, we have that the kth row of $\mathbf{C}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T$ is indeed $\mathbf{0}$. Lastly, as $\mathbf{D}_S = \mathbf{C}_S\mathbf{C}_S^T$, we have that $i \notin S \Rightarrow \mathbf{d}_k(i) = 0$ thus all the rows whose indices aren't in S indeed do stay $\mathbf{0}$.

For the second part, we can see that

$$\begin{aligned}\mathbf{D}_{S'} &= \mathbf{C}_{S'}\mathbf{C}_{S'}^T\\ &= \left(\mathbf{C}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\right)\left(\mathbf{C}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\right)^T\\ &= \left(\mathbf{C}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\right)\left(\mathbf{C}_S^T - \frac{1}{\mathbf{d}_k(k)}\mathbf{c}_k\mathbf{d}_k^T\right)\\ &= \mathbf{C}_S\mathbf{C}_S^T - \frac{1}{\mathbf{d}_k(k)}\mathbf{C}_S\mathbf{c}_k\mathbf{d}_k^T - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\mathbf{C}_S^T + \frac{1}{\mathbf{d}_k(k)^2}\mathbf{d}_k\mathbf{c}_k^T\mathbf{c}_k\mathbf{d}_k^T\\ &= \mathbf{D}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{d}_k^T - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{d}_k^T + \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{d}_k^T\\ &= \mathbf{D}_S - \frac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{d}_k^T\end{aligned}$$

□

Now let's see why this is useful, that is, how to compute $\mathbf{v}^{\perp}(t)$ and $\mathbf{u}_t$ from these matrices.

Lemma 10. *Assume we're currently in a step of the GSW where the active vectors are* $A_t = \{\boldsymbol{v}_s : s \in S\}$, *and* $\boldsymbol{v}_k \in A_t$ *is the pivot. If* $\boldsymbol{d}_k$ *and* $\boldsymbol{c}_k$ *are defined similarly as in Lemma 9, we have that*

$$\boldsymbol{v}^{\perp}(t) = \frac{\boldsymbol{c}_k}{\|\boldsymbol{c}_k\|^2} \text{ and } \boldsymbol{u}_t = \frac{\boldsymbol{d}_k}{\boldsymbol{d}_k(k)}.$$

Proof. Let $S' = S \setminus \{k\}$. Starting from the definition of $\mathbf{v}^{\perp}(t)$ with pivot $p(t) = k$, we have that

$$\begin{aligned}
\mathbf{v}^{\perp}(t) &= \Pi_{V_t^{\perp}}(\mathbf{v}_k) && \\
&= \mathbf{v}_k - \Pi_{V_t}(\mathbf{v}_k) && \text{span } V_t \cup V_t^{\perp} = \mathbb{R}^d \\
&= \mathbf{v}_k - \mathbf{B}_{S'}\mathbf{C}_{S'}\mathbf{v}_k && \text{Definition of } \Pi_{V_t} \text{ with pivot } k \\
&= \mathbf{v}_k - \mathbf{B}_{S'}\left(\mathbf{C}_S - \tfrac{1}{\mathbf{d}_k(k)}\mathbf{d}_k\mathbf{c}_k^T\right)\mathbf{v}_k && \text{Lemma 9} \\
&= \mathbf{v}_k - \mathbf{B}_{S'}\mathbf{e}_k + \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_{S'}\mathbf{d}_k\mathbf{c}_k^T\mathbf{v}_k && \mathbf{C}_i\mathbf{v}_k^T = \mathbb{1}_{i=k} \\
&= \mathbf{v}_k + \mathbf{0} + \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_{S'}\mathbf{d}_k && k\text{th line of } \mathbf{B}_{S'} \text{ is } \mathbf{0} \text{ as } k \notin S' \\
&= \mathbf{v}_k + \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_{S'}(\mathbf{e}_k^T\mathbf{C}_S\mathbf{C}_S^T)^T && \mathbf{d}_k \text{ is the } k\text{th row of } \mathbf{D}_S = \mathbf{C}_S\mathbf{C}_S^T \\
&= \mathbf{v}_k + \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_{S'}\mathbf{C}_S\mathbf{C}_S^T\mathbf{e}_k && (\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T \\
&= \mathbf{v}_k + \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_{S'}\mathbf{C}_S\mathbf{c}_k && \\
&= \mathbf{v}_k + \tfrac{1}{\mathbf{d}_k(k)}\left(\mathbf{B}_S - \mathbf{v}_k\mathbf{e}_k^T\right)\mathbf{C}_S\mathbf{c}_k && \text{Definition of } \mathbf{B}_{S'} \\
&= \mathbf{v}_k + \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_S\mathbf{C}_S\mathbf{c}_k^T - \tfrac{1}{\mathbf{d}_k(k)}\mathbf{v}_k\mathbf{c}_k^T\mathbf{c}_k && \\
&= \mathbf{v}_k + \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_S\mathbf{C}_S\mathbf{c}_k^T - \tfrac{\mathbf{d}_k(k)}{\mathbf{d}_k(k)}\mathbf{v}_k && \mathbf{D}_S = \mathbf{C}_S\mathbf{C}_S^T \\
&= \tfrac{1}{\mathbf{d}_k(k)}\mathbf{B}_S\mathbf{C}_S\mathbf{c}_k^T && \\
&= \tfrac{1}{\mathbf{d}_k(k)}(\mathbf{B}_S\mathbf{C}_S)^T\mathbf{c}_k^T && \text{Pseudo-inverse definition} \\
&= \tfrac{1}{\mathbf{d}_k(k)}\mathbf{C}_S^T\mathbf{B}_S^T\mathbf{c}_k^T && (AB)^T = B^TA^T \\
&= \tfrac{1}{\mathbf{d}_k(k)}\mathbf{c}_k^T && \text{Pseudo-inverse definition} \\
&= \tfrac{1}{\|\mathbf{c}_k\|^2}\mathbf{c}_k^T && \mathbf{D}_S = \mathbf{C}_S\mathbf{C}_S^T
\end{aligned}$$

For the second part, we have that $\mathbf{u}_t$ is the solution to the system

$$\begin{aligned}
\mathbf{u}_t(k) = 1 \\
\mathbf{u}_t(i) = 0 \text{ if } i \notin A_t \\
\mathbf{v}^{\perp}(t) = \mathbf{v}_k + \sum_{i \in A_t \setminus \{p(t)\}} \mathbf{u}_t(i)\mathbf{v}_i = \mathbf{B}_S\mathbf{u}_t
\end{aligned}$$

But we have that

$$\mathbf{v}^{\perp}(t) = \frac{\mathbf{c}_k}{\|\mathbf{c}_k\|^2} = \frac{\mathbf{C}_S^T\mathbf{e}_k}{\|\mathbf{c}_k\|^2}$$

if we multiply by $\mathbf{C}_S$ we get $\frac{\mathbf{C}_S\mathbf{C}_S^T\mathbf{e}_k}{\|\mathbf{c}_k\|^2} = \mathbf{u}_t$ But as $\mathbf{C}_S\mathbf{C}_S^t = \mathbf{D}_S$ and $\|\mathbf{c}_k\|^2 = \langle \mathbf{c}_k, \mathbf{c}_k \rangle = \mathbf{d}_k(k)$, we have that indeed $\frac{\mathbf{d}_k}{\mathbf{d}_k(k)} = \mathbf{u}_t$. □

These lemmas mean that once we have computed $\mathbf{C}_S$ and $\mathbf{D}_S$ at the beginning of the GSW in time $O(n^2d + n^3)$, we update them at most n times in $O(nd + n^2)$ operations and store them with a similar quantity of memory, resulting in a total time of running of $O(n^2(d + n))$ as computing δ_t and updating the coloring takes $O(n)$ time.

This happens to not be much faster in practice than a clever implementation using rank-one update to compute the inverse necessary to compute the least squares solution. That is because in the latter case, the size of the inverse matrix decreases at each step of the algorithm in that case, thus the one matrix multiplication necessary to compute the update direction becomes faster than the rank one updates described in this section quickly when n is small, as $\mathbf{C}_S$ and $\mathbf{D}_S$ keep the same size throughout the whole algorithm run.

5 Generalizing to any Hyperparallelepiped

The idea is to allow the algorithm to produce colorings not only on the hypercube $[-1, 1]^n$, but also on any hyperparallelepiped spanned by a set $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$ of n linearly independent vectors, denoted as the basis $\mathcal{B}$. The assignment returned then won't be a coloring per se but just a linear combination of the vectors with some coefficients corresponding to a vertex of the hyperparallelepiped formed by the basis vectors that should hopefully yield a somewhat short vector of imbalances. We will still denote it by the term coloring for clarity reasons.

To do so, we have to adapt the algorithm at several points. Firstly, an element k associated to the basis vector $\mathbf{b}_k$ should be alive only if the current coloring is not on one of the 2 facets such that when expressed in the basis $\mathcal{B}$, the kth coordinate is -1 or 1.

Secondly, choosing the update direction is then different as we need to stay orthogonal from every fixed vector, but these vectors do not correspond to coordinates anymore as we're not using the trivial basis $\mathbf{e}_1, \ldots, \mathbf{e}_n$ anymore.

Thirdly, the computation of the two potential δ is different as well for some similar reasons: you have to compute the maximum delta such that the coloring stays in the hyperparallelepiped defined by the basis.

All these issues can be solved via some basis changes in the right spots and the modified algorithm is very similar, and should work exactly the same way when given the orthonormal canonical basis of $\mathbb{R}^n$, $\mathbf{e}_1, \ldots, \mathbf{e}_n$. The main idea is that we will keep two separate coloring, $\mathbf{z}$ and $\mathbf{z}'$, where $\mathbf{z}$ is expressed in the canonical basis and $\mathbf{z}'$ is expressed as coordinates in $\mathcal{B}$. We will update $\mathbf{z}'$ with $\mathbf{u}'_t$, that is $\mathbf{u}_t$ except it is expressed via its coordinates in $\mathcal{B}$. Computing δ_t is then fairly easy as we just have to replace $\mathbf{z}$ by $\mathbf{z}'$ in the normal computation.

Let $\mathbf{B}_{\mathcal{B}}$ be the matrix containing the basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ as columns. In order to compute $\mathbf{u}'_t$, we replace the matrix containing our input vectors as columns, $\mathbf{B}$, by $\mathbf{B}\mathbf{B}_{\mathcal{B}}$. We then apply the same method as usual to find $\mathbf{u}'_t$, and change its basis to get $\mathbf{u}_t$.

Putting it all in pseudocode and highlighting the difference with the original algorithm (1) in red, we get the Algorithm 3.

Algorithm 3: The Gram-Schmidt walk on a different basis

```
Input : $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^d$, an initial coloring $\mathbf{z}_0 \in [-1,1]^n$, $n$ linearly independent basis vectors $\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^n$
Output: a coloring $\mathbf{z}_n \in \mathbb{R}^n$
$\mathbf{z}'_0 = \mathbf{B}_{\mathcal{B}}^{-1}\mathbf{z}_0$
$A_1 = \{i \in [n] : |\mathbf{z}'_0(i)| < 1\}$, $n(1) = \max\{i \in A_1\}$ and $t = 1$.
$\mathbf{B} = (\mathbf{v}_1, \dots, \mathbf{v}_n)\mathbf{B}_{\mathcal{B}}$
while $A_t \neq \emptyset$ do
    Compute $\mathbf{u}'_t \in \mathbb{R}^n$ such that $\begin{cases} \mathbf{u}'_t(p(t)) = 1 \\ \mathbf{u}'_t(i) = 0 \text{ if } i \notin A_t \\ \mathbf{v}^{\perp}(t) = \mathbf{B}\mathbf{u}'_t \end{cases}$
    $\Delta = \{\delta : \mathbf{z}'_{t-1} + \delta\mathbf{u}'_t \in [-1,1]^n\}$, let $\begin{cases} \delta_t^+ = \max \Delta \\ \delta_t^- = \min \Delta \end{cases}$ then $\delta_t = \begin{cases} \delta_t^+ \text{ w.p. } \frac{-\delta_t^-}{(\delta_t^+ - \delta_t^-)} \\ \delta_t^- \text{ w.p. } \frac{\delta_t^+}{(\delta_t^+ - \delta_t^-)} \end{cases}$
    $\mathbf{z}'_t = \mathbf{z}'_{t-1} + \delta_t\mathbf{u}'_t$, $t \leftarrow t+1$, $A_t = \{i \in [n] : |\mathbf{z}'_{t-1}(i)| < 1\}$, $p(t) = \max\{i \in A_t\}$.
end
Output $\mathbf{B}_{\mathcal{B}}\mathbf{z}'_t \in \mathbb{R}^n$.
```

An example of output of the GSW with a different coloring basis can be found in figure 4, that contains a plot of vectors of imbalances of the classical GSW, and the GSW with two different basis; one that is all positive vectors of norm 1 and one that is orthonormal. We see that the vectors of imbalances of the latter resemble closely those of the classical GSW, while those obtained using the former have a skewed distribution.

6 Experiments and Properties

In this section, we try to see how the GSW behaves, both generally and when given some specific inputs. We also try some modifications that aim to modify its behavior for some specific purposes.

6.1 How good is the GSW at minimizing output the norm of the vector of imbalances in practice ?

We want to see how well the GSW actually minimizes the norm of the resulting vector of imabalances in comparison with other variants and algorithms. We will compare it to the naive walk (Algorithm 2), the deterministic GSW (also referred to as DGSW, defined in section 3.4) and the assignment that has the actual lowest vector of imbalances norm, computed via brute force, but only for small values of n for this last one.

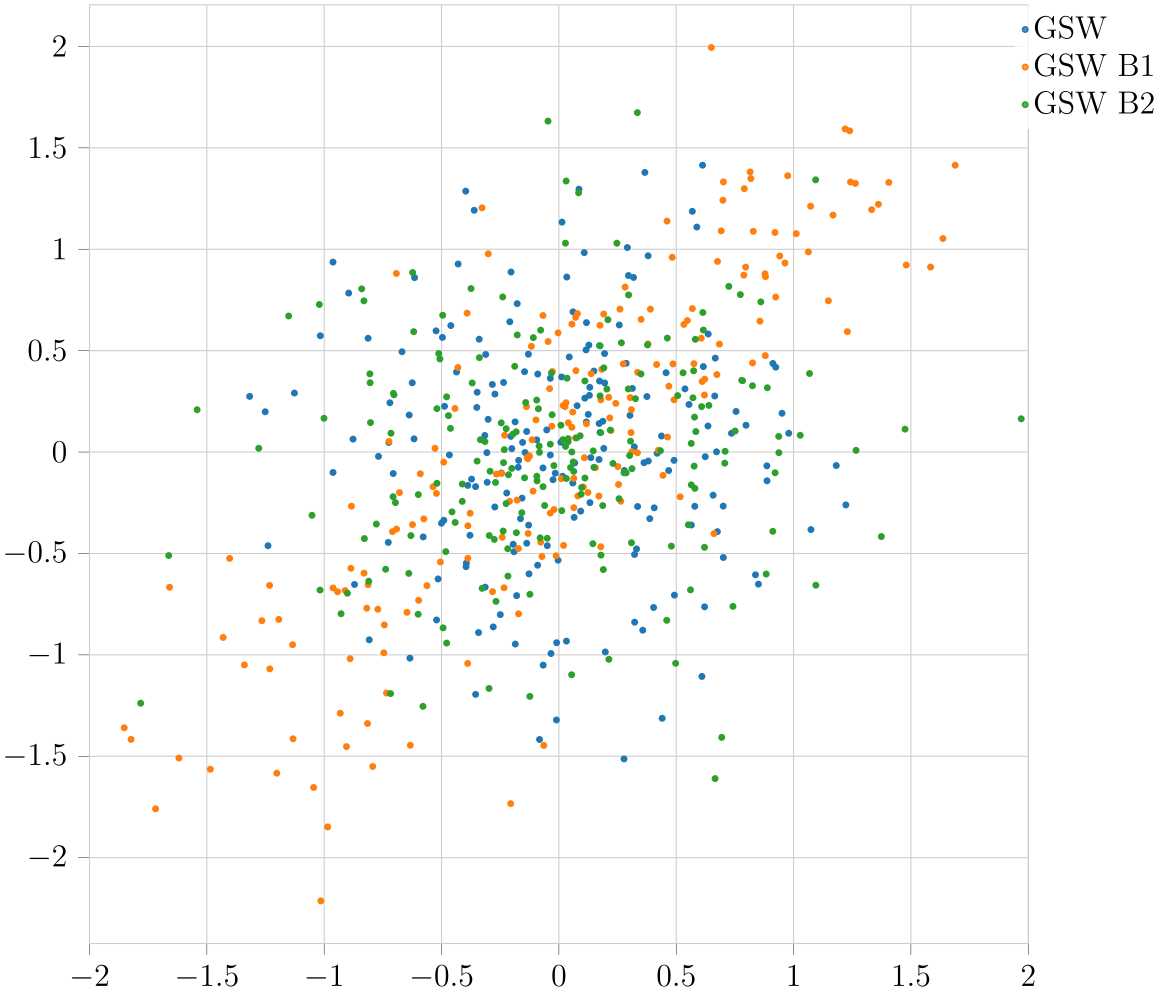


Figure 4: Plot of vectors of imbalances of group assignments from the GSW, the GSW with a basis where every vector has only positive coordinates and is of norm 1 (**B1**) and the GSW with a random orthonormal basis (**B2**), using the generalization from section 5. There are 200 input vectors from $\mathbb{R}^2$, and 200 outputs of each type.

6.1.1 Experiment 1

We compare the output norm of the vector of imbalances of the GSW, DGSW, the naive walk, **NW**, and for some small n also the best assignment found via brute forcing on all possibilities, **LDA**. We do this for $n = 5, 10, 20, 40, 80, 160$, and with $d = 2^i$ for $i \in [15]$, where we sample n vectors from the d-dimensional ball of radius 1. We do one thousand runs for each d and each n, each with a different set of n vectors sampled from the ball of radius 1.

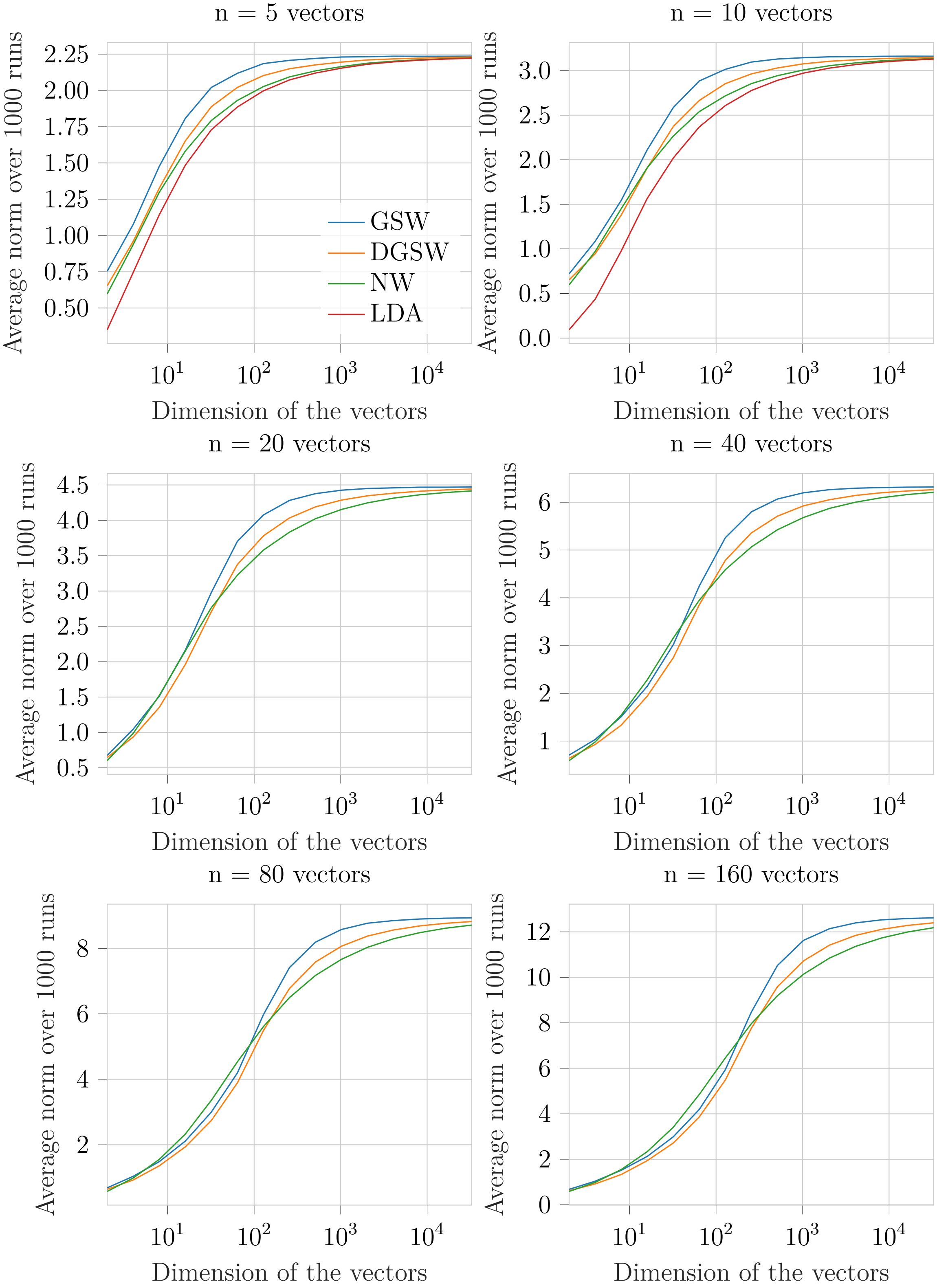


Figure 5: Results of experiment 1 from section 6.1.1: comparison of different norm of vector of imbalances minimizing algorithms for $n = 5, 10, 20, 40, 80, 160$ vectors of dimension between 2 and 2^{15}. Results were averaged over 1000 runs.

6.1.2 Results

Our results are visible in figure 5. We can see that GSW actually gives the worst results in terms of minimization of the norm of the vector of imbalances, but that when the dimension of the vector grows all methods seem to give similar results asymptotically. Note that we cannot say that the naive walk is just a better minimizing algorithm for this problem as these results would probably be different if we modified the distribution of input vectors.

6.1.3 Experiment 2

While the GSW obviously has some other desirable properties, there should be input vector sets on which the GSW returns shorter vector of imbalances on average than the naive walk. If the vectors aren't shuffled before the naive walk, it is trivial to find a set on which the GSW produces shorter vectors of imbalances. If they are, it isn't as trivial but it still isn't too hard. For example let's run the GSW and the naive walk on the set containing 100 copies of both v_1 and v_2 where

$$v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_2 = \begin{pmatrix} \frac{10}{\sqrt{2}} \\ \frac{10}{\sqrt{2}} \end{pmatrix}.$$

We will record the norm of the resulting vector of imbalances for 100 runs of each walk and observe the average.

6.1.4 Results 2

The resulting average norm is exactly 0 for the GSW and 3.47 for the naive walk, showing that the naive walk can yield much higher norms when the input vectors are less well-behaved. Indeed, the naive walk doesn't use the knowledge of the full input vector set like the GSW does and thus fails to balance this set well.

6.2 Does translation of the input vectors have an effect on the balance of the assignment ?

By *balance of the assignment*, we mean the absolute difference between the number of 1s and -1s. An assignment is perfectly balanced if its sum is 0, that is the groups produced by the assignment have equal size.

If our initial group of vector is centered around $\mathbf{0}$, we would expect that translating it away will force it to have a greater balance between -1s and 1s in order to balance the translation part added to each vector.

6.2.1 Experiment

We sample 200 input vectors from the ball of radius 1 in dimension 200. We then run the GSW and DGSW 100 times each on those vector translated by some random norm 1 vector multiplied by some factor. We use the factors 0,1,2,5 and 10 and compare the results.

6.2.2 Results

Table 1: Results of our experiment from section 6.2 on the balance of assignments depending on how much the vectors are translated. The numbers shown are the average absolute value of the sums of output vectors. A smaller number indicates that the assignment is more balanced, that is the number of 1's and -1's are closer.

Factor	0	1	2	5	10
GSW	9.7	0.96	0.46	0.1	0.06
DGSW	44.5	1.78	0.32	0.08	0.04

Results are available in table 1. We can see that indeed the further away from 0 our input vectors are translated the more balanced the assignments are, as expected. It's interesting to notice that assignments from the DGSW are way less balanced than with the GSW. I currently do not have any hypothesis on why that is.

These experiments make us want to try to build a variant of GSW that can have a balance parameter thanks to the balancing properties of translation. That is what we try in the following experiment.

6.3 Can we add a parameter to balance assignments ?

Inspired by section 6.2, we propose a slight modification of the GSW that pushes toward balanced assignments. The idea is to add a coordinate to the input vector and give more or less importance to that coordinate similarly to how the balance-robustness tradeoff is implemented in [6], except here we implement a tradeoff between assignment balance and norm of the vector of imbalances.

Given input vectors $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^d$ and a parameter $\mu \in [0, 1]$, we define $\mathbf{w}_1, \dots, \mathbf{w}_n \in \mathbb{R}^{d+1}$ as

$$\mathbf{w}_i = \begin{pmatrix} \sqrt{1-\mu}\mathbf{v}_i \\ \sqrt{\mu} \end{pmatrix}.$$

This way the $\mathbf{w}_i$'s have similar norm to the $\mathbf{v}_i$'s, but it is possible that a different normalization depending on the norm of the $\mathbf{v}_i$'s could be better. We then run the GSW on the $\mathbf{w}_i$'s and use the output assignment on the $\mathbf{v}_i$'s. Choosing $\mu = 0$ is equivalent to doing the classical GSW algorithm, while using $\mu = 1$ is equal to forcing maximum balance. We run experiments to determine how much balance in the assignment we gain and how much further from $\mathbf{0}$ our output is for different μ's.

6.3.1 Experiment

We use our construction from the previous paragraph to compute an assignment on the $\mathbf{w}_i$'s using GSW or DGSW, then record how balanced this assignment is, and how the norm of its vector of imabalances on the original $\mathbf{v}_i$'s varies. We also program the fixed size GSW described in [6] and compare it with our balancing method. We use n vectors sampled from the ball of radius 1 in dimension n for $n = 100$ and $\mu \in \{0, 0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99, 0.999, 1\}$. The results presented are averaged over 1000 runs.

6.3.2 Results

Table 2: Results of our experiment on the balance of assignments with our new balance-norm of the vector of imbalances tradeoff design. The balance measure shown (lines starting with AB for average balance) are the average absolute value of the sums of assignment vectors. The norm of the vector of imbalances numbers shown (lines starting with AN for average norm of imbalance) are the norms of the sum of $\mathbf{v}_i \cdot \mathbf{z}_i$, $\mathbf{z}$ being the assignment produced by GSW or DGSW for the $\mathbf{w}_i$'s. The D at the end of the last two lines indicates that in this case, the deterministic GSW algorithms was used.
A smaller balance measure indicates that the assignment has a more balanced assignment, that is the number of 1s and -1s are closer. A smaller norm of imbalances measure indicates that the vectors are better balanced among the groups, that is the sum of each coordinate is closer to be the same in each group. FSD references the fixed size design from [6], which was used for the rightmost column in order to compare it with what is described in section 6.3.

μ	0	0.001	0.01	0.1	0.25	0.5	0.75	0.9	0.99	0.999	1	FSD
AB	8.09	6.25	4.16	1.9	1.11	0.58	0.27	0.11	0.01	0.002	0	0
AN	5.26	5.28	5.27	5.31	5.34	5.32	5.34	5.33	5.33	5.34	9.92	5.32
ABD	20.58	18.7	11.91	3.89	2.04	0.99	0.33	0.12	0.02	0.002	0	0
AND	4.86	4.84	4.78	4.82	4.89	4.94	4.99	5.0	5.0	5.0	9.85	5.01

We can see that we can massively increase assignment balance without making the norm of the vector of imbalances much larger. For $\mu = 0.999$ for example, it is very likely that an assignment is exactly balanced and thus this variant of the algorithm can provide balanced GSW assignments with high probability while keeping most properties of the classical GSW, as opposed to the balanced variant described in [6], for which we don't know if some of the original properties still hold.

6.4 Does the norm of a vector affect when it is colored ?

The expected heuristic would have been that bigger vectors are colored earlier and the algorithm then colors the smaller ones to balance the longer ones in order to minimize the norm of the vector of imbalances, as that is what we'd instinctively do. It turns out that the algorithm actually does the reverse and colors the smaller vectors earlier than the bigger ones.

We performed two experiments to observe this behavior. In both experiments, we have vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{200}$ of increasing norm and observe how close the coloring order is to $\mathcal{R} = \{200, 199, \ldots, 1\}$. To do so, if $\mathcal{O}$ is the observed order, we look at the quantity

$$\Delta_o = \sum_{i=1}^{200} |\mathcal{R}_i - \mathcal{O}_i|. \tag{1}$$

The smaller it is, the closer the 2 orders are. For each of the two experiments below, we ran the GSW 100 times and the deterministic GSW (DGSW) 100 times and recorded Δ_o.

6.4.1 Experiment 1

We sampled 200 vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{200}$ in the ball of radius 1 of dimension 200. For each $\mathbf{v}_i$, we replaced it by $i \cdot \mathbf{v}_i / \|\mathbf{v}_i\|$ so that $\|\mathbf{v}_i\| = i$ for each vector. We call this vector set the incrementally growing norm set (IGN).

6.4.2 Experiment 2

We sampled 200 vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{200}$ in the ball of radius 1 of dimension 200. For each $\mathbf{v}_i$, we replaced it by $X_i \cdot \mathbf{v}_i / \|\mathbf{v}_i\|$ so that $\|\mathbf{v}_i\| = X_i$ for each vector, where $X_i = 1 + 199 \cdot \mathbb{1}_{i<n/2}$. We call this vector set the two group set (2G).

6.4.3 Results

We also ran the same experiments with 200 vectors of constant norm 1 as a comparison, which we call the all equal norm set (AEN). The results are summarized in table 3.

Table 3: Results of our experiments on the moment of coloring depending on the norm of the vectors. The numbers shown are the Δ_o as defined in equation 1.

	IGN	2G	AEN
GSW	18664.4	19997.32	13607.06
DGSW	18641.6	19996.16	13431.68

We can see that the vector orders are actually further away from $\mathcal{R}$ than the random orders produced with constant vectors, which means that bigger vectors actually get colored later in the process. Additionally, the DGSW doesn't seem to yield significantly different results. While this is not what we expected upon a first glance at the problem, this behavior actually makes sense, because if we multiply $\mathbf{v}_i$ by μ, it's corresponding coordinate in $\mathbf{u}_t$ is going to be divided by μ, thus it will move less towards the facet of the hypercube corresponding to the color of $\mathbf{v}_i$ and thus be colored later than the shorter vectors.

This motivates us to try to find a variant of the algorithm that would color bigger vectors earlier and thus yield shorter vector of imbalances on an input such as $\{\mathbf{v}, \mathbf{v}, \mathbf{v}, 3\mathbf{v}\}$ for some $\mathbf{v} \in \mathbb{R}^d$ and an arbitrary $d \in \mathbb{N}$. One way would be to choose the pivot through some smart condition or to choose another feasible $\mathbf{u}_t$ than the default least square solution with minimal norm, as in that specific adversarial example there are an infinite number of valid $\mathbf{u}_t$, some of which solve our problem in one step with a perfectly balanced groups.

One such idea is to force the pivot to be the largest norm vector, and then, when $\mathbf{v}_\perp = \mathbf{0}$, to select $\mathbf{u}_t$ through lasso regression with a very small alpha ($\alpha = 10^{-32}$ for example) in order to ensure that the smallest possible number of coordinates are nonzero, and finally to select δ_t by taking it to be of the same sign as the coordinate of x corresponding to the pivot (or randomly if that coordinate is 0). This variant solves the issue mentioned in the previous paragraph but trades a lot of randomness for some additional complexity in the process, so there probably exists some different additional constraint to add when computing $\mathbf{u}_t$ in colinear cases that could work even better.

A second variant that might help is to do quadratic programming instead of simple least squares when $\mathbf{v}_\perp(t) = \mathbf{0}$. This way, we can force the quadratic program to minimize both $\|\mathbf{B}\mathbf{u}_t\|^2$ but also $\|\mathbf{u}_t\|^2$. This could be achieved by adding a line of 1's to the matrix $\mathbf{B}$ containing all input vectors as column, and adding as conditions that the chosen $\mathbf{u}_t$ must have a 1 in the pivot coordinate and 0's in already colored coordinates, similarly to the conditions in the original GSW system of equation. Then if the $\mathbf{u}_t$ chosen through this quadratic program doesn't yield $\mathbf{B}\mathbf{u}_t = \mathbf{0}$, we discard it and compute $\mathbf{u}_t$ through the usual method as the modification would then interfere with the normal properties of the GSW instead of just improving it by adding constraints when there are an infinite number of optimal solutions. This technique coupled with choosing the longest alive vector as pivot actually solves adversarial cases similar to those mentioned three paragraphs ago, as well as more complicated ones such as $\{\mathbf{v}, \mathbf{v}, \mathbf{v}, 3\mathbf{v}, \mathbf{w}, \mathbf{w}, 2\mathbf{w}\}$, and doesn't modify the behavior of the algorithm when $\mathbf{v}^\perp(t) \neq \mathbf{0}$. The only small issue is that the matrix $\mathbf{B}^T\mathbf{B}$ then needs to be regularized by adding some very small constant times the identity or it isn't strictly positive definite, which is necessary for our quadratic program solver.

A third potential method would be to again do quadratic programming, but this time minimize $|(|u| - 1)|$ or $(|u| - 1)^2$ in addition to $\|\mathbf{B}\mathbf{u}\|$, with the same constraints as previously. Despite not being a quadratic or linear function, absolute value in the objective function can be included, it is just necessary to transform it into linear constraints and additional variables, which is a common linear programming trick. This technique would function without forcing a specific pivot and seems thus superior to the previous one, but I didn't have time to program it and test it to make sure that my intuition is indeed correct.

This last variant seems interesting as it intuitively makes sense to color as much as possible in the early steps, as we have more degrees of freedom during these steps. I have not been able to find a situation where that seemed worse than the classical GSW so far.

Another interesting experiment in the same vein would be to check if these results change with the dimension.

6.5 Do longer vectors stay pivot for longer ?

As longer vectors are colored later in the algorithm on average, one could think that they're staying as the pivot for longer. To test this hypothesis we design the following experiment.

6.5.1 Experiment

We sample 200 vectors of norm 1 in dimension 200 and multiply 100 of them by 200 (G1 with norm 1, G2 with norm 200). We then run the GSW 100 times with all these vectors as input and record for how long vectors of different norms (1 and 200) stay pivot once they become pivot. We call this measure the **ALOSAP** (average length of stay at pivot), and it is measured in number of while loop steps.

6.5.2 Results

Table 4: Results of our experiments from section 6.5 on whether long and short vectors stay for longer as the pivot. Results are averaged across groups from 100 runs.

	ALOSAP G1	ALOSAP G2
GSW	6.101	2.281
DGSW	5.989	2.277

Results are visible in table 4. We see that shorter vectors tend to stay pivot for much longer than their longer counterparts. This could be explained by the fact that, as seen in section 6.4, longer vectors are colored later. This means that in the later part of a GSW run, most alive elements are in the second group and will thus have similar scale and color each other, whereas in the early steps of a run short vectors are colored by any pivot, and thus short vectors would be mostly chosen as pivot at the beginnning of a run of the GSW.

It would be interesting to confirm or deny that hypothesis with another experiment looking at a more detailed distribution of the colorings and pivots, but we didn't have the time to perform something like that.

6.6 Can we force bigger vectors to be colored earlier ?

Another technique that we could use would be to modify the choice of the direction. Currently, the bigger a vector is the later in the process it will be colored.

One could multiply the computed direction in each of its coordinate by the norm of the vector corresponding to that coordinate, or the squared norm. That would be a way of trying to compensate for the update coordinates of bigger vectors being smaller. This would remove the orthogonality of updates, but it doesn't mean that the assignment doesn't balance the vectors, even though we would expect the vector of imbalances obtained via this method to be somewhat longer still.

We also study how that affects the order of the coloring in experiments similar to those done in subsection 6.4.

6.6.1 Experiment 1

We sample 100 points from the ball of radius 1 in dimension $d = 2$ and run the classical GSW, as well as the two variant described in the current section 100 times each. We plot all vectors of imbalances obtained this way to see how these modifications affected the algorithm.

6.6.2 Experiment 2

We sample vectors similarly to the experiments performed in section 6.4 and measure similarly how close the coloring order is to the order $\mathcal{R} = \{200, 199, \ldots, 1\}$.

6.6.3 Results

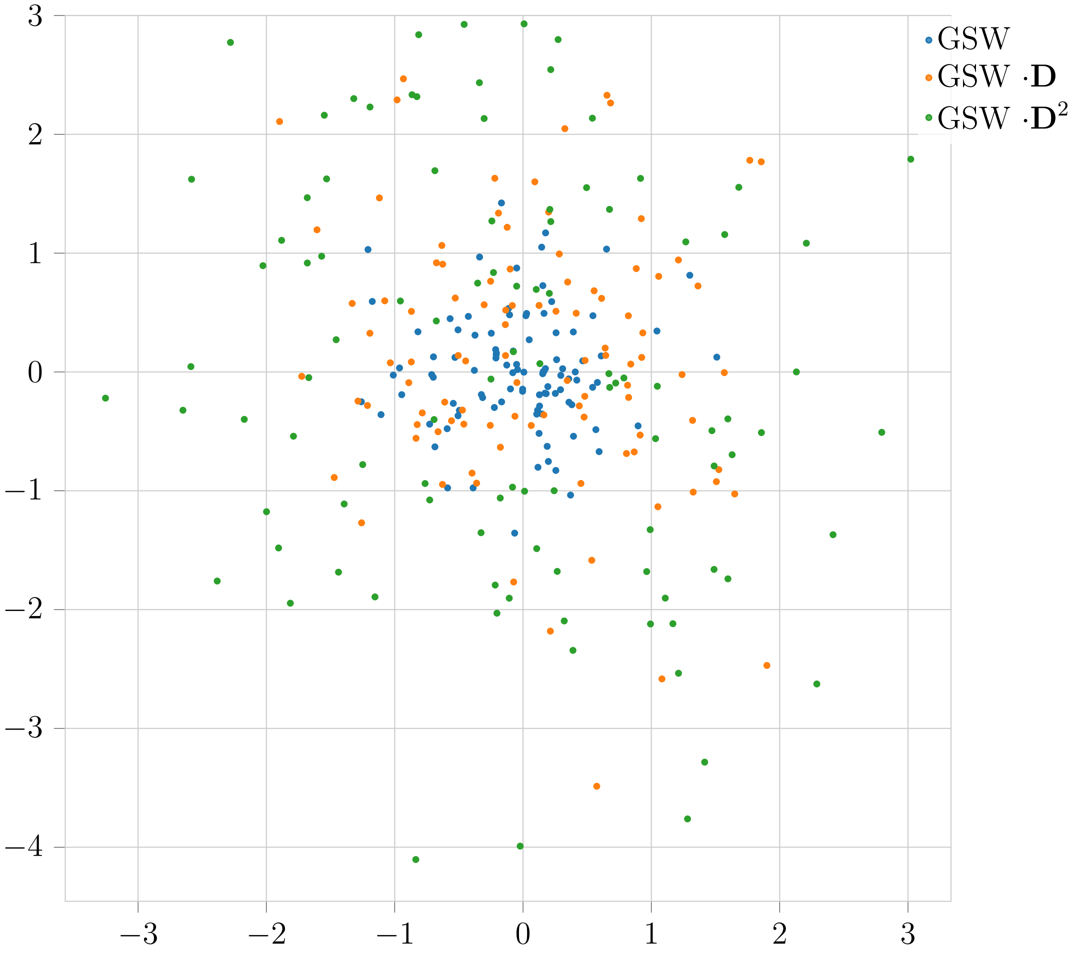


Figure 6: Results of experiment 1 from section 6.6.1: plot of vectors of imbalances of group assignments from the GSW, the GSW with update direction multiplied coordinatewise by the norm of the corresponding vector, and the GSW with update direction multiplied coordinatewise by the square of the norm of the corresponding vector. There are 100 of each kind of points and input vectors are in dimension 2.

Results from experiment 1 can be seen in figure 6. The modification of the computation of the update direction didn't seem to completely ruin how far the sum of outputs were from $\mathbf{0}$ if we compare it with figure 2 which contains random assignments. Still, the vectors of imbalances from modified GSW runs are more spread out than those from classical runs,

Table 5: Results of experiment 2 from section 6.6.2: trying to fix the later coloring of bigger norm vectors. The numbers shown are the Δ_o as defined in equation 1.

	IGN w/ D	IGN w/ D^2	2G w/ D	2G w/ D^2	AEN w/ D	AEN w/ D^2
GSW	13246.72	6078.28	13246.74	6697.1	13444.94	13208.1
DGSW	13100.02	6808.96	13597.86	6830.82	13303.18	13344.14

which is because we forced our modifications onto the algorithm without letting it adapt. We would need to find a smarter way that modifies the computation without just modifying the results at each step carelessly.

We can see from the results of experiment 2 in table 5. Our modifications indeed do remove the late coloring. Multiplying once makes it so the vectors are colored approximately randomly and multiplying twice makes it so the bigger vectors are colored earlier, as intended. Still, the norm of the vector of imbalances can be seen to be greater in the previous experiment thus this method isn't good.

6.7 Can we find another way of computing the update direction ?

As we saw that larger vectors get colored later in the process on average when using the classical algorithm, one could ask themselves how to revert this effect. Let $\mathbf{B}$ be the matrix containing our input vectors as columns. Using a singular value decomposition, we have that $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ where $\mathbf{U} \in \mathbb{R}^{d\times d}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are orthonormal and $\mathbf{\Sigma} \in \mathbb{R}^{d\times n}$ is all zeros except for the diagonal elements which are positive singular values. Using this decomposition, we can see that $\mathbf{B}^\dagger = \mathbf{V}\mathbf{\Sigma}^\dagger\mathbf{U}^T$ where $\mathbf{\Sigma}^\dagger$ is $\mathbf{\Sigma}^T$ except nonzero entries σ_i are replaced by their inverse $1/\sigma_i$. But if one replaced $\mathbf{\Sigma}^\dagger$ by $\mathbf{\Sigma}$, then the matrix multiplying the pivot vector would be $\mathbf{B}^T$ instead of $\mathbf{B}^\dagger$. This suggestion from Pr. Marcus turned out not to work if we want to balance the vectors, but it actually does something close to the opposite which is very interesting.

6.7.1 Experiment 1

We sampled 200 vectors in dimension 200 in the ball of radius 1. We then ran the modified GSW algorithm where the next direction is computed via $\mathbf{u}_t(\mathcal{A}_t \setminus \{p(t)\}) = \mathbf{B}_t^T\mathbf{v}_{p(t)}$ and $\mathbf{u}_t(p(t)) = 1, \mathbf{u}_t(i) = 0 \ \forall i \notin \mathcal{A}_t$, where $\mathbf{B}_t$ is the matrix containing the vectors that are still alive and not the pivot as columns. Everything else is kept similar. We also plot the maximizing naive walk (**MNW**), that is for this experiment we switch the "<" to a ">" in the inequality on line 4 of the pseudocode of the naive walk available in section 3.4, as a comparison between its outputs and the results is interesting.

6.7.2 Experiment 2

We run a similar experiment as in 6.1.1 except we look for the assignment maximizing the norm of the vector of imbalances via bruteforcing (**HDA**) and look at the naive walk trying to maximize the output norm (**MNW**), similarly to what is described in section 6.7.1.

6.7.3 Results

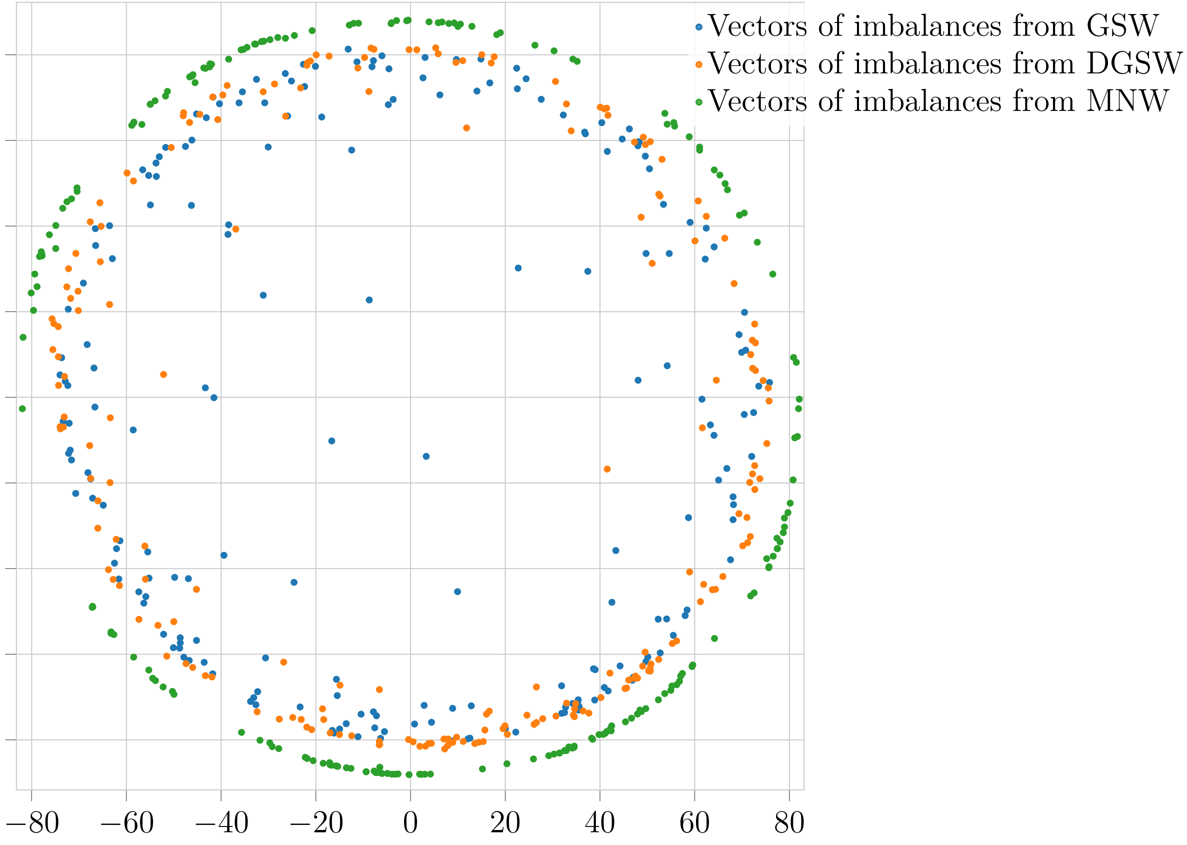


Figure 7: Results of experiment 1 from section 6.7.1. Vectors of imbalances using the modified GSW and DGSW where the update direction is computed by multiplying the pivot vector by $\mathbf{B}^T$, as well as from the maximizing naive walk.

The plot of the outputs from experiment 1 are shown in figure 7. We can see that this modification seems like it now minimizes output balance by pushing the vector of imbalances away from $\mathbf{0}$, which was surprising to us at least. This seems like it could be useful to sample from unbalanced group assignments, or to find a subset to remove to maximize along one dimension for example. This might be equivalent to an already known algorithm, but if not I think there might be interesting applications to this, as pseudoboolean maximization of positive definite quadratic forms is an NP problem as can be seen in [5].

For this exact distribution of input vectors we can see that the maximizing naive walk is always a step further from $\mathbf{0}$, as if there was a limit that this GSW variant could not go over. It would be interesting to understand why that is, and whether that could be used to sample on the outer part of a zonotope for example. We can notice as well that the DGSW vectors of imbalances seem to be more often on the circle than the GSW vectors of imbalances which sometimes ended up in the middle.

Results of experiment 6.7.2 are available in figure 8. We can see that for small n, using

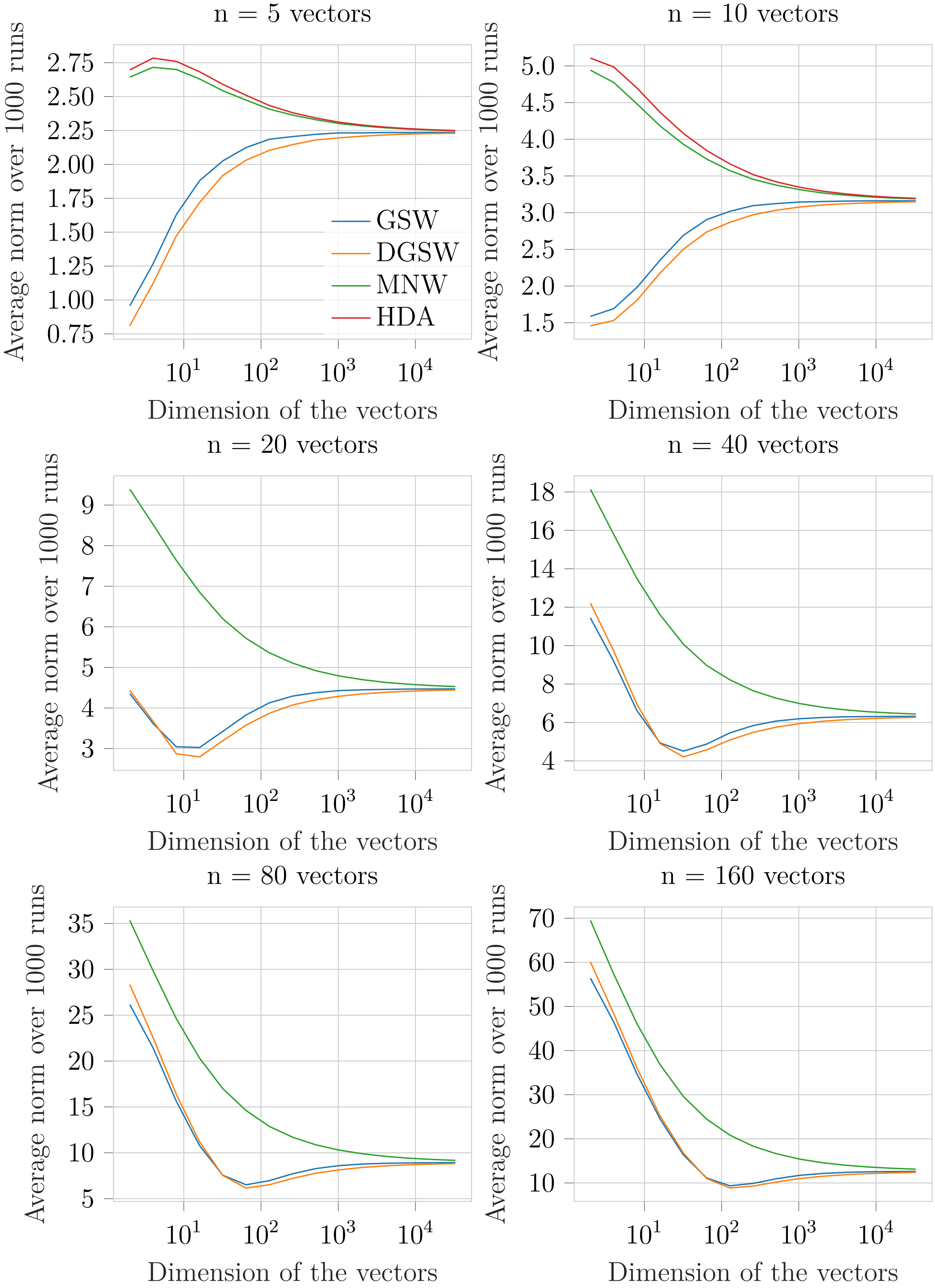


Figure 8: Results of experiment 2 (section 6.7.2): comparing the modified GSW and DGSW with norm of the vector of imbalances maximizing algorithms.

$\mathbf{B}^T$ doesn't actually maximize the norm for small dimensions even though it seems to asymptotically do so when the dimension grows. For bigger n's though, this variant seems to be decent at maximizing the norm of the vector of imbalances, even though it is worse at it than the naive walk for this particular distribution of input vectors. It is also interesting to notice that there seems to be a change of regime around the point $n = d$, where after that point the average norm of the vector of imbalances goes up with the dimension whereas it seems to go down before.

6.8 Are sparse vectors colored at the same moment as non-sparse vectors ?

We want to know if vectors with a lot of 0's can be found among vectors that are less sparse, as that could be very interesting to solve various problems such as the planted clique. As sparse vectors should be easier to color due to their smaller number of degrees of freedoms, we will investigate how early they're colored on average.

6.8.1 Experiment

We sample 100 vectors of dimension 100 from the ball of radius 1 but with 100 additional coordinates locked to zero. We also sample 100 vectors of dimension 200 from the ball of radius 1. We're interested in comparing whether vectors in one group get colored earlier than vectors from the other group on average. To do so we do 100 runs with each of three variants. In the first variant (V1), the vectors aren't changed. In the second one (V2), every vector is normalized. In the third one (V3), every vector is normalized but the non-sparse vectors are normalized to a norm of 2 so that the scale of the elements are similar to the sparse vectors normalized to a norm of 1. The last three variants are respective copies of the first three except the coordinates of each vectors are shuffled by a different random permutation.

6.8.2 Results

Table 6: Results of our experiments on whether sparse vectors are colored earlier from section 6.8. The numbers shown are the average coloring step of sparse vectors over 100 runs of the GSW.

	V1	V2	V3	V4	V5	V6
GSW	75.781	75.281	60.726	98.163	100.268	68.584
DGSW	75.393	76.594	61.692	97.231	98.883	68.088

Results are available in table 6. We can see that sparse vectors are colored much earlier in the first three variants, and even earlier in the third variant, which might be explained also by their smaller norm relative to the non-sparse vectors, similarly to what was observed in section 6.4. The last three variant show use that the earliness effect seems to be mostly linked to the fact that the coordinates of the sparse vectors were not shuffled in the first three variants, as the sparse vectors are colored very close to the average of 100 in the fourth and fifth variant.

The sixth variant has results that are higher than the third but still lower than 100, which is probably because of the difference in norm between the two groups.

It could also be interesting to investigate how balanced each vector group is and how noise affects these results, as this could help us discover a hidden group in a larger set. Another interesting thing would be to see how varying the relative size of the two groups would modify the phenomenon.

6.9 How often is the pivot vector colored ?

It would be interesting to know how often the pivot is colored, and whether that depends on the pivot choice rule, or just on the vector set. To do so we try to run the GSW with a couple different pivot choice rules and on various vector sets.

6.9.1 Experiment

We use three pivot choice rules: the **random** (R) rule, that is the classic one where the pivot is chosen uniformly at random when a new pivot is needed, the **maximum norm** one (MN) where the pivot is always chosen as the vector that has the greatest norm among all vectors that are alive, and the **norm-proportional** (NP) rule that makes it so each vector that is alive has a probability of being picked as the next pivot that is proportional to its norm.

We use three different sets of 100 vectors in dimension $d \in \{2, 10, 100\}$, which correspond to those used in section 6.4, that is the all equal norm set (AEN), the two group set (2G) where half the vectors have size 1 and the other half size $n = 100$, and the incrementally growing norm set (IGN) where the vectors have respectively norms 1,2,...,100. Outside of their norms which are modified, all vectors from these sets are initially sampled from the ball of radius 1 in dimension d. We observe the proportion of time steps during which the pivot is colored among all time steps and try to see whether the pivot rule seems to make that proportion vary or whether it only depends on the vector set.

6.9.2 Results

Results are available in table 7. Notice that when using the maximum norm mode with DGSW we lose all randomness, which is probably undesirable but these were still included for completeness.

The DGSW proportions all seem a little higher in the dimension 2 and 10 cases, while in dimension 100 there seems to be no significant differences. So coloring the closest point on the update direction seems to push toward coloring the pivot most of the time in smaller dimensions, but when the dimension grows this effect appears to vanish.

We would expect the 2G set to not have much difference across the maximum norm and norm-proportional pivot choice rules as the vectors only have 2 potential different norms that are very different in size, and that is indeed the case.

Table 7: Results of our experiments from section 6.9 on how often is the pivot colored as a function of the pivot choice rule and the input vector set. Results are averaged over 100 runs of the GSW each and are proportions between 0 and 1.

Dimension	**2**								
Set	AEN			2G			IGN		
Rule	R	MN	NP	R	MN	NP	R	MN	NP
GSW	0.958	0.949	0.939	0.920	0.474	0.505	0.946	0.912	0.892
DGSW	0.976	0.969	0.952	0.944	0.504	0.509	0.971	0.940	0.914
Dimension	**10**								
Set	AEN			2G			IGN		
Rule	R	MN	NP	R	MN	NP	R	MN	NP
GSW	0.822	0.798	0.822	0.657	0.357	0.346	0.761	0.578	0.628
DGSW	0.870	0.841	0.862	0.705	0.383	0.374	0.811	0.638	0.673
Dimension	**100**								
Set	AEN			2G			IGN		
Rule	R	MN	NP	R	MN	NP	R	MN	NP
GSW	0.532	0.527	0.537	0.434	0.438	0.420	0.463	0.362	0.421
DGSW	0.521	0.538	0.525	0.424	0.445	0.431	0.461	0.387	0.441

We can also see that the pivot rule does not seem to change the pivot coloring rate significantly in the AEN case, which makes sense given that the pivot rules depend on the norm and all the norms are equal.

We can see on the 2G and IGN vector sets that the proportion of colored pivots is lower when using non-uniformly random pivot rules which favor longer vectors, which is consistent with the observation that longer vectors get colored later from 6.4. This is absent on the AEN vector set as the norms are all equal.

Finally, the main observation seems to be that the pivot is colored more often when the dimension is small, and I currently do not have any hypothesis on why that is the case.

6.10 What if we choose the pivot as a function of the fractional coloring ?

One could think of seeing the GSW as a rounding algorithm of some sort, and then, as the pivot is often colored as seen in section 6.9, we could want that the vectors closest to being colored are pivot as they're closer to being rounded. That is what we try in this section.

We introduce two new pivot choice rules: the **maximum absolute coloring** (MAC) rule that chooses as pivot the alive vector that is closest to -1 or 1 and separates ties uniformly randomly, and the **coloring proportional** (CP) rule that chooses the pivot according to a distribution such that the probability that an alive vector is chosen as the pivot is proportional to the absolute value of its current fractional coloring value, and choses uniformly randomly if the fractional coloring value of every alive vector is 0.

6.10.1 Experiment 1

We sample 100 vectors of dimension 2 and run both the classical GSW with random pivot and the classical GSW with **maximum absolute coloring** (MAC), as well as the DGSW with the MAC pivot rule which makes it a deterministic algorithm given the first pivot (assuming we don't come across a situation with equality in absolute value after the very first step). We do 100 runs per variant (with a different starting pivot for each run) and plot the results to see whether there is some pattern and how it looks like, similarly to figure 1.

6.10.2 Results 1

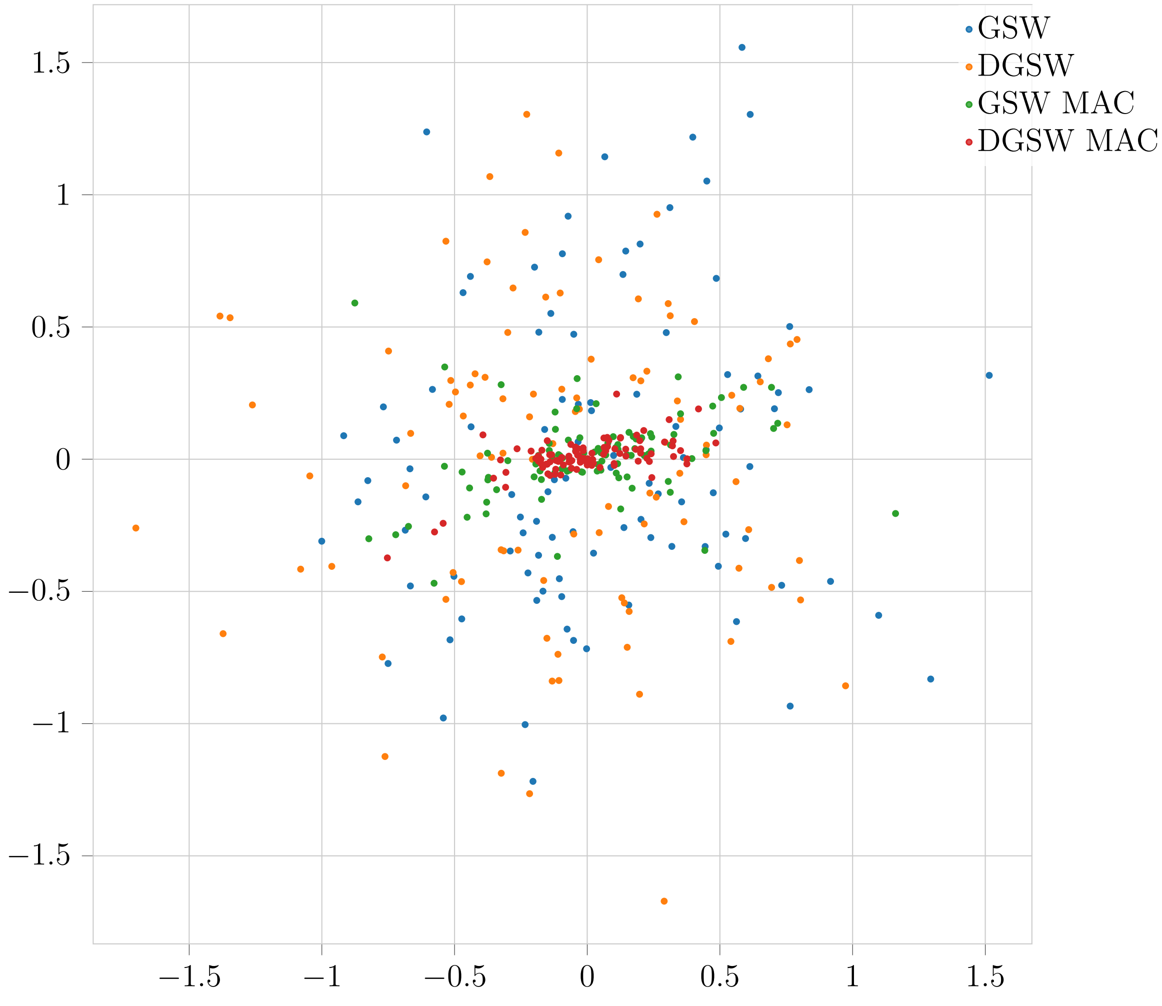


Figure 9: Results of experiment 1 from section 6.10.1. Plot of vectors of imbalances for the GSW and DGSW with random pivot rule and MAC pivot rule. 100 points for each variant on the graph.

Results are visible in figure 9. We can see that as previously, the vector of imbalances from DGSW run are slightly closer to **0** than those from GSW runs. Additionally, we can see that the MAC pivot rule seems to give us shorter vector of imbalances as well, with MAC DGSW runs being visibly much closer to $\mathbf{0}$. While this effect could be expected as we color the vectors that allow us to pick smaller deltas and thus move less away from $\mathbf{0}$, it is actually very visible here and we will investigate it further in the next experiment.

We also see that the vectors of imbalances from the MAC GSW and especially the MAC DGSW seem visibly skewed along the x-axis. While this is not as present if we reproduce this plot with a different input vector sets, we can sometimes see that MAC (D)GSW vectors of imbalances are somewhat more centered along some direction. It would be interesting to see how this direction is related to the input vectors, maybe it is one of the eigenvectors of $\sum_{i\in[n]} \mathbf{v}_i\mathbf{v}_i^T$?

6.10.3 Experiment 2

We run a similar experiment as in section 6.1.1, except the algorithms we use are the classical GSW, the GSW with MAC and the DGSW with MAC.

6.10.4 Results 2

We can see the results in Figure 10. The GSW and the GSW with MAC are very close, but the DGSW with MAC produces significantly smaller vector of imbalances. While for large dimensions, it doesn't make an enormous difference, for smaller dimensions and larger numbers of vectors, this seems to be divide by more than 2 in comparison to the classical GSW which is a sizable improvement.

6.10.5 Experiment 3

We reproduce the experiment in section 6.9 but with the MAC and CP rules.

We would expect these methods to have higher pivot-coloring rates as the previous ones, as they're literally choosing elements that are more likely than average to be colored in the next step as their absolute value is closer to 1.

6.10.6 Results 3

Results are visible in table 8. We can see that as predicted, the pivot coloring rates are higher in this experiment than in section 6.9, but they actually aren't for the 2 groups (2G) vector set using the random pivot. One potential explanation is that the longer vectors are often close to being colored but still harder to color than the small ones despite that, where the random pivot rule actually chooses a smaller vector 50% of the time.

For dimension 100 we see that the maximum absolute coloring rule actually yields a much higher pivot coloring rate than its proportional variant, which could be an indication that the fractional coloring become less sure, that is closer to 0 on average, the higher the dimension goes.

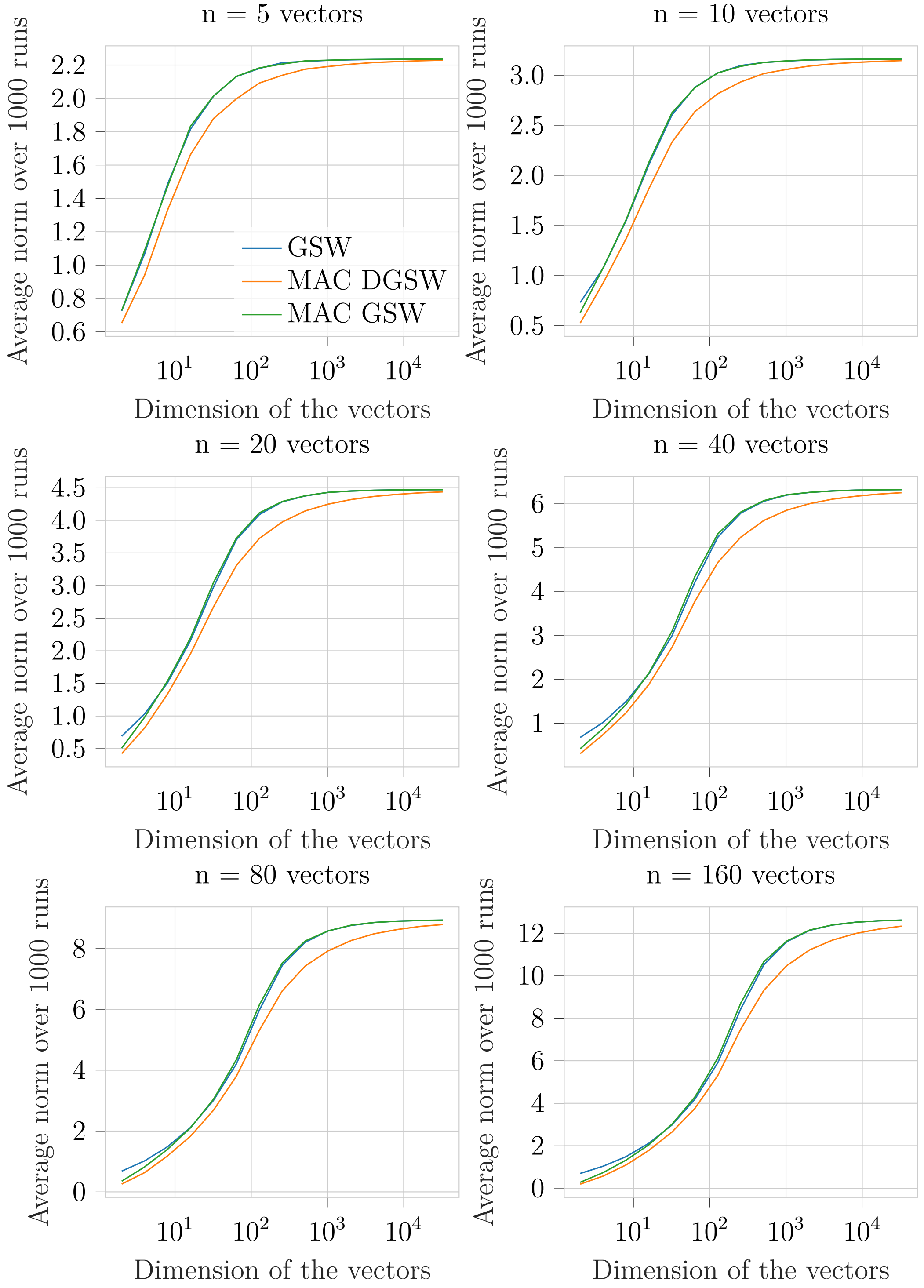


Figure 10: Results of experiment 2 from section 6.10.3. Average norm of 1000 vector of imbalances for $n = 5, 10, 20, 40, 80, 160$ input vectors and in dimensions $d = 2$ to $d = 2^{15}$, using GSW with both the random pivot rule and the maximal coloring rule, and DGSW with the maximal coloring rule.

Table 8: Results of our experiments from section 6.10.5 on how often is the pivot colored as a function of the pivot choice rule and the input vector set, now with coloring-dependent pivot choice rules. Results are averaged over 100 runs of the GSW each and are proportions between 0 and 1.

Dimension			**2**			
Set	AEN		2G		IGN	
Rule	MAC	CP	MAC	CP	MAC	CP
GSW	0.950	0.945	0.590	0.550	0.939	0.900
DGSW	0.965	0.968	0.600	0.566	0.954	0.927
Dimension			**10**			
Set	AEN		2G		IGN	
Rule	MAC	CP	MAC	CP	MAC	CP
GSW	0.889	0.840	0.406	0.386	0.758	0.711
DGSW	0.930	0.887	0.444	0.419	0.798	0.769
Dimension			**100**			
Set	AEN		2G		IGN	
Rule	MAC	CP	MAC	CP	MAC	CP
GSW	0.834	0.595	0.697	0.532	0.800	0.535
DGSW	0.884	0.623	0.722	0.542	0.825	0.544

We can also notice that the DGSW leads to a higher coloring rate, which makes sense as we choose a pivot that's close to being colored and the DGSW will thus finish coloring it more often that the classical GSW.

6.11 Can we choose the pivot from the potential update directions or $\mathbf{v}^{\perp}(t)$?

Section 4 lets us compute the update direction and $\mathbf{v}^{\perp}(t)$ for every potential pivot, so we could exploit that and choose among potential update directions, potentially only when we need a new pivot but not necessarily.

One way to do this is to compute $\mathbf{u}_t$ for every potential pivot, compute the resulting two potential moves, choose a δ for each pivot through the usual martingale setting, then among the chosen $\delta_t \mathbf{u}_t$ moves, use the shortest one. We call that pivot rule the random minimum move rule (**RMM**). Its DGSW equivalent just chooses the shortest move with access to every δ instead of by having chosen one δ for each potential pivot preemptively.

Another way to do so is to choose a pivot with a probability proportional to the inverse of the norm of the expected move, that is, $(2\frac{|\delta_t^+ \delta_t^-|}{|\delta_t^+|+|\delta_t^-|}\|\mathbf{u}_t\|)^{-1}$. We call this pivot rule the inverse move proportional (**IMP**).

Both of these rules could be applied to choose a new pivot at each step, regardless of whether or not the current pivot has been colored, or only when a new pivot is needed. Note that it is necessary for the current subgaussianity proofs that the pivot stays the same for as long as it isn't colored, but in this case it could be interesting to change at each time step. When we always choose a new pivot we will add **ANP** for "always new pivot".

As returning all update directions in a computationally efficient way requires to have $n \leq d$ as that is necessary for the implementation described in section 4, we cannot test these rules on similar experiments as those previously done. Thus we test them in section 7.3.

7 Finding Structured Subgroups

We will try several experiments aiming to identify structured subgroups hidden in sets of random vectors. The idea is that the GSW should color members of the subgroup early as they should be easier to color as they should balance each other out, and we will look at when the GSW colors which vectors over a large number of runs of the GSW and with various sets of vector and configurations. The ideal case would be if the GSW could help us solve a problem such as the planted clique.

This section is inspired by results such as those in section 6.8.

One problem that we would like to solve is, given vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{n-k}$ in a ball of radius 1 in some dimension d and $\mathbf{v}_{n-k+1}, \ldots, \mathbf{v}_n$ vectors in the same ball such that all their coordinates are positive, we apply a single random rotation to $\mathbf{v}_{n-k+1}, \ldots, \mathbf{v}_n$. The random rotation is a random orthonormal matrix obtained via the Gram-Schmidt process from vectors whose coordinates each are uniformly picked in $[0, 1]$. We would then like to use the GSW to find $\mathbf{v}_{n-k+1}, \ldots, \mathbf{v}_n$, by hoping that they would be colored early in a GSW run on average to balance each other as they are close, similarly to what was done in section 6.8. We call this problem the **hidden cluster**.

While one could just compute the norms difference between all pairs of vectors and probably find the cluster, one other way to do so would be to look for the vector that's colored the earliest on average after some number r of GSW runs, and return that vector and the $k-1$ vector that are the closest to it. There are lots of moving parameters here, including n, k, d and the specific GSW variant used. We will investigate when that approach works and when it doesn't. We call this technique the **early coloring**.

7.1 How many runs are needed to get a good idea of when a vector is colored on average ?

In order to investigate that, we created four instances of the hidden cluster with $n = 200, k \in \{5, 10\}, d = 200$. We then used our early coloring method five times on each instance with $r = 200$ and saw if the results differed over the five runs for each set. As the end results were the same for each time we ran the method independently of which run it was, we concluded that $r = 200$ was enough to get a good approximation of the average coloring times for this parametrization at least.

This does not mean that the average coloring time of each vector were very close for each time we ran the technique, but it means that the earliest vector was always the same. While this was for a specific instance of the problem and on a small sample size of different instances, each try took a long time and we weren't able to run more to increase our certainty. There probably exists a smart statistics proof for showing what number is a good number of runs, but that wasn't the focus here.

7.2 Does the early coloring technique work ?

We want to know if the technique makes sense and works, so we created 100 instances of the hidden cluster with $n = 200, k \in \{5, 10, 15\}, d = 200$, (so 100 for each k) and tried our technique with GSW, DGSW, MAC GSW and MAC DGSW, where MAC is defined in section 6.10.5. We then used our early coloring method with $r = 200$ and recorded how often the technique successfully identified the hidden cluster. For DGSW with MAC which is actualy fully deterministic given the first pivot assuming random vector input, we made sure to use each of the 200 vectors as starting pivot exactly once.

7.2.1 Results

Table 9: Results of our experiments from section 7.2 on the early coloring method to solve the hidden cluster problem. The results are the proportion of instances for which the early coloring method worked out of 100 runs for each of the 12 cases.

k	5		10		15	
Rule	R	MAC	R	MAC	R	MAC
GSW	0.23	0.29	0.39	0.46	0.71	0.63
DGSW	0.28	0.22	0.5	0.52	0.69	0.7

Results are available in table 9. We see that the bigger the hidden cluster, the easier it seems to be to find which is expected, among other things because the random chance of finding it is also higher. Nevertheless for each dimension the proportion of runs in which the method works is way higher than the proportion of points that are part of the hidden cluster, so the method seems effective. Using the MAC pivot rule seems to help a little bit, but that is hard to tell for certain and I doubt that the difference is statistically significant with our sample size. On the other hand, the DGSW variant successfully identifies the cluster very slightly more often than the original GSW variant, but this could very well again be because of the small sample size.

7.3 Can we improve early coloring by choosing the pivot differently ?

We used the two pivots rule described in section 6.11 and the early coloring technique to see whether they improved the results. For the random minimum move rule, one can notice that as in the first step $\delta^+ = -\delta^-$, the first vector picked will always be the same, which let us test very quickly as only a single run was necessary instead of the 200 used previously.

Additionally, one can see that there are no difference between the GSW and the DGSW for the early coloring method with this pivot rule, again because the first vector picked is always the same. We do 1000 runs for each case.

For the inverse move proportional rule, we cannot use $r = 1$ and the tries take much longer. We are only able to do 50 runs for each case.

We reproduce the experiment from section 7.2, except with the RMM and IMP pivot rules, and for the former, $r = 1$ and 1000 runs. Additionally, we try both with and without changing the pivot at every step.

7.3.1 Results

Table 10: Results of our experiments from section 7.3 on the early coloring method to solve the hidden cluster problem using the RMM and IMP pivot choice rule. The results are the proportion of instances for which the early coloring method worked out of 1000 runs for each RMM case and 50 runs for each IMP case.

k	5		10		15	
Rule	RMM	IMP	RMM	IMP	RMM	IMP
GSW	0.279	0.28	0.55	0.54	0.687	0.68
GSW ANP	0.26	0.24	0.537	0.54	0.691	0.66

Results are available in table 10. Changing the pivot at each iteration doesn't seem to change much, perhaps because in RMM at least, the same pivot could be chosen again as it could still yield the shortest $\mathbf{v}^{\perp}(t)$. The proportions are slightly higher in dimension 10 compared to those observed in section 7.2, but that doesn't seem to be the case for the other k's who produced similar proportions, so this might just be because of the sample size. Overall this doesn't appear to be an improvement, especially because there are slightly more computations in when using that variant.

8 Next Interesting Directions

It would be interesting to modify the algorithm to be able to divide vectors into k balanced groups rather than only two. While this can be achieved using multiple runs of the algorithm and starting points different from $\mathbf{0}$, it loses some of the important properties of the original algorithm. One idea would be to have the coloring live in $(S_{m-1})^n$ instead of $[-1, 1]^n$, where S_{m-1} is the $(m-1)$-dimensional regular simplex centered in $\mathbf{0}$ and m is the number of groups we want to separate our vectors into, but several aspects aren't easy to generalize with this framework.

Another interesting direction would be to explore adding coordinates to the input vectors to convey information, and more generally modifying the vector set to influence the outcome. For example, it is possible to make sure that a pair of vectors are together or not together in a GSW assignment by replacing both vectors in the set by a single vector that is either their sum or difference. There are probably other similar possibilities that could be interesting to explore in that direction.

Yet another interesting area where improvements could be made is using quadratic programming when computing the update direction, similarly to what we do at the end of section 6.4. While it might increase the asymptotic running time of the algorithm, doing it only when $\mathbf{v}^{\perp}(t) = \sum_{i=1}^{n} \mathbf{u}_t(i)\mathbf{v}_i = \mathbf{0}$ means that most of the results about the classical GSW still hold, and this is just a way to force some good properties on our solution by restricting partially the infinity of equally valid solution of the original least-squares problem.

There probably exists a different method to solve the issue described in section 6.6 that allows us to keep more guarantees and increase the norm of the resulting vectors of imbalances less, potentially by using the tools mentioned in the previous two paragraphs.

There could be a way to adapt what was done in section 7 to problems such as the planted clique, using a smart normalization and a specialized version of the GSW. I did not succeed in consistently solving instances of that problem with a method similar to the one presented for the hidden cluster, but using the GSW to identify structured subsets seems promising. One could also look at coloring step correlations between pairs of element, as one could hope that these measures could be high between two elements of a structured subgroup.

Being able to explicitly compute the distribution of assignments or of vectors of imbalances produced via the GSW without enumerating all potential possibilities could greatly help understand it.

Another interesting experiment could be to actually see how the discrepancy, that is the maximum coordinate in the vector of imbalances rather than the norm of that vector, is actually minimized in some of the experiments we performed. The potential difference in results compared to the experiments already performed would help us understand the behavior of each variant with more details.

It would be interesting to see what the applications of the generalization described in section 5 could be, and how the results about subgaussianity can generalize to this variant depending on the basis $\mathcal{B}$.

Finally, applying this algorithm in various field where it could make an impact and seeing how it improves them would be very interesting. For example, using this algorithm to provide balanced training and testing sets when doing machine learning with small data sets could be useful. Additionally, maybe forcing deep learning batches to be balanced or not at different moments of the training procedure could improve training time and help get out of local minimas or do some fine tuning, but given the size of the datasets generally used in deep learning, this might be too computationally expensive to run if done on the whole dataset.

References

[1] Wojciech Banaszczyk. Balancing vectors and gaussian measures of n-dimensional convex bodies. *Random Structures & Algorithms*, 12(4):351–360, 1998.

[2] Nikhil Bansal, Daniel Dadush, Shashwat Garg, and Shachar Lovett. The gram-schmidt walk: A cure for the banaszczyk blues. *CoRR*, abs/1708.01079, 2017.

[3] Gaëtan Bossy. Gswalk. `https://github.com/gbossy/GSWalk`, 2022.

[4] Daniel Dadush, Shashwat Garg, Shachar Lovett, and Aleksandar Nikolov. Towards a constructive version of banaszczyk's vector balancing theorem. 12 2016.

[5] Peter Gritzmann and Victor Klee. On the 0–1-maximization of positive definite quadratic forms. In *Operations Research Proceedings 1988*, pages 222–227. Springer, 1989.

[6] Christopher Harshaw, Fredrik Sävje, Daniel Spielman, and Peng Zhang. Balancing covariates in randomized experiments with the gram–schmidt walk design. *arXiv preprint arXiv:1911.03071*, 2019.

[7] J v Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928.

[8] Michel Talagrand. *The generic chaining: upper and lower bounds of stochastic processes*. Springer Science & Business Media, 2005.