Studying the Gram-Schmidt Walk

Gaëtan Bossy, under the supervision of Adam Marcus
Chair of Combinatorial Analysis

May 25, 2022

Abstract

TODO

1 Introduction

2 The Algorithm

Input : $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^d$ with ℓ_2 norm at most 1, an initial coloring $\mathbf{z}_0 \in [-1,1]^n$
Output: a coloring $\mathbf{z}_n \in \{-1,1\}^n$

$A_1 = \{i \in [n] : |\mathbf{z}_0(i)| < 1\}$, $n(1) = \max\{i \in A_1\}$ and $t = 1$.
while $A_t \neq \emptyset$ **do**

  Compute $\mathbf{u}_t \in \mathbb{R}^n$ such that $\begin{cases} \mathbf{u}_t(n(t)) = 1 \\ \mathbf{u}_t(i) = 0 \text{ if } i \notin A_t \\ \mathbf{v}^\perp(t) = \mathbf{v}_{n(t)} + \sum_{i \in A_t \setminus \{n(t)\}} \mathbf{u}_t(i)\mathbf{v}_i \end{cases}$

  $\Delta = \{\delta : \mathbf{z}_{t-1} + \delta\mathbf{u}_t \in [-1,1]^n\}$, let $\begin{cases} \delta_t^+ = \max \Delta \\ \delta_t^- = \min \Delta \end{cases}$ then $\delta_t = \begin{cases} \delta_t^+ & \text{w.p. } \frac{-\delta_t^-}{(\delta_t^+ - \delta_t^-)} \\ \delta_t^- & \text{w.p. } \frac{\delta_t^+}{(\delta_t^+ - \delta_t^-)} \end{cases}$

  $\mathbf{z}_t = \mathbf{z}_{t-1} + \delta_t \mathbf{u}_t$, $t \leftarrow t+1$, $A_t = \{i \in [n] : |\mathbf{z}_{t-1}(i)| < 1\}$, $n(t) = \max\{i \in A_t\}$.

Output $\mathbf{z}_t \in \{-1,1\}^n$.

3 Generalizing to any hyperparallelepiped

4 Generalizing to more than two groups

Discrepancy minimization is generally set in a 2-group paradigm, but it could be interesting to generalize the GSW to separate into more than 2 groups. For example, if one wanted to separate

n vectors in 3 groups, the GSW could first be used to separate into groups G_+, G_- such that $\sum_{v \in G_+} v - 2 \cdot \sum_{v \in G_-} v \approx \mathbf{0}$, by starting at $x_0 = (1/3, 1/3, \ldots, 1/3) \in \mathbb{R}^n$. Then the group G_+ could be again inputed into the GSW to separate it into G_{++} and G_{+-}, and we would expect $G_-, G_{++} and G_{+-}$ to be roughly balanced mutually but also all together.

But how do we know if a 3 group assignment G_i for $i \in \{0, 1, 2\}$ is balanced ? For 3, one could use the complex roots of 1, $\omega_0 = 1, \omega_1$, and ω_2, and check that

$$\sum_{i \in \{0,1,2\}} \omega_i \sum_{v \in G_i} v \approx \mathbf{0}.$$

One issue it that this seems like a bandaid method, and it seems like it would yield before grou assignment to have an algorithm that separates in m groups from the get go.

Another issue is that this doesn't generalize to a higher number of groups, which is why seeing we need another perspective. One idea is to link each group with a vertex of the $(m-1)$-dimensional regular simplex centered in $\mathbf{0}$ where m is the number of groups we want to separate our vectors into. So we would want to assign each group to a vertex and verify that the sum of our vectors is close to $\mathbf{0}$ in each of the $m-1$ dimensions. This would mean that our coloring would live in S^n_{m-1}, each vector moving in its personal copy of the simplex until it gets fixed to one of the vertices.

We would have to adapt the choice of the update direction and the choice of δ which could maybe also be multi-dimensional. One big issue then is that choosing an update direction is far from obvious. Should we force the multidimensional vector of update of the pivot to be of norm 1 ? If so how to choose it ? Additionally, assuming we have an update direction, what should one do when the border of the simplex is hit but not the vertex ? Should we now force the coloring to stay fixed to that border and now move in that border ? Should we choose the update direction and δ in a way that borders are never hit outside of vertices ?

All these questions are tough to answer, and generalizing the GSW to separate in m groups would require understanding them deeply. Sadly, I did not succeed in finding such a generalization.

5 Experiments and Properties

By balance of the assignment, we mean the difference between the number of 1s and -1s. An assignment is perfectly balanced if its sum is 0.

5.1 How good is the GSW at minimizing output discrepancy in practice ?

We're interested in seeing how well does the GSW actually perform in minimizing the norm. We will compare it to the naive walk defined in **??**, the deterministic GSW **??** and the actual best computed via bruteforcing.

Figure 1

5.1.1 Experiment

We compare the output discrepancy of GSW, DGSW, the naive walk and for some small n also the best assignment found via brute forcing on all possibilities. We do this for$n = 5, 10, 15, 20$ and 40, and with $d = 2^i$ for $i \in \{1, \ldots, 15\}$, where we sample n vectors from the d-dimensional ball of radius 1.

5.1.2 Results

Our results are visible in Figure **??**. We can see that GSW actually gives the worst results in terms of discrepancy minimization, 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 discrepancy-minimizing algorithm as these results would probably be different if we modified the distribution of input vectors.

5.2 Does translation affect the balance of the assignment ?

If your initial group of vector is centered around 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.

5.2.1 Experiment

We sample 200 input vectors from the ball 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.

5.2.2 Results

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

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

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. 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.

5.3 A parameter to balance assignments

Inspired by section 5.2, 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 [5], except here we implement a tradeoff between assignment balance and output balance.

Given input vectors $v_1, \dots, v_n \in \mathbb{R}^d$ and a parameter $\mu \in [0, 1]$, we define $w_1, \dots, w_n \in \mathbb{R}^{d+1}$ as

$$w_i = \begin{pmatrix} \sqrt{1-\mu} v_i \\ \sqrt{\mu} \end{pmatrix}.$$

This way the w_i's have similar norm to the v_i's, but maybe a different normalization depending on the norm of the v_i's could be better. We then run the GSW on them and use the output assignment on the original vectors. Choosing $\mu = 0$ is equivalent to doing the classical GSW algorithm, while using $\mu = 1$ is equal to forcing exact 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 μs.

5.3.1 Experiment

We use our just explained construction to compute an assignment on the w_i's using GSW or DGSW, then see how this assignment performs on the original v_i's in terms of balance of the assignment and discrepancy. We also program the fixed size GSW described in [5] and compare it. We use $\mu \in \{0, 0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99, 0.999, 1\}$, and n vectors sampled from the ball of radius 1 in dimension n for $= 100$. The results presented are averaged over 1000 runs.

5.3.2 Results

We can see that we can massively increase assignment balance without making the output norm 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 all properties of the classical GSW, as opposed to the balanced variant described in [5], for which we don't know if some of the original properties still hold. The high probability comes from the subgaussianity bound.

μ	0	0.001	0.01	0.1	0.25	0.5	0.75	0.9	0.99	0.999	1	FSD [5]
AB	8.086	6.248	4.164	1.898	1.108	0.576	0.266	0.106	0.01	0.002	0	0
AD	5.259	5.275	5.266	5.311	5.341	5.321	5.336	5.331	5.334	5.337	9.917	5.317
ABD	20.584	18.698	11.908	3.886	2.044	0.994	0.328	0.116	0.018	0.002	0	0
ADD	4.863	4.837	4.775	4.821	4.885	4.938	4.991	5.004	5.008	5.002	9.851	5.01

Table 2: Results of our experiment on the balance of assignments with our new balance-discrepancy tradeoff design. The balance numbers shown (lines starting with AB for average balance) are the average absolute value of the sums of output vectors, and the discrepancy numbers shown (lines starting with AD for average discrepancy) are the norms of the sum of $v_i \cdot x_i$, x being the assignment produced by GSW for the w_i's except for the last column in which we used the fixed size GSW design from [5]. The D at the end of the last two lines indicates that in this case, the deterministic GSW algorithms was used.
A smaller balance number indicates that the assignment has a more balanced assignment, that is the number of 1s and -1s are closer. A smaller discrepancy number 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 [5] references the fixed size design from [5].

5.4 What else can we control by adding coordinates ?

We could think of a design where we want several subgroups, potentially of only 2 vectors per subgroup, to be balanced. We could then add a coordinate for each subgroup and put the same number on that coordinate for each member of the subgroup, while putting 0 for every vector that isn't in the subgroup. This would push towards balancing within any subgroup we want, and could be done via some parametering similar to what was done in section 5.3.

Similarly, if we want 2 vectors to be in the same group, we could add a dimension and assign some number x and $-x$ to these vectors in that new dimension while giving 0 to every other vector in that dimension. Adding dimension could be used to translate knowledge about the vector set into usable information for the algorithm.

5.5 Does norm affect the balance of the assignment ?

I would expect the norms not to affect the balance of the assignment, as multiplying every input vector by the same vector should mean the algorithm runs similarly.

5.6 Does norm affect when a vector is colored ?

The expected heuristic would have been that bigger vectors are colored earlier and the algorithm then colors the smaller ones to minimize discrepancy as that is what I'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 $v_1, \dots, v_200$ of increasing norm and observe how close the coloring order is to $\mathcal{R} = \{200, 199, \dots, 1\}$.

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.

5.6.1 First Experiment

We sampled 200 vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{200}$ in the ball of radius 1 of dimension 200. For each v_i, we replaced it by $i \cdot \mathbf{v}_i/\|\mathbf{v}_i\|$ so that $\|\mathbf{v}_i\| = i$ for each vector.

5.6.2 Second Experiment

We sampled 200 vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{200}$ in the ball of radius 1 of dimension 200. For each 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 $X_i = 1$ if $i < n/2$ and 200 otherwise.

5.6.3 Results

We also ran the same experiments with 200 vectors of constant norm as a comparison. The results are summarized in Figure 3.

	Exp 1	Exp 2	Control
GSW	18664.4	19997.32	13607.06
DGSW	18641.6	19996.16	13431.68

Table 3: Result 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.

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 I expected, this behavior actually makes sense, because if we multiply $\mathbf{v}_i$ by $\mu\mathbf{v}_i$, it's corresponding coordinate in $\mathbf{u}$ is going to be divided by μ, thus it will move less towards the border of the hypercube 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 perform better on an input such as $\{v, v, v, 3v\}$ for $v \in \mathbb{R}^d$ for an arbbitrary $d \in \mathbb{N}$. One way would be to choose the pivot through some smart condition or to choose another feasible $\mathbf{u}$ than the default least square solution with minimal norm, again through a smart criterion.

One such idea is to force the pivot to be the largest norm vector, hen, when $v_\perp = \mathbf{0}$, to select u_t through lasso with a very small alpha ($\alpha = 10^{-32}$ for example) in order to ensure that the smallest number possible 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 loses a lot of randomness in the process, so there probably exist some different additional constraint to add when computing u_t in colinear cases that could work even better.

Another variant that works but only in this trivial example and not in slightly more complicated examples with for examples 2 groups of vectors is to just force the largest alive vector to be the pivot at every step. This is just a product of the solution of the least squares we're choosing as there are infinitely many that wouldn't work.

A third variant that might help is to do quadratic programming instead of simple least squares when $v_\perp = \mathbf{0}$. This way, we can force the quadratic program to minimize both $\|Bu\|$ but also $\|u\|$. This could be achieved by adding a line of 1's to the matrix B containing all input vectors as column, and adding as conditions that the chosen u must have a 1 in the pivot coordinate and 0s in already colored coordinates. Then if the u chosen through this quadratic program doesn't yield $Bu = \mathbf{0}$, we discard it and compute u through the usual method. This technique coupled with choosing the longest alive vector as pivot actually solves trivial adversarial cases, and doesn't modify too much the algorithm.

5.7 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.

5.7.1 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 the become pivot.

5.7.2 Results

	ALOSAP G1	ALOSAP G2
GSW	6.101	2.281
DGSW	5.989	2.277

Table 4: Result of our experiments on whether long and short vectors stay for longer as the pivot. ALOSAP stands for average length of stay at pivot and is measured as the average number of steps over 100 runs.

Results are visible in Table 6. We sede that shorter vectors tend to stay pivot for much longer than their longer counterparts. This could be explained by the fact that coordinates of the update direction for all long vectors are very small so longer vectors rarely get colored by a small vector pivot, but shorter vectors do get colored while the longer vectors are pivot.

5.8 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. This would remove the orthogonality of updates, but in practice it didn't seem to change significantly how far the sum of outputs were from $\mathbf{0}$. We can study how that affects the order of the coloring in experiments similar to those done in subsection 5.6.

5.8.1 Experiments

We sample vectors similarly to the experiments performed in subsection 5.6 and measure similarly how close the coloring order is to the order $\mathcal{R} = \{200, 199, \dots, 1\}$.

5.9 Results

We also perform the same experiments with a group of constant norm vectors

	E1 with D	E1 with D^2	E2 with D	E2 with D^2	Control with D	Control with D^2
GSW	13246.72	6078.28	13246.74	6697.1	13444.94	13208.1
DGSW	6808.96	13100.02	13597.86	6830.82	13303.18	13344.14

Table 5: Result of our experiments on trying to fix the later coloring of bigger norm vectors. The numbers shown are the Δ_o as defined in equation 1.

We can see that our modifications indeed remove the late coloring. Multiplying once makes it so the vectors are colored approximately randomly and the multiplying twice makes it so the bigger vectors are colored earlier, as intended. There are very likely other ways of getting a similar effect, potentially by adding coordinates smartly.

5.10 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 A be the matrix containing our input vectors as columns. Using a singular value decomposition, we have that $A = U\Sigma V^T$ where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthonormal and $\Sigma \in \mathbb{R}^{m\times n}$ is all zeros except for the diagonal elements which are positive singular values. Using this decomposition, we can see that $A^+ = V\Sigma^+U^T$ where Σ^+ is Σ^T except nonzero entries σ_i are replaced by their inverse $1/\sigma_i$. But if one replaced Σ^+ by Σ, then the matrix multiplying the pivot vector would be A^T instead of A^+. This suggestion from Pr. Marcus turned out not to work if we want to balance the vectors, but it actually does the opposite which is very interesting.

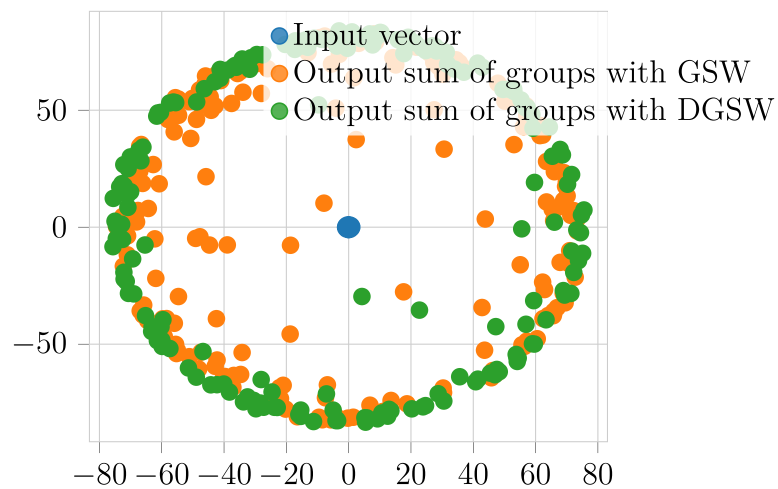

Figure 2: Output sums using the modified GSW and DGSW where the update direction is computed by multiplying the pivot vector by A^T.

5.10.1 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 $u_t(\mathcal{A}_t \setminus \{p(t)\}) = B_t v_{p(t)}$ and $u_t(p(t)) = 1, u_t(i) = 0 \forall i \notin \mathcal{A}_t$. Everything else is kept similar.

5.10.2 Experiment 2

We run as similar experiment as in 5.1.1 except we look for the discrepancy maximizing assignment via bruteforcing and look at the naive walk trying to maximize the output norm.

5.10.3 Results

The outputs are shown in the figure 2. We can see that this modification seems like it now minimizes output balance instead of maximizing it, which was surprising to me at least. This seems like it could be useful to sample from unbalanced group assignments, or to find a subset to remove to maximize something. This might be equivalent to an already known algorithm, but if not I think there are probably interesting applications of this.

5.11 Are vectors with smaller dimensionality colored at the same moment as vectors with more dimensions ?

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. We will investigate how early they're colored on average.

5.11.1 Experiment

We sample 100 vectors of dimension 100 from the ball 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 the sparse vectors are shuffled so that the 0's aren's uniformly placed in the sparse group.

5.11.2 Results

	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

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

We can see that sparse vectors are colored much earlier in the first 3 variants, and even earlier in the third variant, which might be explained also by their smaller norm relative to the non-sparse vectors. The last three variant show use that the earliness effect seems to be nearly completely linked to the fact that the sparse vector were not shuffled in the first three variants, as the sparse vectors are colored very close to the average of 100.

It could also be interesting to investigate how balanced each vector group is and how noise affect this result, 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 affects the phenomenon.

5.12 How often is the pivot vector colored ?

5.13 Can we force the algorithm to color the pivot vector +?

References

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

[2] Dimitris Bertsimas, Mac Johnson, and Nathan Kallus. The power of optimization over randomization in designing experiments involving small samples. *Operations Research*, 63(4):868–876, 2015.

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

[4] David A Freedman. On tail probabilities for martingales. *the Annals of Probability*, pages 100–118, 1975.

[5] 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.

[6] Abba M Krieger, David Azriel, and Adam Kapelner. Nearly random designs with greatly improved balance. *Biometrika*, 106(3):695–701, 2019.

[7] Kari Lock Morgan and Donald B Rubin. Rerandomization to improve covariate balance in experiments. *The Annals of Statistics*, 40(2):1263–1282, 2012.

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

[9] Joel Spencer. *Ten lectures on the probabilistic method*. SIAM, 1994.

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