Unbounded Frank-Wolfe Algorithms applied to the generalized lasso problem

Furkan Can Algan
2085308

Esteban Ortega Dominguez
2084248

Kaan Daibasoglu
2072039

July 2023

Contents

1 Introduction

In this project, the applications of a variant of one of the most popular optimization algorithm: Frank-Wolfe, on the Least Absolute Shrinkage and Selection Operator (LASSO) is going to be examined. LASSO is a popular method for lots of areas such as statistics, machine learning and signal processing. It performs variable selection and regularization to enhance accuracy and interpretability of the results. It is originally generated for linear regression models but it can be extended to other models such as generalized linear models, and, generalized estimating equations. The generalized lasso is encapsulated by the formulation:

$$\min_{x \in R^n} \frac{1}{2}||b - Ax||_2^2 + \lambda||Hx||_1 \tag{1}$$

Where $H \in R^{m \times n}$ is a specified penalty matrix. Depending on the application, we choose H so that the sparsity of Hx corresponds to some desired behaviour for x[2].

Suppose that our penalty matrix H is given by $D_n^{(r)}$, which denotes the r-order discrete derivative operator, that is $D_n^{(1)}$ is a matrix in $R^{(n-1)\times n}$ with (i,i)-entry being 1 and $(i,i+1)$ entry being -1 (for $i \in [n,1]$) and all other entries being 0. For $r \geq 2$, it is defined recursively by $D_n^{(r+1)} = D_{n-r}^{(1)} \cdot D_n^{(r)}$ [3].

The aim of this project is to apply the unbounded Frank Wolfe algorithm for the solution of the generalized lasso with a constraint matrix $D_n^{(r)}$ when r = 1, also known as *fussed lasso*, and r = 2, also known as *linear l1 trend filtering* i.e. when x is given by a picewise constant function and a picewise linear function respectively.

In the formulas, regarding x, coefficients or signal of interest, is the decision variable and belongs to set of real numbers R^n, model matrix $A \epsilon R^{(N \times n)}$, and the response, $b \epsilon R^N$. $\lambda \geq 0$ is a tuning parameter.

The algorithm calculates the gradient of the objective function at each iteration and identifies a feasible point that maximizes the dot product between the gradient and the feasible point obtained. However, this traditional Frank-Wolfe Algorithm is only applicable to bounded regions. In our research, we investigated the generalized version of the Frank-Wolfe Algorithm for unbounded feasible regions. The unbounded version of the algorithm has various applications, such as the L1 Trend Filtering Problem and Matrix Optimization Problems with Generalized Nuclear Norm Constraints, and it shows potential for other optimization problems with non-convex or non-smooth objective functions. [1]

At each iteration, it computes the gradient of the objective function at the current iterate and finds a feasible point which maximizes the transpose of the gradient times feasible point we found. But this traditional Frank-Wolfe Algorithm is useful only when region is bounded. In our study, we examined the generalized version of the Frank-Wolfe Algorithm to the unbounded feasible region. The unbounded version of the Frank Wolfe algorithm has several

applications as, for example, the L1 Trend Filtering Problem, and Matrix Optimization Problems with Generalized Nuclear Norm Constraints, and potential for other Optimization Problems with Non-Convex or Non-Smooth Objective Function.

In our report, we initially generated synthetic data to assess the performance of our models, similar to the approach taken by the article authors. Next, we examined two algorithms: the Unbounded Frank-Wolfe Algorithm with simple step-size and linesearch step-size rules, and the Unbounded Away-Step Frank-Wolfe Algorithm with the linesearch rule. These algorithms were employed to solve the LASSO problem with unbounded feasible regions. We also discussed the applications of both algorithms. Lastly, we validated our algorithms by conducting tests and presenting our results using synthetic data generated by us, following the methodology described in the article. To substantiate our findings, we utilized the Optimality Gap, $G_k - gap$, and $H_k^2 - gap$plots for r=1 and r=2, which represent the order of the difference matrix utilized in the L1 trend filtering problem.

2 Unbounded Frank-Wolfe algorithms

2.1 Unbounded Frank-Wolfe Algorithm

Due to the projection-free nature of the traditional Frank-Wolfe algorithm, there has been an increased demand in the optimization community for methods to solve large-scale optimization problems. In the article we focused on, an approach to handle unbounded constraint sets using a single loop algorithm. This algorithm addresses the unbounded constraint in each subproblem by employing a Frank-Wolfe step on the intersection and a Euclidean ball centered at the initial problem. By adopting this approach, the method solves the same number of linear oracles, which are comparatively easier to solve. Theoretically, this should effectively enhance the algorithm's performance, as its efficacy relies on the difficulty of solving the linear oracles.

The generalized lasso problem can be rewritten as:

$$\min_{x \in R^n} f(x) \quad \text{s.t.} \quad x \in T \oplus S \tag{2}$$

Where $f(x)$ if our objective function and the constraint can be formulated as:

$$T \oplus S = \{x + y \mid x \in T, y \in S\}$$

$$T = \ker(D_n^{(r)}), \quad S = \left\{ x \in R^n \;\middle|\; \left\| D_n^{(r)} x \right\|_1 \leq \delta, x \in \left(\ker(D_n^{(r)}) \right)^{\perp} \right\}$$

First, the gradient step is taken in the linear subspace T, then projected on linear subspace(which is generally computationally cheaper compared to a projection onto a bounded set) directed onto negative projected gradient with

the step size η. The solution of the projection is noted as s^k Eventually, the Frank-Wolfe step is taken on the bounded subspace, S, onto the direction between the chosen extreme point and the current solution projected on S which corresponds to

$$s_k - P_T^T x_k$$

. However, the step size in Frank-Wolfe is not as direct as the gradient step. Although the existence of various methods, the 2 different step sizes (simple and line search) will be compared in this study. Calculations for the step size is made considering the following formulas.
For simple step-size rule:

$$\alpha_k = 2k + 2$$

For linesearch step-size we have:

$$\alpha_k \in \arg\min_{\alpha \in [0,1]} f(y_k + \alpha(s_k - P_T^T x_k))$$

We thus arrive at the unbounded FW algorithm:

Algorithm 1 Unbounded Frank-Wolfe Algorithm

for $k = 0, 1, 2, ...$ **do**

1. Gradient descent step (along subspace T): $y_k = x_k \eta \mathcal{P}^\top \nabla f(x_k)$
2. Solve linear oracle: $s_k = \arg\min_{s \in S} \langle \nabla f(y_k), s \rangle$.
3. Frank–Wolfe step (along subset S): x_{k+1}

end for

This method does not converge linearly even for the strongly convex objective functions like gradient step does. Thus resulting in a zigzagging phenomenon between the two extreme points of the constraint set [1]. An elegant way to improve the convergence without a feasibility oracle requirement is Away-Step Frank-Wolfe Algorithm.

2.2 Unbounded Away-Step Frank-Wolfe Algorithm

When the optimal solution of the problem lies at the boundary of the convex set, the convergence rate of the iterates is slow i.e. it is sub linear . This is because the iterates of the classical FW algorithm zig-zag between the vertices of the face containing the optimal solution. To address the zig-zagging problem, Wolfe proposed the possibility to move away from the active vector, which results in a modified FW algorithm called the Away-Step Frank-Wolfe. This modification of the possible away step makes the algorithm linearly convergent for strongly convex functions. [1]

When expressing the constraint set of the generalized lasso problem and expressing the feasible regions as the direct sum of a linear subspace T and a bounded set S . The vertex set V is the set of the orthogonal projection onto x.

Algorithm 2 Unbounded Away-Step Frank-Wolfe Algorithm

Initialization with $x^0 \in T \oplus S$ such that $P_T^{\perp} x^0$ is a vertex of S, vertex set $V(x^0) = \{P_T^{\perp} x^0\}$, weight parameters $\lambda_v(x^0) = 1$ for $v \in V(x^0)$, and gradient descent step-size η

for k = 0,1,2,... **do**

1. Gradient descent (along subspace T): $y^k = x^k - \eta P_T \nabla f(x^k)$.

2. Linear oracle: $s^k := argmin_{s \in S} \langle \nabla f(y^k), s \rangle$, $v^k := argmax_{v \in V(x^k)} \langle \nabla f(y^k), v \rangle$.

3. Find the moving direction:

if $\langle \nabla f(y^k), s^k - P_T^{\perp} x^k \rangle < \langle \nabla f(y^k), P_T^{\perp} x^k - v^k \rangle$: (Frank-Wolfe step)

$d^k := s^k - P_T^{\perp} x^k$; $\alpha_{max} = 1$.

else (away-step)

$d^k := P_T^{\perp} x^k v^k$; $\alpha_{max} = \lambda v^k(x^k)/(1 \lambda v^k(x^k))$.

4. Frank–Wolfe/away-step (along subset S): $xk + 1 = yk + \alpha k dk$, where $\alpha k \in [0, \alpha_{max}]$ is chosen by line-search.

5. Update vertices $V(x^{k+1})$ and weights $\lambda_v(x^{k+1})$ for $v \in V(x^{k+1})$

end for

The unbounded Away-Step Frank-Wolfe Algorithm (uAFW) can be expressed as[3]:

The uAFW follows a similar structure where first a gradient descent step along subspace T is taken. The moving direction is found in step (3) by comparing the dot product of the gradient y^k with the either direction. The direction whose projection onto the gradient is the greatest is the chosen direction. Its important to note that when taking the away step the step size is limited by the weights.

In the uAFW, we have to update the vertex and weight sets by the following formulas:

$$V(x^k + 1) = \{s^k\} if \alpha k = 1 \quad and \quad V(x^{k+1}) = V(x^k) \cup \{s^k\} \quad if \quad \alpha_k \neq 1 \quad (3)$$

$$\lambda_{s^k}(x^{k+1}) = (1\alpha k)\lambda s^k(x^k) + \alpha_k, \quad \text{and} \quad \lambda_v(x^{k+1}) = (1 - \alpha_k)\lambda v(x^k) \quad \text{for} \quad v \in V(x^k) \backslash \{s^k\}, \quad (4)$$

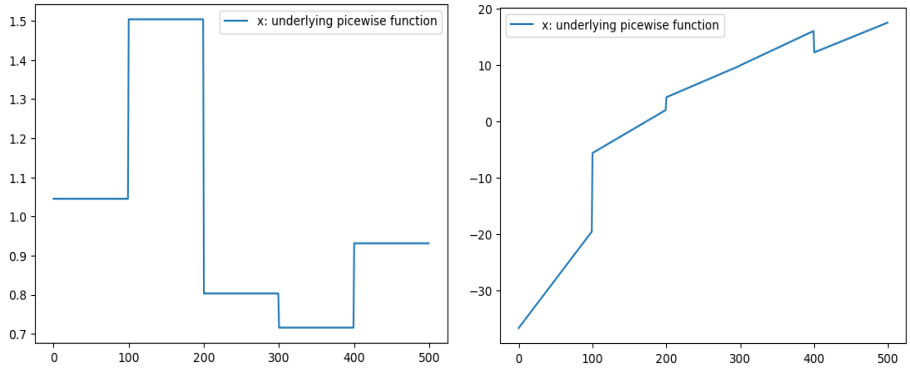
$$V(x^{k+1}) = V(x^k) \backslash \{v^k\} if \alpha_k = \alpha_{max} \quad and \quad V(x^k + 1) = V(x^k) \quad if \quad \alpha_k \neq \alpha_{max} \quad (5)$$

$$\lambda_{v^k}(x^{k+1}) = (1 + \alpha_k)\lambda_{v^k}(x^k) - \alpha_k, \quad and \quad \lambda_v(x^k + 1) = (1 + \alpha_k)\lambda_{v^k} \quad for \quad v \in V(x^k) \backslash \{v^k\} \quad (6)$$

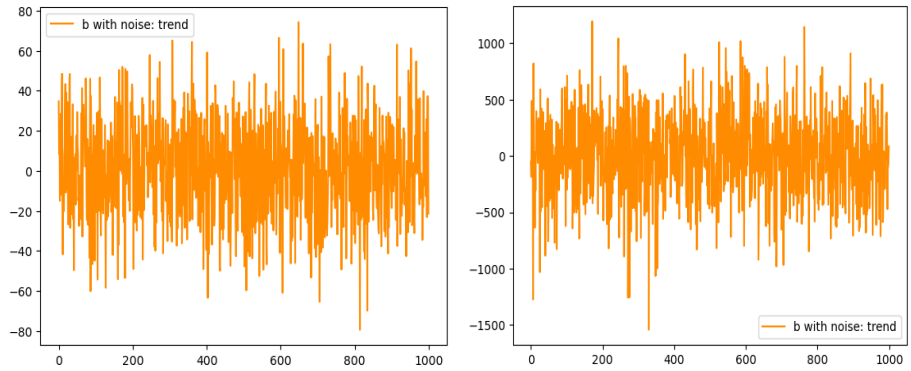
3 Methodology

3.1 Synthetical Data Generation

We generated synthetic data following the methodology described in Section 5.1 of the article. The authors used N = 1000, n = 500, and SNR = 1 (signal-to-noise ratio) for their synthetic data generation. We replicated their experiment and used the same values. Initially, we generated matrix A as a combination of the vectors $[a_1, a_2, ..., a_{(500)}]^T \in R^{(1000x500)}$, where each a_i represents a vector of regression coefficients. These vectors were independently and identically distributed with N(0,1) entries. Next, we introduced the noise term denoted as ϵ_i for $i \in [1000]$. The noise followed a normal distribution with variance σ^2. The determination of σ^2 will be discussed later. Additionally, we obtained the underlying model coefficient vector $x^* \in R^{500}$, which was generated as a piecewise constant vector for r=1 and a piecewise linear vector for r=2. The corresponding graphics for the x values in the r=1 and r=2 cases are shown below.



Then, we can calculate the response with following model: $b_i = a_i^T x^* + \epsilon_i$ where $i \in [0, 999]$. Here are the graphics for b values for r=1 and r=2 cases respectively.



Now we can focus on how we compute σ^2. We computed the value of σ^2 with using SNR = 1. Below the section 5.1 they defined SNR.

$$SNR := ||Ax^*||^2/(n\sigma^2)$$

For our model, we can write $500\sigma^2 = ||Ax^*||^2$. And $\sigma^2 = 500/||Ax^*||^2$.

3.2 Unbounded Frank-Wolfe Algorithms implementation

The uFW and uAFW algorithms are implemented to solve for the generalized lasso problem with constraint matrices $D_n^{(r)}$, $r = 1, 2$.

An important part of this implementation is the characterization of $T = ker(D_n^{(r)})$

3.2.1 Computing projections onto T and $T^\perp$

Let $U \in R^n$ be an upper triangular matrix with all its upper triangular entries being 1.

Then

$$T = ker(D_n^{(r)}) = span\{\mathbf{1}_n, U\mathbf{1}_n, ..., U^{r-1}\mathbf{1}_n\}$$

The proof of this can be done by induction.

Let $A := [1_n; U1_n; ; U^{r1}1_n] \in R^{n\times r}$. To compute the projection P_T a QR decomposition of of $A = QR$. Thus the projection is given by $P_T x = QQ^T x$. In our implementation of the uFW algorithms, x^0 was initialized directly without its derivation from A.

3.2.2 Solving the linear programming sub problem

The linear oracle in the uFW algorithm has the form:

$$\min_x \langle c, x\rangle \quad s.t \quad ||D_n^{(r)}||_1 \quad \leq \quad \delta, \quad x \in (ker(D_n^{(r)})_\perp$$

Where $c = \nabla f(y^K)$. This admits a closed form solution.

The above can be written as:

$$\min_x \langle c, x\rangle \quad s.t \quad ||D_n^{(r)}||_1 \quad \leq \quad \delta, \quad Q^T x = 0$$

Where Q is computed in the QR decomposition of A as above.

We can apply a variable transformation via $x = U^r y$. Now define $\tilde{c} := (U^r)^T, c = [\tilde{c_1}; \tilde{c_2}]$ where $\tilde{c_1} \in R^{n-r}$ is the first $n - r$ components of $\tilde{c}$ and $\tilde{c_2} \in R^r$ is the last r components of . Similarly, define $y = [z; w]$ where $z \in R^{r\times r}$ is the first $n - r$ components of y and $w \in R^r$ is the last r components of y. We let $B_1 \in R^{r\times(n-r)}$ and $B_2 \in R^{r\times r}$ be such that $Q^T U^r = [B_1, B_2]$. Then the problem can be converted to:

$$\min_{z,w} \langle \tilde{c_1}, z \rangle + \langle \tilde{c_2}, w \rangle \quad s.t \quad ||z||_1 \leq \delta, \quad B_1 z + B_2 w = 0$$

.

We can reduce the problem even further from the equality constraints where we have $w = -B_2^{-1} B_1 z$ and the problem reduces to:

$$\min_{z} \langle \tilde{c_1} - B_1^T (B_2^{-1})^T \tilde{c}_2, z \rangle s.t \quad ||z||_1 \leq \delta$$

.

Denote $\widetilde{c} := \hat{c}_1 B_1^\top (B_2)^\top \hat{c_2}$ and let $j = \text{argmax}_i \, |\widetilde{c}_i|$ be its absolute singular value, the the solution of our linear programming sub problem is:

$$z = \text{-sign}(\widetilde{c}_j) \delta e_j$$

if $\widetilde{c} \geq 0, z = -\delta e_j$; otherwise $z = \delta e_j$

Reversing the process mentioned above, led us to the solution of linear oracle. A similar approach is taken in the Unbounded Away Step Frank-Wolfe algorithm, where we have to solve a second linear oracle to compute v^k in step 2 of Algorithm 2, as well as the updates of the vertex set and weights.

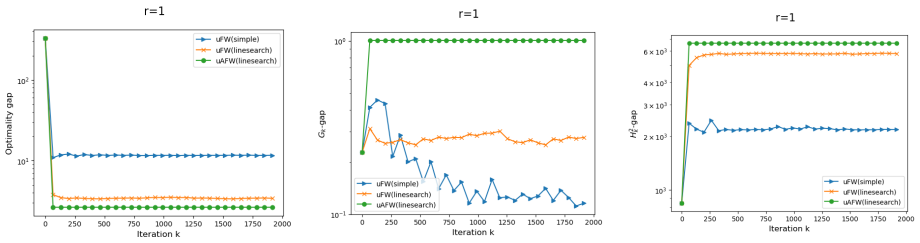
The cost of computing the linear oracle above is $O(nr)$. In addition, there is a one-time cost of $O(nr2)$ for computing the QR decomposition of A and $B_1^T (B_2^{-1})^T$ [3]

3.2.3 Choosing the gradient step size

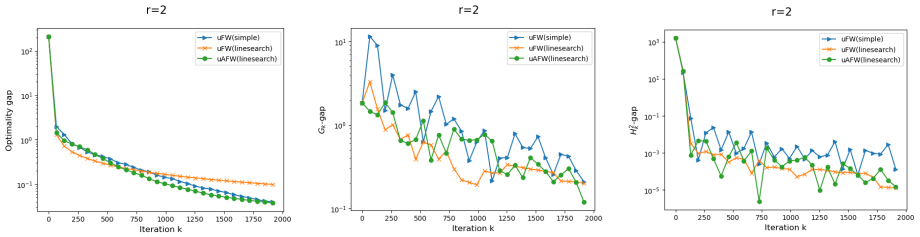
Choosing an appropriate gradient step size for step (1) of the algorithms is an important parameter for the correct convergence of the FW algorithm, using any complex gradient step size such as an adaptive step size would defeat the purpose of an efficient algorithm. The authors propose to choose a gradient steps size $\eta = 1/||A||^2$ where $||A||$ is the operator norm of the matrix A. During testing of the algorithms we encountered the exploding gradient problem where this proposed step size was too small and thus adopted another one that is $\tilde{\eta} = \frac{1}{\nabla f(x^k)}$

4 Results

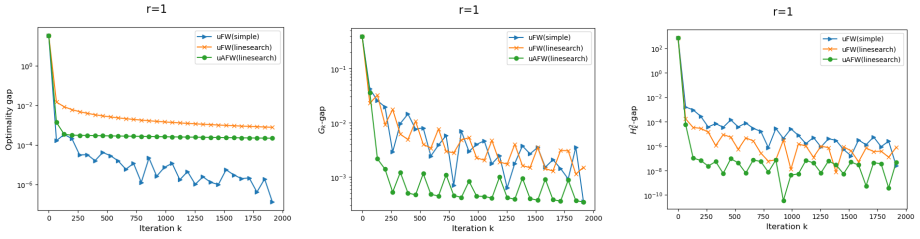
Our algorithms are written in the Python 3.11 and the optimality gap comparison is made as the result of MOSEK algorithm called through CVXPY library being the pivotal point, just like in the article. The other control parameters are the possible termination rules mentioned in the article. The G_k-gap is defined as G_k-gap $= \frac{G_k}{\max\{1, |f_k|\}}$, and the (relative) $H2_k$-gap is defined as $\frac{H2_k}{\max\{1, |f_k|\}}$. The algorithms are compared based on the criterion given under the synthetically generated dataset for r=1 which is the fused lasso case and the results are presented below in Figure 1.



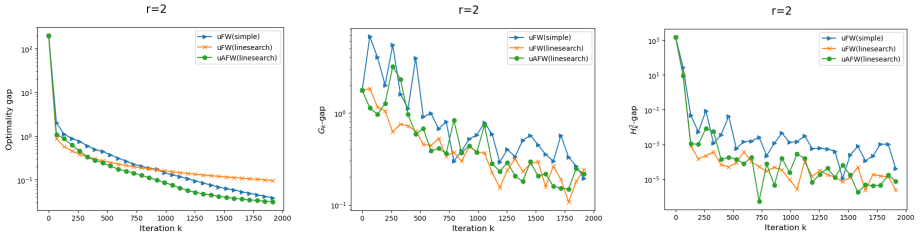
The same procedure repeated for r=2 case as well.



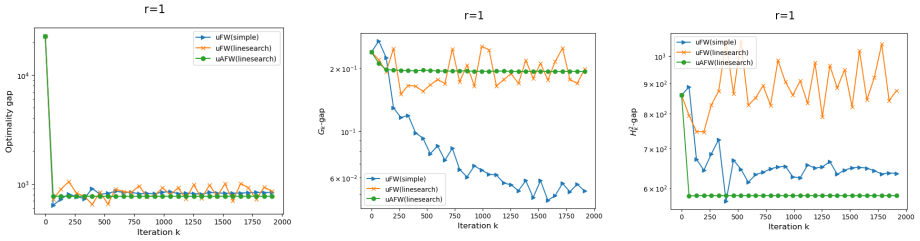
Nonetheless, we noticed that the results shown is highly dependent to the random seed (first seed were 681) fed into the model. Below, there is an example of the same results with a different seed (random seed= 122) fed into data generation.



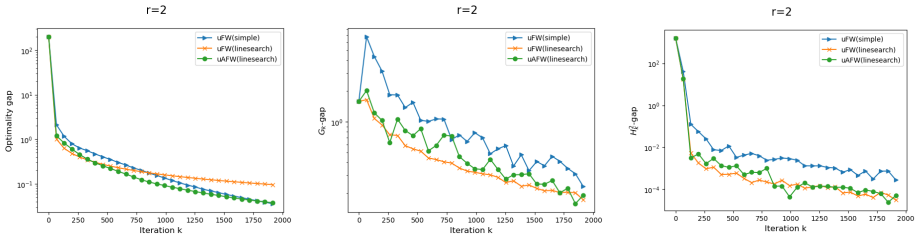
Shown above there is a clear difference between the results. The same procedure repeated for r=2 case as well.



In order to fix the issue, we decided to use average results of 10 repeats with random seeds ([619, 344, 268, 406, 327, 761, 358, 746, 668, 650]). Figure 3 shows the results gathered as an average of 10 repeats when r=1.



The same procedure applied for the case r=2.



Even though there are altercations in results, all of the models showed significant progress on closing the optimality gap.

In addition to these metrics, we keep record of the computation time of each model. The results are gathered the total time spent (seconds) on each method with maximum of 2000 iterations. The first table below presents the time spent for reaching the maximum number of iterations assigned for the first one seed method.

seed = 681	UFW simple	UFW linesearch	UAFW
r=1	5.70	5.85	10.04
r=2	5.61	5.70	8.99

For the other one seed approach the time required was:

seed = 122	UFW simple	UFW linesearch	UAFW
r=1	6.28	6.44	10.58
r=2	5.70	5.86	8.76

Finally, the total time required for 10 repeats were recorded as follows:

seed (10 seeds combined)	UFW simple	UFW linesearch	UAFW
r=1	58.28	59.8	101.6
r=2	55.4	57.6	87.4

5 Conclusion

In our study, we applied two different Frank-Wolfe Algorithms, namely Unbounded Frank-Wolfe and Unbounded Away-Step Frank-Wolfe, to solve the LASSO linear approximation model in unbounded regions. We tested these algorithms using synthetic data generated by us, considering different seeds and averaging results across ten seeds due to inconsistent outcomes with a single seed. The performance of the algorithms was evaluated using the Optimality Gap, $G_k - gap$, $H_k^2 - gap$, and, time spent.

For r=1, the Optimality Gap plot showed that Unbounded Frank-Wolfe with simple step size and Unbounded Away-Step Frank-Wolfe with linesearch converged to similar values, while Unbounded Frank-Wolfe with linesearch exhibited some fluctuations. In the $G_k - gap$ plot, Unbounded Frank-Wolfe with simple step size outperformed the other two algorithms, indicating closer proximity to the actual solution. However, in the $H_k^2 - gap$ plot, which measures the distance between the current solution and the best quadratic approximation of the objective function, Unbounded Away-Step Frank-Wolfe with linesearch performed better and converged.

For r=2, all the algorithms converged, but Unbounded Frank-Wolfe with linesearch had the worst performance in the Optimality Gap plot. In the other two plots, Unbounded Frank-Wolfe with linesearch, Unbounded Away-Step Frank-Wolfe with linesearch, and Unbounded Frank-Wolfe with simple step size showed the best results, respectively.

Across all three different seed trials, the r=2 cases demonstrated the fastest results within a maximum of 2000 iterations. Consistency was prioritized, thus averaging the results over 10 runs.

Furthermore, we found that the uFW algorithms are highly sensitive to the gradient step size η, suggesting that investigating more optimal values for η could benefit this family of algorithms.

References

[1] Martin Jaggi Simon Lacoste-Julien. "On the Global Linear Convergence of Frank-Wolfe Optimization Variants". In: (2015).

[2] Ryan J Tibshirani. *The solution path of the generalized lasso*. Stanford University, 2011.

[3] Haoyue Wang, Haihao Lu, and Rahul Mazumder. "Frank–Wolfe Methods with an Unbounded Feasible Region and Applications to Structured Learning". In: *SIAM Journal on Optimization* 32.4 (2022), pp. 2938–2968.