Approximate Frank-Wolfe Algorithms over Graph-structured Support Sets

4 Team Members

Abstract

In this project, we reviewed a paper that deals graph-structured convex optimization (GSCO) problem with the approximate Frank-Wolfe (FW) algorithm. We analyzed and re-implemented the original algorithm and introduced some extensions based on that. Then we conducted experiments to compare the results and concluded that our backtracking line-search method effectively reduced the number of iterations, while our new DMO method (Top-g+ optimal visiting) did not make satisfying enough improvements.

1   Introduction

The paper we reviewed introduces a kind of approximate Frank-Wolfe (FW) algorithm to solve convex optimization problems over graph-structured support sets where the linear minimization oracle (LMO) cannot be efficiently obtained in general.

1.1   Problem Statement

This paper deals with the following graph-structured convex optimization (GSCO) problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x}), \text{subject to } \boldsymbol{x} \in \mathcal{D}(C, \mathbb{M}), \tag{1}$$

where $\mathbb{M} := \{S_1, S_2, ..., S_m\}$, a collection of subsets of $[d]$, with $\cup_i S_i = [d]$, and $\mathcal{D}(C, \mathbb{M}) \triangleq \text{conv}\{\boldsymbol{x} : \|\boldsymbol{x}\|_2 \leq C, \text{supp}(\boldsymbol{x}) \in \mathbb{M}\}$ is a convex hull of the graph-structured support set described by $\mathbb{M}$, which contains a collection of allowed structures of the problem, and f is a convex differentiable function. The support of $\boldsymbol{x}$, i.e, $\text{supp}(\boldsymbol{x}) \triangleq \{i : \boldsymbol{x}_i \neq 0\}$, encodes the sparsity pattern of $\boldsymbol{x}$, which can be defined by interesting graph structures such as a path, tree, or cluster over an underlying graph. Model $\mathbb{M}$ describes many interesting scenarios where graph structures serve as a powerful prior.
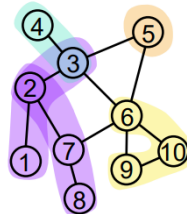

Figure 1: An element of a g-subgraph model $\mathbb{M}(\mathbb{G}, s = 5, g = 4)$ defined on a 10-node graph where each colored region is a subgraph.

As the authors mentioned, a natural idea to solve the GSCO problem is to use projected gradient descent (PGD) where a projection oracle finds a point in D per per-iteration. In order to obtain

approximate convergence guarantees, the existing works of this type generally have the assumption that projection oracles can be solved exactly or with very high approximation guarantees. However, projections satisfying these requirements are usually hard to find for the GSCO problem and multiple projections may be needed at per-iteration. These are the two main issues that the authors look forward to solving with alternative methods.

The authors stated that Frank-Wolfe(FW) methods (Frank et al., 1956), different from PGD-based methods, at each iteration, find a point using the linear minimization oracle (LMO), which for many constraints may enjoy a much cheaper per-iteration cost than the projection oracle (Combettes & Pokutta, 2021), and often obtain high-quality sparser solutions in early iterations. The fact that the FW methods are less explored for graph-structured optimization problems motivates the authors to tackle the GSCO problem using FW-type methods.

1.2 Motivation

The rate of the standard FW method can be improved to $\mathcal{O}(1/t^2)$ if $\mathcal{D}$ is a strongly convex set. However, sparsity-including sets are often not strongly convex so the standard FW method is inapplicable to GSCO problems. To achieve the $\mathcal{O}(1/t^2)$ rate without a strongly convex set, the authors proposed a novel method called approximate FW-type method.

The standard FW method gets a descent direction by using the linear minimization oracle (LMO), which has a cheaper iteration cost than the projection oracle for many constraints.

$$LMO: \boldsymbol{v}_t \in \operatorname*{argmin}_{\boldsymbol{v} \in \mathcal{D}(C,\mathbb{M})} \langle \nabla f(\boldsymbol{x}_t), \boldsymbol{v} \rangle \tag{2}$$

In general, the LMO returns an extreme point of $\mathcal{D}$. Therefore in the case of the $\mathcal{D}$ that is defined previously, this problem can be reformulated as a subspace identification problem

$$\min_{\boldsymbol{v} \in \mathcal{D}(C,\mathbb{M})} \langle \nabla f(\boldsymbol{x}_t), v \rangle = \min_{S^* \in \mathbb{M}, \|\boldsymbol{v}\|_2 \leq \mathcal{C}} \langle \nabla f(\boldsymbol{x}_t)_{S^*}, \boldsymbol{v} \rangle \tag{3}$$

where S^* is an optimal support set minimizing the inner product in (2). Hence, the minimizer is

$$\boldsymbol{v}_t = -\frac{C \cdot \nabla f(\boldsymbol{x}_t)_{S^*}}{\|\nabla f(\boldsymbol{x}_t)_{S^*}\|_2}, S^* \in \operatorname*{argmax}_{S \in \mathbb{M}} \|\nabla f(\boldsymbol{x}_t)_S\|_2^2 \tag{4}$$

The computational complexity of minimizing the inner product (2) increases dramatically due to finding S^*, when $\mathbb{M}$ is the g-subgraph model or other types of models. To decrease the computational complexity, the authors propose an approximate LMO to find $\boldsymbol{v}_t$. For our project, in order to extend upon the original authors' work, we planned to modify the process after Top-g+ visiting. In the original algorithm, we arbitrarily added feasible elements to $\boldsymbol{v}$ so that there would be multiple choices for $\boldsymbol{v}$. Instead of arbitrarily choosing $\boldsymbol{v}$ as the proximal gradient, we determined $\boldsymbol{v}$ by comparing the result of the objective function. The convergence rate can be expected to be improved.

2 Related work

The FW method (Franket et al., 1956) and its variants for convex constrained problems have recently received popularity mainly due to two advantages. First, it is projection free –the LMO is often much cheaper to compute than the projection oracle. Second, in applications with desired structured sparsity, early FW iterations tend to be naturally sparse. Inspired from these advantages, we seek to propose FW-type methods for GSCO problems.

Because of the properties of GSCO problems, using exact FW methods can be impossible. Recent studies mainly focus on the inexact FW methods to solve GSCO problems. The study of inexact FW methods tend to center on two types of LMO errors: gap-additive (Dunn and Harshbarger, 1978; Jaggi, 2013) and gap-multiplicative (Locatello et al., 2017; Pedregosa et al., 2020). Both methods can be hard to solve the LMO without exact support. Gap-additive bound cannot decay properly. Gap-multiplicative estimate could be negative. Instead, rather than approximating the gap, dual maximization oracle (DMO) turns to approximating $\langle \nabla f(\boldsymbol{x}_t), \boldsymbol{v}_t \rangle$(Zhou et al., 2021). An efficient

method for DMO to solve GSCO problems is Top-g+ visiting proposed by Zhou. To improve the convergence rate of FW method based on Top-g+ visiting, we propose a new visiting method and combine backtracking linesearch method with standard FW method.

3 Method

3.1 Original Algorithm

To overcome the computational barrier and improve the convergence rate of FW method, the authors used an inner product operator (IPO) to approximate $\langle \nabla f(\boldsymbol{x}_t), \boldsymbol{v}_t \rangle$. Given gradient $\nabla f(\boldsymbol{x}_t)$, constraint set $\mathcal{D}$ and approximation factor $\delta \in (0, 1]$ the approximated IPO returns $\boldsymbol{v}_t$ such that

$$Approximate\ IPO \quad \langle \nabla f(\boldsymbol{x}_t), \boldsymbol{v}_t \rangle \leq \delta \cdot \min_{\boldsymbol{s} \in \mathcal{D}} \langle \nabla f(\boldsymbol{x}_t), \boldsymbol{s} \rangle \tag{5}$$

We denote such set of $\boldsymbol{v}_t$ as $(\delta, \nabla f(\boldsymbol{x}_t), \mathcal{D})$ IPO.

The approximate IPO can be easier to obtain via DMO. Given the structure support set $\mathbb{M}$, the DMO finds a set $S \in \mathbb{M}$ such that

$$\|\nabla f(\boldsymbol{x}_t)_S\|_2 \geq \delta \cdot \max_{S' \in \mathbb{M}} \|\nabla f(\boldsymbol{x}_t)_{S'}\|_2 \tag{6}$$

where approximation factor $\delta \in (0, 1]$. We denote such set S as the $(\delta, \nabla f(\boldsymbol{x}_t), \mathcal{D})$ DMO.

In practice, the authors used the $(\delta, \nabla f(\boldsymbol{x}_t), \mathcal{D})$ DMO for the g-subgraph model $\mathbb{M}$ by visiting Top-g+ neighbors. This DMO algorithm takes in the underlying graph $\mathbb{G}$, the sparsity k, the number of CCs g, and the gradient vector $\boldsymbol{z}$ as inputs. It first sorts the entries of $\boldsymbol{z}$ by magnitudes in order of large to small and picks the g largest magnitude elements. We initialized the output S to be the indices of these g elements in the original vector. Then we numbered these connected components by ID 1 to g. As a next step, we set a graph $\mathbb{F}$ with edges that were in g components and then added any feasible additional elements to $\mathbb{F}$ and S until the number of nonzero elements in S was equal to s (the maximum sparsity of $S \in \mathbb{M}$). This can be done by just picking any neighbor nodes to the first g "seed" nodes.

Compared with the standard FW method, the approximate FW-type method via DMO has 2 more steps (line 5 and line 6 in Alg. 1) to gain an approximal gradient $\boldsymbol{v}_t$.

Algorithm 1 FW-type methods for GSCOs

1: **Input:** step size $\{\eta_t\}$, δ, L, C, and $\mathbb{M}$
2: pick any point $\boldsymbol{x}_0$ in $\mathcal{D}$
3: **for** $t = 0, 1, \ldots,$ **do**
4: $\quad \boldsymbol{z}_t = \begin{cases} \text{DMO-FW} := -\nabla f(\boldsymbol{x}_t) \\ \text{DMO-AccFW} := -\left(\boldsymbol{x}_t - \frac{\nabla f(\boldsymbol{x}_t)}{L\eta_t}\right) \end{cases}$
5: $\quad S_t = (\delta, -\boldsymbol{z}_t, \mathcal{D})\text{-DMO}$
6: $\quad \tilde{\boldsymbol{v}}_t = \frac{C \cdot (\boldsymbol{z}_t)_{S_t}}{\|(\boldsymbol{z}_t)_{S_t}\|_2}$
7: $\quad$ Option I: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \eta_t(\tilde{\boldsymbol{v}}_t - \boldsymbol{x}_t)$
8: $\quad$ Option II: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \eta_t(\tilde{\boldsymbol{v}}_t/\delta - \boldsymbol{x}_t)$
9: **end for**
10: **Return** $(\boldsymbol{x}_{\bar{t}}, f(\boldsymbol{x}_{\bar{t}})), \bar{t} \in \underset{t}{\operatorname{argmin}} f(\boldsymbol{x}_t)$

3.2 Our Extension

To further improve the rate of approximate FW-type method, we made two changes to the original algorithm.

First, we added a backtracking line search in determining the optimal step size η_t in every iteration. This could potentially accelerate the convergence of the algorithm, because it ensures that the objective function decreases sufficiently with a properly chosen step size.

The second change we made is optimizing the process of Top-g+ visiting. In the original function, a support vector S is returned whenever it is found. However, multiple latent support vectors are available in the constraint set. The support vector first found can not guarantee to be the optimal descent direction. To return a better choice of support vector S, we compared various feasible support vectors S instead of immediately returning whenever one feasible vector is found.

We presented the original Top-g+ visiting method in Alg. 3 (in Appendix) while our new method Top-g+ optimal visiting method is in Alg. 4 (in Appendix). There are 2 design differences. First, our new method needs an additional input theta which decides the number of support vectors expected to be compared. Second, we modified the process of finding feasible support vectors (lines 14-31 in Alg. 3, lines 18-47 in Alg. 4). In our new method, every $(s - g)$ iteration could be viewed as one experiment. After we stayed the g largest magnitude elements in the dual vector z, we randomly added feasible adjacent nodes to the support vector S until $|S| = s$ (line 19 in Alg. 4). The process of finding enough feasible nodes to constitute a support vector S needed $(s - g)$ iterations. Once a support vector S was found, we recorded S and the gradient descent based on the support vector (line 25,37 in Alg. 4). After $(s - g) \times \theta$ iterations, we recorded θ possible support vectors. In the end, we chose the support vector S that had the largest magnitude of gradient descent (line 45,46 in Alg. 4) and returned S. Since the magnitude of gradient descent became larger, the convergence rate could be expected to be improved.

4 Experiment

4.1 Dataset and Optimization Problem

We finally used the graph from https://github.com/baojian/verse/tree/master/data, and blogcatalog.mat to be specific. There are 10312 nodes, 333983 links, and 39 categories in our dataset. It is composed of 2 sparse matrices, one for the sub-graph information and the other representing the node-to-node connection. Note that the original dataset in the paper is MNIST, which is actually not a graph-like dataset. However, the authors considered it as such for convenience! So we decided to use a real graph-like dataset for the sake of rigor.

The objective function we wanted to optimize is a least-square loss function, i.e.

$$\min_{\boldsymbol{x} \in \mathcal{D}(\mathcal{C}, \mathbb{M})} f(\boldsymbol{x}) := \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \tag{7}$$

where $\mathcal{D} := \{\boldsymbol{x} : \text{supp}(\boldsymbol{x}) \in \mathbb{M}, \|\boldsymbol{x}\|_2 \leq C\}$ with $C = 1$ and $\mathbb{M} = \{S \subseteq [d] = [10312] : |S| \leq s = 1623\}$. $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ is a normalized Gaussian sensing matrix where each entry $a_{ij} \sim N(0, 1)$, $\boldsymbol{y}$ is the observation vector, generated as $\boldsymbol{y} = \langle \boldsymbol{A}, \boldsymbol{x}^* \rangle + \boldsymbol{e}, \boldsymbol{e} \sim N(0, \sigma^2 \boldsymbol{I}_d)$, and $\boldsymbol{x}^*$ is a normalized sparse vector with norm 1 that we would like to recover from the observations using the least-square method. We set the number of observations n to 100 manually.

4.2 Baseline Implementation

We first implemented this baseline algorithm with the pseudo-code listed in section 4.2 of the original paper (as well as section 3.1 in our report). The baseline algorithm is a variation of the Frank-Wolfe method, with a DMO method to adapt variables on a graph data set. There are 2 variations with the computation of $\boldsymbol{z}_t$ in step 4, called DMO-FW and DMO-AccFW respectively, as well as another 2 variations for the update of $\boldsymbol{x}$ in steps 7 and 8. Besides, we added a stop criterion as $|(Obj(x_{t+1}) - Obj(x_t))/Obj(x_t)| \leq 10^{-6}$. As a default option for the Frank-Wolfe method, the step size η_t of the t's iteration here is set as $2/(t + 2)$. There are potentially more iteration step size choices, such as exact line search where $\eta_t = \underset{\eta \geq 0}{\text{argmin}} f(\boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t))$ (which could be expensive to solve), Demyanov-Rubinov step size, where $\eta_t = min\{\frac{\langle -\nabla f(\boldsymbol{x}_t), \tilde{\boldsymbol{v}}_t - \boldsymbol{x}_t \rangle}{L \|\tilde{\boldsymbol{v}}_t - \boldsymbol{x}_t\|^2}, 1\}$. L is the Lipschitz constant, in the least-square objective function here which can be expressed as $\sigma_1(\boldsymbol{A}^T \boldsymbol{A})$, which is

the largest singular value of $A^T A$. Because of the time limit and the huge computation cost to get the Lipschitz constant in singular value decomposition, we did not choose the exact line search and Demyanov-Rubinov step size.

4.3 Evaluation

To solve the GSCO problem, a natural idea is to use the projected gradient descent (PGD) method, where a *projection oracle* finds a point in $\mathcal{D}$ at per-iteration. In our Frank-Wolfe method, we can also use the PGD method to find a mask for the variable to satisfy the graph structure. We compared the objective function values for the Frank-Wolfe method via DMO, the random projected gradient descent method, and the best-projected gradient descent by comparing all possible cases (which could be expensive). We didn't implement the DMO-Acc method for it would be impossible to calculate the Lipschitz constant. For all three implementations including the original method and the two improvements, we implemented them based on Option I in Algorithm 1. Here are our results. We can see that the outputs of the Frank-Wolfe method via DMO are very close to the best-projected gradient descent by comparing all possible cases:
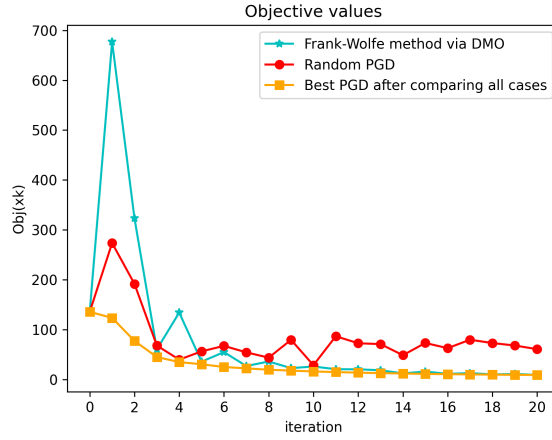

Figure 2: Objective function values for the Frank-Wolfe method via DMO, the random projected gradient descent method, and the best-projected gradient descent by comparing all possible cases.
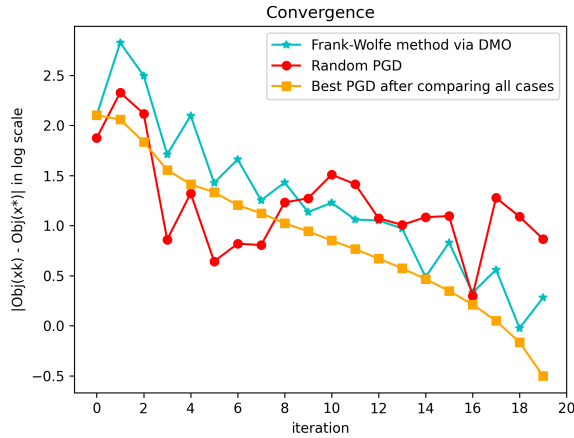

Figure 3: Convergence plot for the Frank-Wolfe method via DMO, the random projected gradient descent method, and the best-projected gradient descent by comparing all possible cases.

5

As seen in Fig. 2 and Fig. 3, Best PGD after comparing all the cases has the smallest objection function value in every iteration, and its objective function value steadily decreases as iteration goes. This is anticipated as Best PGD is the most expensive method in computing. Random PGD performs the worst as it has the highest objective function value at the end and the objective function value does not decrease significantly during the iteration process. The objective function value of random PGD shows great fluctuation and even increases significantly at some points. Random PGD projects the gradient on a lower-dimensional subspace randomly, so it may not be as accurate as other algorithms, which can result in slower convergence or lower-quality solutions. Frank-Wolfe method shows similar performance as Best PGD. It achieves a similar final objective function value and generally decreases as iteration goes on. However, we can see some increasing parts of the curve, and where its objective function values are much larger than Best PGD. This is probably due to a too-large step size selection. It seems like it's time to do some improvements.

4.4 Improvements

Backtracking line-search Armijo, L., 1966) " is a method used in many iterative algorithms to find an appropriate step size for the next iteration of the optimization process. The idea behind it is to ensure that the objective function decreases sufficiently by iteratively decreasing the step size until a suitable step size is found that satisfies certain conditions.

The traditional backtracking line-search is based on the gradient of current $\boldsymbol{x}_t$. However, we may find the position of $\boldsymbol{x}_t - \boldsymbol{v}_t$ in the updating step $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t(\boldsymbol{x}_t - \boldsymbol{v}_t)$ very similar to that of $\nabla f(\boldsymbol{x}_t)$ in gradient decent method. So here $\nabla f(\boldsymbol{x}_t)$ were replaced with $\boldsymbol{x}_t - \boldsymbol{v}_t$ in traditional backtracking line-search, and it worked well. The number of iterations required was reduced successfully!

Algorithm 2 Backtracking Line-search for FW-type method

1: **Input:** current location $\boldsymbol{x}_t$, new direction $\boldsymbol{v}_t$, previous step size η_t, decay parameter $\beta \in (0, 1)$
2: **while** $Obj(\boldsymbol{x}_t - \eta_t(\boldsymbol{x}_t - \boldsymbol{v}_t)) > Obj(\boldsymbol{x}_t) - 0.5\eta_t \|\boldsymbol{x}_t - \boldsymbol{v}_t\|_2^2$ **do**
3: $\quad \eta_{t+1} \leftarrow \beta\eta_t$
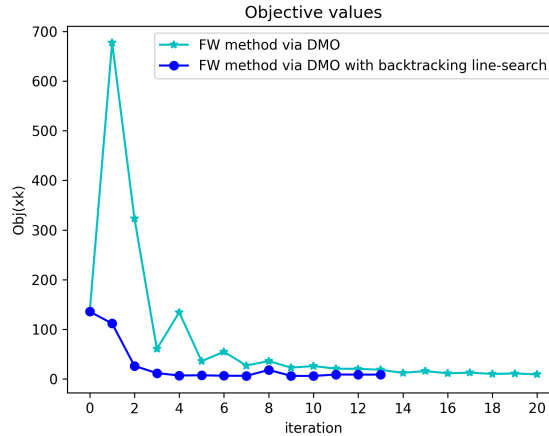4: **end while**
5: **Return** η_t

Figure 4: Objective function values for the Frank-Wolfe methods via DMO, without and with backtracking line-search.
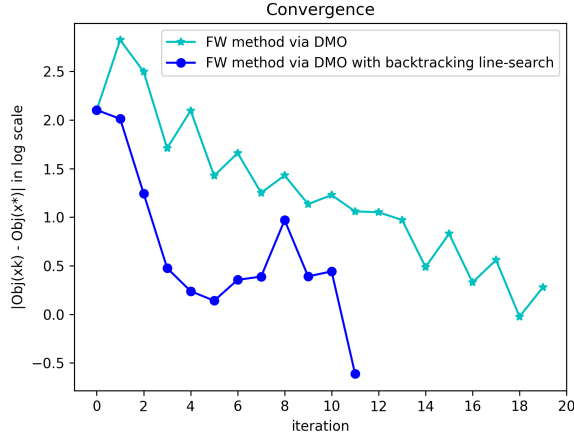
Figure 5: Convergence plot for the Frank-Wolfe methods via DMO, without and with backtracking line-search.

In Fig. 4 and Fig. 5, it is apparent that the FW method via DMO with backtracking line search greatly reduces the objective function value at every iteration. The iteration times needed for convergence are also greatly reduced. This can be partially explained by the nature of the method because it ensures that the objective function value decreases sufficiently with the chosen step size, rather than a possible increase because of a larger step size. If the condition is not satisfied, the step size is reduced and the process is repeated until the condition is met. By using a backtracking line-search method to select the step size in the Frank-Wolfe algorithm, it can make better progress toward the solution with fewer iterations, leading to faster convergence.

To verify Top-g+ optimal visiting method we propose converges faster, we compared the result of Alg. 3 and Alg. 4 (in Appendix). Blue line denotes the result of FW method via Top-g+ visiting while purple line denotes the result of FW method via Top-g+ optimal visiting.
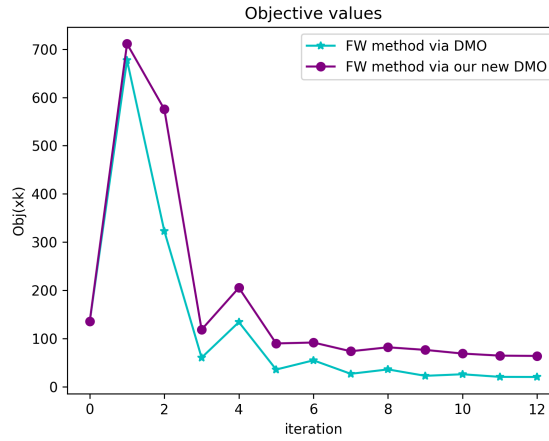

Figure 6: Objective function values for the Frank-Wolfe methods via original DMO and our modified DMO.
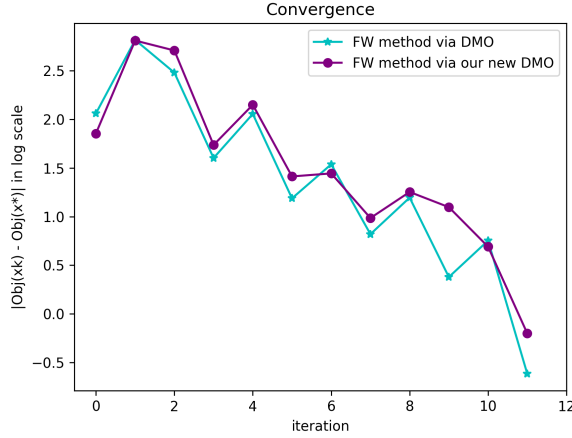
Figure 7: Convergence plot for the Frank-Wolfe methods via original DMO and our modified DMO.

The comparison of objective values and convergence is shown in Fig. 6 and Fig. 7. Compared with FW method via original DMO method (Top-g+ visiting), our new DMO method (Top-g+ optimal visiting) converges slightly faster in some points while the convergence rate can even be slower in other points. One possible reason is that the expected quantity of support vectors we want to compare is not enough. Without adequate support vectors to compare, we can not guarantee finding the optimal support vector. However, the computational complexity will dramatically increase if we expand the search area for support vectors.

5 Conclusion

In this report, we first introduced a kind of graph-structured convex optimization (GSCO) problem, which is relatively not easy to solve using traditional methods because the solution should satisfy the graph structure. It will always be a huge pain to obtain the optimal support set by traversing every group using LMO. Then we introduced a new DMO method to get an approximate IPO, which has more feasibility in some cases. In our experiments, we compared the objective function values for the FW method via DMO, the random PGD method, and the best PGD by comparing all possible cases. And we got a relatively good result: the outputs via the DMO method are very close to the Best PGD method. During the experiments, we also found that the authors of this paper didn't use a proper graph-like dataset, so they got an extremely perfect result. In the end, to deal with the shortcomings in the original algorithm, we proposed two improvements. The backtracking line-search method effectively reduced the number of iterations, while our new DMO method (Top-g+ optimal visiting) did not make many contributions. We notice that the improvement of convergence rate is slight, but the running time will dramatically increase if we want Top-g+ optimal visiting to have better performance. Our experiments indicate that it is not reasonable to improve convergence rate at the huge cost of running time. However, at several points Top-g+ optimal visiting method indeed improves the convergence rate. The result proves that the optimal support vector exists, but the method to find it still needs more studies.

Contribution By Group Members

- Author 1: Proposed Top-g+ optimal visiting method. Proposed to combine backtracking line-search with FW method. Participated in completing the new DMO algorithm. Wrote the 'Method' section and the relevant part of the 'Conclusion' section.

- Author 2: Participated in searching for the project topic. Wrote the Plan of Experiments section in the proposal. Participated in the implementation of the outer framework and ran the final program. Completed the relevant parts of the report. Add more complementary content to the report.

- Author 3: Participated in completing the outer framework of the algorithm. Implemented the backtracking line-search algorithm. Designed experiments to compare the performance of different methods. Completed the relevant part of the report.
- Author 4: Participated in searching for project topic; Wrote the 'Introduction' and 'Related work' sections of the proposal; Participated in implementing the original and new DMO algorithm. Revised the 'Introduction' section from the proposal, and wrote the "Original algorithm" sub-section of the report.

References

[1] Frank, M., Wolfe, P., et al. "An algorithm for quadratic programming". Naval research logistics quarterly, 3(1-2): 95–110, 1956.

[2] Demyanov, Vladimir and Rubinov, Aleksandr. "Approximate Methods in Optimization Problems". Elsevier (1970).

[3] Pedregosa, Fabian and Askari, Armin and Negiar, Geoffrey and Jaggi, Martin (2018) "Step-Size Adaptivity in Projection-Free Optimization". arXiv:1806.05123

[4] Lacoste-Julien, Simon. "Convergence rate of Frank-Wolfe for non-convex objectives". arXiv preprint arXiv:1607.00345 (2016).

[5] Armijo, L. "Minimization of functions having Lipschitz continuous first partial derivatives". Pacific Journal of Mathematics, 16(1), 1-3.(1966).

6 Appendix

Algorithm 3 ORIGINAL DMO WITH $\delta = \sqrt{1/\lceil s/g \rceil}$ approximation guarantee

---

 1: **Input**: underlying graph $\mathbb{G}(\mathbb{V}, \mathbb{E})$, sparsity k, number of CCs g, input vector $\boldsymbol{z}$
 2: Sort entries of $\boldsymbol{z}$ by magnitudes such that $|z_{\tau_1}| \geq |z_{\tau_2}| \geq \ldots \geq |z_{\tau_g}| \geq |z_{\tau_{g+1}}|$
 3: $I_g = [\tau_1, \tau_2, \ldots, \tau_g], S = I_g$
 4: $\boldsymbol{c} = \boldsymbol{0}$ {Initially, all nodes have same connected component ID}
 5: $i = 1$ // Tracking the ID of connected component
 6: **for** $v \in S$ **do**
 7:    $c_v = i$ // Node v has a component ID i
 8:    $i = i + 1$
 9: **end for**
10: $\mathbb{F} = \emptyset$ // Keep edges that are in g components
11: **if** $|S| = s$ **then**
12:    **Return** S // We assume $g \leq s$
13: **end if**
14: **for** $(u, v) \in \mathbb{E}$ **do**
15:    **if** $c_u == 0$ and $c_v \neq 0$ **then**
16:       $S = S \cup \{u\}$
17:       $\mathbb{F} = \mathbb{F} \cup (u, v)$
18:       $c_u = c_v$ // u is added to c_v-th component
19:    **end if**
20:    **if** $|S| = s$ **then**
21:       **Return** S
22:    **end if**
23:    **if** $c_u \neq 0$ and $c_v == 0$ **then**
24:       $S = S \cup \{v\}$
25:       $\mathbb{F} = \mathbb{F} \cup (u, v)$
26:       $c_v = c_u$ // v is added to c_u-th component
27:    **end if**
28:    **if** $|S| = s$ **then**
29:       **Return** S
30:    **end if**
31: **end for**

Algorithm 4 NEW DMO WITH $\delta = \sqrt{1/\lceil s/g \rceil}$ approximation guarantee

---

1: **Input**: underlying graph $\mathbb{G}(\mathbb{V}, \mathbb{E})$, number of CCs g, input vector $\boldsymbol{z}$, expected quantity of support sets θ
2: Sort entries of $\boldsymbol{z}$ by magnitudes such that $|z_{\tau_1}| \geq |z_{\tau_2}| \geq \ldots \geq |z_{\tau_g}| \geq |z_{\tau_{g+1}}|$
3: $I_g = [\tau_1, \tau_2, \ldots, \tau_g], S = I_g$
4: $\boldsymbol{c} = \boldsymbol{0}$ {Initially, all nodes have same connected component ID}
5: $i = 1$ // Tracking the ID of connected component
6: **for** $v \in S$ **do**
7:     $c_v = i$ // Node v has a component ID i
8:     $i = i + 1$
9: **end for**
10: $\mathbb{F} = \emptyset$ // Keep edges that are in g components
11: **if** $|S| = s$ **then**
12:     **Return** S // We assume $g \leq s$
13: **end if**
14: $j = 0$
15: $\mathbb{E}_0 = \mathbb{E}$
16: $S_0 = S$
17: $n = (s - g) * \theta$
18: **for** $t = 0, 1, \ldots, n$ **do**
19:     randomly select $(u, v) \in \mathbb{E}$
20:     **if** $c_u == 0$ and $c_v \neq 0$ **then**
21:         $S = S \cup \{u\}$
22:         $\mathbb{F} = \mathbb{F} \cup (u, v)$
23:         $c_u = c_v$ // u is added to c_v-th component
24:     **end if**
25:     **if** $|S| = s$ **then**
26:         $out[j] = \|z_S\|_2$
27:         $S_{list}[j] = S$ // record possible support vectors
28:         $j = j + 1$
29:         $\mathbb{E} = \mathbb{E}_0$
30:         $S = S_0$
31:     **end if**
32:     **if** $c_u \neq 0$ and $c_v == 0$ **then**
33:         $S = S \cup \{v\}$
34:         $\mathbb{F} = \mathbb{F} \cup (u, v)$
35:         $c_v = c_u$ // v is added to c_u-th component
36:     **end if**
37:     **if** $|S| = s$ **then**
38:         $out[j] = \|z_S\|_2$
39:         $S_{list}[j] = S$ // record possible support vectors
40:         $j = j + 1$
41:         $\mathbb{E} = \mathbb{E}_0$
42:         $S = S_0$
43:     **end if**
44: **end for**
45: d=index(max(out))
46: $S = S_{list}[d]$
47: **Return** S