Dual Averaging Method for Online Graph-structured Sparsity

Baojian Zhou
bzhou6@albany.edu
University at Albany, SUNY
Albany, NY, USA

Feng Chen
fchen5@albany.edu
University at Albany, SUNY
Albany, NY, USA

Yiming Ying
yying@albany.edu
University at Albany, SUNY
Albany, NY, USA

ABSTRACT

Online learning algorithms update models via one sample per iteration, thus efficient to process large-scale datasets and useful to detect malicious events for social benefits, such as disease outbreak and traffic congestion on the fly. However, existing algorithms for graph-structured models focused on the offline setting and the least square loss, incapable for online setting, while methods designed for online setting cannot be directly applied to the problem of complex (usually non-convex) graph-structured sparsity model. To address these limitations, in this paper we propose a new algorithm for graph-structured sparsity constraint problems under online setting, which we call GraphDA. The key part in GraphDA is to project both averaging gradient (in dual space) and primal variables (in primal space) onto lower dimensional subspaces, thus capturing the graph-structured sparsity effectively. Furthermore, the objective functions assumed here are generally convex so as to handle different losses for online learning settings. To the best of our knowledge, GraphDA is the first online learning algorithm for graph-structure constrained optimization problems. To validate our method, we conduct extensive experiments on both benchmark graph and real-world graph datasets. Our experiment results show that, compared to other baseline methods, GraphDA not only improves classification performance, but also successfully captures graph-structured features more effectively, hence stronger interpretability.

KEYWORDS

online learning; dual averaging; graph-structured sparsity

ACM Reference Format:
Baojian Zhou, Feng Chen, and Yiming Ying. 2019. Dual Averaging Method for Online Graph-structured Sparsity. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330915

1 INTRODUCTION

As a new paradigm in machine learning, convex online learning algorithms have received enormous attention [11, 17, 28, 38, 40, 44, 45]. These algorithms update learning models sequentially by using one training sample at each iteration, which makes them applicable to large-scale datasets on the fly and still enjoy *non-regret* property. For better interpretability and less computational complexity in high dimension data, many online learning algorithms [11, 30, 40, 41] exploit ℓ_1 norm or ℓ_1/ℓ_2 mixed norm to achieve sparse solution [25, 40, 41]. However, these sparsity-inducing models cannot characterize more complex (usually non-convex) graph-structured sparsity constraint, hence, unable to use some important priors such as graph data.

Graph-structured sparsity models have significant real-world applications, for example, social events [37], disease outbreaks [36], computer viruses [10], and gene networks [9]. These applications all contain graph structure information and the data samples are usually collected on the fly, i.e., the training samples have been received and processed one by one. Unfortunately, most of the graph-structured (non-convex) methods [1, 7, 22, 23] are batch learning-based, which cannot be applied to the online setting. The past few years have seen a surge of convex online learning algorithms, such as online projected gradient descent [48], AdaGrad [11], Adam [28], ℓ_1-RDA [40], FOBOS [13], and many others (e.g. [17, 38]). However, they cannot be used to tackle online graph-structured sparsity problems due to the limitation of sparsity-inducing norms.

In recent years, machine learning community [8, 14, 15, 18, 29, 42] have made promising progress on online non-convex optimization with regards to algorithms and *local-regret* bounds. Nonetheless, these algorithms cannot deal with graph-structured sparsity constraint problems due to the following two limitations: 1) The existing non-convexity assumption is only on the loss functions subject to a convex constraint; 2) Most of these proposed algorithms are based on online projected gradient descent (PGD), and cannot explore the structure information, hardly workable for graph-structured sparsity constraint. To the best of our knowledge, there is no existing work to tackle the combinatorial non-convexity constraint problems under online setting.

In this paper, we aim to design an approximated online learning algorithm that can capture graph-structured information effectively and efficiently. To address this new and challenging question, the potential algorithm has to meet two crucial requirements: 1) *graph-structured:* The algorithm should effectively capture the latent graph-structured information such as trees, clusters, connected subgraphs; 2) *online:* The algorithm should be efficiently applicable to online setting where training samples can only be processed one by one. Our assumption on the problem has a non-convex constraint but with a convex objective, which will sustain higher applicability in the practice of our setting. Inspired by the success of dual-averaging [34, 40], we propose the Graph Dual Averaging Algorithm, namely, GraphDA. The key part in GraphDA is to keep track of both averaging gradient via dual variables in dual space and primal variables in primal space. We then use two approximated projections to project both primal variables and dual variables onto

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330915

low dimension subspaces at each iteration. We conduct extensive experiments to demonstrate that by projecting both primal and dual variables, GraphDA captures the graph-structured sparsity effectively. Overall, our contributions are as follows:

- We propose a dual averaging-based algorithm to solve graph-structured sparsity constraint problems under online setting. To the best of our knowledge, it is a first attempt to establish an online learning algorithm for the graph-structured sparsity model.
- We prove the minimization problem occurring at each dual averaging step, which can be formulated as two equivalent optimization problems: minimization problem in primal space and maximization problem in dual space. The two optimization problems can then be solved approximately by adopting two popular projections. Furthermore, we provide two exact projection algorithms for the non-graph data.
- We conduct extensive experiments on both synthetic and real-world graphs. The experimental results demonstrate that GraphDA can successfully capture the latent graph-structure during online learning process. The learned model generated by our algorithm not only achieves higher classification accuracy but also stronger interpretability compared with the state-of-the-art algorithms.

The rest of the paper is organized as follows: Related work is teased out in Section 2. Section 3 gives the notations and problem definition. In Section 4, we present our main idea and algorithms. We report and discuss the experiment results in comparison with other baseline methods in Section 5. A short conclusion ensues in Section 6. Due to space limit, the detailed experimental setup and partial experimental results are supplied in Appendix. Our source code including baseline methods and datasets are accessible at: https://github.com/baojianzhou/graph-da.

2 RELATED WORK

In line with the focus of the present work, we categorize highly related researches into three sub-topics for the sake of clarity.

Online learning with sparsity. Online learning algorithms [6, 17, 38, 43, 48] try to solve classification or regression problems that can be employed in a fully incremental fashion. A natural way to solve online learning problem is to use stochastic gradient descent by using one sample at a time. However, this type of methods usually cannot produce any sparse solution. The gradient of only one sample has such a large variance that renders its projection unreliable. To capture the model sparsity, ℓ_1 norm-based [5, 11, 13, 30, 40] and ℓ_1/ℓ_2 mixed norm-based [41] are used under online learning setting; the dual-averaging [40] adds a convex regularization, namely ℓ_1-RDA to learn a sparsity model. Based on the dual averaging work, online group lasso and overlapping group lasso are proposed in [41], which provides us a sparse solution. However, the solution cannot produce methods directly applicable to graph-structured data. For example, as pointed out by [40], the levels of sparsity proposed in [13, 30] are not satisfactory compared with their batch counterparts.

Model-based sparsity. Different from ℓ_1-regularization [39] or ℓ_1-ball constraint-based method [12], model-based sparsity are non-convex [4, 21–23]. Using non-convex such as ℓ_0 sparsity based methods [3, 35, 46, 47] becomes popular, where the objective function is assumed to be convex with a sparsity constraint. To capture graph-structured sparsity constraint such as trees and connected graphs, a series of work [4, 20, 22, 23] has proposed to use structured sparsity model $\mathbb{M}$ to define allowed supports $\mathbb{M} = \{S_1, S_2, \ldots, S_k\}$. These complex models are non-convex, and gradient descent-based algorithms involve a projection operator which is usually NP-hard. [21–23] use two approximated projections (head and tail) without sacrificing too much precision. However, the above research work cannot be directly applied to online setting and the objective function considered is not general loss.

Online non-convex optimization. The basic assumption in recent progress on online non-convex optimization [8, 14, 15, 18, 29, 42] is that the objective considered is non-convex. *Local-regret* bound has been explored in these studies, most of which are based on projected gradient descent methods, for example, online projected gradient descent [18] and online normalized gradient descent [14]. However, these online non-convex algorithms cannot deal with our problem setting where there exists a combinatorial non-convex structure.

3 PRELIMINARIES

We begin by introducing some basic mathematical terms and notations, and then define our problem.

3.1 Notations

An index set is defined as $[p] = \{1, \ldots, p\}$. The bolded lower-case letters, e.g., $\boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^p$, denote column vectors where their i-th entries are w_i, x_i. The ℓ_2-norm of $\boldsymbol{w}$ is denoted as $\|\boldsymbol{w}\|_2$. The inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ on $\mathbb{R}^p$ is defined as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = x_1 y_1 + \cdots + x_p y_p$. Given a differentiable function $f(\boldsymbol{w}) : \mathbb{R}^p \to \mathbb{R}$, the gradient at $\boldsymbol{w}$ is denoted as $\nabla f(\boldsymbol{w})$. The support set of $\boldsymbol{w}$, i.e., $\mathrm{supp}(\boldsymbol{w}) := \{i | w_i \neq 0\}$, is defined as a subset of indices which index non-zero entries. If $|\mathrm{supp}(\boldsymbol{w})| \leq s$, $\boldsymbol{w}$ is called an s sparse vector. The upper-case letters, e.g., Ω, denote a subset of $[p]$ and its complement is $\Omega^c = [p] \backslash \Omega$. The restricted vector of $\boldsymbol{w}$ on Ω is denoted as $\boldsymbol{w}_\Omega \in \mathbb{R}^p$, where $(\boldsymbol{w}_\Omega)_i = w_i$ if $i \in \Omega$; otherwise 0. We define the undirected graph as $\mathbb{G}(\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = [p]$ is the set of nodes and $\mathbb{E}$ is the set of edges such that $\mathbb{E} \subseteq \{(u, v) | u \in \mathbb{V}, v \in \mathbb{V}\}$. The upper-case letters, e.g., H, T, S, stand for subsets of $[p] := \{1, 2, \ldots, p\}$. Given the standard basis $\{\boldsymbol{e}_i : 1 \leq i \leq p\}$ of $\mathbb{R}^p$, we also use H, T, S to represent subspaces. For example, the subspace S is the subspace spanned by S, i.e., $\mathrm{span}\{\boldsymbol{e}_i : i \in S\}$. We will clarify the difference only if confusion occurs.

3.2 Problem Definition

Here, we study an online non-convex optimization problem, which is to minimize the regret as defined in the following:

$$R(T, \mathcal{M}(\mathbb{M})) := \sum_{t=1}^{T} f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\}) - \min_{\boldsymbol{w} \in \mathcal{M}(\mathbb{M})} \sum_{t=1}^{T} f_t(\boldsymbol{w}), \quad (1)$$

where each $f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\})$ is the loss that a learner predicts an answer for the question $\boldsymbol{x}_t$ after receiving the correct answer y_t, and $\min_{\boldsymbol{w} \in \mathcal{M}(\mathbb{M})} \sum_{t=1}^{T} f_t(\boldsymbol{w})$ is the minimum loss that the learner can potentially get. To simplify, we assume f_t is convex differentiable. For example, we can use the least square loss $f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\}) = (\boldsymbol{w}_t^\top \boldsymbol{x}_t - y_t)^2$ for the online linear regression problem and logistic loss $f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\}) = \log(1 + \exp(-y_t \cdot \boldsymbol{w}^\top \boldsymbol{x}_t))$ for online binary

classification problem where $y_t \in \{\pm 1\}$. The goal of the learner is to minimize the regret $R(T, \mathcal{M}(\mathbb{M}))$. Different from the online convex optimization setting in [17, 38], $\mathcal{M}(\mathbb{M}) \subseteq \mathbb{R}^p$ is a generally non-convex set. To capture more complex graph-structured information, in a series of seminal work [4, 21, 22], a structured sparsity model $\mathcal{M}(\mathbb{M})$ is proposed as follows:

$$\mathcal{M}(\mathbb{M}) := \{\boldsymbol{w} | \text{supp}(\boldsymbol{w}) \subseteq S \text{ for some } S \in \mathbb{M}\}, \tag{2}$$

where $\mathbb{M} = \{S_1, S_2, \ldots, S_k\}$ is the collection of allowed structure supports with $S_i \in [p]$. Basically, $\mathcal{M}(\mathbb{M})$ is the union of k subspaces. Each subspace is uniquely identified by S_i. Definition (2) is so general that it captures a broad spectrum of graph-structured sparsity models such as trees [20], connected subgraphs [7, 22]. We mainly focus on the Weighted Graph Model(WGM) proposed in [22].

Definition 1 (Weighted Graph Model [22]). *Given an underlying graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \boldsymbol{c})$ defined on the coefficients of the unknown vector $\boldsymbol{w}$, where $\mathbb{V} = [p]$, $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ and associated cost vector $\boldsymbol{c}$ on edges, then the weighted graph model $(\mathbb{G}, s, g, B)$-WGM can be defined as the following set of supports:*

$$\mathbb{M} = \{F : |F| \leq s, \text{ there is an forest } \mathcal{F} \text{ with } \mathbb{V}_{\mathcal{F}} = F, \gamma(\mathcal{F}) = g, \text{ and } \boldsymbol{c}(\mathcal{F}) \leq B\},$$

where B is the budget on cost of edges in forest $\mathcal{F}$, $\gamma(\mathcal{F})$ is the number of connected component in forest $\mathcal{F}$ denoted as g, and s is the sparsity. To clarify, forest $\mathcal{F}$ is the subgraph induced by its nodes set F, i.e. $\mathcal{F} := \mathbb{G}(F, \mathbb{E}')$, where $\mathbb{E}' = \{(u, v) : u \in F, v \in F, (u, v) \in \mathbb{E}\}$. $\boldsymbol{c}(F)$ is the total edge costs in forest $\mathcal{F}$.

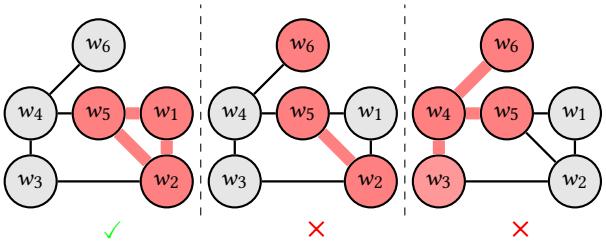


Figure 1: A toy example of Weighted Graph Model

$(\mathbb{G}, s, g, B)$-WGM captures a broad range of graph structures such as groups, clusters, trees, and subgraphs. A toy example is given in Figure 1 where we define a graph with 6 nodes in $\mathbb{V}$, 7 edges in $\mathbb{E}$, and the cost of all edges is set to 1. Let $\mathbb{V}$ be associated with a vector $\boldsymbol{w} \in \mathbb{R}^6$. Suppose we are interested in connected subgraphs[1] with at most 3 nodes, to capture these subgraphs, $\mathbb{M}$ can be defined as $\mathbb{M} = \{S_i | \mathbb{G}(S_i, \mathbb{E}_i) \text{ is connected }, |S_i| \leq 3\}$. By letting the budget $B = 3$, and the sparsity parameter $s = 3$, we can clearly use $(\mathbb{G}, 3, 1, 3)$-WGM to represent this $\mathbb{M}$. Figure 1 shows three subgraphs formed by red nodes and edges. The subgraph induced by $\{w_1, w_2, w_5\}$ on the left is in $\mathbb{M}$. However, the subgraph induced by $\{w_2, w_5, w_6\}$ in the middle is not in $\mathbb{M}$ because of the non-connectivity. The subgraph formed by $\{w_3, w_4, w_5, w_6\}$ on the right is not in $\mathbb{M}$ either, as it violates the sparsity constraint, i.e., $s \leq 3$.

After defining the structure-sparsity model $\mathcal{M}(\mathbb{M})$, we explore how to design an efficient and effective algorithm to minimize the regret under model constraint. An intuitive way to do this is to use online projected gradient descent [48] where the algorithm needs to solve the following projection at iteration t:

$$\boldsymbol{w}_{t+1} = \text{P}(\boldsymbol{w}_t - \eta_t \nabla f_t(\boldsymbol{w}_t), \mathcal{M}(\mathbb{M})), \tag{3}$$

where η_t is the learning rate and P is the projection operator onto $\mathcal{M}(\mathbb{M})$, i.e., $\text{P}(\cdot, \mathcal{M}(\mathbb{M})) : \mathbb{R}^p \to \mathbb{R}^p$ is defined as

$$\text{P}(\boldsymbol{w}, \mathcal{M}(\mathbb{M})) = \underset{\boldsymbol{x} \in \mathcal{M}(\mathbb{M})}{\arg\min} \|\boldsymbol{w} - \boldsymbol{x}\|^2. \tag{4}$$

However, there are two essential drawbacks of using (3): First, the projection in (3) only uses single gradient $\nabla f_t(\boldsymbol{w}_t)$ which is too noisy (large variance) to capture the graph-structured information at each iteration; Second, the training samples coming later are less important than these coming earlier due to the decay of learning rate η_t. Recall that η_t needs to decay asymptotically to $O(1/\sqrt{t})$ in order to achieve a non-regret bound. Fortunately, inspired by [34, 40], the above two weaknesses can be successfully overcome by using dual averaging. The main idea is to keep tracking both primal vectors (corresponding to $\boldsymbol{w}_t$ in primal space) and dual variables (corresponding to gradients, $\nabla f_t(\boldsymbol{w}_t)$ in dual space[2]) at each iteration. In the next section, we focus on designing an efficient algorithm by using the idea of dual averaging to capture graph-structured sparsity under online setting.

4 ALGORITHM: GRAPHDA

We try to develop a dual averaging-based method to minimize the regret (1). At each iteration, the method updates $\boldsymbol{w}_t$ by using the following minimization step:

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w} \in \mathcal{M}(\mathbb{M})}{\arg\min} \left\{ \frac{1}{t+1} \sum_{i=0}^{t} \langle \boldsymbol{g}_i, \boldsymbol{w} \rangle + \frac{\beta_t}{2t} \|\boldsymbol{w}\|_2^2 \right\}, \tag{5}$$

where β_t is to control the learning rate implicitly and $\boldsymbol{g}_i$ is a subgradient in $\partial f_i(\boldsymbol{w}, \{\boldsymbol{x}_i, y_i\}) = \{\boldsymbol{g} : f_i(\boldsymbol{z}, \{\boldsymbol{x}_i, y_i\}) \geq f_i(\boldsymbol{w}, \{\boldsymbol{x}_i, y_i\}) + \langle \boldsymbol{g}, \boldsymbol{z} - \boldsymbol{w} \rangle, \forall \boldsymbol{z} \in \mathcal{M}(\mathbb{R})\}$[3]. Different from the convexity explored in [34] and [40], the problem considered here is generally non-convex, which makes it NP-hard to solve. Initially, the solution of the primal is set to zero, i.e., $\boldsymbol{w}_0 = \boldsymbol{0}$.[4] Then at each iteration, it computes a subgradient $\boldsymbol{g}_t$ based on current data sample $\{\boldsymbol{x}_t, y_t\}$ and then updates $\boldsymbol{w}_t$ by using (5) averaging gradient from the dual space. The algorithm terminates after receiving T samples and returns the model $\boldsymbol{w}_T$ or $\bar{\boldsymbol{w}}_T = 1/T \sum_{t=0}^{T} \boldsymbol{w}_t$ depending on needs. The dual averaging step (5) has two advantages: 1) The gradient information of training samples coming later will not decay when new samples are coming; 2) The averaging gradient can be accumulated during the learning process; hence we can use it to capture graph-structure information more effectively than online PGD-based methods.

Due to the NP-hardness to compute (5), it is impractical to directly use (5). Thus, we have to treat this minimization step more

[1] A connected subgraph is the subgraph which has only 1 connected component.

[2] Notice that we use ℓ_2-norm, i.e. $\|\cdot\|_2$, which is defined in the Euclidean space $X = \mathbb{R}^p$. By definition, the dual norm of ℓ_2 is identical to itself, i.e., $\|\cdot\|_* = \|\cdot\|_2$. Also, recall that the dual space X^* of the Euclidean space is also identical with each other $X^* = X = \mathbb{R}^p$.

[3] As we assume f_i is convex differentiable, then we have $\partial f_i(\boldsymbol{w}, \{\boldsymbol{x}_i, y_i\}) = \{\nabla f_i(\boldsymbol{w}, \{\boldsymbol{x}_i, y_i\})\}$, i.e. $\boldsymbol{g}_i = \nabla f_i(\boldsymbol{w}, \{\boldsymbol{x}_i, y_i\})$.

[4] There are two advantages: 1. $\boldsymbol{w}_0 = \boldsymbol{0}$ is trivially in the $\mathcal{M}(\mathbb{M})$; 2. $\boldsymbol{w}_0 = \boldsymbol{0} \in \arg\min_{\boldsymbol{w}} \|\boldsymbol{w}\|_2^2$ under convex setting [40] so that sublinear regret can obtain.

carefully for $\mathcal{M}(\mathbb{M})$. The minimization step (5) has the following equivalent projection problems, specified in Theorem 1.

THEOREM 1. *Assume $\beta_t = \gamma\sqrt{t}$, where $\gamma > 0$ and denote $\bar{\boldsymbol{s}}_{t+1} = \frac{1}{t+1}\sum_{i=0}^{t} \boldsymbol{g}_i$. The minimization step of (5) can be expressed as the following two equivalent optimization problems:*

$$\max_{S\in\mathbb{M}} \|P(-\frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}, S)\|_2^2 \tag{6}$$

$$\min_{S\in\mathbb{M}} \| - \frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma} - P(-\frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}, S)\|_2^2, \tag{7}$$

where $P(\boldsymbol{s}, S)$ is the projection operator that projects $\boldsymbol{s}$ onto the subspace spanned by S.

PROOF. The original minimization problem in (5) can be equivalently expressed as

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \arg\min_{\boldsymbol{w}\in\mathcal{M}(\mathbb{M})} \left\{\langle \bar{\boldsymbol{s}}_{t+1}, \boldsymbol{w}\rangle + \frac{\gamma}{2\sqrt{t}}\|\boldsymbol{w}\|_2^2\right\} \\
&= \arg\min_{\boldsymbol{w}\in\mathcal{M}(\mathbb{M})} \left\{\frac{\sqrt{t}}{2\gamma}\|\bar{\boldsymbol{s}}_{t+1}\|_2^2 + \langle \bar{\boldsymbol{s}}_{t+1}, \boldsymbol{w}\rangle + \frac{\gamma}{2\sqrt{t}}\|\boldsymbol{w}\|_2^2\right\} \\
&= \arg\min_{\boldsymbol{w}\in\mathcal{M}(\mathbb{M})} \frac{\gamma}{2\sqrt{t}}\left\|\boldsymbol{w} - \left(-\frac{\sqrt{t}}{\gamma}\bar{\boldsymbol{s}}_{t+1}\right)\right\|_2^2 \\
&= \arg\min_{\boldsymbol{w}\in\mathcal{M}(\mathbb{M})} \left\|\boldsymbol{w} - \left(-\frac{\sqrt{t}}{\gamma}\bar{\boldsymbol{s}}_{t+1}\right)\right\|_2^2,
\end{aligned} \tag{8}$$

where the second equality follows by adding a constant to the minimization objective and (8) follows by multiplying $2\sqrt{t}/\gamma$ on the third equation. Hence, (5) is equivalent to the minimization of (8). Clearly, (8) is essentially the projection $\mathrm{P}(-(\sqrt{t}\boldsymbol{s}_{t+1})/\gamma, \mathcal{M}(\mathbb{M}))$ defined in (4). To further explore (8), notice that one needs to solve the following equivalent minimization problem:

$$\min_{\boldsymbol{w}\in\mathcal{M}(\mathbb{M})} \| - \frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma} - \boldsymbol{w}\|_2^2 \Leftrightarrow \min_{S\in\mathbb{M}} \| - \frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma} - \mathrm{P}(-\frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}, S)\|_2^2.$$

Here, for any $\boldsymbol{x}$, $\mathrm{P}(\boldsymbol{x}, S)$ is an orthogonal projection operator that projects $\boldsymbol{x}$ onto subspace spanned by S. By the projection theorem, for any $\boldsymbol{x}$, it always has the following property:

$$\|\boldsymbol{x}\|_2^2 - \|\mathrm{P}(\boldsymbol{x}, S)\|_2^2 = \|\boldsymbol{x} - \mathrm{P}(\boldsymbol{x}, S)\|_2^2.$$

Replacing $\boldsymbol{x}$ by $-\sqrt{t}\bar{\boldsymbol{s}}_{t+1}/\gamma$ and adding minimization to both sides with respect to subspace S, we obtain:

$$\begin{aligned}
&\min_{S\in\mathbb{M}} \left\{\| - \frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}\|_2^2 - \|\mathrm{P}(-\frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}, S)\|_2^2\right\} \\
&= \min_{S\in\mathbb{M}} \| - \frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma} - \mathrm{P}(-\frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}, S)\|_2^2.
\end{aligned}$$

By moving the minimization into the negative term, we obtain

$$\begin{aligned}
&\| - \frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}\|_2^2 + \max_{S\in\mathbb{M}} \|\mathrm{P}(-\frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}, S)\|_2^2 \\
&= \min_{S\in\mathbb{M}} \| - \frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma} - \mathrm{P}(-\frac{\sqrt{t}\bar{\boldsymbol{s}}_{t+1}}{\gamma}, S)\|_2^2.
\end{aligned} \tag{9}$$

We prove the theorem. □

The above theorem leads to a key insight that the NP-hard problem (8) can be solved either by maximizing $\|\mathrm{P}(\sqrt{t}\bar{\boldsymbol{s}}_{t+1}/\gamma, S)\|_2^2$ or by minimizing $\|\sqrt{t}\bar{\boldsymbol{s}}_{t+1}/\gamma - \mathrm{P}(\sqrt{t}\bar{\boldsymbol{s}}_{t+1}/\gamma, S)\|_2^2$ over S. Inspired by [21–23], instead of solving these two problems exactly, we apply two approximated algorithms provided in [22] to solve the problem approximately. We present the following two assumptions:

ASSUMPTION 1 (HEAD PROJECTION [23]). *Let $\mathbb{M}$ and $\mathbb{M}_H$ be the predefined subspace models. Given any $\boldsymbol{w}$, there exists a $(c_H, \mathbb{M}, \mathbb{M}_H)$ Head-Projection which is to find a subspace $H \in \mathbb{M}_H$ such that*

$$\|\mathrm{P}(\boldsymbol{w}, H)\|^2 \geq c_H \cdot \max_{S\in\mathbb{M}} \|\mathrm{P}(\boldsymbol{w}, S)\|^2, \tag{10}$$

where $0 < c_H \leq 1$. We denote $\mathrm{P}(\boldsymbol{w}, H)$ as $\mathrm{P}(\boldsymbol{w}, \mathbb{M}, \mathbb{M}_H)$.

ASSUMPTION 2 (TAIL PROJECTION [23]). *Let $\mathbb{M}$ and $\mathbb{M}_T$ be the predefined subspace models. Given any $\boldsymbol{w}$, there exists a $(c_T, \mathbb{M}, \mathbb{M}_T)$ Tail-Projection which is to find a subspace $T \in \mathbb{M}_T$ such that*

$$\|\mathrm{P}(\boldsymbol{w}, T) - \boldsymbol{w}\|^2 \leq c_T \cdot \min_{S\in\mathbb{M}} \|\boldsymbol{w} - \mathrm{P}(\boldsymbol{w}, S)\|^2, \tag{11}$$

where $c_T \geq 1$. We denote $\mathrm{P}(\boldsymbol{w}, T)$ as $\mathrm{P}(\boldsymbol{w}, \mathbb{M}, \mathbb{M}_T)$.

To minimize the regret $R(T, \mathcal{M}(\mathbb{M}))$, we propose the approximated algorithm, presented in Algorithm 1 below. Initially, the primal vector $\boldsymbol{w}_0$ and dual vector $\bar{\boldsymbol{s}}_0$ are all set to $\mathbf{0}$. At each iteration, it works as the following four steps:

- **Step 1**: The learner receives a question $\boldsymbol{x}_t$ and makes a prediction based on $\boldsymbol{x}_t$ and $\boldsymbol{w}_t$. After suffering a loss $f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\})$, it computes the gradient $\boldsymbol{g}_t$ in Line 4;
- **Step 2**: In Line 5, the current gradient $\boldsymbol{g}_t$ has been accumulated into $\bar{\boldsymbol{s}}_{t+1}$, which is ready for the next head projection[5];
- **Step 3**: The head projection inputs the accumulated gradient $\bar{\boldsymbol{s}}_{t+1}$ and outputs the vector $\boldsymbol{b}_{t+1}$ so that $\mathrm{supp}(\boldsymbol{b}_{t+1}) \in \mathbb{M}_H$;
- **Step 4**: The next predictor $\boldsymbol{w}_{t+1}$ is then updated by using the tail projection, i.e., $\mathrm{supp}(\boldsymbol{w}_{t+1}) \in \mathbb{M}_T$. The weight $-\sqrt{t}/\gamma$ is to control the learning rate.

The algorithm repeats the above four steps until some stop condition is satisfied. The main difference between our method and the methods in [34, 40] lies in that, we, as a first attempt, use two projections (Line 6 and Line 7), to project dual vector $\bar{\boldsymbol{s}}_{t+1}$ and primal vector $\boldsymbol{w}_{t+1}$ onto a graph-structured subspaces $\mathbb{M}_H$ and $\mathbb{M}_T$ respectively. In dual projection step, most of the irrelevant gradient entries have been effectively set to zero values. In primal tail projection step, we make sure $\boldsymbol{w}_{t+1}$ has been projected onto $\mathbb{M}_T$ so that the constraint of interest is satisfied.

Algorithm 1 GRAPHDA: Online Graph Dual Averaging Algorithm

1: **Input**: γ, $\mathbb{M}$
2: $\bar{\boldsymbol{s}}_0 = \mathbf{0}, \boldsymbol{w}_0 = \mathbf{0}$
3: **for** $t = 0, 1, 2, \ldots$ **do**
4: receive $\{\boldsymbol{x}_t, y_t\}$ and compute $\mathbf{g}_t = \nabla f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\})$
5: $\bar{\boldsymbol{s}}_{t+1} = \bar{\boldsymbol{s}}_t + \boldsymbol{g}_t$
6: $\mathbf{b}_{t+1} = \mathrm{P}(\bar{\mathbf{s}}_{t+1}, \mathbb{M})$
7: $\boldsymbol{w}_{t+1} = \mathrm{P}(-\frac{\sqrt{t}}{\gamma}\boldsymbol{b}_{t+1}, \mathbb{M})$
8: **end for**

[5]Pseudo-code of these two projections are provided in Appendix A for completeness.

In real applications, graph data is not always available, i.e., $\mathbb{M}$ cannot be explicitly constructed by $\mathbb{G}(\mathbb{V}, \mathbb{E})$, so we often have to deal with non-graph data but still with the aim to pursue structure sparsity constraint. To compensate, we provide Dual Averaging Iterative Hard Thresholding, namely DA-IHT, presented in Theorem 2, to handle non-graph data cases.

Theorem 2. *Assume that the graph information is not available or the graph is a complete graph and the budget B is large enough. We can define our model $\mathbb{M}$ such that it includes all possible s-sparse subgraphs, i.e., $\mathbb{M} = \{S : |S| \leq s\}$. Then there exists exactly head and tail projection algorithm such that*

$$\|\mathrm{P}(\boldsymbol{w}, H)\|_2^2 = \max_{S \in \mathbb{M}} \|\mathrm{P}(\boldsymbol{w}, S)\|_2^2, \tag{12}$$

and

$$\|\mathrm{P}(\boldsymbol{w}, T) - \boldsymbol{w}\|_2^2 = \min_{S \in \mathbb{M}} \|\boldsymbol{w} - \mathrm{P}(\boldsymbol{w}, S)\|_2^2. \tag{13}$$

Proof. Since the graph is a complete graph (i.e., all subgraphs are connected.) and the budget constraint B is large enough, any subset S that has s elements belongs to $\mathbb{M}$. In this case, $\mathbb{M}$ contains all s-subsets, i.e., $\mathbb{M} = \{S_i : |S_i| \leq s\}$. By sorting the magnitudes of $\boldsymbol{w}$ in a descending manner, we have

$$|w_{\tau_1}| \geq |w_{\tau_2}| \geq \ldots \geq |w_{\tau_s}| \geq \ldots \geq |w_{\tau_p}|.$$

Let $H = T = \{\tau_1, \tau_2, \ldots, \tau_s\}$. For any s-sparse set S, by the fact that $|w_{\tau_1}|, |w_{\tau_2}|, \ldots, |w_{\tau_s}|$ are the largest magnitude s entries, we always have

$$\|\mathrm{P}(\boldsymbol{w}, H)\|_2^2 \geq \|\mathrm{P}(\boldsymbol{w}, S)\|_2^2.$$

At the same time, $H \in \mathbb{M}$, then

$$\|\mathrm{P}(\boldsymbol{w}, H)\|_2^2 \leq \max_{S \in \mathbb{M}} \|\mathrm{P}(\boldsymbol{w}, S)\|_2^2.$$

Hence, we prove (12). In a similar vein, we can also prove (13). □

By Theorem 2, one can implement the two projections in Line 6 and 7 of Algorithm 1 by sorting the magnitudes $\bar{\boldsymbol{s}}_{t+1}$ and $-\sqrt{t}/\gamma \boldsymbol{b}_{t+1}$ respectively, to deal with non-graph data. DA-IHT will be used as a baseline in our experiment to compare with the graph-based method, GraphDA.

Time Complexity. At each iteration of GraphDA, the time complexity of two projections depends on the graph size p and the number of edges $|\mathbb{E}|$. As proved in [22], two projections have the time complexity $O(|\mathbb{E}| \log^3(p))$. In many real-world applications, the graphs are usually sparse, i.e., $O(p)$, and then the total complexity of each iteration of GraphDA is $O(p + p \log^3(p))$. Our method is characterized by two merits: 1) The time cost of each iteration is nearly-linear time; 2) At each iteration, it only has $O(p + |\mathbb{E}|)$ memory cost, where $O(p)$ stores the averaging gradient and current solution and $O(|\mathbb{E}|)$ is to save the graph. For DA-IHT, we need to select the top s largest magnitude entries at each iteration. Thus, the time cost of per-iteration is $O(sp)$ with $O(p)$ memory cost.

Regret Discussion. Given any online learning algorithm, we are interested in whether the regret $R(T, \mathcal{M}(\mathbb{M}))$ is sub-linear and whether we can bound the *estimation error* $\|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2$. We first assume the primal vectors are $\{\boldsymbol{w}_t\}_{t=0}^T$ and the dual gradient sequences $\{\boldsymbol{g}_t\}_{t=0}^T$. We then assume that the potential solution $\boldsymbol{w}$ is always bounded in D, i.e., $\|\boldsymbol{w}\|_2 \leq D$ and gradients are also bounded, i.e., $\|\boldsymbol{g}_t\|_2 \leq L$. Then for any $T \geq 1$ and any $\boldsymbol{w} \in \mathcal{M}(\mathbb{M})$, the regret in [40] can be bounded as the following:

$$R(T, \mathcal{M}(\mathbb{M})) \leq 2DL\sqrt{T}. \tag{14}$$

Given any optimal solution $\boldsymbol{w}^* \in \arg\min_{\boldsymbol{w} \in \mathcal{M}(\mathbb{M})} \sum_{i=1}^T f_i(\boldsymbol{w})$ and the solution $\boldsymbol{w}_T$, the *estimation error*, i.e., $\|\boldsymbol{w}_{T+1} - \boldsymbol{w}^*\|$ is bounded as the following:

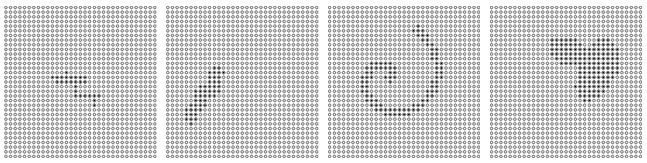
$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}^*\|_2^2 \leq 2\Big(D^2 + \frac{L^2}{\gamma^2} - \frac{1}{\gamma\sqrt{T}} R(T, \mathcal{M}(\mathbb{M}))\Big). \tag{15}$$

However, the regret bound (14) and *estimation error* (15) are under the assumption that the constraint set $\mathcal{M}(\mathbb{M})$ is convex. For GraphDA, an approximated algorithm, it is difficult to establish a sublinear regret bound. The reasons are two-fold: 1) Due to the non-convexity of $\mathcal{M}(\mathbb{M})$, it is possible that GraphDA converges to a local minimal, so the regret will potentially be non-sublinear; 2) The solution of model projection is approximated, making the regret analysis harder. Although recent work [14, 18] shows that it is still possible to obtain a *local-regret* bound when the objective function is non-convex, it is different from our case since we assume the objective function convex subject to a non-convex constraint. We leave the theoretical regret bound analysis of GraphDA an open problem.

5 EXPERIMENTS

To corroborate our algorithm, we conduct extensive experiments, comparing GraphDA with some popular baseline methods. Note DA-IHT derived from Theorem 2 is treated as a baseline method. We aim to answer the following questions:

- ***Question Q1*:** Can GraphDA achieve better classification performance compared with baseline methods?
- ***Question Q2*:** Can GraphDA learn an stronger interpretative model through capturing more meaningful graph-structure features compared with baseline methods?



(a) *Graph01* (b) *Graph02* (c) *Graph03* (d) *Graph04*

Figure 2: Four benchmark graphs from [2]

5.1 Datasets and evaluation metrics

Datasets. We use the following three publicly available graph datasets: 1) **Benchmark Dataset** [2]. Four benchmark graphs [2] are shown in Figure 2. The four subgraphs are embedded into 33×33 graphs with 26, 46, 92, and 132 nodes respectively. Each graph has $p = 1,089$ nodes and $m = 2,112$ edges with unit weight 1.0. We use the Benchmark dataset to learn an online graph logistic regression model; 2) **MNIST Dataset** [31]. This popular hand-writing dataset is used to test GraphDA on online graph sparse linear regression. It contains ten classes of handwritten digits from 0 to 9. We randomly choose each digit as our target graph. Each pixel stands for a node. There exists an edge if two nodes are neighbors. We set the weights

to 1.0 for edges; 3) **KEGG Pathway Dataset** [32]. The Kyoto Encyclopedia of Genes and Genomes (KEGG) dataset contains 5,372 genes. These genes (nodes) form a connected graph with 78,545 edges. The edge weights stand for correlations between two genes. We use KEGG to detect a related pathway.

Evaluation metrics. We have two categories of metrics to answer **Question Q1** and **Question Q2** respectively. To measure classification performance of $\boldsymbol{w}_t$ or $\bar{\boldsymbol{w}}_t$[6], we use classification Accuracy(*Acc*), the Area Under Curve (*AUC*) [16], and the number of Misclassified samples (*Miss*). To evaluate feature-level performance (interpretability), we use Precision (*Pre*), Recall (*Rec*), F1-score (*F1*), and Nonzero Ratio (*NR*). To clarify, given any optimal $\boldsymbol{w}^* \in \mathbb{R}^p$ and learned model $\boldsymbol{w}_t$, *Pre*, *Rec*, *F1*, and *NR* are defined as follows:

$$Pre_{\boldsymbol{w}_t} = \frac{|\text{supp}(\boldsymbol{w}^*) \cap \text{supp}(\boldsymbol{w}_t)|}{|\text{supp}(\boldsymbol{w}_t)|}, \; Rec_{\boldsymbol{w}_t} = \frac{|\text{supp}(\boldsymbol{w}^*) \cap \text{supp}(\boldsymbol{w}_t)|}{|\text{supp}(\boldsymbol{w}^*)|}$$
$$F1_{\boldsymbol{w}_t} = \frac{2|\text{supp}(\boldsymbol{w}^*) \cap \text{supp}(\boldsymbol{w}_t)|}{|\text{supp}(\boldsymbol{w}^*)| + |\text{supp}(\boldsymbol{w}_t)|}, \; \text{NR}_{\boldsymbol{w}} = \frac{|\text{supp}(\boldsymbol{w})|}{p}. \tag{16}$$

5.2 Baseline methods

We consider the following eight baseline methods: 1) ℓ_1-RDA [40]. We use the enhanced Regularized Dual-Averaging (ℓ_1-RDA) method in Algorithm 2 of [40]; 2) DA-GL [41]. Online Dual Averaging Group Lasso (DA-GL) is the dual averaging method with group Lasso; 3) DA-SGL [41]. It also uses dual averaging, but with sparse group Lasso as the regularization; 4) AdaGrad [11]. The adaptive gradient with ℓ_1 regularization is different from ℓ_1-RDA [40]. AdaGrad yields a dedicated step size for each feature inversely. In order to capture the sparsity, we use its ℓ_1 norm-based method for comparison; 5) Adam [28]. Since there is no sparsity regularization in Adam, it generates totally dense models. We use its online version[7] to compare with these sparse methods; 6) StoIHT [35]. We use this method with block size 1, which can be treated as online learning setting; 7) DA-IHT, derived from Theorem 2 in this paper. We use it to compare with GraphDA, which has graph-structure constraint; 8) GraphStoIHT. We apply the head and tail projection to StoIHT to generate GraphStoIHT.

Online Setting. All methods are completely online, i.e., all learning algorithms receive a single sample per-iteration. Due to the space limit, the parameters of all baseline methods including GraphDA are in Appendix A. All numerical results are averaged from 20 trials. The following three sections report and discuss the experimental results on each dataset to answer **Q1** and **Q2**.

5.3 Results from Benchmark dataset

Given the training dataset $\{\boldsymbol{x}_i, y_i\}_{i=1}^t$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{\pm 1\}$ on the fly, the online graph sparse logistic regression is to minimize the regret $R(\boldsymbol{t}, \mathcal{M}(\mathbb{M}))$ where $f_t(\boldsymbol{w}_t)$ is a logistic loss defined as

$$f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\}) = \log(1 + \exp(-y_t \cdot \boldsymbol{w}_t^\top \boldsymbol{x}_t)).$$

We simulate the negative and positive samples as done in [2]. $y_t = -1$ stands for no signals or events ("business-as-usual"). $y_t = +1$ means a certain event happens such as disease outbreak/computer virus hidden in current data sample $\boldsymbol{x}_t$, and feature values in subgraphs are abnormally higher. That is, if $y_t = -1$, then $x_{v_i} \sim \mathcal{N}(0,1) \; \forall \, v_i \in \mathbb{V}$; and if $y_t = +1$, then,

$$(x_i)_{v_i} \sim \begin{cases} \mathcal{N}(\mu, 1) & v_i \in F \\ \mathcal{N}(0,1) & v_i \neq F, \end{cases} \tag{17}$$

where F stands for the nodes of a specific subgraph showcased in Figure 2. Then each entry $(\boldsymbol{w}^*)_i$ is μ if $i \in F$; otherwise 0. We first fix $\mu = 0.3$ and then generate validating, training, and testing samples, each with 400 samples. All methods stop at $t = 400$ after seeing all training samples once. Parameters are tuned on 400 validating samples. We test $\boldsymbol{w}_t$ and $\bar{\boldsymbol{w}}_t$ on testing samples.

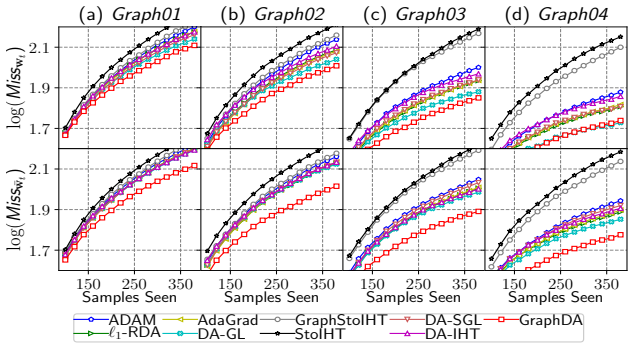


Figure 3: The logarithm of the number of misclassified samples as a function of samples seen

Classification Performance on fixed μ. Table 1 shows that four all three indicators of classification performance, GraphDA scores higher than the other baseline methods. Specifically, it has the highest *Acc* (0.749, 0.739) and *AUC* (0.749, 0.739) with respect to $\boldsymbol{w}_t$ and $\bar{\boldsymbol{w}}_t$. The averaged number of misclassified samples (*Miss*) is lower (133.45, 136.20), than other methods by quite a large margin. Figure 3 further shows that the number of misclassified samples of GraphDA keeps the lowest during the entire online learning course for all four graphs [2].

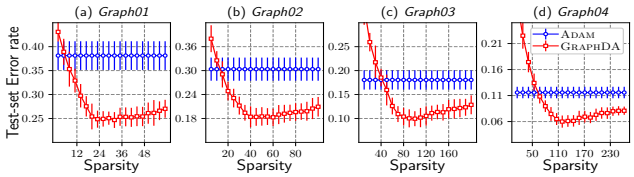


Figure 5: Test dataset error rates as a function of sparsity s

The sparsity s is an important parameter for GraphDA. We explore how s affects the test error rate[8]. We compare the error rate of GraphDA with that of the non-sparse method Adam. Figure 5 clearly demonstrates that GraphDA has the least test error rate corresponding to the true model sparsity (26, 46, 92, 132 for these four subgraphs). When s reaches the true sparsity (26, 46, 92, 132 respectively), the testing error rate of GraphDA is the minimum.

Classification Performance on different t and μ. We explore how different numbers of training sample and different μ affects the performance of each method similarly done in [41]. First, we choose t from set $\{100, 200, 300, \ldots, 1000\}$, and tune the model based on classification accuracy. Results in Figure 6 show that when the number of training sample increases, the classification accuracy of all methods are increasing accordingly, but GraphDA enjoys

[6] For the comparison, we also evaluate the averaged decision $\bar{\boldsymbol{w}}_t$ similar as done in [40].
[7] One can find more details of the online version in Section 4 of [28].
[8] The test error rate is calculated as $1 - Acc_{\boldsymbol{w}_t}$ similar as done in [11].

Table 1: Classification performance on *Graph01* of Benchmark dataset

Method	$Pre_{\boldsymbol{w}_t}$ ±std	$Rec_{\boldsymbol{w}_t}$ ±std	$F1_{\boldsymbol{w}_t}$ ±std	$AUC_{\boldsymbol{w}_t,\bar{\boldsymbol{w}}_t}$	$Acc_{\boldsymbol{w}_t,\bar{\boldsymbol{w}}_t}$	$Miss_{\boldsymbol{w}_t,\bar{\boldsymbol{w}}_t}$	$NR_{\boldsymbol{w}_t,\bar{\boldsymbol{w}}_t}$
Adam	0.024±0.00	**1.000**±0.00	0.047±0.00	(0.618, 0.603)	(0.619, 0.603)	(166.35, 173.10)	(100.0%, 100.0%)
ℓ_1-RDA	0.267±0.11	0.863±0.09	0.389±0.13	(0.693, 0.672)	(0.694, 0.673)	(155.30, 166.05)	(11.58%, 83.60%)
AdaGrad	0.256±0.11	0.877±0.09	0.379±0.13	(0.696, 0.636)	(0.696, 0.637)	(156.00, 166.00)	(11.33%, 100.0%)
DA-GL	0.176±0.11	0.967±0.04	0.283±0.12	(0.735, 0.666)	(0.735, 0.667)	(142.90, 162.20)	(15.99%, 100.0%)
DA-SGL	0.523±0.40	0.854±0.14	0.506±0.35	(0.699, 0.647)	(0.699, 0.647)	(151.00, 165.50)	(25.54%, 100.0%)
StoIHT	0.057±0.04	0.150±0.08	0.072±0.03	(0.552, 0.523)	(0.553, 0.523)	(194.55, 195.25)	(7.79%, 40.62%)
GraphStoIHT	0.151±0.12	0.356±0.16	0.194±0.12	(0.603, 0.570)	(0.602, 0.570)	(174.65, 181.40)	(7.84%, **22.06**%)
DA-IHT	0.507±0.20	0.744±0.12	0.566±0.11	(0.697, 0.666)	(0.697, 0.666)	(155.65, 162.85)	(4.35%, 39.50%)
GraphDA	**0.869**±0.13	0.906±0.04	**0.880**±0.08	(**0.749**, **0.739**)	(**0.749**, **0.739**)	(**133.45**, **136.20**)	(**2.56**%, 32.12%)

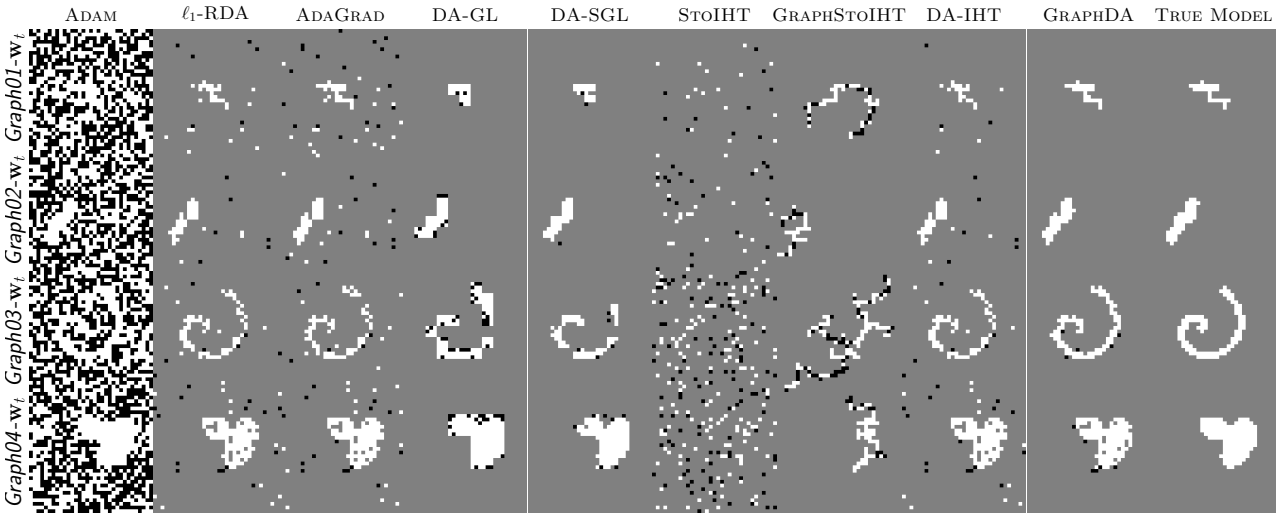


Figure 4: The learned models $\boldsymbol{w}_t$ of four benchmark graphs. These models are from the first trial of all 20 trials. For each pixel i, black stands for $(\boldsymbol{w}_t)_i < 0$, gray $(\boldsymbol{w}_t)_i = 0$, and white $(\boldsymbol{w}_t)_i > 0$.

the highest classification accuracy on both $\boldsymbol{w}_t$ and $\bar{\boldsymbol{w}}_t$. Second, we choose μ from the set $\{0.1, 0.2, \ldots, 1.0\}$ and fix $t = 400$. As is reported in Figure 7, when μ is small (a harder classification task), all methods achieve lower accuracy; when μ is large (an easier task), all methods can obtain very high accuracy except StoIHT and GraphIHT. Again, Acc of GraphDA is the highest.

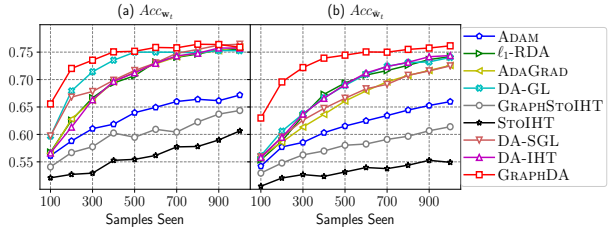


Figure 6: The classification accuracy on testing dataset as a function of number of training samples seen

Model interpretability. Table 1 shows that three out of the four indicators of feature-level performance for GraphDA score higher than for the other baseline methods. To be more specific, our method has the highest *F1*, 0.880, exceeding other methods by a large margin, which means that graph-structured information does help improve its performance. It also testifies that the head/tail projection during the online learning process does help capture more meaningful features than others. The nonzero ratio (*NR*) of $\boldsymbol{w}_t$ and $\bar{\boldsymbol{w}}_t$ is the least. The learned $\boldsymbol{w}_t$ in Figure 4 shows GraphDA successfully captures these subgraphs in $\boldsymbol{w}_t$, which are the closest to true models in terms of shapes and values (white colored pixels). Adam learns a totally dense model, and hence has worse performance. DA-IHT and ℓ_1-RDA obtain very similar performance, probably because both of them use the dual averaging techniques. The results of StoIHT and GraphStoIHT testify that the online PGD-based methods hardly learn an interpretable model. In brief, our algorithm exploits the least number of features to learn the best model among all of the methods.

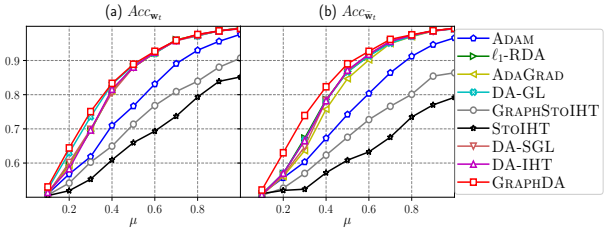


Figure 7: The classification accuracy on testing dataset as a function of μ.

5.4 Results from MNIST dataset

The goal of online graph sparse linear regression is to minimize the regret where each $f_t(\boldsymbol{w}_t)$ is the least square loss defined as

$$f_t(\boldsymbol{w}_t, \{\boldsymbol{x}_t, y_t\}) = (y_t - \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle)^2. \tag{18}$$

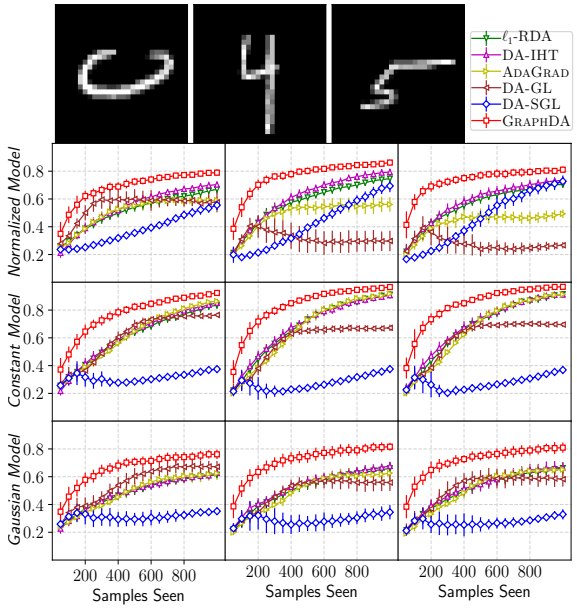


Figure 8: Three handwritten digits 0, 4 and 5 (top row) and the *F1* score as a function of samples seen (2nd to 4th row).

On this dataset, we use the least square loss as the objective function. The experiment is to compare the feature-level *F1* score of different algorithms. We generate 1,400 data samples by using the following linear relation:

$$y_t = \boldsymbol{x}_t^\top \boldsymbol{w}^*,$$

where $\boldsymbol{x}_t \in \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. We use three different strategies to obtain $\boldsymbol{w}^*$. The first one is to directly use the sparse images and then normalize them to the range $[0.0, 1.0]$, which we call *Normalized Model*. The second is to generate $\boldsymbol{w}^*$ by letting all non-zeros be 1.0, which is called *Constant Model*. The third is to generate the nonzero nodes by using Gaussian distribution $(\boldsymbol{w}^*)_i \sim \mathcal{N}(0, 1)$ independently, which is *Gaussian Model*. Again, our dataset is partitioned into three parts: training, validating and testing samples. We increase the number of training samples n from $\{50, 100, \ldots, 1000\}$ and then use 200 samples as validating dataset to tune the model. For all the eight online learning algorithms, we pass each training sample once and stop training when all training samples are used. The results shown in Figure 8 are generated from the 200 testing samples.

Adam, StoIHT and GraphStoIHT are excluded from comparison because of their inferior performance. From Figure 8, we can observe that when the training samples increase, the *F1* score of all methods is increasing correspondingly. But the *F1* score values of GraphDA in *Normalized Model*, *Constant Model*, and *Gaussian Model* are the highest among the six methods.

5.5 Results from KEGG dataset

To demonstrate that GraphDA can capture more meaningful features during online learning process, we test it on a real-world protein-protein iteration (PPI) network in [32][9]. This online learning scenario could be realistic since the training samples can be collected on the fly. More Details of the dataset including the data preprocessing are in Appendix B.3. We explore a specific gene pathway, HSA05213, related with endometrial cancer[10]. Due to the lack of true labels and ground truth features (genes), we directly use the two data generation strategies in [32], namely *Strategy 1* (corresponding to a hard case) and *Strategy 2* (corresponding to an easy case). After the data generation, we have 50 ground truth features. The number of positive and negative samples are both 100. The goal is to find out how many endometrial cancer-related genes are learned from different algorithms as done by [32]. All algorithms stop when they have seen 200 training samples.

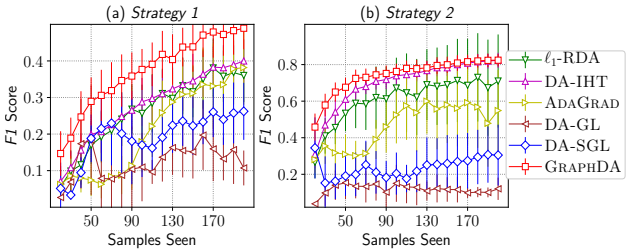


Figure 9: Node level *F1* score as a function of the number of training samples have seen.

We report the feature *F1* score in Figure 9. GraphDA outperforms the other baseline methods in terms of both two strategies, with *F1* score about 0.5 for *Strategy 1* and about 0.9 for *Strategy 2*, higher than the rest methods. Interestingly, DA-OL and DA-SOL achieve better results only between 60 and 70 training samples and then become worse between 70 and 100. A possible explanation is that the learned model selected by the tuned parameters is not steady when the number of training samples seen is small. In addition to a better *F1* score, another strength of GraphDA and DA-IHT is that the standard deviation of *F1* score is smaller than other convex-based methods, including ℓ_1-RDA, DA-GL, and DA-SGL.

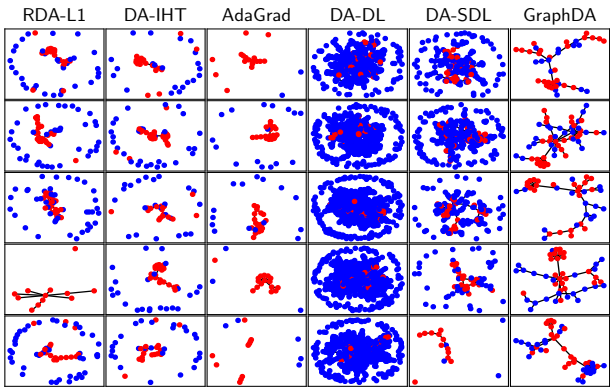


Figure 10: HSA05213 pathway detected by different methods. The red nodes are the genes in HSA05213 while blue nodes are the genes not in HSA05213. Results of each row are from a specific trial (from trial 1 to trial 5). Each column shows the results found by a specific method.

In Figure 10, we show the identified genes by different methods. Clearly, GraphDA can find more meaningful genes, indicated by less blue nodes (genes found but not in HSA05213) and more

[9]It was originally provided in KEGG [27]

[10]Details of pathway HSA05213(50 genes) can be found in https://www.genome.jp/dbget-bin/www_bget?hsa05213

red nodes (genes found and in HSA05213). However, all the other five baseline methods have many isolated nodes (not connected to cancer-related genes).

6 CONCLUSION AND FUTURE WORK

In this paper, we propose a dual averaging-based method, GraphDA, for online graph-structured sparsity constraint problems. We prove that the minimization problem in the dual averaging step can be formulated as two equivalent optimization problems. By projecting the dual vector and primal variables onto lower dimensional subspaces, GraphDA can capture graph-structure information more effectively. Experimental evaluation on one benchmark dataset and two real-world graph datasets shows that GraphDA achieves better classification performance and stronger interpretability compared with the baseline methods so as to answer the two questions raised at the beginning of the experiment section. It remains interesting if one can prove that both the exact and approximated projections have non-regret bound under some proper assumption, and if one can explore learning a model under the setting that true features are time evolving [24].

7 ACKNOWLEDGEMENTS

The work of Yiming Ying is supported by the National Science Foundation (NSF) under Grant No #1816227. The work of Baojian Zhou and Feng Chen is supported by the NSF under Grant No #1815696 and #1750911.

REFERENCES

[1] Cem Aksoylar, Lorenzo Orecchia, and Venkatesh Saligrama. 2017. Connected Subgraph Detection with Mirror Descent on SDPs. In *ICML*. PMLR, 51–59.
[2] Ery Arias-Castro, Emmanuel J Candes, Arnaud Durand, et al. 2011. Detection of an anomalous cluster in a network. *The Annals of Statistics* 39, 1 (2011), 278–304.
[3] Sohail Bahmani, Bhiksha Raj, and Petros T Boufounos. 2013. Greedy sparsity-constrained optimization. *JMLR* 14, Mar (2013), 807–841.
[4] Richard G Baraniuk, Volkan Cevher, Marco F Duarte, and Chinmay Hegde. 2010. Model-based compressive sensing. *IEEE Transactions on Information Theory* 56, 4 (2010), 1982–2001.
[5] Léon Bottou. 1998. Online learning and stochastic approximations. *On-line learning in neural networks* 17, 9 (1998), 142.
[6] Léon Bottou and Yann L Cun. 2004. Large scale online learning. In *Advances in neural information processing systems*, Vol. 16. MIT Press, 217–224.
[7] Feng Chen and Baojian Zhou. 2016. A generalized matching pursuit approach for graph-structured sparsity. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. AAAI Press, 1389–1395.
[8] Lin Chen, Hamed Hassani, and Amin Karbasi. 2018. Online Continuous Submodular Maximization. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1896–1905.
[9] Han-Yu Chuang, Eunjung Lee, Yu-Tsueng Liu, Doheon Lee, and Trey Ideker. 2007. Network-based classification of breast cancer metastasis. *Molecular systems biology* 3, 1 (2007), 140.
[10] Moez Draief, Ayalvadi Ganesh, and Laurent Massoulié. 2006. Thresholds for virus spread on networks. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*. ACM, 51.
[11] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12, Jul (2011), 2121–2159.
[12] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. Efficient projections onto the l1-ball for learning in high dimensions. In *ICML*. ACM, 272–279.
[13] John Duchi and Yoram Singer. 2009. Efficient online and batch learning using forward backward splitting. *JMLR* 10, Dec (2009), 2899–2934.
[14] Xiand Gao, Xiaobo Li, and Shuzhong Zhang. 2018. Online Learning with Non-Convex Losses and Non-Stationary Regret. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 235–243.
[15] Alon Gonen and Elad Hazan. 2018. Learning in Non-convex Games with an Optimization Oracle. *arXiv preprint arXiv:1810.07362* (2018).
[16] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36.
[17] Elad Hazan et al. 2016. Introduction to online convex optimization. *Foundations and Trends® in Optimization* 2, 3-4 (2016), 157–325.
[18] Elad Hazan, Karan Singh, and Cyril Zhang. 2017. Efficient Regret Minimization in Non-Convex Games. In *ICML*. PMLR, 1433–1441.
[19] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 2014. A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem. In *Workshop of the 11th DIMACS Implementation Challenge*.
[20] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 2014. A fast approximation algorithm for tree-sparse recovery. In *Information Theory (ISIT), 2014 IEEE International Symposium on*. IEEE, 1842–1846.
[21] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 2015. Approximation algorithms for model-based compressive sensing. *IEEE Transactions on Information Theory* 61, 9 (2015), 5129–5147.
[22] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 2015. A nearly-linear time framework for graph-structured sparsity. In *ICML*. PMLR, 928–937.
[23] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 2016. Fast recovery from a union of subspaces. In *NIPS*. 4394–4402.
[24] Bo-Jian Hou, Lijun Zhang, and Zhi-Hua Zhou. 2017. Learning with Feature Evolvable Streams. In *NIPS*. 1416–1426.
[25] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. 2009. Group lasso with overlap and graph lasso. In *ICML*. ACM, PMLR, 433–440.
[26] David S Johnson, Maria Minkoff, and Steven Phillips. 2000. The prize collecting Steiner tree problem: theory and practice. In *SODA*. Society for Industrial and Applied Mathematics, 760–769.
[27] Minoru Kanehisa, Miho Furumichi, Mao Tanabe, Yoko Sato, and Kanae Morishima. 2016. KEGG: new perspectives on genomes, pathways, diseases and drugs. *Nucleic acids research* 45, D1 (2016), D353–D361.
[28] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[29] Jean Lafond, Hoi-To Wai, and Eric Moulines. 2015. On the online Frank-Wolfe algorithms for convex and non-convex optimizations. *arXiv:1510.01171* (2015).
[30] John Langford, Lihong Li, and Tong Zhang. 2009. Sparse online learning via truncated gradient. *JMLR* 10, Mar (2009), 777–801.
[31] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).
[32] Alexander LeNail, Ludwig Schmidt, Johnathan Li, Tobias Ehrenberger, Karen Sachs, Stefanie Jegelka, and Ernest Fraenkel. 2017. Graph-Sparse Logistic Regression. *arXiv preprint arXiv:1712.05510* (2017).
[33] Taibo Li, Rasmus Wernersson, Rasmus B Hansen, Heiko Horn, Johnathan Mercer, Greg Slodkowicz, Christopher T Workman, Olga Rigina, Kristoffer Rapacki, Hans H Stærfeldt, et al. 2017. A scored human protein–protein interaction network to catalyze genomic interpretation. *Nature methods* 14, 1 (2017), 61.
[34] Yurii Nesterov. 2009. Primal-dual subgradient methods for convex problems. *Mathematical programming* 120, 1 (2009), 221–259.
[35] Nam Nguyen, Deanna Needell, and Tina Woolf. 2017. Linear convergence of stochastic iterative greedy algorithms with sparse constraints. *IEEE Transactions on Information Theory* 63, 11 (2017), 6869–6895.
[36] Jing Qian, Venkatesh Saligrama, and Yuting Chen. 2014. Connected Sub-graph Detection.. In *AISTATS*, Vol. 14. 22–25.
[37] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. 2014. Event detection in activity networks. In *KDD*. ACM, 1176–1185.
[38] Shai Shalev-Shwartz et al. 2012. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning* 4, 2 (2012), 107–194.
[39] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.
[40] Lin Xiao. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *JMLR* 11, Oct (2010), 2543–2596.
[41] Haiqin Yang, Zenglin Xu, Irwin King, and Michael R Lyu. 2010. Online learning for group lasso. In *ICML*. PMLR, 1191–1198.
[42] Lin Yang, Lei Deng, Mohammad H Hajiesmaili, Cheng Tan, and Wing Shing Wong. 2018. An optimal algorithm for online non-convex learning. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 2 (2018), 25.
[43] Yiming Ying and Massimiliano Pontil. 2008. Online gradient descent learning algorithms. *Foundations of Computational Mathematics* 8, 5 (2008), 561–596.
[44] Yiming Ying and D-X Zhou. 2006. Online regularized classification algorithms. *IEEE Transactions on Information Theory* 52, 11 (2006), 4775–4788.
[45] Yiming Ying and Ding-Xuan Zhou. 2017. Unregularized online learning algorithms with general loss functions. *Applied and Computational Harmonic Analysis* 42, 2 (2017), 224–244.
[46] Xiaotong Yuan, Ping Li, and Tong Zhang. 2014. Gradient hard thresholding pursuit for sparsity-constrained optimization. In *ICML*. PMLR, 127–135.
[47] Pan Zhou, Xiaotong Yuan, and Jiashi Feng. 2018. Efficient Stochastic Gradient Hard Thresholding. In *NIPS*. Curran Associates, Inc., 1985–1994.
[48] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*. PMLR, 928–936.

A REPRODUCIBILITY

A.1 Implementation details

All experiments are tested on a server of Intel Xeon(R) 2.40GHZ E5-2680 with 251GB of RAM. The code is written in Python2.7 and C language with the standard C11. The implementation of the head and tail projection follows the original implementation[11]. We present the pseudo code in Algorithm 2 below. The two projections are essentially two binary search algorithms. Each iteration of the binary search executes the Prize Collecting Steiner Tree (PCST) algorithm [26] on the target graph. Both projections have two main parameters: a lower bound sparsity s_l and an upper bound sparsity s_h. In all of the experiments, two sparsity parameters have been set to $s_l = p/2$ and $s_h = s_l * (1 + \omega)$ for the head projection, where ω is the tolerance parameter set to 0.1. For the tail projection, we set $s_l = s$ and $s_h = s_l * (1 + \omega)$. The binary search algorithm terminates when it reaches $max\_iter = 20$ maximum iterations. Line 7 of Algorithm 2 is the PCST algorithm proposed in [19]. We use a non-root version and Goemans-Williamson pruning method to prune the final forest.

Algorithm 2 Head/Tail Projection (P($\boldsymbol{w}, \mathbb{M}$)) [22]

1: **Input**: $\boldsymbol{w}$, max_iter, $\mathbb{M} = (\mathbb{G}(\mathbb{V}, \mathbb{E}, \boldsymbol{c}), s_l, s_h, g)$
2: $\boldsymbol{\pi} = \boldsymbol{w} \cdot \boldsymbol{w}$ // vector dot product, i.e., $\pi_i = w_i * w_i$
3: $\lambda_l = 0, \lambda_h = \max\{\pi_1, \pi_2, \ldots, \pi_p\}, \lambda_m = 0, t = 0$
4: **repeat**
5: $\lambda_m = (\lambda_l + \lambda_h)/2$
6: $\boldsymbol{c}_m = \lambda_m \cdot \boldsymbol{c}$ // scale dot product, i.e., $(\boldsymbol{c}_m)_i = \lambda_m * c_i$
7: $\mathcal{F}$ = PCST($\mathbb{G}(\mathbb{V}, \mathbb{E}, \boldsymbol{c}_m), \boldsymbol{\pi}, g$)
8: **if** $s_l < |\mathcal{F}| < s_h$ **then**
9: **return** $\boldsymbol{w}_{\mathcal{F}}$
10: **end if**
11: **if** $|\mathcal{F}| > s_h$ **then**
12: $\lambda_l = \lambda_m$
13: **else**
14: $\lambda_h = \lambda_m$
15: **end if**
16: $t = t + 1$
17: **until** t > max_iter
18: $\boldsymbol{c}_h = \lambda_h \cdot \boldsymbol{c}$
19: $\mathcal{F}$ = PCST($\mathbb{G}(\mathbb{V}, \mathbb{E}, \boldsymbol{c}_h), \boldsymbol{\pi}, g$)
20: **return** $\boldsymbol{w}_{\mathcal{F}}$

A.2 Parameter tuning

Initial parameters $\mathbf{w}_0$ of all baseline methods and proposed algorithms are zero vectors $\mathbf{w}_0 = \mathbf{0}$, which means we train all methods starting from a zero point. We list all related methods and their corresponding parameter settings below. (1) ℓ_1-RDA is the enhanced version provided in Algorithm 2 of [40]. There are three parameters: The ℓ_1-regularization parameter λ is chosen from {0.0001, 0.0005, 0.001, 0.005, 0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 1.0, 3.0, 5.0, 10.0} which is a superset used in [40]. The parameter γ to control the learning rate is chosen from {1.0, 5.0, 10.0, 50.0, 100.0, 500.0, 1000.0, 5000.0, 10000.0 }, and the sparsity-enhancing parameter ρ is chosen from {0.0, 0.00001, 0.000005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0}, where 0.0 is for the basic regularization. All the three parameter sets are supersets used in [40]. (2) Adam. We directly use the parameters $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ provided in [28]. For the magnitude of steps in parameter space α, we choose it from {0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5 }. (3) DA-GL/SGL have two main parameters, λ to control the sparsity and γ to control the learning rate. We choose 3×3 grids as groups for Benchmark dataset and choose 2×2 grids for MNIST dataset. (4) DA-SGL has an additional parameter γ_g, which is set to 1.0 for all groups as done in [41]. For each group i, there exists an additional parameter r_i for DA-SGL. We set it as default value $r_i = 1$ as recommended by the authors. (5) AdaGrad has two main parameters, λ to control sparsity and η to control the learning rate, which is from {0.0001 , 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0, 10.0, 50.0, 100.0, 500.0, 1000.0, 5000.0}. (6) StoIHT has two parameters: sparsity s from {5, 10, 15, 20, 25, 26, 30, 35, 40, 45, 46, 50, 55, 60, 65, 70, 75, 80, 85, 90, 92, 95, 100, 105, 110, 115, 120, 125, 130, 132, 135, 140, 145, 150}, and γ to control the learning rate. (7) GraphStoIHT shares the same parameter settings (sparsity s and γ) as GraphDA. The block size of GraphStoIHT and StoIHT are set to 1. (8) GraphDA has parameters γ and s.

[11] The two projections were originally implemented in C++, which are available at: https://github.com/ludwigschmidt/cluster_approx. We implement them by using C language. Taking the advantage of the continuous memory of arrays in C, our code is faster than original one.

B MORE EXPERIMENTAL RESULTS

B.1 More results from Benchmark dataset

We present the results of *Graph02*, *Graph03* and *Graph04* in Table 2, 3, 4, respectively. Basically, we show the classification performance (*Acc*, *Miss*, *AUC*) and feature-level performance (*Pre*, *Rec*, *F1*, *NR*). The size of validating and testing dataset are both 400. All results are averaged from 20 trials of experiment.

Table 2: Performance of *Graph02*

Method	$Pre_{\mathbf{w}_t}$	$Rec_{\mathbf{w}_t}$	$F1_{\mathbf{w}_t}$	$AUC_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$Acc_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$Miss_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$NR_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$
Adam	0.042	**1.000**	0.081	(0.697, 0.663)	(0.696, 0.663)	(144, 151)	(100.0%, 100.0%)
ℓ_1-RDA	0.371	0.876	0.494	(0.772, 0.732)	(0.772, 0.731)	(127, 140)	(13.31%, 96.47%)
AdaGrad	0.342	0.888	0.470	(0.771, 0.711)	(0.771, 0.711)	(125, 141)	(14.43%, 100.0%)
DA-GL	0.270	0.976	0.415	(0.809, 0.755)	(0.809, 0.755)	(114, 138)	(17.07%, 100.0%)
DA-SGL	0.283	0.948	0.314	(0.777, 0.738)	(0.777, 0.737)	(123, 141)	(45.42%, 100.0%)
StoIHT	0.102	0.217	0.132	(0.586, 0.557)	(0.586, 0.557)	(171, 179)	(9.48%, 45.60%)
GraphStoIHT	0.279	0.355	0.287	(0.669, 0.620)	(0.669, 0.620)	(150, 158)	(7.31%, **19.29**%)
DA-IHT	0.679	0.741	0.694	(0.776, 0.733)	(0.776, 0.733)	(132, 141)	(4.86%, 42.86%)
GraphDA	**0.855**	0.870	**0.850**	**(0.811, 0.799)**	**(0.811, 0.799)**	**(106, 107)**	(**4.55**%, 43.89%)

Table 3: Performance of *Graph03*

Method	$Pre_{\mathbf{w}_t}$	$Rec_{\mathbf{w}_t}$	$F1_{\mathbf{w}_t}$	$AUC_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$Acc_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$Miss_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$NR_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$
Adam	0.084	**1.000**	0.156	(0.820, 0.789)	(0.820, 0.788)	(104, 116)	(100.0%, 100.0%)
ℓ_1-RDA	0.340	0.940	0.488	(0.870, 0.833)	(0.869, 0.833)	(88, 104)	(26.15%, 99.51%)
AdaGrad	0.318	0.942	0.462	(0.872, 0.825)	(0.872, 0.824)	(88, 106)	(29.21%, 100.0%)
DA-GL	0.289	0.990	0.443	(0.894, 0.853)	(0.894, 0.853)	(78, 100)	(32.07%, 100.0%)
DA-SGL	0.166	0.990	0.283	(0.883, 0.829)	(0.883, 0.828)	(89, 111)	(52.19%, 100.0%)
StoIHT	0.156	0.208	0.175	(0.635, 0.593)	(0.634, 0.592)	(162, 173)	(11.62%, 50.60%)
GraphStoIHT	0.276	0.223	0.217	(0.666, 0.640)	(0.667, 0.640)	(154, 163)	(8.93%, **23.20**%)
DA-IHT	0.716	0.782	0.734	(0.865, 0.834)	(0.865, 0.834)	(95, 103)	(9.59%, 63.20%)
GraphDA	**0.856**	0.881	**0.864**	**(0.898, 0.885)**	**(0.897, 0.885)**	**(72, 80)**	(**8.82**%, 63.14%)

B.2 More results from MNIST Dataset

We show the results on image id {1, 2, 3, 6, 7, 8, 9}. To make the task more challenging, these 7 images are the sparsest images (with

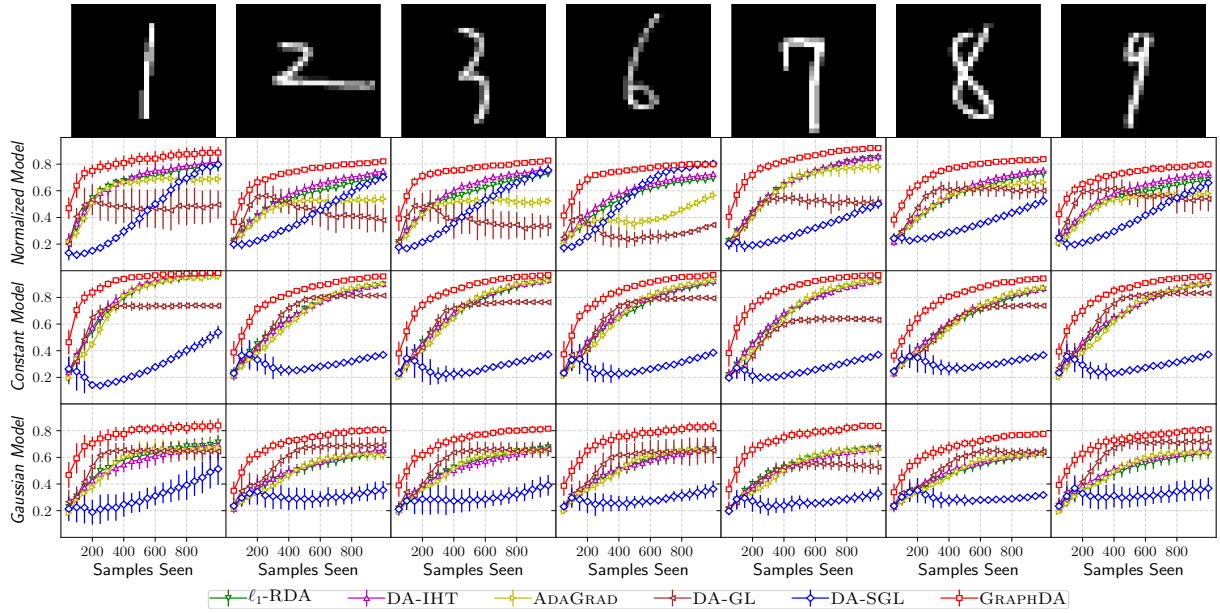


Figure 11: Seven handwritten digits 1, 2, 3, 6, 7, 8 and 9 (top row) and the *F1* score as a function of samples seen (2nd to 4th row)

Table 4: Performance of *Graph04*

Method	$Pre_{\mathbf{w}_t}$	$Rec_{\mathbf{w}_t}$	$F1_{\mathbf{w}_t}$	$AUC_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$Acc_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$Miss_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$	$NR_{\mathbf{w}_t,\bar{\mathbf{w}}_t}$
Adam	0.121	**1.000**	0.216	(0.884, 0.858)	(0.884, 0.858)	(77, 90)	(100.0%, 100.0%)
ℓ_1-RDA	0.361	0.961	0.513	(0.917, 0.896)	(0.917, 0.896)	(66, 79)	(36.19%, 99.21%)
AdaGrad	0.376	0.961	0.528	(0.919, 0.889)	(0.919, 0.889)	(67, 81)	(35.43%, 100.0%)
DA-GL	0.476	0.994	0.640	(**0.942**, 0.918)	(**0.941**, 0.918)	(**54**, 73)	(26.03%, 100.0%)
DA-SGL	0.238	0.988	0.379	(0.931, 0.894)	(0.931, 0.894)	(65, 85)	(53.31%, 100.0%)
StoIHT	0.207	0.203	0.204	(0.689, 0.639)	(0.689, 0.639)	(148, 160)	(11.86%, 47.88%)
GraphStoIHT	0.439	0.245	0.299	(0.743, 0.699)	(0.743, 0.699)	(131, 143)	(**7.77%**, **19.96%**)
DA-IHT	0.780	0.801	0.788	(0.919, 0.898)	(0.919, 0.899)	(74, 82)	(12.51%, 72.72%)
GraphDA	**0.931**	0.865	**0.895**	(0.939, **0.925**)	(0.939, **0.925**)	(56, **61**)	(11.30%, 72.80%)

the digits forming a connected component) selected from MNIST dataset. The sparsity parameter s of DA-IHT and GraphDA is chosen from $\{30, 32, \ldots, 100\}$ with step size 2. Figure 11 reports the results.

B.3 More results from KEGG Pathways

This PPI network contains a total of 229 pathways. Each pathway often involves a specific biological function, e.g. metabolism. We restrict our analysis on 225 pathways (by removing 4 empty pathways), which contains 5,374 genes with 78,545 edges. These genes form a connected graph. There exists an edge if two proteins (genes) physically interact with each other [33]. Weights of edges stand for the confidence of these interactions. There are 7,368 genes with null values. We sample these null values from $\mathcal{N}(0, 1)$.

Due to the inferior performance of Adam, StoIHT and GraphStoIHT, we exclude them from experimental evaluation. Notice that DA-GL and DA-SGL need groups as priors. However, the groups (pathways) of this PPI network have overlapping features. To remedy this issue, we simply replicate these overlapping features as suggested in [25, 41], and by doing so, the two baselines are still applicable. For these two non-convex methods, we choose the sparsity parameter from {40, 45, 50, 55, 60 }. We report the averaged results from 20 trials in Figure 12.

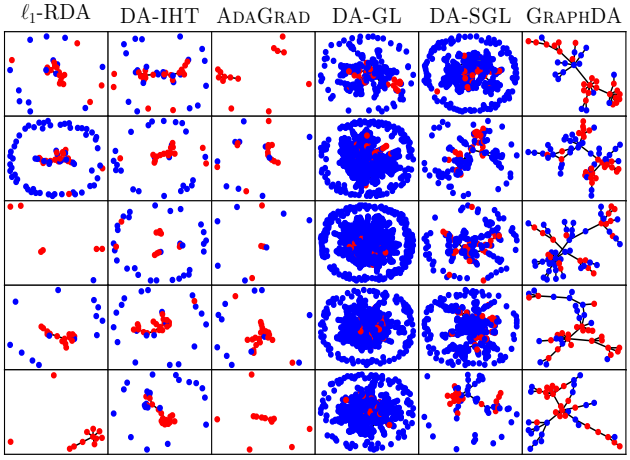


Figure 12: HSA05213 pathway detected by different methods. The red nodes are the genes in HSA05213 while blue nodes are the genes not in HSA05213. Results of each row are from a specific trial (from trial 6 to trial 10).