Problems in Learning under Limited Resources and Information

by

Yi Huang
B.Sc., Nankai University, Tianjin, China, 2004
M.S., Chinese Academy of Sciences, Beijing, China, 2008
M.S., Georgia Institute of Technology, Atlanta, US, 2010

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mathematics
in the Graduate College of the
University of Illinois at Chicago, 2017

Chicago, Illinois

To my parents

ACKNOWLEDGMENTS

I cannot express enough gratitude to Lev Reyzin, my academic advisor. Through the many projects I worked under him, Lev has shown me how to break a problem down to small manageable pieces and how we can add them up to something meaningful in the end. He taught me to always be positive and optimistic, not with blank words, but with hands-on experience of overcoming obstacles with persistency and patience, especially when progress was hopeless to make. I also want to thank him for encouraging and coaching me to demonstrate my research and build connections in the broader community of researchers, both of which presented no less challenge for me than research itself. Lev wrote in the acknowledgement of his own Ph.D. dissertation that he wanted to emulate the qualities he learned from his advisor in his career ahead. I would like to say that is a hope fulfilled from my experience as his student.

I also want to thank faculty from mathematical computer science at UIC, especially György Turán and Dhruv Mubayi, whose masterly teaching in computer science and combinatorics laid a solid foundation to my research, and to faculty from statistics at UIC, especially Jie Yang, Min Yang, and Cheng Ouyang for the many helpful discussions.

My fellow graduate students and collaborators Brain Powers, Ben Fish, and Mano Vikash Janardhanan had made research so stimulating and enjoyable. We have different styles towards research but we always complemented each other's approaches and worked in harmony. The companion of so many other spirited and friendly graduate students at the math department of UIC had made the overly stressful life of a graduate student mellow. I cannot fit all their names into this acknowledgement, but I thank them all wholeheartedly.

I was not the typical model student throughout my school years, which makes me even more grateful to my many teachers, especially Mr. Mingjie Gu from my high school, Dr. Matthew Baker and Dr. Igor Belegradek from Georgia Institute of Technology, and Dr. Benson Farb from the University of Chicago, who managed to see my potential through suboptimal academic record and, with many heartwarming compliments and constructive criticisms, made sure I set out to fulfill it.

Finally, I want to thank my family who has been an endless source of support and encouragement. I thank my husband whose unfailing love of math has always been an inspiration to me, especially during hard times in research. I owe an incredible debt to my parents who did everything they could to help me pursue my ambitions in science and always kept faith in me.

CONTRIBUTION OF AUTHORS

Chapter 3 represents the published manuscript (1): Yi Huang, Brian Powers, Lev Reyzin: Training-Time Optimization of a Budgeted Booster. *International Joint Conferences on Artificial Intelligence Organization (IJCAI)* 2015: 3583-3589.

Chapter 4 represents the published manuscript (2): Benjamin Fish, Yi Huang, Lev Reyzin: Recovering Social Networks by Observing Votes. *International Conference on Autonomous Agents & Multiagent Systems (AAMAS)* 2016: 376-384.

Chapter 5 represents the preprint manuscript (3): Yi Huang, Mano Vikash Janardhanan, Lev Reyzin: Network Construction with Ordered Constraints. *arXiv preprint: 1702.07292*

In all the papers list above, all the work, including the literature review, the formulation of definitions, lemmas, propositions, and theorems together with their proofs, the design of algorithms and their implementation, the making of figures and tables, the collection of data, was done jointly with the other coauthors.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The main theme of this thesis is to investigate how learning problems can be solved in the face of limited resources and with limited information to base inferences on. We study feature-efficient prediction when each feature comes with a cost and our goal is to construct a good predictor during training time with total cost not exceeding the given budget constraints. We also study complexity-theoretic properties of models for recovering social networks with knowledge only about how people in the network vote or how information propagates through the network.

In our study of feature-efficient prediction – a setting where features have costs and the learner is limited by a budget constraint on the total cost of the features it can examine in test time, we focus on solving this problem with boosting by optimizing the choice of weaker learners in the training phase and stopping the boosting process when the budget runs out. We experimentally show that our method improves upon the boosting approach `AdaBoostRS` (4) and in many cases also outperforms the recent algorithm `SpeedBoost` (5). We provide a theoretical justification for our optimization method via the margin bound. We also experimentally show that our method outperforms CART decision trees, a natural budgeted classifier.

We then study on the problem of reconstructing social networks from voting data. In particular, given a voting model that considers social network structure, we aim to find the network that best explains the agents' votes. We study two plausible voting models, one edge-centric and the other vertex-centric. For these models, we give algorithms and lower bounds, characterizing cases where network recovery is possible and where it is computationally difficult. We also test our algorithms on United

States Senate voting data. Despite the similarity of the two models, we show that their respective network recovery problems differ in complexity and involve distinct algorithmic challenges. Moreover, the networks produced when working under these models can also differ significantly. These results indicate that great care should be exercised when choosing a voting model for network recovery tasks.

In the final part of the thesis, we explore problem of constructing a network by observing ordered connectivity constraints, where ordered constraints are made to capture properties of real-world problems. We give hardness of approximation results and nearly-matching upper bounds for the offline problem, and we study the online problem in both general graphs and restricted sub-classes. In the online problem, for general graphs, we give exponentially better upper bounds than exist for algorithms for general connectivity problems. For the restricted classes of stars and paths we are able to find algorithms with optimal competitive ratios, the latter of which involve analysis using a potential function defined over PQ-trees.

CHAPTER 1

INTRODUCTION

Learning theory seeks to answer the questions: What are proper models for learning, and for a given learning model, is a given class of functions learnable? However in the real world, problems generally do not fall perfectly into any learning model, and we cannot afford to stop researching even when problem of interest is known not to be learnable. Solving problems when observations are incomplete and flawed, and when features are pricey or risky to obtain is an everyday occurrence of all fields of scientific and technological endeavors. In a clinical trial, when subjects are in short supply, Wald (6) studies the sequential approach that keeps on testing more patients only when the uncertainty of the hypothesis in question is sufficiently large. To combat the problems of applying learning models that are trained in perfect conditions to imperfect real world setting, Globerson and Roweis (7) study the problem of whether a robust predictor can be built with resiliency to missing or corrupted features. Not only do we try to find solutions in imperfect conditions we also need to understand the boundary of what we can learn with the scarcity of resources and information. Efforts in this direction are organized into the field of approximation theory, which is the study of approximation algorithms to computationally hard problems and their performance guarantees, and into the field of competitive analysis, the study of online algorithms that must respond promptly to current input without seeing the entire input. It is well-known in the study of differential equations that, as it is hard enough to find a solution to a well-defined equation, the inverse problem of recovering the parameters of an unknown equation from observations of its solution is almost always ill-defined and can be significantly harder in both analysis and computation.

We have the same problem in the setting of network analysis. While finding the behavior of people in a given model from a known underlying network between them is computationally hard, as discussed in (8), the question of recovering the unknown network from observing people's behavior always lead to more intractable computational tasks, and small difference in the models can result in big difference in the recovered networks. However, as we cannot force earthquakes or floods to follow some arbitrary equations we have in mind, we cannot force people to live in a certain social network and in a particular way. We need to reveal the mechanisms of earthquakes and floods by measuring them, and we need to recover social network structure and models of how people interact in it by measuring behavior. So, learning under inadequate knowledge is not just practical, it is at the heart of all scientific exploration.

In this thesis we aim to solve three problems in the area of learning under limited resource and information.

The AdaBoost algorithm formulated by Freund and Schapire (9) is the bases for many state-of-the-art solvers for hard prediction problems. However, one problem that limits its application in many real-world scenarios is that boosting doesn't consider cases where features have costs. In the field of medical diagnoses, prices of tests vary vastly, from taking temperature, which costs practically nothing, to an MRI, which could easily cost thousands of dollars. In medical emergencies, not only do the monetary costs count, the amount of time used for making diagnoses can also be a matter of life or death. The World Wide Web is another source of prediction tasks that are time-sensitive; for example, no matter how strong a search engine algorithm is, if it runs too long, it risks people dropping out before seeing the results. By overlooking feature costs, the plain boosting algorithm may fail to predict at test time when the total cost of the features it uses exceeds some budget. In light of this, we set off in

Chapter 3 to construct predictors in the boosting framework that are feature-efficient, i.e. responsive to costs and budgets. The problem of feature-efficient learning is approached from many different angles in the theory of learning (10; 11) and studied under a variety of machine learning models (12; 13; 14; 15). One approach, `AdaBoostRS`, proposed by Reyzin (4), from where we directly get our inspiration is that, we first run the normal boosting algorithm to get a predictor, which is a linear combination of weak learners, then "prune" the linear combination to met the budget. This is done via random sampling with a bias towards the larger coefficients and smaller costs. However this approach is not satisfactory in the following aspects: first, since the coefficient is not necessarily an indicator of the relative importance of the weak learner, the practice of biasing towards terms with larger coefficients is not well-founded; second, boosting chooses weak learners to back up each other, so arbitrarily removing terms may cause the mutually supporting structure to fall apart; finally, Reyzin's optimization is done in two separate steps. This motivates us to find algorithms that can optimize with respect to both accuracy and cost at the same time while the predictor is being trained. We show that not only is this possible, the performance of the two approaches we propose, `AdaBoostBT_Greedy` and `AdaBoostBT_Smooth`, improves greatly over the two-staged optimization on both synthetic and real-world datasets.

Next, in Chapter 4, we study the computational complexity and algorithms for recovering networks from only observations of their members' behavior. In order to carry out the inference, we first have to hypothesize the ways social networks can impact the formation of members' opinion. We focus on two plausible and previously studied models, one is an edge-centric model called the independent conversation model (8) and the other vertex-centric called the common neighbor model (16). While in both models, edges of networks are used to represent some form of influence between pairs of vertices, the difference in the exact formulations causes the two network recovery problems to differ in complexity and produces distinct algorithmic challenges. In the independent conversation model we characterize cases where network recovery is possible and where it is computationally difficult. To analyze the complexity of the recovery problem under the common neighbor model, we compare the problem to the task of recovering graph from the square of its adjacency matrix, whose closely related problem we show to be hard. We also give a heuristic algorithm to the network recovery problem under the common neighbor model. In experimental results, we test our algorithms on United States Senate data. We show that the networks produced when working under the two models can also differ significantly. These results indicate that great care should be exercised when choosing a model for network recovery tasks.

Networks can also manifest their structure through how things, such as substances, information, or cases of disease, spread through them. Inspired by the works on connectivity constraints (17; 18; 19), in Chapter 5 we study the problem of inferring network from ordered connectivity constraints, which are formulated to capture the propagation pathways through a network over time. One salient feature of the task of reconstructing networks from ordered constraints is that the real-world problems it tries to imitate are very often of time-sensitive nature. For example, when a new transmission pathway of an epidemic is reported, prevention and control measures should be put into action as soon as possible in response, because although waiting for more information may benefit the optimization of the cost by restricting redundant actions, it potentially put at risk something we cannot afford to sacrifice, such as human lives. This motivated us to devote a large part of Chapter 5 on the comparative analysis of the reconstruction problem in the online setting, in which constraints are given one at a time, and an algorithm is required to satisfy each constraint upon receiving it with the understanding that no previous actions can be revoked even they turn out to be unnecessary later on. To briefly sum up the results we get in Chapter 5: in the offline problem, we give hardness of approximation results and nearly-matching upper bounds; in the online problem, for general graphs, we give exponentially better upper bounds for competitive ratios than those exist for general connectivity problems, and for the restricted classes of stars and paths, we are able to find algorithms with optimal competitive ratios.

CHAPTER 2

BACKGROUND

2.1 Learning theory

2.1.1 AdaBoost

Boosting was first introduced by Schapire in 1990 (20) as an algorithmic proof to the theoretical question "whether weak and strong PAC-learnablility are equivalent". The power of boosting as a solver for real-world problems was established when Freund and Schapire formulated the AdaBoost, short for "adaptive boosting", in 1995 (21).

AdaBoost takes as input a set of labeled examples S, a collection of weak classifiers $\mathcal{H}$, and number of rounds T, and outputs a classifier that is the signature of a linear combination of the weak classifiers. By weak classifier, we mean a classifier that is better than random guess. As in real life we form a strong team by choosing teammates that can back each other up, AdaBoost forms the linear combination so that weak classifiers in it complement each other. This backing-up mechanism is driven by a probability distribution on the examples. In each round t, a weak classifier h_t is chosen from $\mathcal{H}$, and a combination coefficient, or weight, α_t is computed using the h_t's error rate so as to minimize an exponential loss function. After adding the weighted weak classifier to the combination, AdaBoost mulitiplicatively diminishes the probability mass of examples that h_t classifies correctly, and amplifies those of examples that h_t gets wrong, and normalizes the masses to make it a probability distribution again. See Algorithm 1 for a pseudo code of AdaBoost.

Algorithm 1 `AdaBoost(S)`, where: $S \subset X \times \{-1,+1\}$, and $\mathcal{H}$ a set of weak classifiers, $T \in \mathbb{N}$

```
Given: $(x_1, y_1), ..., (x_m, y_m) \in S$
Initialize $D_1(i) = \frac{1}{m}$.
for $t = 1, \dots, T$ do
    Train base learner using distribution $D_t$, get weak classifier
        $h_t \in \mathcal{H} : X \to \{-1, +1\}.$
    Set
        $\alpha_t = \frac{1}{2} \ln \frac{1 + \gamma_t}{1 - \gamma_t},$
    where $\gamma_t = \sum_i D_t(i) y_i h_t(x_i)$ is the margin of the weak classifier with respect to $D_t$.
    Update
        $D_{t+1}(i) = D_t(i) \exp(\alpha_t y_i h_t(x_i)) / Z_t,$
    where $Z_t$ is the normalization factor.
end for
Output the final classifier
    $H(x) = \text{sign}\left( \sum_{t=1}^{T} \alpha_t h_t(x) \right).$
```

We note that, by updating the probability distribution in such a way that misclassified examples increase in importance, AdaBoost signals the process of weak classifier selection on exactly where the previous rounds fail to do well. It also worth noting that, although the greedy approach of choosing the weak classifier with the smallest margin works well in general, it is *not* necessarily the way AdaBoost makes the selection, especially when there are more criteria for a good weak classifier than low classification error. In Chapter 3, we generalize AdaBoost to deal with the case when each weak classifiers has a cost and there is a budget limit on the total costs of the combination. The way we tackle the problem is to select weak classifiers with respect to their performance, cost, and remaining budget.

An interested reader will find (22) a great reference to AdaBoost, and to machine learning in general. We also point out that, although we treat AdaBoost as a binary classification algorithm in this thesis, it has been generalized to many highly successful algorithms for multi-classification (23) and regression (24).

2.1.2 Decision tree learning

Decision tree learning is a supervised learning method that uses decision trees to do classification and regression. The method takes as input a training set of labeled examples with their observed features, and outputs a tree with each leaf node identified with a predicted label and each interior node, the root included, a test on a particular feature. When a new example comes, it traverses through the tree from the root to a leaf, and the path it takes is determined by the outcomes of the tests it encounters along the way. A test on feature is also called a split, because examples that go through the test will be split into two sets depended on their values of the selected feature. The choice of the feature used as a split is determined by how informative the feature is on predicting the labels of the subset of examples that go through the test, with informativeness measured by a given impurity function. An interested reader will find (25) a great reference to decision tree learning.

The following is a toy example of a decision tree predicting the gender of an imaginary species given an animal's weight, length, and color:

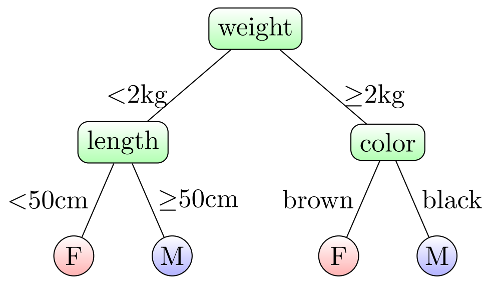


Figure 1: A toy example of decision tree predicting the gender of a species

In the example above, an animal that is 3.2kg and brown will be predicted as a female, and one that is 1.8kg and 60cm long will be predicted as a male.

There are many algorithms that compute decision trees, and the one we use in Section 3.7 is called the CART decision tree, a detailed description of which can be found in (26).

2.2 Computational complexity theory and competitive analysis

2.2.1 Complexity classes

All discussions on computational complexity theory we are going encounter in this thesis are in the computational model of Turing Machine, which we won't introduce in detail, but refer an interested reader to (27). In complexity theory, there are two big categories of problems, **decision problems** and **function problems**. For every input, a decision problem expects an output that can be recorded by a single bit, i.e. either 'YES' or 'NO', while a function problem expects an output that is more complex than a yes-or-no answer.

Definition 1 (Complexity class P and NP)**.** *A decision problem is in the complexity class* P *if it can be decided in polynomial time of its input size by a deterministic Turing machine. A decision problem is in the complexity class* NP *if it can be decided in polynomial time of its input size on an nondeterministic Turing machine.*

In computational complexity theory, a reduction is an algorithm converting one problem to another. Reductions are used to provide an order of hardness on problems. Intuitively, if we can reduce a problem A to a problem B, and by solving B we get the result for A in turn, B should be at least as hard as A. Since a reduction is meaningless if it takes longer to run than solving the original problem directly, the reductions we consider in this thesis all run in polynomial time in some way or another:

Definition 2 (Polynomial-time Karp reduction)**.** *A polynomial-time Karp reduction from a decision problem A to a decision problem B is an algorithm that, on input an instance x of problem A, given an* *instance y of B in polynomial time, such that the answer to y is 'YES' if and only if the answer to x is also 'YES'.*

Definition 3 (Complexity class NP-hard and NP-complete)**.** *An* NP *problem B is* NP*-complete if A can be reduced to B for all $A \in$* NP.

Definition 4 (Polynomial-time Turing reduction)**.** *A polynomial-time Turing reduction from a problem A to a problem B is an algorithm that solves problem A using polynomially many of calls to a subroutine for problem B (assumed to output result in unit time) on an oracle Turing machine, and polynomial time beside the calls to the subroutine.*

The Karp reduction is also called **many-one reduction**, and the Turing reduction is also called **oracle reduction**. We note that Turing reduction is more general than Karp reduction in that, first, Turing reduction can also be applied to function problems, and second, Karp reduction can be viewed as a restricted case of the Turing reduction in which the subroutine can only be called once.

In Definition 4 above, assume that the Turing machine is probabilistic, i.e. one that chooses from a set of instructions according to some probability distribution at each transition, and we only insist on that the solution is correct with probability $\geq 2/3$, then we have a **randomized polynomial-time Turing reduction**. The $2/3$ in the definition can be replaced by any probability strictly greater $1/2$, or even by $1-\varepsilon$ for arbitrarily small $\varepsilon > 0$, in which case the running time is allowed to depend polynomially on $1/\varepsilon$.

Definition 5 (Complexity class #P)**.** *A function problem is in* #P *if its output is the number of accepting paths of an* NP *problem running on a nondeterministic Turing Machine.*

We note that unlike the classes P and NP, and most other complexity classes, #P is a class of *function problems*.

Definition 6 (The Hitting Set problem)**.** *Let $U = \{u_1, \ldots, u_n\}$ be a set (also called a universe), and $\mathcal{S} = \{S_1, \ldots, S_m\}$ be a set of subsets of U. A subset $H \subset U$ is called a hitting set if $H \cap S_i \neq \emptyset$ for all $i = 1, \ldots, m$. The **optimization or search** version of Hitting Set problem is the function problem of finding a smallest hitting set. The **decision** version of the Hitting Set problem asks whether there exists a hitting set of size $\leq k$, with some $k \in \mathbb{N}^+$.*

Theorem 1. *The decision version of the Hitting Set problem is* NP*-complete (27), while the optimization version is* **NP-*hard***, *since its solution can be translated to a solution to the corresponding decision problem immediately.*

Since it is unlikely we can find efficient, i.e. polynomial-time, exact algorithms for NP-hard problems, polynomial-time approximation algorithms that render reasonably good sub-optimal solutions for hard problems become the next best thing we can hope for. To measure the performance of a polynomial-time approximation algorithm, we consider the **approximate ratio**, the quotient of the output given by the approximation in the worst case to the optimum, as a function of the input size. There are extensive researches in the area of approximation algorithms and an interested reader can find, for example, (28) as good reference. In this thesis, we are going to use the following result on the approximability of the hitting set problem, whose proof can be found in (29).

Theorem 2. *If* $\mathrm{P} \neq \mathrm{NP}$*, the approximation ratio of the Hitting Set problem is* $\Omega(\log n)$.

2.2.2 Competitive analysis

An **online algorithm** is an algorithm that must process input that is given in a one-by-one fashion. For example, an online algorithm for the hitting set problem, when given a sequence of sets $S_1, S_2, \ldots$ one at a time, should be able to pick up an element to hit S_i upon seeing it. Note that actions an online algorithm has taken will not be revoked even if they are later shown to be unnecessary. The performance of an online algorithm is measured by **competitive ratio**, which is the quotient of its output to that of an optimal offline algorithm.

When the online algorithm we consider is randomized, we have to specify how much knowledge the adversary has on the randomness of the algorithm.

Definition 7 (Oblivious adversary and adaptive adversary)**.**

- *An **oblivious adversary** knows the algorithm but not the randomized results of the algorithm. Hence an oblivious adversary must fix an input sequence, presumably the worst one, before feeding it to the online algorithm, but cannot alter it afterwards.*
- *An **adaptive adversary** knows the algorithm and also the randomized result of the algorithm up to the current round and can use it to determine its input to the algorithm in the next round.*

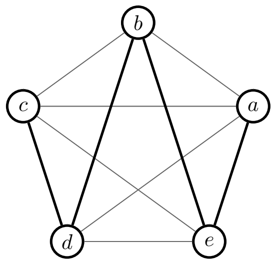
The adaptive adversary defined above is also called **adaptive online adversary**, to distinguish itself from **adaptive offline adversary** that knows, in addition, the random number generator of the algorithm. Since it can be shown that we can no long benefit from randomization when facing adaptive offline adversaries, by adaptive we usually mean implicitly adaptive online.

Assume that the objective function of an online problem is stated in the form of minimizing a quantity. To find a lower bound to the competitive ratio, we find a strategy that the adversary can take such that the competitive ratio cannot go bellow a certain value l no matter how smart an online algorithm is. A stronger lower bound can be achieved if there is a smarter strategy the adversary can take to force the best algorithm to do worse, i.e. to have a competitive ratio that is $\omega(l)$. To find an upper bound in the same setting, on the contrary, we find an online algorithm for the problem such that the competitive ratio cannot be forced to go above a certain value u by any strategy the adversary takes. By a stronger upper bound, we mean an algorithm that is more powerful such that the competitive ratio is $o(u)$ for the shrewdest adversary. From the definition of lower and upper bounds, we can derive that $l = O(u)$, i.e. $u = \Omega(l)$, holds for all pairs of lower l and upper bounds u. When we find a pair of lower and upper bounds, l and u, that match each other in their orders of magnitude of the input size, we know both l and u are the strongest, because for any lower bound l', we have $l' = O(u) = O(l)$, and for any upper bound u', we have $u' = \Omega(l) = \Omega(u)$. In this case, we say that we find the **optimal competitive ratio**.

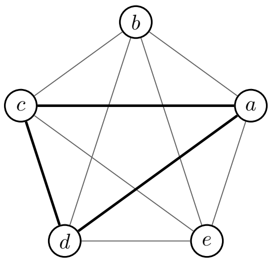
2.3 Graph theory

A graph $G = (V, E)$ is defined by a set V of objects, usually called vertices or nodes, together with a collection E of edges, where each $e \in E$ is a subset of V. In this thesis, we only consider **simple graphs**, whose edges contain exactly two distinct elements. The edges are used to represent some sort of relation between objects – in a social network, where V is a group of people, an edge may represent a channel for information and opinion exchange between two people; in a traffic network, where V is a collection of locations, an edge may represent a direct access from one location to another.

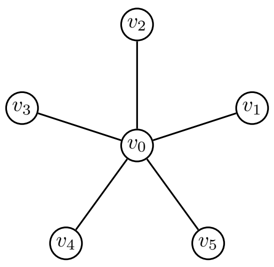
A **complete graph** has an edge between each pair of vertices. A **path** is a sequence of vertices and edges $v_1, e_1, v_2, e_2, v_3, \ldots v_{k-1}, e_{k-1}, v_k$, where v_1 through v_k are distinct elements, except for possibly the first and the last one, and edge $e_i = \{v_i, v_{i+1}\}$ for all $i = 1, \ldots, k-1$. When $v_1 = v_k$, the path is called a **cycle**. We say a graph is **connected**, if for any pair of vertices u and v, there is a path connecting them. We call a graph a path, if $V = \{v_1, \ldots, v_k\}$ and $E = \{e_1, \ldots, e_{k-1}\}$. An **acyclic graph** contains no cycle. A **tree** is a connected acyclic graph. A **leaf** in a tree can have only one edge incident to it, while an interior vertex must serve as the endpoint of at least two edges. A **star** is a graph that has a center v_0 connecting to all vertices, and the edge set $E = \{\{v_0, v\} \mid \text{for all } v \in V\}$. Examples of the concepts introduced above can be found in Figure 2.



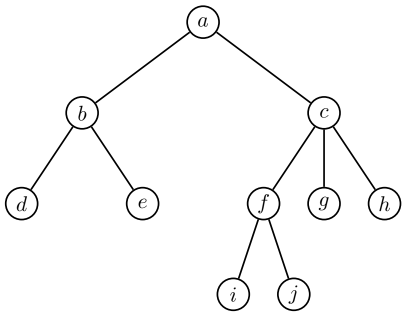
(a) A path c, d, b, e, a in a complete graph with 5 vertices



(b) A cycle c, a, d, c



(c) A star with v_0 as the center

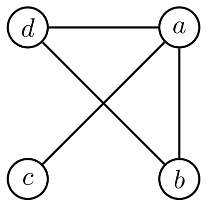


(d) A tree

Figure 2: Path, cycle, star, tree

The **neighborhood** $N(v)$ of a vertex v is defined to be the subset of all vertices that are connected to v, i.e. $N(v) = \{u \in V | \{u,v\} \in E\}$. The **degree** of a vertex v, denoted by d_v, is defined to be the size of its neighborhood, i.e. $d_v = |N(v)|$. For two distinct vertices u, v we denote by d_{uv} the size of the their common neighborhood, i.e. $d_{uv} = |N(u) \cap N(v)|$. The **adjacency** matrix $A(G)$ of a graph G, or simply A when G is clear from the context, is a $\{0,1\}$-valued matrix with rows and columns indexed by the vertices, ordered in the same way. The (u,v)-entry of $A(G)$ equals to 1 if and only if $\{u,v\}$ is in E. The adjacency matrix for the graph in Figure 3(a) is given by Figure 3(b). For a simple graph, A is a symmetric matrix, and the diagonal entries of the adjacency matrix are all 0. In this thesis, we are especially interested in A^2, the square of the adjacency matrix, since it encodes several important information of the graph. In particular, the (v,v)-entry of A^2 is d_v, and the (u,v)-entry of A^2 is d_{uv}.

The complexity of the decision problem of whether a given nonnegative integral matrix is the square of the adjacency matrix of some graph is a long lasting open problem.



(a) A graph G on 4 vertices

	a	b	c	d
a	0	1	1	1
b	1	0	0	1
c	1	0	0	0
d	1	1	0	0

(b) $A(G)$

	a	b	c	d
a	3	1	0	1
b	1	2	1	1
c	0	1	1	1
d	1	1	1	2

(c) $(A(G))^2$

Figure 3: The adjacency matrix $A(G)$ and its square

A **subgraph** $G' = (V', E')$ of a graph $G = (V, E)$ is formed with vertext set $V' \subset V$ and edge set $E' \subset E$, together with the constraint that all endpoints of edges in E' are in V'. An **induced subgraph** $G[V']$ of a subset $V' \subset V$ is a subgraph of G with the edge set consisting of all edges of G that have both endpoints in V'. We say a subset of vertices V' form a **clique** in G if the induced subgraph $G[V']$ is complete. The **clique number** $\kappa(G)$ of a graph G is defined to be the size of its largest clique. The **k-CLIQUE** problem asks whether a graph G has clique number $\kappa(G) \geq k$, for some $k \in \mathbb{N}^+$. We are going to use the following result in this thesis.

Theorem 3. *The k-*CLIQUE *problem is* NP*-complete.*

2.4 Probability theory

In this part, we give a brief review to probability theory on finite set that are used throughout this thesis, especially in Chapter 4.

2.4.1 Basic concepts

A **finite probability space** $\Omega = (S, \mathbb{P})$ is composed of a set S of objects, and a map, called distribution, $\mathbb{P} : S \to \mathbb{R}^{\geq 0}$, such tha $\sum_{s\in S} \mathbb{P}(s) = 1$. We call $\mathbb{P}(s)$ the probability of s. An **event** A is a subset of S, and we defined the probability of an event A to be

$$\mathbb{P}(A) = \sum_{s\in A} \mathbb{P}(s). \tag{2.1}$$

A random variable X is a map $S \to \mathbb{R}$. The **expectation** of a random variable X is given by $\mathbb{E}(X) = \sum_{s\in S} \mathbb{P}(s)X(s)$.The **variance** of a random variable X is given by

$$Var\,(X) = \mathbb{E}\left((X - \mathbb{E}(X))^2\right) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2\,. \tag{2.2}$$

Let X and Y be two random variables, we define the **covariance** of X and Y to be

$$Cov\,(X,Y) = \mathbb{E}\left((X - \mathbb{E}(X))\,(Y - \mathbb{E}(Y))\right) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y). \tag{2.3}$$

We can define events by conditions on random variables. For example, we can define an event $A = [1 < X \leq 4]$, where $[1 < X \leq 4]$ is the shorthand for $\{s \in S | 1 < X(s) \leq 4\}$, or an event $B = [X = 3, Y \geq 8]$, where $[X = 3, Y \geq 8]$ is the shorthand for $\{s \in S | X(s) = 3, \text{and } Y(s) \leq 8\}$. We say a

$\{0,1\}$-valued random variable X_A is an **indicator random variable** for an event A if $X_A(s) = 1$ if and only if $s \in A$. The subscript A in X_A is sometime omitted if A is obvious from the context. The **probability mass function, or pmf**, f_X of an random variable X is defined to be the function that gives the probability of the event $[X = a]$ for all $a \in \mathbb{R}$, i.e. $f_X(a) = \mathbb{P}[X = a]$. In the case when Ω is finite, we say two random variables X and Y are **identical** if and only if $[X = a] = [Y = a]$ for all possible a, and are **independent** if and only if $\mathbb{P}[X = a, Y = b] = \mathbb{P}[X = a]\,\mathbb{P}[Y = b]$ for all possible a and b. We say a sequence of random variables $X_1, ..., X_n$ are **independently identically distributed, or i.i.d.**, if they are all identical, and $\mathbb{P}[X_1 = a_1, \ldots, X_n = a_n] = \prod_{i=1}^{n} \mathbb{P}[X_i = a_i]$ for all possible $(a_1, \ldots, a_n)$. The expectation has the **linearlity** property, which says that the expectation of random variables is equal to the sum of their expectation, regardless of whether they are independent or not. The probability that an event A happens conditioned on another event B also happens, denoted by $\mathbb{P}[A|B]$ and called the **conditional probability** of A given B, is equal to $\mathbb{P}[A \cap B] / \mathbb{P}[B]$ if $\mathbb{P}[B] \neq 0$, and 0 if otherwise. Sometimes, the probability space Ω is just an element $\Omega(\text{`}) = (S, \mathbb{P}_{\text{`}})$ in a family of probability spaces parameterized by `} = (\theta_1, \ldots, \theta_k)$, and our task is to estimate $\text{` by sampling. Suppose we have a collection of samples $\{s_1, \ldots, s_m\}$ drawn independently from $\Omega(\text{`})$ of some unknown $\text{`}$, we define the **likelihood** of the samples to be $L(\theta) = \prod_{i=1}^{m} \mathbb{P}_{\text{`}}(s_i)$, and the **maximum likelihood estimate** of $\text{`}$ to be $\operatorname{argmax}_{\text{`}} L(\text{`})$. An estimator got by maximizing the likelihood function is called a **maximum likelihood estimator**.

For an example demonstrating the concepts above, we consider the probability space Ω of all possible graphs on three vertices a, b, and c. For a distribution on Ω, we assume that $\{a, b\}$ exists with probability 0.5, $\{b, c\}$ 0.2, and $\{c, a\}$ 0.8. We further assume that the edges exist independently of each other. We list the totally $2^3 = 8$ possible graphs in Ω and their probability in Figure 4.

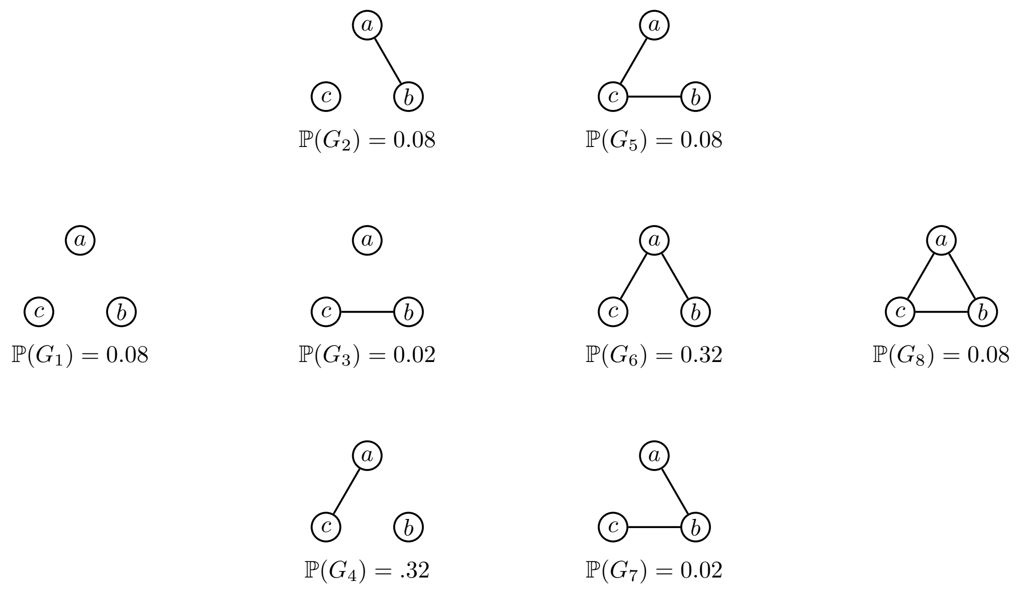


Figure 4: A distribution on the set of all graphs on three vertices

Let us consider the event A that the graph having two edges. Then we have $A = \{G_5, G_6, G_7\}$ and $\mathbb{P}(A) = 0.08 + 0.32 + 0.02 = 0.42$. Given that we have a graph having two edges, the probability that the graph is actually G_5 is given by $\mathbb{P}(G_5|A) = 0.08/0.42$. Let X be the random variable of the number of edges in graphs in Ω, then we have $\mathbb{E}(X) = 0.08 \cdot 0 + (0.08 + 0.02 + 0.32) \cdot 1 + (0.08 + 0.32 + 0.02) \cdot 2 + 0.08 \cdot 3 = 1.5$, while $Var(X) = 0.08 \cdot 0 + (0.08 + 0.02 + 0.32) \cdot 1 + (0.08 + 0.32 + 0.02) \cdot$

$4 + 0.08 \cdot 9 - 1.5^2 = 0.57$. In terms of X, the event A is defined by $[X = 2]$. Let X_{ab}, X_{bc}, X_{ca} be the indicator random variables of the existence of edge $\{a, b\}, \{b, c\}$ and $\{c, a\}$, respectively. We have $X = X_{ab} + X_{bc} + X_{ca}$. It is easy to calculate that $\mathbb{E}(X_{ab}) = 0.5$, $\mathbb{E}(X_{bc}) = 0.2$, and $\mathbb{E}(X_{ca}) = 0.8$, and hence by linearity of expectation we get, again, $\mathbb{E}(X) = 0.5 + 0.2 + 0.8 = 1.5$.

Note in the example above we can view Ω as an element $\Omega(0.5, 0.2, 0.8)$ in the family of probability spaces parameterized by the existing probability of the three edges. Assume for simplicity that the edges exist with probability p independently, and we want to estimate p by sampling three graphs from the space. Suppose the three graphs we get are G_2, G_5, G_6, we can calculate that the probably of getting these three graphs is $q = 2p^2(1 - p) + p(1 - p)^2$. Since q is maximized at $p = \sqrt{1/3}$, we have the maximum likelihood estimates of p is $\sqrt{1/3}$.

2.4.2 Chernoff-Hoeffding bound

By the central limit theorem, we know that the sample mean converges to $\mathbb{E}(X)$ as the size of the samples becomes large. But more often than not, we need a quantitative understanding on the convergence rate and, in turn, on the sample size that would guarantee an acceptably low chance that our estimate deviates from $\mathbb{E}(X)$ by too much. Because the ubiquitous appearance of the problem above in the study of learning, especially in the PAC model, we introduce the Chernoff-Hoeffding bound that is the one technique in cracking the problem. We begin with the original form of the Chernoff-Hoeffding bound on $\{0, 1\}$-value random variables, i.e. Bernoulli random variables, and proceed to variations of the bound used in this thesis directly.

Theorem 4 (Chernoff-Hoeffding Theorem)**.** *Suppose $X_1, ..., X_n$ are i.i.d. Bernoulli random variables. Let $Y = \frac{1}{n}\sum_{i=1}^{n} X_i$ and let $p = \mathbb{E}(X_i)$, then for any $\varepsilon > 0$ we have*

$$\mathbb{P}\left(Y \geq p + \varepsilon\right) \leq e^{-D(p+\varepsilon\|p)n}, \quad \mathbb{P}\left(Y \leq p - \varepsilon\right) \leq e^{-D(p-\varepsilon\|p)n},$$

where

$$D(p\|q) = p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q},$$

is the Kullback-Leibler divergence between two Bernoulli random with variables expectations p and q, respectively.

In case that the random variable of interest is $\{-1, 1\}$-valued, as those appear in Chapter 4, we have

Corollary 5. *Suppose $X_1, ..., X_n$ are i.i.d. $\{-1, 1\}$-valued random variables. Let $Y = \frac{1}{n}\sum_{i=1}^{n} X_i$ and let $p = \mathbb{E}(X_i)$, then for any $\varepsilon > 0$ we have*

$$\mathbb{P}\left(Y \geq p + \varepsilon\right) \leq e^{-D(p+\varepsilon/2\|p)n}, \quad \mathbb{P}\left(Y \leq p - \varepsilon\right) \leq e^{-D(p-\varepsilon/2\|p)n}.$$

Proof. Let $X_i' = (X_i + 1)/2$ and $Y' = \frac{1}{n}\sum_{i=1}^{n} X_i'$. Applied Theorem 4 to Y' with $p' = \mathbb{E}(X_i') = (p+1)/2$. $\square$

With the inequality

$$D(p + \varepsilon\|p) < 2\varepsilon^2,$$

we get a simpler form of the Chernoff-Hoeffding bound for $\{-1, 1\}$-valued random variables

$$\mathbb{P}\left(Y \geq p + \varepsilon\right), \quad \mathbb{P}\left(Y \leq p - \varepsilon\right) \leq e^{-n\varepsilon^2/2} \tag{2.4}$$

CHAPTER 3

TRAINING-TIME OPTIMIZATION OF A BUDGETED BOOSTER

This chapter was previously published as Yi Huang, Brian Powers, Lev Reyzin: Training-Time Optimization of a Budgeted Booster. *International Joint Conferences on Artificial Intelligence Organization (IJCAI)* 2015: 3583-3589.

3.1 Introduction

The problem of budgeted learning centers on questions of resource constraints imposed on a traditional supervised learning algorithm. Here, we focus on the setting where a learner has ample resources during training time but is constrained by resources in predicting on new examples. In particular, we assume that accessing the features of new examples is costly (with each feature having its own access cost), and predictions must be made without running over a given budget. This budget may or may not be known to the learner. Learners that adhere to such budget constraints are sometimes called **feature-efficient**.

A classic motivation for this problem is the medical testing setting where features correspond to the results of tests that are often costly or even dangerous to perform. Diagnoses often need to be made on incomplete information and doctors must order tests thoughtfully in order to stay within whatever budgets the world imposes. In internet-scale applications this problem also comes up. The cost to access certain features of a document or website is often used as a proxy for computing time which is crucially important to minimize. To address this issue, Yahoo! has recently released datasets which include feature costs (15).

Here, we focus on boosting methods, in particular `AdaBoost`, to make them feature-efficient predictors. This line of work was started by Reyzin (4), who introduced the algorithm `AdaBoostRS`, a feature-efficient version of `AdaBoost`. While `AdaBoostRS` provably converges to the behavior of `AdaBoost` as the feature budget is increased, it only considers feature costs and budget at test time. Reyzin left open the problem of whether optimizing during training can improve performance. Here, we answer this question with a resounding yes, giving algorithms that clearly outperform `AdaBoostRS`, especially when costs vary and budget limits are small.

Our approach relies mainly on two observations. The first is that when all features have equal costs, stopping the training of `AdaBoost` early, once the budget runs out, will outperform `AdaBoostRS`. Second, when features have different costs, which is the setting that chiefly concerned Reyzin, one can still run `AdaBoost`but choose weak learners as to better trade-off their cost against contribution to the performance of the ensemble.

3.2 Past work

Research on this problem goes back at least to Wald (6), who considered the problem of running a clinical trial sequentially, only testing future patients if the validity of the hypothesis in question is still sufficiently uncertain. This question belongs to the area of sequential analysis (30).

Ben-David and Dichterman (10) examined the theory behind learning using random partial information from examples and discussed conditions for learning in their model. Greiner et al. (11) also considered the problem of feature-efficient prediction, where a classifier must choose which features to examine before predicting. They showed that a variant of PAC-learnability is still possible even without access to the full feature set.

In related settings, Globerson and Roweis (7) looked at building robust predictors that are resilient to corrupted or missing features. Cesa-Bianchi et al. (31) studied how to efficiently learn a linear predictor in the setting where the learner can access only a few features per example. In the multi-class setting, Gao and Koller (12) used a parameter-tuned value-theoretic computation to create efficient instance-specific decision paths. In a similar vein, Schwing et al. (13) trained a random forest to adaptively select experts at test-time via a tradeoff parameter. He et al. (14) trained an MDP for this task, casting it as dynamic feature selection – their model is a variant of ours, except that they attempt to jointly optimize feature costs and errors, whereas our model has a strict bound on the budget. Finally, Xu et al. (15) tackled a related a feature-efficient regression problem by training CART decision trees with feature costs incorporated as part of the impurity function.

In the area of boosting, Pelossof et al. (32) analyzed how to speed up margin-based learning algorithms by stopping evaluation when the outcome is close to certain. Sun and Zhou (33) also considered how to order base learner evaluations so as to save prediction time.

However, our main motivation is the work of Reyzin (4), who tackled the feature-efficient learning problem using ensemble predictors. He showed that sampling from a weights distribution of an ensemble yields a budgeted learner with similar properties to the original ensemble, and he tested this idea experimentally on `AdaBoost`. The goal of this paper is to improve on Reyzin's approach by considering the feature budget during training.

We also compare to the recent work of (5), who also focused on this setting. Their algorithm, which is called `SpeedBoost`, works by sequentially choosing weak learners and voting weight α as to greedily optimize the improvement of a loss function (e.g. exponential loss) per unit cost, until the budget runs out.

3.3 AdaBoost and early stopping

Our goal in this paper is to produce an accurate classifier given a budget B and a set of m training examples, each with n features, and each feature with a cost via cost function $C : [n] \to \mathbb{R}^+$.

Reyzin's `AdaBoostRS` (4) takes the approach of ignoring feature cost during training and then randomly selecting hypotheses from the ensemble produced by `AdaBoost` until the budget is reached.

Here we look at a different approach – to optimize the cost efficiency of boosting during training, so the ensemble classifier that results is both relatively accurate and affordable.

One straightforward approach is to run `AdaBoost`, paying for the features of the weak learners chosen every round, bookkeeping expenditures and the features used, until we cannot afford to continue. In this case we are simply stopping `AdaBoost` early. We call this algorithm the "basic" `AdaBoostBT` for Budgeted Training. Surprisingly, this albeit simple methodology produces results that are significantly better than `AdaBoostRS` for both features with a uniform cost and features with random cost across a plethora of datasets.

We note that, in `AdaBoost`, since training error is upper bounded by

$$\widehat{\mathbb{P}}[H(x) \neq y] \leq \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \sqrt{1 - \gamma_t^2},$$

at each round t of boosting one typically greedily chooses the base learner that minimizes the quantity Z_t, which is equivalent to choosing the base learner that maximizes γ_t. This is done in order to bound the generalization error, which was shown by Freund and Schapire (21) to be bounded by

$$\mathbb{P}\left[H(x) \neq y\right] \leq \widehat{\mathbb{P}}[H(x) \neq y] + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right).$$

In these bounds $\widehat{\mathbb{P}}$ refers to the probability with respect to the training sample, and d is the VC-dimension of $\mathcal{H}$.

Hence, one can simply choose h_t in step 4 of `AdaBoostBT` according to this rule, which amounts to stopping `AdaBoost` early if its budget runs out. As we show in Section 3.6, this already yields an improvement over `AdaBoostRS`. This is unsurprising, especially when the budget or number of allowed rounds is low, as `AdaBoost` aggressively drives down the training error (and therefore generalization error), which `AdaBoostRS` does not do as aggressively. A similar observation will explain why the methods we will introduce presently also outperform `AdaBoostRS`.

However, this approach turns out to be suboptimal when costs are not uniform. Namely, it may sometimes be better to choose a worse-performing hypothesis if its cost is lower. Doing so may hurt the algorithm on that current round, but allow it to afford to boost for longer, more than compensating for the locally suboptimal choice.

3.4 A better trade-off

Here we focus on the problem of choosing a weak learner when feature costs vary. Clearly, higher values of γ_t are still preferable, but so are lower feature costs. Both contribute to minimizing the quantity

$\prod_{t=1}^{T} Z_t$, which upper bounds the training error. High γ_ts make the product small term-wise. Lower costs, on the other hand, allow for more terms in the product.[1] The goal is to strike the proper balance between the two.

One problem is that it is difficult to know exactly how many future rounds of boosting can be afforded under most strategies. If we make the assumption that we expect the costs of base learners selected in future rounds to be similar to the cost c in this round, we could afford B_t/c additional rounds of boosting. Assuming that future rounds will incur the same cost and achieve the same γ_t as in the current round, minimizing the quantity $\prod_{t=1}^{T} Z_t$ is equivalent to minimizing the quantity

$$\left(1 - \gamma_t(h)^2\right)^{T/2} = \left(1 - \gamma_t(h)^2\right)^{B_t/2c}.$$

Since $B_t/2$ is common to all hypotheses, h_t is chosen using the following rule:

$$h_t = \operatorname{argmin}_{h \in \mathcal{H}} \left((1 - \gamma_t(h)^2)^{\frac{1}{c(h)}} \right), \tag{3.1}$$

where

$$\gamma_t(h) = \sum_i D_t(i) y_i h(x_i),$$

and $c(h)$ is the cost of the features used by h. This is our first algorithm criterion for modifying base learner selection in step 4 of `AdaBoostBT` (boxed for emphasis) and name it `AdaBoostBT_Greedy`.

[1]This also creates a slightly more complex classifier, which factors into the generalization error bound, but this effect has been observed to not be very significant in practice (34; 35).

Algorithm 2 `AdaBoostBT(S,B,C)`, where: $S \subset X \times \{-1,+1\}$, $B > 0$, $C : [n] \to \mathbb{R}^+$

```
Given: $(x_1, y_1), ..., (x_m, y_m) \in S$
Initialize $D_1(i) = \frac{1}{m}$, $B_1 = B$
for $t = 1, \ldots, T$ do
    Train base learner using distribution $D_t$, get
        $\boxed{h_t \in \mathcal{H} : X \to \{-1,+1\}}$
    if the total cost of the unpaid features of $h_t$ exceeds $B_t$ then
        set $T = t - 1$ and break
    else
        set $B_{t+1}$ as $B_t$ minus the total cost of the unpaid features of $h_t$, marking them as paid
    end if
    Set
        $\alpha_t = \frac{1}{2} \ln \frac{1 + \gamma_t}{1 - \gamma_t},$
    where $\gamma_t = \sum_i D_t(i) y_i h_t(x_i)$
    Update
        $D_{t+1}(i) = D_t(i) \exp(\alpha_t y_i h_t(x_i)) / Z_t,$
    where $Z_t$ is the normalization factor
end for
Output the final classifier
    $H(x) = \text{sign}\left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$
```

There is a potential pitfall with this approach: if we mark every used feature down to cost 0 (since we don't re-pay for features), then the optimization will collapse since every base learner with cost 0 will be favored over all other base learners, no matter how uninformative it is. We can obviate this problem by considering the original cost during the selection, but not paying for used features again while updating B_t, as is done in our Algorithm.

As optimizing according to Equation 3.1 makes a very aggressive assumption of future costs, we consider a smoother optimization for our second approach. If in round t, we were to select h_t with cost c, the average cost per round thus far is then

$$\frac{(B - B_t) + c}{t}.$$

If we expect future rounds to have this average cost, we get a different estimate of the number of additional rounds we are able to afford. Specifically, in step 4 of `AdaBoostBT`, we select a base learner according to

$$h_t = \text{argmin}_{h \in \mathcal{H}} \left(\left(1 - \gamma_t(h)^2\right)^{\frac{1}{(B-B_t)+c(h)}} \right). \tag{3.2}$$

Selecting according to Equation 3.2 is less aggressive, because as more of the budget is used, current costs matter less and less. Hence, we call this second algorithm `AdaBoostBT_Smoothed`. While some other feature-efficient algorithms require a non-heuristic tuning parameter to control trade-off, this equation continually revises the trade-off as the budget is spent. Our experimental results show that it pays to be less greedy for larger budgets. To further customize this trade-off, however, a parameter

$\tau \in (0,1)$ may be used to scale $(B - B_t)$ allowing greed to drive the decision for more rounds of boosting.

One could even implement a hybrid approach, in which `AdaBoostBT_Greedy` is used at beginning to select weak learners and `AdaBoostBT_Smoothed` is used later on when the quality (a function of γ and c) of the unused features drops below a certain threshold. Exploring this hybrid idea, however, is outside the scope of this paper.

3.5 Additional theoretical justification

While the theory behind optimizing the bound of $\prod_{t=1}^{T} Z_t$ on the training error is clear, we can borrow from the theory of margin bounds (35) to understand why this optimization yields improved results for generalization error.

One might be concerned that in finding low cost hypotheses, we will be building too complex a classifier, which will not generalize well. In particular, this is the behavior that the (21) generalization bound would predict. Fortunately, the margin bound theory can be used to alleviate these concerns.

The following bound, known as the margin bound, bounds the probability of error as a sum of two terms.

$$\mathbb{P}\left[yf(x) \leq 0\right] \;\; \leq \;\; \widehat{\mathbb{P}}[yf(x) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right),$$

where $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$, as in Algorithm 2. The first term is the fraction of training examples whose margin is below a given value and the second term is independent of the number of weak learners.

It can be shown (22) that the first term can be bounded as follows

$$\widehat{\mathbb{P}}[yf(x) \leq \theta] \leq e^{\theta \Sigma \alpha_i} \prod_{t=1}^{T} Z_t,$$

where α_i is defined in Algorithm 2. For small θ this tends to

$$\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \sqrt{1 - \gamma_t^2}.$$

This well-known viewpoint provides additional justification for optimizing $\prod_{t=1}^{T} Z_t$, as is done in the preceding section.

3.6 Experimental results

Although there are a number of feature-efficient classification methods (12; 13; 15), we directly compare the performance of `AdaBoostBT`, `AdaBoostBT_Greedy` and `AdaBoostBT_Smoothed` to `AdaBoostRS` and `SpeedBoost` as both are feature-efficient boosting methods which allow for any class of weak learners.

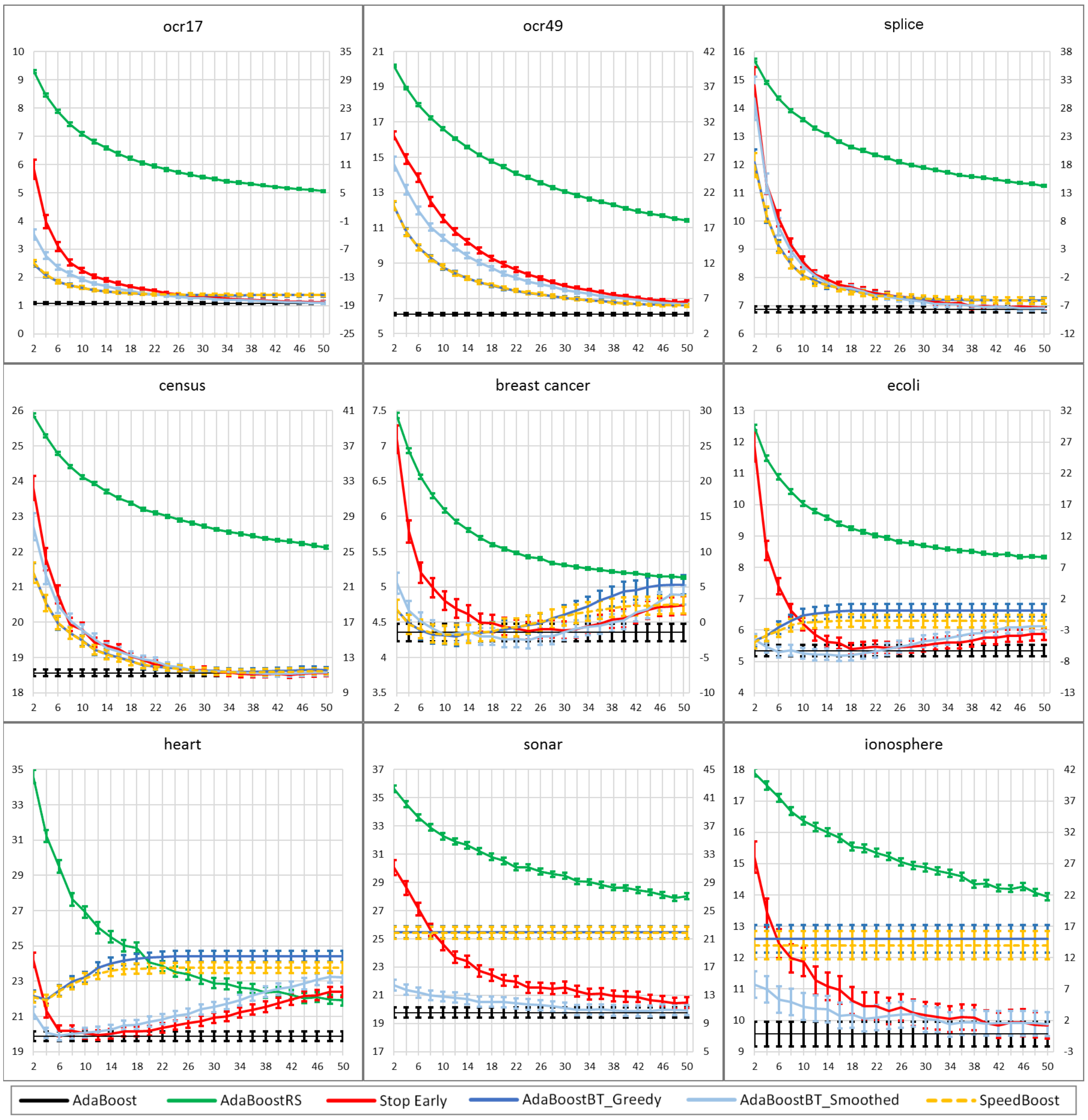


Figure 5: Experiments on nine UCI machine learning datasets

In Figure 5, `AdaBoostBT_Greedy` and `AdaBoostBT_Smoothed` are compared to `AdaBoostRS` and `SpeedBoost` on nine UCI machine learnig datasets. Test error is calculated at budget increments of 2. The feature costs are uniformly distributed in the interval $[0,2]$. The horizontal axis has the budget, and the vertical axis has the test error rate. `AdaBoostRS` test error rate uses the secondary vertical axis (on the right hand side) for all data sets except for heart. Error bars represent a 95% confidence interval. For our experiments we first used datasets from the UCI repository, as shown in Table I. The features and labels were collapsed into binary categories, and decision stumps were used for the hypothesis space.

In the table bellow we summarized the statistics (dataset sizes, numbers of features for training and test, and number of rounds when running the `AdaBoost` predictor.) of experiments demonstrated above.

TABLE I Statistics of the experiments on nine UCI machine learning datasets

	num features	training size	test size	AdaBoost rounds (optimized)	trials
ocr17	403	1000	5000	400	100
ocr49	403	1000	5000	200	100
splice	240	1000	2175	75	100
census	131	1000	5000	880	100
breast cancer	82	500	199	500	400
ecoli	356	200	136	50	400
sonar	11196	100	108	99	400
heart	371	100	170	15	400
ionosphere	8114	300	51	400	400
webscope set2	519	7000	15439	500	20

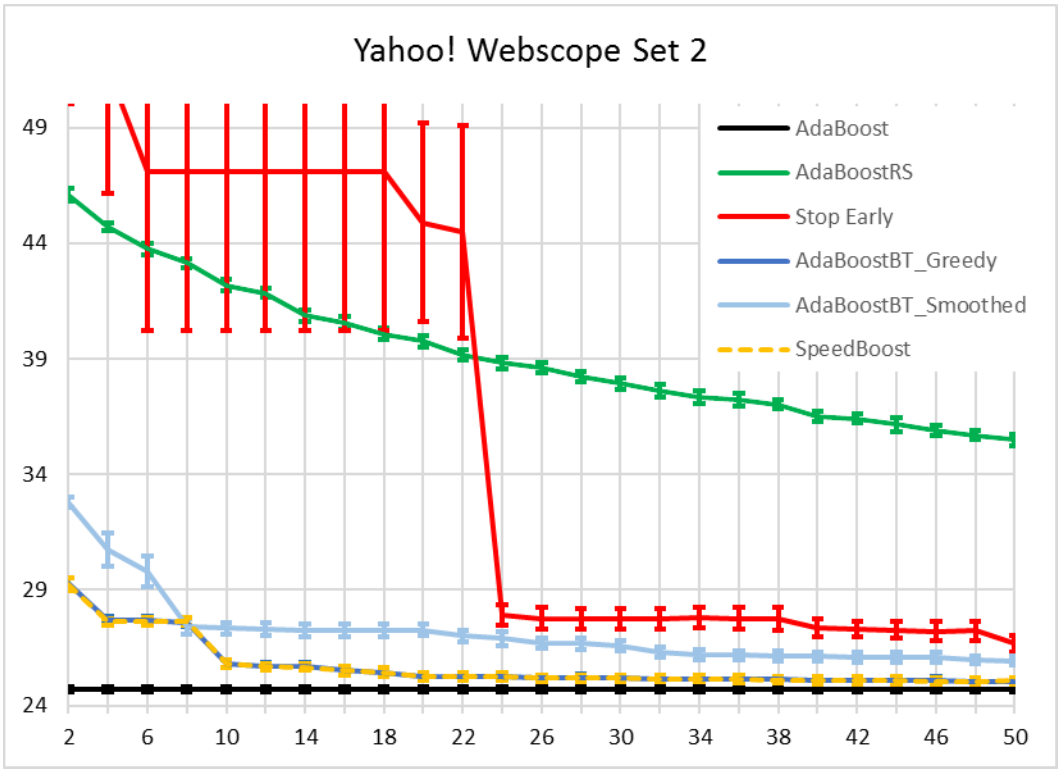


Figure 6: Experiments on the Yahoo! Webscope data set 2

In Figure 6, `AdaBoostBT_Greedy` and `AdaBoostBT_Smoothed` are compared to `AdaBoostRS` and `SpeedBoost` on the Yahoo! Webscope data set 2. Test error is calculated at budget increments of 2. Error bars represent a 95% confidence interval.

Experimental results, given in Figure 5, compare average generalization error rates over multiple trials, each with a random selection of training examples. Features are given costs uniformly at random on the interval $[0,2]$. For comparison, `AdaBoost` was run for a number of rounds that gave lowest test error, irrespective of budget. This setup was chosen to compare directly against the results of Reyzin (4) who also used random costs. We test on all the datasets Reyzin used, plus others.

Then, to study our algorithms on real-world data, we used the Yahoo! Webscope dataset 2, which includes feature costs (15). The data set contains 519 features, whose costs we rescaled to costs to the set {.1, .5, 1, 2, 5, 10, 15, 20}. Examples are query results labeled 0 (irrelevant) to 5 (perfectly relevant). We chose to collapse labels 1-5 to be binary label 1 (relevant) to test our algorithms. Results are given in Figure 6.

The most apparent conclusion from our experiments is that it is not only possible to improve upon `AdaBoostRS` by optimizing base learner selection during training, but that the improvement is dramatic. Further modifications of the basic `AdaBoostBT` tend to yield additional improvements.

`AdaBoostBT_Greedy` often performs better than the basic `AdaBoostBT` for small budgets, but it chooses base learners quite aggressively – a low cost base learner is extremely attractive at all rounds of boosting. This makes it possible that the algorithm falls into a trap, as we see in the sonar and ionosphere datasets where a huge number of features (11,196 and 8,114 respectively) lead to many features with costs close to zero (due to the cost distribution). Even after 500 rounds of boosting on the sonar dataset, this approach still does not spend the budget of 2 because the same set of cheap features are re-used round after round leading to a deficient classifier. Similar behavior is seen for the ecoli (356) and heart (371) datasets which also have relatively small training sizes, leading to over-fitting on small budgets.

`AdaBoostBT_Smoothed` avoids this trap by considering the average cost instead. The appeal of cheap base learners is dampened as the boosting round increases, with its limiting behavior to choose weak learners that maximize γ. Thus, we can see that using `AdaBoostBT_Smoothed`, while tending to perform worse than `AdaBoostBT_Greedy` for low budgets, tends to exceed its accuracy for larger budgets.

In cases when `AdaBoost` will noticeably over-fit with larger budgets (breast cancer, ecoli, heart - note the behavior of Stopping Early) we note that `AdaBoostBT_Smoothed` achieves the same (or better) error rate as `AdaBoost` at much lower budgets. For example, on the ecoli data set, `AdaBoost` needs a budget of 18 to achieve what `AdaBoostBT_Smoothed` does with a budget of 6.

On the Yahoo! Webscope data set, we see a dramatic difference between `AdaBoostBT` and our other optimizations. However, this is understandable because `AdaBoost` typically includes the expensive feature (cost of 20) in early rounds of boosting thus failing to produce a feature-efficient classifier for small budgets. Both the `Greedy` and `Smoothed` optimizations, however, effectively select from the less expensive features to create powerful low-budget classifiers.

Comparing to **`Speedboost`**

We also compare to the classification algorithm `SpeedBoost` (5) on all data sets, which was recently designed for tackling the same issue. `SpeedBoost` works by choosing weak learners (together with a voting weight α) so as to greedily optimize the improvement of a loss function per unit cost until the budget runs out. To mirror (5), as well as `AdaBoost`, we use exponential loss.

In our experiments, `SpeedBoost` performs almost identically to `AdaBoostBT_Greedy`, which forces us to use dashed lines for `SpeedBoost` in Figure 5. This phenomenon can be explained as follows. Let $H_t(x_i) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(x_i)$, `SpeedBoost` tries to find

$$\operatorname*{argmax}_{\alpha \in \mathbb{R}^+, h \in \mathcal{H}} \frac{\sum_{i=1}^{m} e^{-y_i H_t(x_i)} - \sum_{i=1}^{m} e^{-y_i (H_t(x_i) + \alpha h(x_i))}}{c(h)}$$

For a fixed $h \in \mathcal{H}$, let

$$I(\alpha) = \sum_{i=1}^{m} e^{-y_i H_t(x_i)} - \sum_{i=1}^{m} e^{-y_i (H_t(x_i) + \alpha h(x_i))},$$

we have

$$\begin{aligned} I'(\alpha) &= -\sum_{i=1}^{m} e^{-y_i H_t(x_i)} (-y_i h(x_i)) e^{-\alpha y_i h(x_i)} \\ &= \sum_{\{i : y_i = h(x_i)\}} e^{-y_i H_t(x_i)} e^{-\alpha} - \sum_{\{i : y_i \neq h(x_i)\}} e^{-y_i H_t(x_i)} e^{\alpha} \end{aligned}$$

which is equal to zero if and only if

$$\alpha = \frac{1}{2} \ln \frac{\sum_{\{i : y_i = h(x_i)\}} e^{-y_i H_t(x_i)}}{\sum_{\{i : y_i \neq h(x_i)\}} e^{-y_i H_t(x_i)}}. \tag{3.3}$$

Plugging in the α above, we get

$$\begin{aligned} &\max_{\alpha \in \mathbb{R}^+, h \in \mathcal{H}} \frac{\sum_{i=1}^{m} e^{-y_i H_t(x_i)} - \sum_{i=1}^{m} e^{-y_i (H_t(x_i) + \alpha h(x_i))}}{c(h)} \\ = \;& \max_{h \in \mathcal{H}} \frac{1}{c(h)} \left(\sum_{i=1}^{m} \ell_t(i) - 2 \left(\sum_{i : y_i = h(x_i)} \ell_t(i) \sum_{i : y_i \neq h(x_i)} \ell_t(i) \right)^{\frac{1}{2}} \right). \end{aligned}$$

Where $\ell_t(i) = e^{-y_i H_t(x_i)}$. Hence, in `SpeedBoost`, we implicitly find

$$\operatorname*{argmax}_{h \in \mathcal{H}} \frac{1 - \sqrt{1 - \gamma(h)^2}}{c(h)}. \tag{3.4}$$

Note that Equation 3.4 does not depend on α, which, after choosing h, can be set according to Equation 3.3. Remember that in `AdaBoostBT_Greedy` we find

$$\text{argmin}_{h \in \mathcal{H}} \left(1 - \gamma(h)^2\right)^{\frac{1}{c(h)}}.$$

Since

$$\min_{h \in \mathcal{H}} \left(1 - \gamma(h)^2\right)^{\frac{1}{c(h)}} = \max_{h \in \mathcal{H}} \frac{-\ln \sqrt{1 - \gamma(h)^2}}{c(h)},$$

and the Taylor series of $-\ln(x)$ is

$$(1 - x) + \frac{1}{2}(1 - x)^2 - o((1 - x)^2),$$

we have when $\gamma(h)$ is close to 0 (the value toward which boosting drives edges by making hard distributions), the two functions `SpeedBoost` and `AdaBoostBT_Greedy` seek to optimize are very similar.

As we can see, both algorithms greedily optimize an objective function without considering the impact on future rounds. Hence, `SpeedBoost` falls into the same trap of copious cheap hypotheses as `AdaBoostBT_Greedy`.

Yet, our approach offers a number of benefits over `SpeedBoost`. First, we have flexibility of explicitly considering future rounds, as `AdaBoostBT_Smoothed` does, in many cases outperforming `SpeedBoost`– e.g. on the ecoli, heart, sonar, and ionosphere data. Second, computational issues (for a discussion, see (5)) surrounding choosing the best α is completely avoided in our weak learner selection that the double optimization of `SpeedBoost` can also be avoided. Hence, our approach is simple, yet powerful at the same time.

3.7 A note on decision trees

One straightforward method for making a budgeted predictor is to use decision trees, and we consider this approach in this section. Decision trees are a natural choice for budgeted predictor since after a tree is constructed, each test example only needs to go through one path of the tree in order the receive a label. Hence, it incurs a cost using features only on that particular path. One may even be tempted to think that simply using decision trees would be optimal for this problem. We experimentally show that this is not the case, especially for larger budgets.

The problem with decision trees is that when one has more budget at hand and is able to grow larger trees, the trees begin to overfit, and this occured on all datasets (Figure 7). In contrast, `AdaBoostBT` performs better with more budget on most datasets. Cost optimization methods, both `Greedy` and `Smoothed`, tend to exploit large budgets to an even greater extent. Budgeted boosting algorithms continue to drive error rates down with higher budgets; decision trees do not.

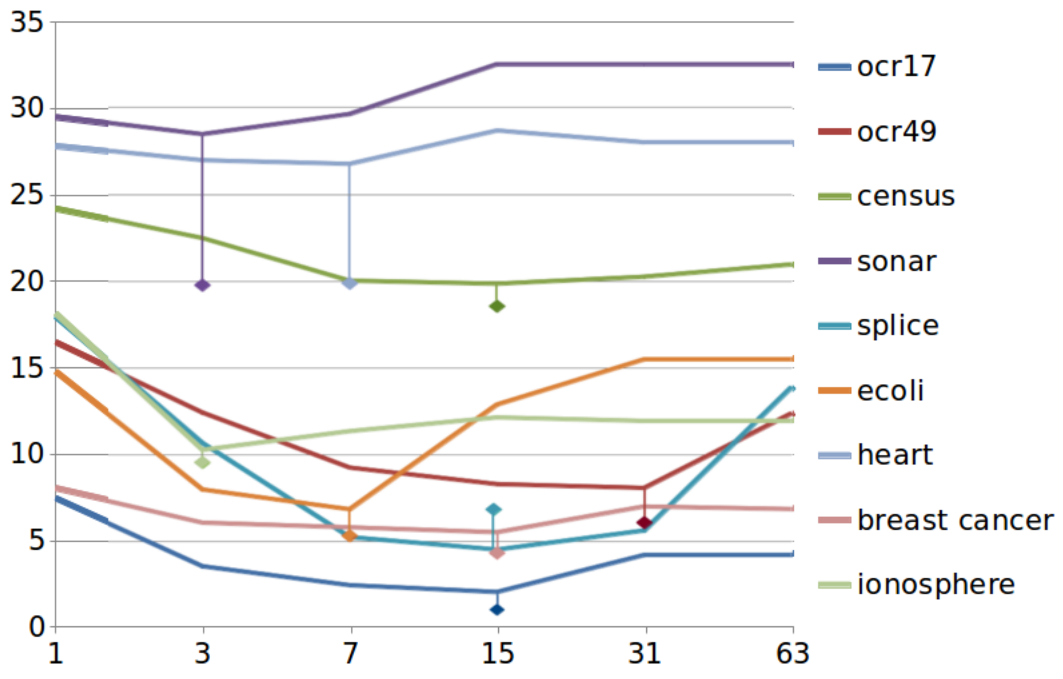


Figure 7: Error Rates of decision trees.

In Figure 7, the horizontal axis is these number of nodes (log scale in number of nodes, linear in expected tree depth). The vertical axis is percent error. Diamonds show the `AdaBoost` error rate for easy comparison.

CHAPTER 4

RECOVERING SOCIAL NETWORKS BY OBSERVING VOTES

This chapter was previously published as Benjamin Fish, Yi Huang, Lev Reyzin: Recovering Social Networks by Observing Votes. *International Conference on Autonomous Agents & Multiagent Systems (AAMAS)* 2016: 376-384.

4.1 Introduction

One approach to investigating voting data assumes that agents' votes are independent of one another, conditioned on some underlying (sometimes probabilistic) model of ground truth. This is usually an unrealistic assumption, leading to a more recent line of inquiry which asks how the social network structure of the voters affects the relationship between votes. Each agent in a social network expresses a position (votes for or against a bill, prefers one brand over another, etc.) that is influenced by their social connections. In this view, it is possible to detect the organization and evolution of voting communities by looking at the social network structure. The literature on congressional and political voting networks focuses on detecting community structure, partisanship, and evolutionary dynamics (36; 37; 38; 39; 40; 41), while the literature on idea propagation investigates how to best maximize the spread of ideas through a population (42; 43).

However, it is often not necessarily clear how to build this social network graph. For example, Macon et al. give a few different variants on how to define the social network of the voters of the United

Nations (38). In this approach, different graphs may reveal different aspects of the social network structure.

This corresponds neatly with a typical view in social choice theory that votes are manifestations of subjective preferences. At the other extreme, a voter votes according to a noisy estimate of the ground-truth qualities of the possible choices on which he or she is voting. While both are over-simplified extremes, it is useful to consider the extremes in order to investigate their consequences (8).

In this paper, as in previous work, we assume there is a fixed probabilistic model which is used to determine the relationship between initial preferences for the possible choices and how each individual ends up voting for those choices. This probabilistic model takes into account the social network structure in which the voters are embedded.

In this approach, it is typically assumed that the social network of the voters is known. The goal is then to find the correct choice from votes, as tackled by Conitzer and then others (8; 44; 45). This can be made more difficult depending on the structure of a social network, which may enforce the wrong choice by aggregating individual opinions over dense subgraphs, leading voters with low degree to possibly change their mind to the majority view of the subgraph.

In practice, the social network is usually not known and it is not necessarily clear how to infer the graph. In this paper, we tackle the problem of inferring the social network from the votes alone. We discuss two similar but distinct voting models in the vein of Conitzer (8), and show how to recover the graph given the votes under these voting models and under several notions of what it means to recover the graph. We show that your ability to learn the graph from the votes is highly dependent on the underlying voting model - in some settings, it is computationally hard to do so but not in others.

Moreover, we demonstrate that the resulting learned graphs can differ significantly depending on which underlying voting model is assumed.

4.2 Models and results

We give results for two similar models: an edge-centric model, which Conitzer calls the *independent conversation model* (8), and a vertex-centric model, introduced in this paper, which we will call the *common neighbor model.*

Similar to some existing models, the common neighbor model is, for instance, equivalent to the "deterministic binary majority process" run for one step (where the initial assignment is random). This process was examined by Goles and Olivos (16) and related work, e.g. by Frischknecht et al. (46), and it has been used in the press to illustrate the disproportionate influence of certain voters (47). The models we consider herein also resemble settings in multiple previous works, e.g. by Grabisch and Rusinowska (48), Grandi et al. (49), and Schwind et al. (50).

In both of our models, there is an unknown simple undirected graph G on n vertices. Each vertex is an agent, who can vote "-1" or "1". Both models describe how each agent votes in one round of voting. We consider m rounds of voting and in each round every vertex votes, leading to a sequence of vote sets $V^{[m]} = V_1, \ldots, V_m$, where each V_i is the set of votes from all voters. The problem is to recover G from $V^{[m]}$.

First, we define the independent conversation model, a two-step process where edges represent conversations between voting agents, and each agent votes according to the majority outcome of his conversations.

Definition 8 (**independent conversation model**). *First, each edge flips a coin i.i.d. that with probability p is 1 and with probability $(1-p)$ is -1. Then each vertex votes according to the majority of its adjacent edges' preferences. If there is a tie, then it is broken in favor of voting 1 with probability q and -1 with probability $1-q$.*

This process is depicted for a particular graph in Figure 8. Note that the set of votes V_i only includes the final votes, not the initial preferences.

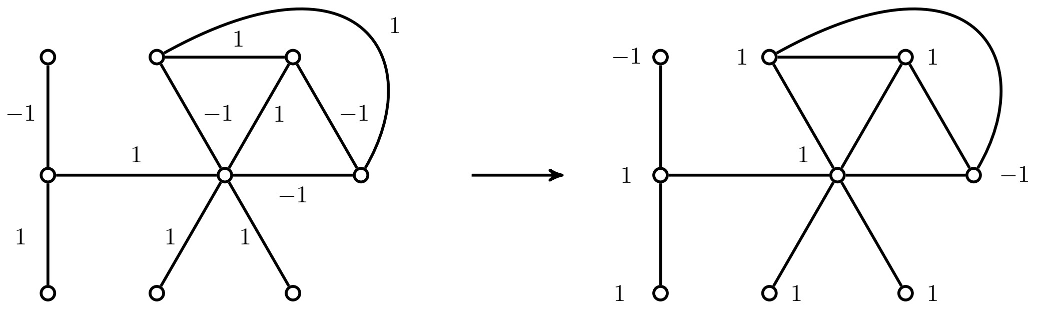


Figure 8: An example of the independent conversation model

On the left of Figure 8, we have the outcome of pairwise "conversations" between connected neighbors. On the right of Figure 8 we show the resulting votes. For simplicity, the edge probabilities are not depicted.

The common neighbor model is similar, except here the initial preferences are on the vertices, not the edges:

Definition 9 (common neighbor model). *Each vertex initially flips a coin i.i.d. that with probability p is 1 and with probability $1 - p$ is -1. Then each vertex votes 1 if more adjacent vertices' initial preferences were 1 then -1 and vice versa. If there is a tie, then it is broken in favor of voting 1 with some probability q and -1 with probability $1 - q$.*

This process is illustrated in Figure 9.

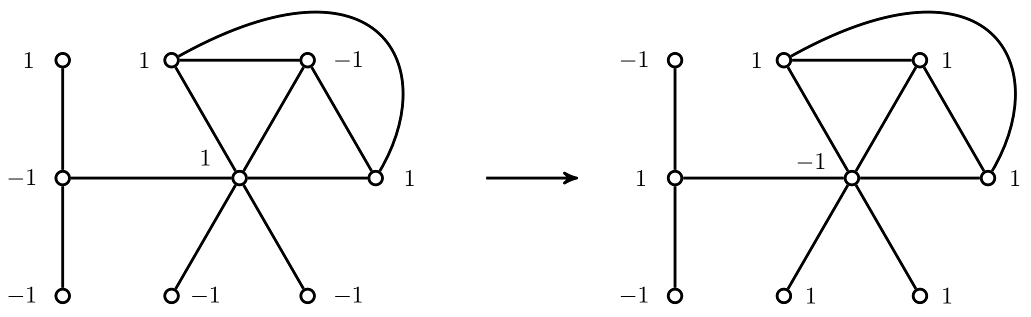


Figure 9: An example of the common neighbor model

On the left of Figure 9 we have the initial preferences of the nodes. On the right of Figure 9, we show the resulting votes. For simplicity, the preference probabilities are not depicted.

It is straightforward to see how they are different. In the independent conversation model, two vertices' votes are independent of each other if and only if they do not share an edge, while in the common neighbor model, they are independent if and only if they have no common neighbors.

Our main contribution consists of algorithms to recover the hidden graph G from the votes, lower bounds, and experiments for both models.

Our results span a few different notions of what it means to recover the unknown graph. First, we ask whether there exists a polynomial-time algorithm that succeeds with high probability (given only a polynomial number of votes in the number of voters) in finding the unknown graph G when the votes were drawn from G. The algorithm must take time polynomial in both the number of votes given as input and the number of vertices. We refer to this as *exact learning*. We show the following:

Result 1. *In the independent conversation model, there is a polynomial-time algorithm that exactly learns the unknown graph when* $p = 1/2$ *(with high probability). Moreover, for constant* $p \neq 1/2$*, an exponential number of votes are required to exactly learn the graph. (See* ***Observation 6*** *and* ***Theorem 7****.)*

Our algorithm is a statistical test for edges between pairs of vertices by calculating sample covariances of their votes, which here measures how likely it is that two voters vote the same way. This is very similar to how voting networks are often constructed in the literature (51; 38; 40). This result can then be seen as a formal motivation for why this type of method should be used.

Result 2. *In the common neighbor model, no algorithm can exactly recover the unknown graph. (See* ***Observation 13****.)*

The above two results motivate us to consider other notions of what it means to recover the graph because the graph is generally not recoverable efficiently here. Moreover, in a setting where there is not necessarily a ground-truth graph from which the votes were drawn, we are still interested in finding a graph that explains the votes.

We make this precise by asking for the maximum likelihood estimator (MLE) graph, that graph that maximizes the probability of the observed votes over the distribution of votes induced by a fixed voting model. As is standard for the maximum likelihood estimator, we assume that the prior over graphs is uniform. We refer to this as *maximum likelihood learning*. Under this model of learning, votes do not necessarily need to come from any particular hidden graph. Nevertheless, the goal is to produce the MLE graph under a voting model for a set of input votes regardless of where the votes came from.

Result 3. *In the independent conversation model, there is a polynomial-time bounded-probability random reduction from a* #P*-complete problem to finding the likelihood of the MLE graph. However, if enough votes are drawn from a hidden graph G when $p = 1/2$, there is a polynomial-time algorithm that finds the MLE graph on the votes with high probability. (See* ***Theorem 10*** *and* ***Theorem 12****.)*

This lower bound is an indication that computing the likelihood of the MLE graph is difficult, since if there were an efficient algorithm to compute this quantity then there would be an efficient algorithm to solve a #P-complete problem that succeeds with high probability.

On the other hand, merely trying to find the MLE graph is possible, at least if there is enough votes given as input (specifically, for n voters, order n^4 votes suffices).

In the common neighbor model, we investigate a third approach to finding a graph that explains the votes. Given that we recover graphs in the independent conversation model using covariances between votes, it is natural to ask whether it is possible to find a graph whose expected covariances are close to the observed covariances in the common neighbor model. We show that even if you were to know the expected covariances exactly, it would still be computationally difficult to find such a graph.

Result 4. *For the common neighbor model, finding a graph with given expected covariances between votes is at least as hard as recovering an adjacency matrix from its square. Moreover, a generalization of this problem, namely the generalized squared adjacency problem, is NP-hard. (See **Observation 14** and **Theorem 16**.)*

The squared adjacency problem and its generalized version are defined as follows:

Definition 10. *The input for the squared adjacency matrix problem is a matrix B, and the decision problem asks if there is an adjacency matrix A of a simple graph such that $A^2 = B$.*

Definition 11. *The input for the generalized squared adjacency problem is a collection of n^2 sets S_{ij}, and the decision problem asks if there is a simple graph whose adjacency matrix is A such that $A^2_{ij} \in S_{ij}$ for each entry A^2_{ij} of A^2.*

To the best of our knowledge, the squared adjacency matrix problem is not known to be NP-hard nor is it known to be in P. It is a difficult open problem in its own right, and other versions of it have been proven NP-hard (52). It is equivalent to the special case of the generalized version where the set sizes are exactly one.

4.3 The independent conversation model

In this section, we show when there is a algorithm to recover the hidden graph. We will also show that it is hard to find the likelihood of the maximum likelihood graph for an input sequence of votes. Before we present those two results, we start with the following observation:

Observation 6. *For constant $p \neq 1/2$, under the independent conversation model, it takes exponentially many votes to distinguish with high probability between the complete graph and the complete graph minus an edge.*

This follows directly from the fact that in both the complete graph and the complete graph minus an edge, if $p \neq 1/2$, with exponentially high probability every voter will vote 1. Our only hope is that it becomes possible to recover the graph G when $p = 1/2$, which we show to be the case.

4.3.1 An algorithm for $p = 1/2$

In this section, we prove the following:

Theorem 7. *Let $p = q = 1/2$. For any graph G on n vertices and $\delta > 0$, if $m = \Omega\left(n^2\left(\ln n + \ln \frac{1}{\delta}\right)\right)$ votes are drawn from G under the independent conversation model, there is a polynomial-time algorithm that will recover G with probability at least $1 - \delta$.*[1]

Let $X_u \in \{1, -1\}$ be the random variable representing the outputted vote of vertex u, so $X_u = 1$ if u votes 1 and -1 otherwise. Now consider two vertices u and v. The votes of u and v are independent if and only if (u, v) is not an edge. This yields a natural approach to determining if (u, v) is an edge of

[1] *This result actually remains true for arbitrary values of q, but we restrict the Theorem to $q = 1/2$ to simplify the proof in this version.*

G: measure the sample covariance between the votes of u and v and if this covariance is sufficiently far away from zero, there must be an edge.

To formalize this, we need to calculate the covariance between X_u and X_v if there is an edge between them:

Lemma 8. *For any edge (u,v) of G, let d_u and d_v be the degrees of u and v. For convenience, let $\rho = (1-2p)q + p$. Then $Cov\left(X_u, X_v\right)$ is*

$$\begin{cases} 4\rho^2\binom{d_u-1}{\frac{d_u-2}{2}}\binom{d_v-1}{\frac{d_v-2}{2}}\left(p(1-p)\right)^{\frac{d_u+d_v-2}{2}}, & \text{even } d_u, d_v \\ 4\rho\binom{d_u-1}{\frac{d_u-2}{2}}\binom{d_v-1}{\frac{d_v-1}{2}}\left(p(1-p)\right)^{\frac{d_u+d_v-1}{2}}, & \text{even } d_u, \text{ odd } d_v \\ 4\rho\binom{d_u-1}{\frac{d_u-1}{2}}\binom{d_v-1}{\frac{d_v-2}{2}}\left(p(1-p)\right)^{\frac{d_u+d_v-1}{2}}, & \text{odd } d_u, \text{ even } d_v \\ 4\binom{d_u-1}{\frac{d_u-1}{2}}\binom{d_v-1}{\frac{d_v-1}{2}}\left(p(1-p)\right)^{\frac{d_u+d_v}{2}}, & \text{odd } d_u, d_v. \end{cases}$$

Proof. Consider an edge $(u,v) \in E(G)$. Since u and v vote independently given the vote of the edge (u,v), we will write the probability that each of these vertices vote 1 given the edge vote.

Namely, call $P_u^1 = \mathbb{P}\left[X_u = 1 | \text{edge } (u,v) \text{ votes } 1\right]$ and $P_u^{-1} = \mathbb{P}\left[X_u = 1 | \text{edge } (u,v) \text{ votes -1}\right]$ and similarly P_v^1, P_v^{-1} the analogous probabilities for v.

We can write the covariance in terms of these four probabilities: $Cov\left(X_u, X_v\right) = 4p(1-p)(P_u^1 - P_u^{-1})(P_v^1 - P_v^{-1})$.

To show this, it suffices to write the covariance as a function of the joint probabilities $\mathbb{P}\left[X_u = X_v = 1\right]$, etc., and then write each joint probability as a function of the probabilities that a vertex votes 1 given that how the adjacent edge votes. For example, by conditioning on the vote of edge (u, v),

$$\mathbb{P}\left[X_u = X_v = 1\right] = pP_u^1 P_v^1 + (1-p)P_u^{-1}P_v^{-1}.$$

The others are similar.

To complete the proof, all we need are formulae for P_u^1 and P_u^{-1} (P_v^1 and P_v^{-1} are calculated analogously). This is done by choosing edges to form the majority vote of u's neighborhood.

Recall $d(u) - 1$ and $d(v) - 1$ are the degrees of u and v, respectively, minus 1 (in order to discount the edge (u, v)). We then have

$$P_u^1 = \begin{cases} \sum_{i=0}^{\frac{d(u)-2}{2}} \binom{d(u)-1}{i}(1-p)^i p^{d(u)-1-i} & \text{even } d(u) \\ + q\binom{d(u)-1}{\frac{d(u)}{2}}(1-p)^{\frac{d(u)}{2}} p^{\frac{d(u)-2}{2}}, & \\ \sum_{i=0}^{\frac{d(u)-1}{2}} \binom{d(u)-1}{i}(1-p)^i p^{d(u)-1-i}, & \text{odd } d(u). \end{cases}$$

and

$$P_u^{-1} = \begin{cases} \sum_{i=0}^{\frac{d(u)-4}{2}} \binom{d(u)-1}{i}(1-p)^i p^{d(u)-1-i} & \text{even } d(u) \\ + q\binom{d(u)-1}{\frac{d(u)-2}{2}}(1-p)^{\frac{d(u)-2}{2}} p^{\frac{d(u)}{2}}, & \\ \sum_{i=0}^{\frac{d(u)-3}{2}} \binom{d(u)-1}{i}(1-p)^i p^{d(u)-1-i}, & \text{odd } d(u), \end{cases}$$

The statement of the lemma then follows. $\square$

In the case where $p = 1/2$, the covariance will be sufficiently large; namely that it will be $\Omega(1/n)$, where n is the number of vertices of G:

Corollary 9. *When $p = 1/2$ and (u, v) is an edge of G,*

$$Cov\left(X_u, X_v\right) \geq \frac{1}{2\pi} \frac{1}{\sqrt{d_u d_v}} \geq \frac{1}{2\pi n}.$$

Proof. We simplify the formula for the covariance derived in Lemma 8 by giving lower bounds for the central binomial coefficients, from which the result immediately follows: For any positive integer k, the central binomial coefficient(s) satisfy

$$\binom{k}{\left\lceil \frac{k-1}{2} \right\rceil} \geq \frac{2^k}{\sqrt{\pi k}}.$$

These lower bounds follow from Sterling's approximation $k! = \sqrt{2\pi k}\left(\frac{k}{e}\right)^k \left(1 + O\left(\frac{1}{k}\right)\right)$. $\square$

Note this lower bound was only polynomial in $1/n$ because $p = 1/2$; otherwise, the exponential term $(p(1-p))^n$, for p constant, ensures that the covariance goes to 0 exponentially quickly in n.

We are now ready to prove Theorem 7, which uses the Hoeffding bound to establish that the sample covariance converges quickly enough to its expectation, which if there is an edge is given in Lemma 8 and if there is no edge is just 0.

Proof of Theorem 7. Recall that in the independent conversation model, we are given m votes $X_{u,i} \overset{i.i.d.}{\sim} X_u$ for $i = 1, \ldots, m$ and all u in G, where X_u is the $\{-1, 1\}$-valued random variable found by taking the majority vote of the initial votes of u's neighborhood.

For $p = q = 1/2$, $\mathbb{E}(X_u) = 0$ for each vertex u, which means that $Cov(X_v, X_u) = \mathbb{E}(X_u X_v)$. This means that the sample covariance between u and v is

$$C^m_{u,v} = \frac{1}{m}\sum_i^m X_{u,i}X_{v,i}.$$

The algorithm to recover G from the m votes is straightforward: For each pair of vertices u, v, calculate the sample covariance $C^m_{u,v}$. If $C^m_{u,v} > \frac{1}{4\pi n}$, then the algorithm claims there is an edge between u and v, and otherwise, the algorithm claims there is no such edge. We call this the *covariance test*. It suffices to show that the probability that the covariance test is wrong is low. Using Corollary 9, we get for edge (u, v),

$$\mathbb{P}\left(C^m_{u,v} < \frac{1}{4\pi n}\right) \leq \mathbb{P}\left(C^m_{u,v} - \mathbb{E}\left(C^m_{u,v}\right) < -\frac{1}{4\pi n}\right),$$

and for $(u, v) \notin E(G)$,

$$\mathbb{P}\left(C^m_{u,v} > \frac{1}{4\pi n}\right) = \mathbb{P}\left(C^m_{u,v} - \mathbb{E}\left(C^m_{u,v}\right) > \frac{1}{4\pi n}\right).$$

By the Hoeffding bound each of these two terms is bounded above by $e^{-\frac{cm}{n^2\pi^2}}$ for some constant c.

Let G' be the network inferred by the above algorithm. Then the probability that G' is not G is no more than

$$\sum_{u,v\in E}\mathbb{P}\left(C^m_{u,v} < \frac{1}{4\pi n}\right) + \sum_{u,v\notin E}\mathbb{P}\left(C^m_{u,v} > \frac{1}{4\pi n}\right),$$

which is bounded from above by $\binom{n}{2}e^{-\frac{cm}{n^2\pi^2}}$.

Hence, for any $\delta > 0$, setting $m = \Omega\left(n^2\left(\ln n + \ln\frac{1}{\delta}\right)\right)$ suffices so that $\binom{n}{2}e^{-\frac{cm}{n^2\pi^2}} < \delta$. $\square$

4.3.2 Moving from exact learning to maximum likelihood learning

In the previous section, we showed that exact learning is possible when $p = q = 1/2$. We now show that it is possible to not only exactly learn the graph, but also find the maximum likelihood graph for the votes when $p = q = 1/2$, assuming we are given enough data. Recall the maximum likelihood graph (which we will also refer to as the MLE graph) is the graph that maximizes the probability of the observed votes over the distribution of votes.

Theorem 10. *Let $p = q = 1/2$. For any graph G on n vertices and $\delta > 0$, if $m = \Omega\left(n^2\left(n^2 + \ln\frac{1}{\delta}\right)\right)$ votes are drawn from G under the independent conversation model, there is a polynomial-time algorithm that will find the maximum likelihood graph on the drawn votes with probability at least $1-\delta$.*

In other words, if the votes really do come from a hidden graph and we are given order n^4 votes, we can find the maximum likelihood graph. Specifically, what we show is that if you are given this many votes from a hidden graph, then the MLE graph *is* the hidden graph with high probability. The proof of Theorem 10 then follows from applying Theorem 7, i.e. using the covariance test to find the hidden graph, which is the MLE graph. This moves a statement about exact learning to a statement about maximum likelihood learning.

We now prove (Lemma 11) that the MLE graph is the hidden graph for a sufficiently large set of votes drawn from a hidden graph. Indeed, we show something stronger: the hidden graph will be more likely (under this model) than any other graph by an arbitrarily large factor α (where the number of votes given as input needs to increase logarithmically in α). This stronger result will also be needed for the proof of Theorem 12.

The statement of this will need some notation: For any graph G on n vertices, let $\mathcal{V}_G$ be the distribution over a set of n votes induced by G under the independent conversation model for $p = q = 1/2$. For convenience we will denote the m-product distribution $\mathcal{V}_G \times \ldots \times \mathcal{V}_G$ as $\mathcal{V}_G^{[m]}$. That is, for any vote $V \in \{-1, 1\}^n$, $\mathbb{P}_{\mathcal{V}_G}(V) = \mathbb{P}(V|G)$ is the probability mass of V under $\mathcal{V}_G$. Similarly, for a sequence of votes $V^{[m]}$, $\mathbb{P}_{\mathcal{V}_G^{[m]}}\left(V^{[m]}\right) = \mathbb{P}\left(V^{[m]}|G\right)$ is the probability mass of $V^{[m]}$ under $\mathcal{V}_G^{[m]}$.

Lemma 11. *For $\delta > 0$, $\alpha > 1$,*

$$\mathop{\mathbb{P}}_{V^{[m]} \sim \mathcal{V}_G^{[m]}}\left(\mathbb{P}\left(V^{[m]}|G\right) \leq \alpha \max_{G' \neq G} \mathbb{P}\left(V^{[m]}|G'\right)\right) < \delta$$

for $m = \Omega\left(n^2\left(n^2 + \ln\frac{1}{\delta} + \ln\alpha\right)\right)$.

Proof. Fix $\alpha > 1$ and denote by E the event that

$$\frac{\max_{G' \neq G} \mathbb{P}(V^{[m]}|G')}{\mathbb{P}(V^{[m]}|G)} \geq \frac{1}{\alpha}.$$

First of all, if $\mathbb{P}(V^{[m]}|G) = 0$, then this event occurs, so we can safely assume the converse. The idea of this proof is that we will show the probability of E happening is small by conditioning on what the vote sequence $V^{[m]}$ looks like when drawn from G. Specifically, in the proof of Theorem 7, we show that the covariance test would have successfully found G with high probability, so we condition on this happening.

Fix a graph $G' \neq G$. We will want to show that the probability that $V^{[m]}$ is pulled from G' (instead of G) is sufficiently small. The covariance test failed if $V^{[m]}$ were pulled from G': the covariance test returned G on $V^{[m]}$ instead of G'. And again, the probability that the covariance test failed is low, as showed in the proof of Theorem 7, so the probability that $V^{[m]}$ is pulled from G' must be small.

We denote the set of vote sequences for which the covariance test returns G by Φ_G. Using this notation, we condition on $V^{[m]}$ being in Φ_G or not and then get an immediate upper bound:

$$\mathbb{P}_{\mathcal{V}_G^{[m]}}(E) \leq \mathbb{P}_{\mathcal{V}_G^{[m]}}\left(E|V^{[m]} \in \Phi_G\right) + \mathbb{P}_{\mathcal{V}_G^{[m]}}\left(V^{[m]} \notin \Phi_G\right).$$

We then bound each of these two terms. The probability that the covariance test failed on $V^{[m]}$ is small: By inspecting the proof of Theorem 7, we have that for some constant c,

$$\mathbb{P}_{\mathcal{V}_G^{[m]}}\left(V^{[m]} \notin \Phi_G\right) \leq \binom{n}{2} e^{-\frac{cm}{n^2\pi^2}}. \tag{4.1}$$

Otherwise, the covariance test succeeded and we condition on $V^{[m]} \in \Phi_G$. We now show that

$$\mathbb{P}_{\mathcal{V}_G^{[m]}}\left(E|V^{[m]} \in \Phi_G\right) \leq \alpha\left(2^{\binom{n}{2}} - 1\right)\binom{n}{2} e^{-\frac{cm}{n^2\pi^2}}. \tag{4.2}$$

Markov's inequality gives

$$\mathbb{P}_{\mathcal{V}_G^{[m]}}\left(\frac{\max_{G'\neq G}\mathbb{P}\left(V^{[m]}|G'\right)}{\mathbb{P}\left(V^{[m]}|G\right)} \geq \frac{1}{\alpha}\,\middle|\, V^{[m]} \in \Phi_G\right)$$
$$\leq$$
$$\alpha \cdot \mathbb{E}_{\mathcal{V}_G^{[m]}}\left(\frac{\max_{G'\neq G}\mathbb{P}\left(V^{[m]}|G'\right)}{\mathbb{P}\left(V^{[m]}|G\right)}\,\middle|\, V^{[m]} \in \Phi_G\right).$$

It is then enough to expand this expected value using the definition to get that

$$\mathbb{P}_{\mathcal{V}_G^{[m]}}\left(E|V^{[m]} \in \Phi_G\right) \leq \alpha \sum_{V^{[m]} \in \Phi_G} \max_{G' \neq G} \mathbb{P}\left(V^{[m]}|G'\right).$$

Now we group the terms of the sum by which graph $G' \neq G$ maximizes the probability $\mathbb{P}\left(V^{[m]}|G'\right)$. There may be many terms in the sum that any one graph G' maximizes, but certainly each vote sequence associated with each term is in Φ_G. There are of course $2^{\binom{n}{2}} - 1$ such graphs, so

$$\begin{aligned}\sum_{V^{[m]} \in \Phi_G} \max_{G' \neq G} \mathbb{P}\left(V^{[m]}|G'\right) &\leq \sum_{G' : G' \neq G} \sum_{V^{[m]} \in \Phi_G} \mathbb{P}\left(V^{[m]}|G'\right) \\ &= \left(2^{\binom{n}{2}} - 1\right) \mathbb{P}\left(V^{[m]} \in \Phi_G|G'\right).\end{aligned}$$

If $V^{[m]}$ were in Φ_G but $V^{[m]}$ was pulled from G', then the covariance test has failed at returning G'. So

$$\mathbb{P}\left(V^{[m]} \in \Phi_G|G'\right) \leq \binom{n}{2} e^{-\frac{cm}{n^2\pi^2}},$$

implying Equation 4.2. Combining Equation 4.1 and Equation 4.2, we get

$$\mathbb{P}_{\mathcal{V}_G^{[m]}}(E) \leq \alpha 2^{\binom{n}{2}} \binom{n}{2} e^{-\frac{cm}{n^2\pi^2}}.$$

For $\mathbb{P}_{\mathcal{V}_G^{[m]}}(E)$ to be upper-bounded by $\delta > 0$, it suffices to set $m = \Omega\left(n^2\left(n^2 + \ln\frac{1}{\delta} + \ln\alpha\right)\right)$. $\square$

4.3.3 Hardness of computing the MLE

As we have seen, when $p = q = 1/2$, distinguishing between graphs can be done in polynomial time. This might give hope that, in this case, computing the likelihood of the MLE graph, given a set of votes, may be easy. That is, given a graph G which is the maximum likelihood graph for a set of input votes $V^{[m]}$ over G, we wish to compute $\mathbb{P}(V^{[m]}|G)$. Alas, we give hardness results indicating this is not easy to do.

We reduce from Conitzer's problem of computing $\mathbb{P}\left(V^*|G\right)$, where V^* is a vote produced by a given graph G (8).[1] He shows that this is problem is #P-hard by reducing from counting the number of perfect matchings in a bipartite graph. Surprisingly, our proof of this hardness result uses the easiness of finding the MLE graph in polynomial time in the case when $p = q = 1/2$. Namely, we use Lemma 11 to be able to say when the input G is the maximum likelihood graph for a set of votes $V^{[m]}$, which in turn says when the oracle will successfully compute $\mathbb{P}(V^{[m]}|G)$. Formally, we prove the following theorem:

Theorem 12. *There is a randomized polynomial-time oracle reduction from counting the number of perfect matchings in a balanced bipartite graph to computing the MLE of the maximum likelihood graph from a sequence of votes with high probability.*

sketch. It suffices to consider the case where $p = q = 1/2$. Instead of directly reducing from the #P-hard problem of counting the number of perfect matchings in a balanced bipartite graph, we reduce

[1]While the problem Conitzer considers is slightly different than computing $\mathbb{P}\left(V^*|G\right)$, in the case where $p = 1/2$, his problem reduces to computing $\mathbb{P}\left(V^*|G\right)$.

from the #P-hard problem of computing $\mathbb{P}\left(V^*|G\right)$ given a graph G and vote V^* on n voters under the independent conversation model.

The idea of the proof is going to be to build a sequence of votes $V^{[m]}$ whose MLE we know to be the input G, and then compute

$$\mathbb{P}(V^*|G) = \frac{\mathbb{P}(V^{[m]}, V^*|G)}{\mathbb{P}(V^{[m]}|G)}.$$

Our oracle will give us the values of the right-hand side. This approach will work if $\mathbb{P}(V^*|G) \neq 0$.

So we first test for the case if $\mathbb{P}(V^*|G) = 0$. Conitzer provides a way to do this for a similar problem when the vertices of the graph have all odd degree: his reduction is from the maximum weighted b-matching problem, which we can adapt to the so-called "c-capacitated" version that we need (8; 53).

Else, $\mathbb{P}\left(V^*|G\right) \neq 0$. We draw a sequence of votes $V^{[m]} \overset{i.i.d.}{\sim} \mathcal{V}_G \times \ldots \times \mathcal{V}_G$. Lemma 11 immediately implies that G will be the MLE for $V^{[m]}$ with failure probability less than $\delta/2$ when $m = \Omega(n^2(n^2 + \ln \frac{2}{\delta}))$. In other words, with just $\Omega(n^4)$ votes we will successfully query the oracle for $\mathbb{P}(V^{[m]}|G)$ with high probability.

It suffices to ensure that with high probability we will also successfully query the oracle for the $(m+1)$-length sequence $V^{[m]}, V^*$. Recall that since $\mathbb{P}(V^*|G)$ is the sum, over all satisfying edge votes, of the quantity $\left(\frac{1}{2}\right)^{|E(G)|}$, where $E(G)$ is the edge set of G and $p = 1/2$. There must be at least one satisfying edge-vote assignment since $\mathbb{P}(V^*|G) \neq 0$, so $\mathbb{P}(V^*|G) \geq \left(\frac{1}{2}\right)^{|E(G)|}$. In addition, again by Lemma 11, for any $G' \neq G, \frac{\mathbb{P}(V^{[m]}|G)}{\mathbb{P}(V^{[m]}|G')} > \alpha$ with failure probability no more than $\delta/2$ when $m = \Omega(n^2(n^2 + \ln\frac{2}{\delta} + \ln\alpha))$. Then for any $G' \neq G$,

$$\begin{aligned}\mathbb{P}(V^{[m]}, V^*|G') &< \left(\frac{1}{\alpha}\mathbb{P}(V^{[m]}|G)\right)\left(2^{|E(G)|}\mathbb{P}(V^*|G)\right)\\ &= \frac{2^{|E(G)|}}{\alpha}\mathbb{P}(V^{[m]}, V^*|G).\end{aligned}$$

Setting $\alpha = \Omega(e^{n^2})$ suffices to ensure that $\alpha > 2^{|E(G)|}$. Thus setting

$$m = \Omega\left(n^2\left(n^2 + \ln\frac{2}{\delta} + \ln\alpha\right)\right)$$

as above for this setting of α, a query to the oracle for $\mathbb{P}(V^{[m]}, V^*|G)$ will fail with probability less than $\delta/2$. Setting δ to be, say, $\Theta(\frac{1}{2^n})$, yields that $m = \Omega(n^4)$. The oracle reduction, once it tests for the existence of at least one valid edge vote, simply consists of drawing m votes from G and then querying the oracle for $\mathbb{P}(V^{[m]}, V^*|G)$ and $\mathbb{P}(V^{[m]}|G)$. The reduction then succeeds with probability at least $1 - \delta$. $\square$

4.4 The common neighbor model

We now turn our attention to the common neighbor model. Again, we ask if it is possible to recover G by seeing only polynomially many votes. In general, it is not possible to recover G at all, let alone with only polynomially many votes:

Observation 13. *Under the common neighbor model, no algorithm can distinguish between two different perfect matchings.*

If G is a matching between the vertices, each vertex will vote how its neighbor votes, meaning that each vertex votes i.i.d. with probability p regardless. Thus there is no way to distinguish between different matchings.

4.4.1 Recovering A^2 from covariances

Given the impossibility of recovering the graph, we relax the problem to the following: Find a graph that is likely to produce the given votes in the sense that the expected covariances of this graph should be as close as possible (under some norm) to the covariances of the observed votes. This problem is motivated by the algorithm for the independent conversation model which finds a graph whose expected covariances match the measured covariances.

Yet even if we were to know the *expected* covariances of the input votes, finding a graph whose expected covariances are close to those input covariances remains challenging:

Observation 14. *For the common neighbor model, finding a graph with given expected vote covariances is at least as hard as recovering an adjacency matrix from its square.*

To prove this observation, it suffices to show that the expected covariances are a function solely of the entries of A^2. Then recovering A from A^2 consists of using the entries of A^2 to compute the expected covariances, at which point the adjacency matrix of a graph with those covariances will be exactly A. The i, jth entry of A^2 is the number of length-two paths between i and j, so it is enough to write the covariances of a graph in terms of the following: For $\Gamma(v)$ the neighborhood of a vertex v, denote $d_{uv} = |\Gamma(v) \cap \Gamma(u)|$, $d_u = |\Gamma(u) \setminus (\Gamma(v) \cap \Gamma(u))|$, and d_v analogously. The covariances are a function of d_u, d_v, and d_{uv}. For the sake of simplicity we will assume that $|\Gamma(u)|$ and $|\Gamma(v)|$ are odd, but it is straightforward to modify the formula given below in the cases when they are not.

Lemma 15. *Assume $|\Gamma(u)|$ and $|\Gamma(v)|$ are odd. For $p = 1/2$,*

$$Cov(X_u, X_v) = \frac{1}{2^{d_{uv}-2}} \left(\sum_{k=0}^{d_{uv}} \binom{d_{uv}}{k} P_{u,v}(k) P_{v,u}(k) \right) - 1,$$

where, for $\theta_{u,v} = (d_{uv} + d_u + 1)/2 - k$,

$$P_{u,v}(k) = \begin{cases} \frac{1}{2^{d_u}} \sum_{i=\theta_{u,v}}^{d_u} \binom{d_u}{i} & \text{if } 0 \le \theta_{u,v} \le d_u \\ 0 & \text{if } \theta_{u,v} > d_u \\ 1 & \text{if } \theta_{u,v} \le 0. \end{cases}$$

Proof. Let X_u represent vertex u's vote. When $p = 1/2$, $\mathbb{E}(X_u) = 0$, and $\mathbb{P}[X_u = X_v = 1] = \mathbb{P}[X_u = X_v = 0]$, so the covariance $Cov(X_u, X_v)$ is

$$\mathbb{E}(X_u X_v) = 2\mathbb{P}[X_u = X_v] - 1 = 4\mathbb{P}[X_u = X_v = 1] - 1.$$

To determine $\mathbb{P}[X_u = X_v = 1]$, we condition on the number of common neighbors that voted 1:

Assuming some k common neighbors vote 1, in order for u to vote 1, u needs an additional $\frac{d_{uv}+d_u+1}{2} - k$ neighbors to vote 1. If k is already at least $\frac{d_{uv}+d_u+1}{2}$, then the probability of voting 1 is already 1; on the other hand if there aren't enough remaining vertices to vote 1, then the probability is 0. This yields $P_{u,v}(k)$ as the probability that u votes 1 given that k common neighbors of u and v voted 1.

Now we can write $\mathbb{P}\left[X_u = X_v = 1\right]$ as

$$\sum_{k=0}^{d_{uv}} \binom{d_{uv}}{k} p^k (1-p)^{d_{uv}-k} P_{u,v}(k) P_{v,u}(k),$$

completing the proof. □

When recovering a graph from a sequence of input votes, we are not even given the expected covariances of the input votes. Instead we can calculate the measured covariances, from which we can determine A^2. At this point, we have a function inversion problem on our hands: We can find A^2 merely by recovering these d_{uv}'s and d_u's from the covariances, but given that the formula given in Lemma 15 is not closed, this is not trivial. Since there are only polynomially many possible values for d_{uv} and d_u, we can simply try all values to find the covariance closest to the observed value. However, there may be covariances that are exponentially close to each other, making it impossible to distinguish between these values for given d_{uv}, d_u. In this case the values recovered for the entries of A^2 may not be unique, which in the worst case leads to the generalized squared adjacency problem. Even in the case when we recover unique values, it still reduces to the squared adjacency problem.

While the squared adjacency problem is open, we show the following:

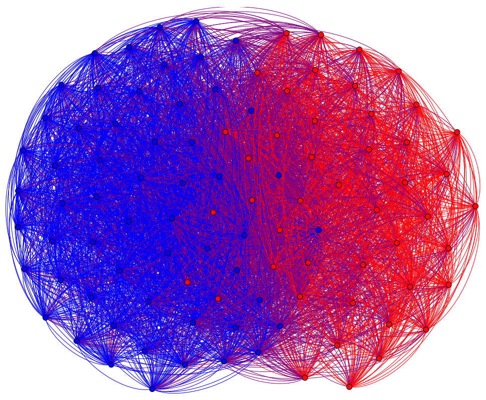
Theorem 16. *The generalized squared adjacency problem is* NP*-hard.*

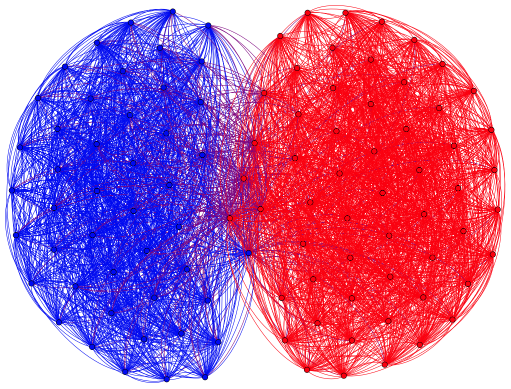
Proof. The reduction is from CLIQUE, which asks if there is a clique of size k on the input graph. Given a graph $G = (V, E)$ and an integer k, we construct a set system $\{S_{ij}\}$ with $(n+1)^2$ sets, where $n = |V|$. We then show that G has a clique of size k if and only if there is a graph $G' = (V \cup \{v\}, E')$ such that the i,jth entry of $A(G')^2$ is in $S_{i,j}$, where $A(G')$ is the adjacency matrix of G'. The $(n+1)^2$-sized set system $\{S_{i,j}\}$ is defined as follows:

$$S_{ij} = \begin{cases} \{0,1\} & \text{if } i \neq j \text{ and } i,j \neq v \text{ and } (i,j) \in E(G) \\ \{0\} & \text{if } i \neq j \text{ and } i,j \neq v \text{ and } (i,j) \notin E(G) \\ \{0\} & \text{if } i = v \text{ and } j \neq v \\ \{0\} & \text{if } j = v \text{ and } i \neq v \\ \{k\} & \text{if } i = j = v \\ \{0,1\} & \text{if } i = j \text{ and } i,j \neq v \end{cases}$$

Assume there is a clique of size k in G. Then G' is defined as follows: Denote the vertex set of the clique in G by C. G' will have an edge between v and all members of C, and no other edges. It is straightforward to check that the i,jth entry of $A(G')^2$ is in S_{ij} by noting that the diagonal entries of $A(G')^2$ are the vertices' degrees and the off-diagonal entries counts the number of common neighbors.

In the other direction, assume there is such a graph G' whose squared adjacency matrix satisfies the constraints imposed by the set system $\{S_{i,j}\}$. In this case, the clique of size k in G will be exactly the neighborhood of v in G' (not including v itself). Call $N(v)$ the neighborhood of v in G'. Note the degree of v in G' must be k, by definition of S_{vv}, i.e. $|N(v)| = k$. Consider a distinct pair of vertices i,j in $N(v)$. The vertices i and j have at least one common neighbor in G', namely v, because both are in $N(v)$, meaning that $A(G')^2_{ij} \geq 1$. But if (i,j) is not an edge in G then $S_{ij} = \{0\}$ by the definition of S_{ij}, a contradiction, forcing $N(v)$ to be a clique as required. $\square$


(a) 101st Senate (1989-1990)


(b) 106th Senate (1999-2000)

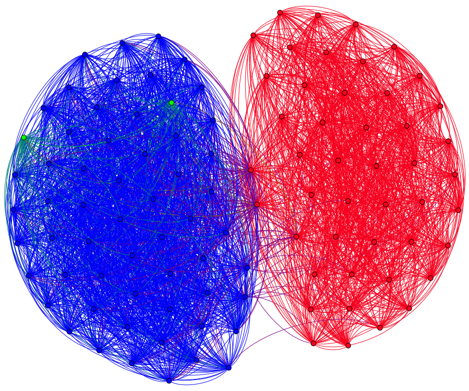
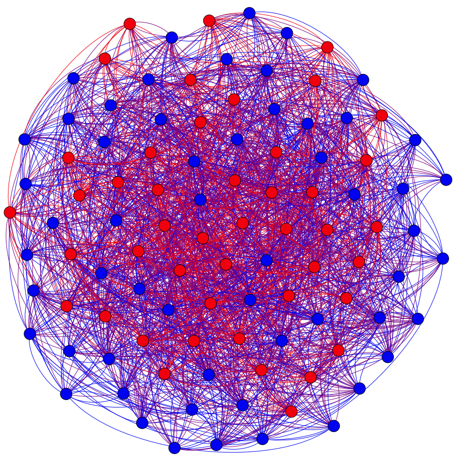
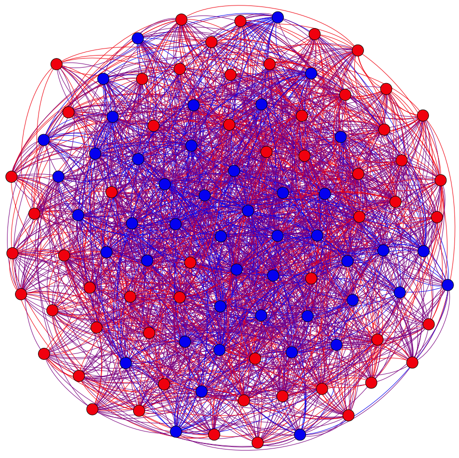

(c) 113th Senate (2013-2014)

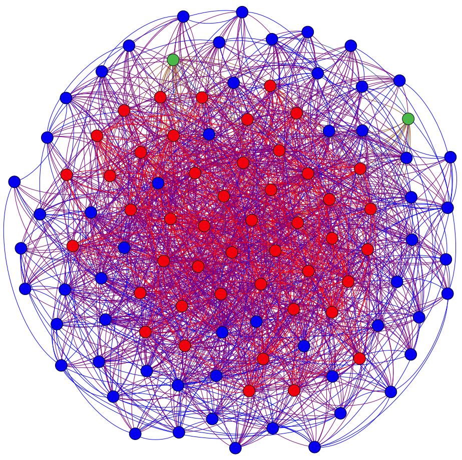
Figure 10: Three US Senates under the independent conversation model



(a) 101st Senate (1989-1990)



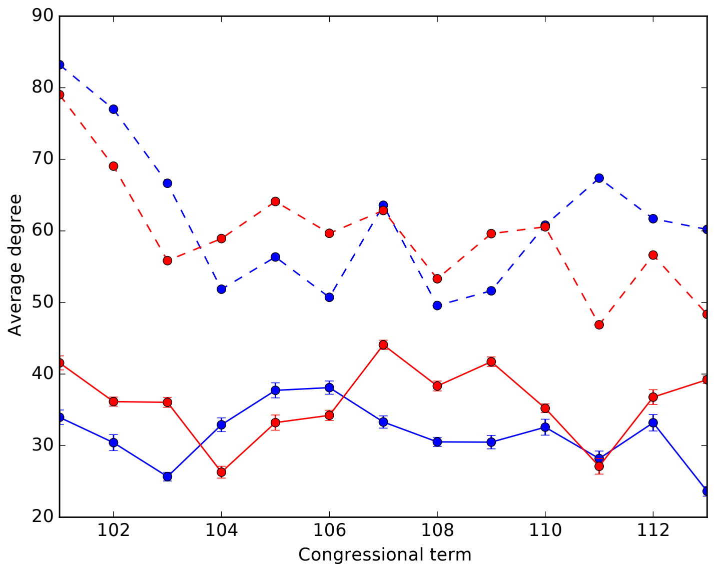
(b) 106th Senate (1999-2000)



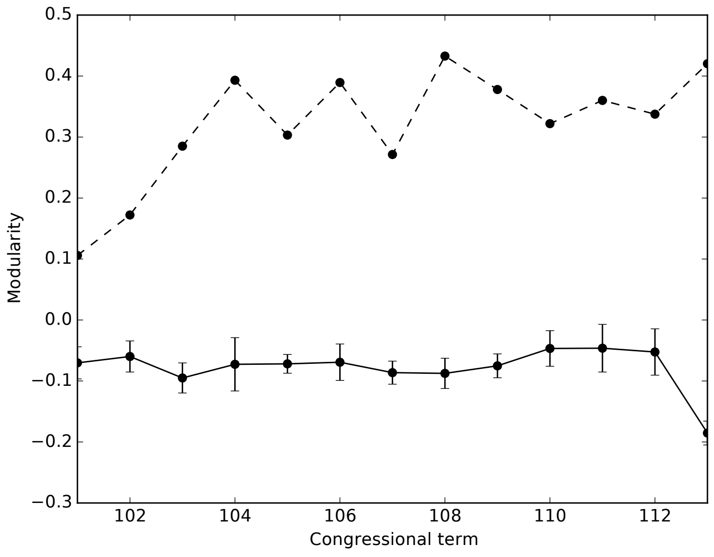
(c) 113th Senate (2013-2014)

Figure 11: Three US Senates under the common neighbor model

In Figure 10 and Figure 11, democrats are colored blue, Republicans are red, and Independents are green.



(a) Average degree. The blue and red lines are the average degree of the Democrats and the Republicans, respectively.



(b) Modularity between Democrats and Republicans. (Independents are included with the party with which they caucus.)

Figure 12: Statistics for the 101st-113th Senates

In Figure 12, dashed and solid lines are statistics for the independent conversation model and common neighbor model, respectively. Error bars represent one standard deviation, over 20 trials.

4.4.2 A heuristic approach

Unlike in the independent conversation model, we have no efficient algorithm for producing the social network under the common neighbor model. Hence, we employ a heuristic to find a graph that satisfies or comes close to satisfying the constraints imposed on it by the measured covariances.

Because of the computational hardness of this problem, we propose a heuristic approach to learn networks under the common neighbor model. This heuristic will be used in our experimental results in Section 4.5. Our heuristic, Algorithm 3, finds those pairs of vertices whose current expected covariance (assuming $p = 1/2$) is farthest away from the measured covariance and modifies the graph to decrease that gap.

Algorithm 3 Common neighbor heuristic

Input: $\{\hat{c}_{ij}\}$, measured covariances between voters i and j, and T, the number of iterations to run.

```
$G \sim \mathcal{G}(n, 1/2)$, $G_{best} := \emptyset$
$\{c_{ij}\} := \{\sigma(i,j)\}$    # calculate expected covariances
for 0 to $T$ do
    $i, j := \text{argmax}_{i,j} |c_{ij} - \hat{c}_{i,j}|$
    $G :=$ Modify$G(G, i, j, \hat{c}_{ij}, c_{ij})$
    $\{c_{ij}\} := \{\sigma(i,j)\}$    # update the expected covariances
    if $\sum_{i,j} c_{ij} - \hat{c}_{ij}$ is smallest so far then $G_{best} := G$
end for
return $G_{best}$
```

Algorithm 4 Modify$G$

```
Input: Graph $G$; vertices $i, j$; $\hat{c}_{ij}$, $c_{ij}$ the measured and expected covariances between $i$ and $j$.
  $unconnected := V(G) \setminus (\Gamma(i) \cup \Gamma(j) \cup \{i, j\})$
  $cn := \Gamma(i) \cap \Gamma(j)$
  if $c_{ij} - \hat{c}_{ij} > 0$ then
    randomize among whichever of these are available:
    1: x := random($i, j$), y := random($unconnected$)
       add edge $(x, y)$ to $G$
    2: x := random($i, j$), y := random($cn$).
       delete edge $(x, y)$ from $G$
    3: y := random($cn$)
       delete edges $(i, y)$ and $(j, y)$ from $G$
  else if $c_{ij} - \hat{c}_{ij} < 0$ then
    randomize among whichever of these are available:
    1: y := random($unconnected$)
       add edges $(i, y)$ and $(j, y)$ to $G$
    2: y := random($\Gamma(i) \setminus cn$)
       add $(i, y)$ or delete $(j, y)$ from $G$ randomly
    3: y := random($\Gamma(j) \setminus cn$)
       add $(j, y)$ or delete $(i, y)$ from $G$ randomly
  end if
  return $G$
  # where random(.) selects an element of its input u.a.r.
```

The local changes we want to make clearly cannot just consist of adding/removing single edges: Say the covariance between a given pair of vertices needs to go up and those vertices' neighborhoods are currently empty. The only way to increase the covariance is to add at least two edges: (i, v) and (v, j) for some other vertex v. The natural compromise is then to add or remove the minimal number of edges (either one or two) to change the covariance, as seen in Algorithm 4.

4.5 Experimental Results

In this section, we test our algorithms on United States Senate roll call votes. We examine each two-year congressional session as one voting network. Each Senate member is an agent who either votes for the bill in question, against, or does not vote (either because the senator served only part of the term or because the senator just didn't vote on the bill), yielding votes from the set $\{-1, 0, 1\}$.

Obviously, our models are simplifications — they don't take into account evolution of opinion, nor do they take into account the possibility of anti-correlated voters. Even assuming that either model is representative, when presented real data, the parameter p is not given, as is assumed above. The algorithms we present assume that $p = 1/2$, which is not necessarily the case. Finally, we are given a fixed amount of data, independent of the number of voters.

Despite these limitations, for the independent conversation model, our covariance test, which forms the basis for Theorem 7, results in intuitive behavior. Note that while our model assumes binary votes, our covariance test is general enough to handle such votes — covariance is calculated between the $\{-1, 0, 1\}$-valued votes and the threshold remains the same as in the original covariance test.[1] Examples

[1]We do, however, use the unbiased sample covariance instead of the biased sample covariance, as the assumption that $p = 1/2$ no longer necessarily holds, despite the analysis of the algorithm assuming it.

of the results from this covariance test on the US Senate are shown in Figure 10. Given the highly structured nature of these graphs, it is possible to recover senators' places on the left/right political spectrum, but since this is not the focus of this paper, we do not go into any further detail here.

For the common neighbor model, we use Algorithm 3. Examples of results of this heuristic run on US Senate data are shown in Figure 11. Graphs under this model appear to be very different from those found using the covariance test under the independent conversation model.

To demonstrate these marked differences, in Figure 12 we give modularity values and average degrees of Democrats and Republicans under both models for the period 1989-2014 (corresponding to the 101st through 113th Congresses). Modularity is a standard measure of the amount of division between communities (54). Both average degree and modularity are much higher under the independent conversation model (dashed lines in Figure 12) than under the common neighbor model (solid lines). Since the heuristic is randomized, we average these statistics over twenty graphs, each of which is an independent run of the heuristic with 100,000 rounds.

4.6 Conclusion

In this paper we derive algorithms and lower bounds for recovering graphs from their vertices' votes under two distinct models. We also present experiments on the U.S. Senate voting network. In the independent conversation model, we show when the graph is recoverable using only a polynomial number of votes. However, if we want to instead take a maximum likelihood approach to recovering graphs, then the task becomes computationally hard.

The common neighbor model, on the other hand, leads to significantly different results. Not only is it impossible to recover the graph using only polynomially many votes, finding a graph whose votes' covariances are close to the observed covariances leads to having to solve a hard problem (the generalized squared adjacency problem).

This implies that these models really are very different from each other, despite their very similar definitions. This is strong evidence that much care needs to be taken when choosing voting models for network inference. Experiments on U.S. Senate roll call data support this conclusion.

CHAPTER 5

NETWORK CONSTRUCTION WITH ORDERED CONSTRAINTS

This chapter was previously published as Yi Huang, Mano Vikash Janardhanan, Lev Reyzin: Network Construction with Ordered Constraints. *arXiv preprint: 1702.07292*

5.1 Introduction

In this paper, we study the problem of recovering a network after observing how information propagates through the network. Consider how a tweet (through "retweeting" or via other means) propagates through the Twitter network – we can observe the identities of the people who have retweeted it and the timestamps when they did so, but may not know, for a fixed user, via whom he got the original tweet. So we see a chain of users for a given tweet. This chain is semi-ordered in the sense that, each user retweets from some one before him in the chain, but not necessarily the one directly before him. Similarly, when a virus such as Ebola spreads, each new patient in an outbreak is infected from some one who has previously been infected, but it is often not immediately clear from whom.

In a graphical social network model with nodes representing users and edges representing links, an "outbreak" illustrated above is captured exactly by the concept of an **ordered constraint** which we will define formally below. One could hope to be able to learn something about the structure of the network by observing repeated outbreaks, or a sequence of ordered constraints.

Formally we call our problem **Network Construction with Ordered Constraints** and define it as follows. Let $V = \{v_1, \dots, v_n\}$ be a set of vertices. An **ordered constraint** $\mathcal{O}$ is an ordering on a subset of V of size $s \geq 2$. The constraint $\mathcal{O} = (v_{k_1}, \ldots, v_{k_s})$ is satisfied if for any $2 \leq i \leq s$, there exists at least one $1 \leq j < i$ such that the edge $e = \left\{v_{k_j}, v_{k_i}\right\}$ is included in a solution. Given a collection of ordered constraints $\{\mathcal{O}_1, \ldots, \mathcal{O}_r\}$, the task is to construct a set E of edges among the vertices V such that all the ordered constraints are satisfied and $|E|$ is minimized.

We can see that our newly defined problem resides in a middle ground between path constraints, which are too rigid to be very interesting, and the well-studied subgraph connectivity constraints (17; 18; 19), which are more relaxed. The established subgraph connectivity constraints problem involves getting an arbitrary collection of connectivity constraints $\{S_1, \ldots, S_r\}$ where each $S_i \subset V$ and requires vertices in a given constraint to form a connected induced subgraph. The task is to construct a set E of edges satisfying the connectivity constraints such that $|E|$ is minimized.

We want to point out one key observation relating the ordered constraint to the connectivity constraint – an ordered constraint $\mathcal{O} = (v_{k_1}, \ldots, v_{k_s})$ is equivalent to $s-1$ connectivity constraints $S_2, \ldots, S_s$, where $S_i = \left\{v_{k_1}, \ldots, v_{k_i}\right\}$. We note that this observation plays an important role in several proofs in this paper which employ previous results on subgraph connectivity constraints – in particular, upper bounds from the more general case can be used in the ordered case (with some overhead), and our lower bounds apply to the general problem.

In the offline version of the Network Construction with Ordered Constraints problem, the algorithm is given all of the constraints all at once; in the **online** version of the problem, the constraints are given one by one to the algorithm, and edges must be added to satisfy each new constraint when it is given. Edges cannot be removed.

An algorithm is said to be **c-competitive** if the cost of its solution is less than c times OPT, where OPT is the best solution in hindsight (c is also called the competitive ratio). When we restrict the underlying graph in a problem to be a class of graphs, e.g. trees, we mean all the constraints can be satisfied, in an optimal solution (for the online case, in hindsight), by a graph from that class.

5.1.1 Past Work

In this paper we study the problem of network construction from ordered constraints. This is an extension of the more general model where constraints come unordered.

For the general problem, Korach and Stern (18) had some of the initial results, in particular for the case where the constraints can be optimally satisfied by a tree, they give a polynomial time algorithm that finds the optimal solution. In subsequent work, in (19) Korach and Stern considered this problem for the even more restricted problem where the optimal solution forms a tree, and all of the connectivity constraints must be satisfied by stars.

Then, Angluin et al. (17) studied the general problem, where there is no restriction on structure of the optimal solution, in both the offline and online settings. In the offline case, they gave nearly matching upper and lower bounds on the hardness of approximation for the problem. In the online case, they give a $O(n^{2/3} \log^{2/3} n)$-competitive algorithm against oblivious adversaries; we show that this bound can be drastically improved in the ordered version of the problem. They also characterized special classes of graphs, i.e. stars and paths, which we are also able to do herein for the ordered constraint case. Independently of that work, Chockler et al. (55) also nearly characterized the offline general case.

In a different line of work Alon et al. (56) explore a wide range of network optimization problems; one problem they study involves ensuring that a network with fractional edge weights has a flow of

1 over cuts specified by the constraints. Alon et al. (57) also study approximation algorithms for the Online Set Cover problem which have been shown by Angluin et al. (17) to have connections with Network Construction problems.

In related areas, Gupta et al. (58) considered a network design problem for pairwise vertex connectivity constraints. Moulin and Laigret (59) studied network connectivity constraints from an economics perspective. Another motivation for studying this problem is to discover social networks from observations. This and similar problems have also been studied in the learning context (60; 61; 62; 63).

Finally, in query learning, the problem of discovering networks from connectivity queries has been much studied (64; 65; 66; 67; 68; 69). In active learning of hidden networks, the object of the algorithm is to learn the network exactly. Our model is similar, except the algorithm only has the constraints it is given, and the task is to output the cheapest network consistent with the constraints.

5.1.2 Our results

In Section 5.2, we examine the offline problem, and show that the Network Construction problem is NP-Hard to approximate within a factor of $\Omega(\log n)$. A nearly matching upper bound comes from Theorem 2 of Angluin et al. (17).

In Section 5.3, we study online problem. For problems on n nodes, for r constraints, we give an $O\left((\log r + \log n)\log n\right)$ competitive algorithm against oblivious adversaries, and an $\Omega(\log n)$ lower bound (Section 5.3.1).

Then, for the special cases of stars and paths (Sections 5.3.2 and 5.3.3), we find asymptotic optimal competitive ratios of $3/2$ and 2, respectively. The proof of the latter uses a detailed analysis involving PQ-trees (70). The competitive ratios are asymptotic in n.

5.2 The offline problem

In this section, we examine the Network Construction with Ordered Constraints problem in the offline case. We are able to obtain the same lower bound as Angluin et al. (17) in the general connectivity constraints case.

Theorem 17. *If P≠NP, the approximation ratio of the* ***Network Construction with Ordered Constraints*** *problem is* $\Omega(\log n)$.

Proof. We prove the theorem by reducing from the Hitting Set problem. Let $(U, \mathcal{S})$ be a hitting set instance, where $U = \{u_1, \ldots, u_n\}$ is the universe, and $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a set of subsets of U. A subset $H \subset U$ is called a hitting set if $H \cap S_i \neq \emptyset$ for $i = 1, \ldots, m$. The objective of the Hitting Set problem is to minimize $|H|$. We know from (71; 72) that the Hitting Set problem cannot be approximated by any polynomial time algorithm within a ratio of $o(\log n)$ unless P=NP. Here we show that the Network Construction problem is inapproximable better than an $O(\log n)$ factor by first showing that we can construct a corresponding Network Construction instance to any given Hitting Set instance, and then showing that if there is a polynomial time algorithm that can achieve an approximation ratio $o(\log n)$ to the Network Construction problem, then the Hitting Set problem can also be approximated within in a ratio of $o(\log n)$, which is a contradiction.

We first define a Network Construction instance, corresponding to a given Hitting Set instance $(U, \mathcal{S})$, with vertex set $U \cup W$, where $W = \{w_1, \ldots, w_{n^c}\}$ for some $c > 2$. Note that we use the elements of the universe of hitting set instance as a part of the vertex set of Network Construction instance. The ordered constraints are the union of the following two sets:

- $\{(u_i, u_j)\}_{1 \le i < j \le n}$;
- $\{(S_k, w_l)\}_{S_k \in \mathcal{S}, 1 \le l \le n^c}$,

where by (S_k, w_l) we mean an ordered constraint with all vertices except the last one from a subset S_k of U, while the last vertex w_l is an element in W. The vertices from S_k are ordered arbitrarily.

We note that the first set of ordered constraints forces a complete graph on U, and the second set of ordering demands that there is at least one edge going out from each S_k connecting each element in W. Let $\mathcal{A}$ be an algorithm solving the Network Construction problem, and let E_l denote the set of edges added by $\mathcal{A}$ incident to w_l. Because of the second set of ordered constraints, the set $H_l = \{u \in U | \{u, w_l\} \in E_l\}$ is a hitting set of $\mathcal{S}$!

Let $H \subset U$ be any optimal solution to the hitting set instance, and denote by $\mathrm{OPT_H}$ the size of H. It is easy to see the two sets of ordered constraints can be satisfied by putting a complete graph on U and a complete bipartite graph between H and W. Hence the optimal solution to the Network Construction instance satisfies

$$\mathrm{OPT} \le \binom{n}{2} + n^c\,\mathrm{OPT_H},$$

where OPT is the minimum number of edges needed to solve the Network Construction instance. Let us assume that there is a polynomial time approximation algorithm to the Network Construction problem that adds ALG edges. Without loss of generality we can assume that the algorithm adds no edge among vertices in W, because any edge within W can be removed without affecting the correctness of the solution, which implies that ALG $= \binom{n}{2} + \sum_{l=1}^{n^c} |E_l|$. Now if ALG is $o(\log n \cdot \text{OPT})$, from the fact that $|H_l| = |E_l|$, we get

$$\begin{aligned}
\min_{1 \le l \le n^c} |H_l| \le \frac{\text{ALG} - \binom{n}{2}}{n^c} &= \frac{o\left(\log n \left(\binom{n}{2} + n^c\, \text{OPT}_\text{H}\right)\right) - \binom{n}{2}}{n^c} \\
&= o\left(\log n \cdot \text{OPT}_\text{H}\right),
\end{aligned}$$

which means by finding the smallest set H_{l_0} among all the H_ls, we get a hitting set that has size within an $o(\log n)$ factor of the optimal solution to the Hitting Set instance, which is a contradiction.

□

We also observe that the upper bound from the more general problem implies a bound in our ordered case. We note the upper and lower bounds match when $r = poly(n)$.

Corollary 18 (of Theorem 2 from Angluin et al. (17))**.** *There is a polynomial time $O(\log r + \log n)$-approximation algorithm for the Network Construction with Ordered Constraints problem on n nodes and r constraints.*

Proof. Observing that r ordered constraints imply at most nr unordered constraints on a graph with n nodes, we can use the $O(\log r)$ upper bound from Angluin et al. (17). □

5.3 The online problem

Here, we study the online problem, where constraints come in one at a time, and the algorithm must satisfy them by adding edges as the constraints arrive.

5.3.1 Arbitrary graphs

Theorem 19. *There is an $O((\log r + \log n)\log n)$ upper bound for the competitive ratio for the* ***Online Network Construction with Ordered Constraints*** *problem on n nodes and r ordered constraints against an oblivious adversary.*

Proof. To prove the statement, we first define the **Fractional Network Construction** problem, which has been shown by Angluin et al. (17) to have an $O(\log n)$-approximation algorithm. The upper bound is then obtained by applying a probabilistic rounding scheme to the fractional solution given by the approximation. The proof heavily relies on arguments developed by Buchbinder and Noar (73), and Angluin et al. (17).

In the **Fractional Network Construction** problem, we are also given a set of vertices and a set of constraints $\{S_1, \ldots, S_r\}$ where each S_i is a subset of the vertex set. Our task is to assign weights w_e to each edge e so that the maximum flow between each pair of vertices in S_i is at least 1. The optimization problem is to minimize $\sum w_e$. Since subgraph connectivity constraint is equivalent to requiring a maximum flow of 1 between each pair of vertices with edge weight $w_e \in \{0, 1\}$, the fractional network construction problem is the linear relaxation of the subgraph connectivity problem. Lemma 2 of Angluin et al. (17) gives an algorithm that multiplicatively updates the edge weights until all the flow constraints are satisfied. It also shows that the sum of weights given by the algorithm is upper bounded by $O(\log n)$ times the optimum.

As we pointed out in the introduction, an ordered constraint $\mathcal{O}$ is equivalent to a sequence of subgraph connectivity constraints. So in the first step, we feed the r sequences of connectivity constraints, each one is equivalent to an ordered constraint, to the approximation algorithm to the fractional network construction problem and get the edge weights. Then we apply a rounding scheme similar to the one considered by Buchbinder and Noar (73) to the weights. For each edge e, we choose t random variables $X(e,i)$ independently and uniformly from $[0,1]$, and let the threshold $T(e) = \min_{i=1}^{t} X(e,i)$. We add e to the graph if $w_e \geq T(e)$.

Since the rounding scheme has no guarantee to produce a feasible solution, the first thing we need to do is to determine how large t should be to make all the ordered constraints satisfied with high probability.

We note that an ordered constraint $\mathcal{O}_i = \{v_{i1}, v_{i2}, \ldots, v_{is_i}\}$ is satisfied if and only if the $(s_i - 1)$ connectivity constraints $\{v_{i1}, v_{i2}\}$, ..., $\{v_{i1}, \ldots, v_{is_i-1}, v_{is_i}\}$ are satisfied, which is equivalent, in turn, to the fact that there is an edge that goes across the $\left(\{v_{i1}, \ldots, v_{ij-1}\}, \{v_{ij}\}\right)$ cut, for $2 \leq j \leq s_i$. For any fixed cut C, the probability the cut is not crossed equals $\prod_{e \in C}(1 - w_e)^t \leq \exp\left(-t \sum_{e \in C} w_e\right)$. By the max-flow min-cut correspondence, we know that $\sum_{c \in C} w_e \geq 1$ in the fractional solution given by the approximation algorithm for all cuts $C = \left(\{v_{i1}, \ldots, v_{ij-1}\}, \{v_{ij}\}\right)$, $1 \leq i \leq r$, $2 \leq j \leq s_i$, and hence the probability that there exists at least one unsatisfied $\mathcal{O}_i$ is upper bounded by $rn \exp(-t)$. So $t = c(\log n + \log r)$, for any $c > 1$, makes the probability that the rounding scheme fails to produce a feasible solution approaches 0 as n increases.

Because the probability that e is added equals the probability that at least one $X(e,i)$ is less than w_e, and hence is upper bounded by $w_e t$, we get the expected number of edges added is upper bounded by $t \sum w_e$ by linearity of expectation. Since the fractional solution is upper bounded by $O(\log n)$ times the optimum of the fractional problem, which is upper bounded by any integral solution, our rounding scheme gives a solution that is $O\left((\log r + \log n) \log n\right)$ times the optimum.

□

Corollary 20. *If the number of ordered constraints $r = \text{poly}(n)$, then the algorithm above gives a $O\left((\log n)^2\right)$ upper bound for the competitive ratio against an oblivious adversary.*

Remark 1. *We can generalize Theorem 19 to the weighted version of the Online Network Construction with Ordered Constraints problem. In the weighted version, each edge $e = (u, v)$ is associated with a cost c_e and the task is to select edges such that the connectivity constraints are satisfied and $\sum c_e w_e$ is minimised where $w_e \in \{0, 1\}$ is a variable indicating whether an edge is picked or not and c_e is the cost of the edge. The same approach in the proof of Theorem 19 gives an upper bound of $O\left((\log r + \log n) \log n\right)$ for the competitive ratio of the weighted version of the Online Network Construction with Ordered Constraints problem.*

For an lower bound for the competitive ratio for the Online Network Construction with Ordered Constraints problem against an oblivious adversary, we study the **Online Permutation Hitting Set** problem defined below: Let k be a positive integer, and let π be a permutation on $[k]$. Define sets

$$P_\pi^i = \{\pi(1), \pi(2), \ldots, \pi(i)\} \quad \text{for } i = 1, \ldots, k,$$

Definition 12 (Online Permutation Hitting Set). *Let π be a permutation on $[k]$ and let $\left(P_\pi^k, P_\pi^{k-1}, \ldots, P_\pi^1\right)$ arrive one at a time. A solution to the Online Permutation Hitting Set problem is a sequence of set $H_1 \subset H_2 \subset \cdots \subset H_k$ such that $H_i \cap P_\pi^{k+1-i} \neq \emptyset$ for $i = 1, \ldots, k$.*

Lemma 21. *The expected size of H_k for any algorithm that solve the* ***Online Permutation Hitting Set*** *problem over the space of all permutations on $[k]$ with uniform distribution is lower bounded by $h_k = \sum_{i=1}^{k} \frac{1}{i}$.*

Proof. We first show that the lower bound is achieved by a randomized algorithm $\mathcal{A}^0$, and then we show that no other randomized algorithm could do better than $\mathcal{A}^0$.

We start by describing $\mathcal{A}^0$. Upon seeing P_π^{k+1-i}, $\mathcal{A}^0$ sets H_i using the following rule:

$$H_i = \begin{cases} H_{i-1} & \text{if } H_{i-1} \cap P_\pi^{k+1-i} \neq \emptyset \\ H_{i-1} \cup \{a\} & \text{if } H_{i-1} \cap P_\pi^{k+1-i} = \emptyset \end{cases},$$

where a is a random point in P_π^{k+1-i}. Let E_i denote the expected size of the final output of $\mathcal{A}^0$ on permutations on $[i]$ for all $i = 1, \ldots, k$. By symmetry, without loss of generality, we assume that $\mathcal{A}^0$ add 1 upon receiving $P_\pi^k = [k]$, then we have

$$E_k = 1 + \sum_{i=1}^{k} E_i \mathbb{P}\left(\pi(i) = 1\right) = 1 + \frac{1}{k} \sum_{i=1}^{k-1} E_i$$

The first equality is because $\mathcal{A}^0$ doesn't need to add more points until it receives P_π^{k-i} if $\pi(i) = 1$, and the second equality is because that all i has the same probability to be mapped to 1. To show that $E_k = h_k$, we first verify that the harmonic series satisfies the same recursive relation. In fact, we have

$$\begin{aligned} 1 + \frac{1}{k}\sum_{i=1}^{k-1} h_i &= \frac{1}{k}\left(k + \sum_{i=1}^{k-1}\sum_{j=1}^{i}\frac{1}{j}\right) = \frac{1}{k}\left(k + \sum_{i=1}^{k-1}(k-i)\cdot\frac{1}{i}\right) \\ &= \frac{1}{k}\left(k + \sum_{i=1}^{k-1}\left(k\cdot\frac{1}{i} - 1\right)\right) = \frac{1}{k}\left(1 + k\sum_{i=1}^{k-1}\frac{1}{i}\right) = h_k. \end{aligned}$$

Since $E_1 = h_1 = 1$, we have $E_k = h_k$ for all $k \in \mathbb{N}^+$.

Next, we show that $\mathcal{A}^0$ is in fact the best randomized algorithm in expectation. To this end, we need to show two things:

i. when $H_{i-1} \cap P_\pi^{k+1-i} \neq \emptyset$, adding point(s) is counter-productive;

ii. when $H_{i-1} \cap P_\pi^{k+1-i} = \emptyset$, adding more than one points is counter-productive.

To show i., let us assume that $H_{i-1} \cap P_\pi^{k+1-i} \neq \emptyset$ and j is the smallest integer in $1 \leq j \leq k+1-i$ such that $\pi(j)$ is contained in H_{i-1}. We show that adding one more point hurts the expectation, and the proof for adding more than one points follows the same fashion. $\mathcal{A}^0$ will wait till P_π^{j-1} and add a random point from $\{\pi(1), \pi(2), \ldots, \pi(j-1)\}$. Hence, $\mathcal{A}^0$ does not assign probabilities to any point in $\{\pi(j+1), \ldots, \pi(k+1-i)\}$. We note that adding any point in $\{\pi(j+1), \ldots, \pi(k+1-i)\}$ wouldn't help. Hence, any algorithm that assigns non-zero probabilities to $\{\pi(j+1), \ldots, \pi(k+1-i)\}$ will do worse than $\mathcal{A}^0$ in expectation.

To show ii., for simplicity, we show that adding two points upon seeing P_π^k hurts the expectation, and the proof for adding more than two points follows the same fashion. We show this by induction assuming that $\mathcal{A}^0$ is the best algorithm in expectation for all $i < k$. By symmetry, without loss of generality, we assume that $H_1 = \{1, 2\}$. We have

$$\begin{aligned}
& \mathbb{E}\left(|H_k| | H_1 = \{1,2\}\right) \\
= \; & 2 + \left(\sum \mathbb{E}\left(|H_k| | \pi^{-1}(1) < \pi^{-1}(2)\right)\right) \mathbb{P}\left(\pi^{-1}(1) < \pi^{-1}(2)\right) \\
& + \left(\sum \mathbb{E}\left(|H_k| | \pi^{-1}(1) > \pi^{-1}(2)\right)\right) \mathbb{P}\left(\pi^{-1}(1) > \pi^{-1}(2)\right)
\end{aligned}$$

$$\geq \; 2+2\cdot\frac{1}{2}\left(\frac{1}{k-1}\sum_{i=1}^{k-2}E_i\right)=2+\frac{1}{k-1}\sum_{i=1}^{k-2}E_i=1+E_{k-1}>E_k,$$

where the first inequality follows from i. and the inductive hypothesis. □

With the help of Lemma 21, we can prove the following lower bound.

Theorem 22. *There exists an $\Omega(\log n)$ lower bound for the competitive ratio for the **Online Network Construction with Ordered Constraints** problem against an oblivious adversary.*

Proof. The adversary divides the vertex set into two parts U and V, where $|U| = \sqrt{n}$ and $|V| = n - \sqrt{n}$, and gives the constraints as follows. First, it forces a complete graph in U by giving the constraint $\{u_i, u_j\}$ for each pair of vertices $u_i, u_j \in U$. At this stage both the algorithm and optimal solution will have a clique in U, which costs $\Theta(n)$.

Then, for each $v \in V$, first fix a random permutation π_v on U and give the ordered constraints

$$\mathcal{O}_{(v,i)} = (\pi_v(1), \pi_v(2), \ldots, \pi_v(i), v) \quad \text{for } i = 1, \ldots, \sqrt{n}.$$

First note that all these constraints can be satisfied by adding $e_v = \{\pi_v(1), v\}$ for each $v \in V$ which costs $\Theta(n)$. However, the adversary gives constraints in the following order:

$$\mathcal{O}_{(v,\sqrt{n})}, \mathcal{O}_{(v,\sqrt{n}-1)}, \ldots, \mathcal{O}_{(v,1)}.$$

To satisfy $O_{(v,i)}$, any algorithm just need to make sure that at least one point in P_π^i is chosen to connect to v, and hence the algorithm is in fact solving an instance of the Online Permutation Hitting Set problem on input π at this stage. By Proposition 21, we know that all algorithm solving the Online Permutation Hitting Set problem will add $\Omega\left(\log\sqrt{n}\right)$ points in expectation, and this shows that any algorithm to the Online Network Construction with Ordered Constraints problem needs to add $\Omega(n + n\log n)$ edges in expectation in total. This gives us the desired result because OPT $= O(n)$.

□

Now we study the online problem when it is known that an optimal graph can be a star or a path. These special cases are challenging in their own right and are often studied in the literature to develop more general techniques (17).

5.3.2 Stars

Theorem 23. *The optimal competitive ratio for the* ***Online Network Construction with Ordered Constraints*** *problem when the algorithm knows that an optimal solution forms a* ***star*** *is asymptotically* $3/2$.

Proof. For the lower bound, we note that the adversary can simply give $\mathcal{O}_i = (v_1, v_2, v_i)$ for all $i = 3, \dots, n$ obliviously for the first $n-2$ rounds. Then an algorithm, besides adding $\{v_1, v_2\}$ in the first round, can only choose from adding either $\{v_1, v_i\}$ or $\{v_2, v_i\}$, or both in each round. Note that v_1 or v_2 have to be the center since the first constraint ensures that $\{v_1, v_2\}$ is an edge. After the first $n-2$ rounds, the adversary counts the number of v_1 and v_2's neighbors, and chooses the one with fewer neighbors, say v_1, to be the center by giving the constraints (v_1, v_i) for all $i = 3, \dots, n$ where no edge (v_1, v_i) exists. Since the algorithm has to add at least $\lceil (n-2)/2 \rceil$ edges that are unnecessary in the hindsight, we get an asymptotic lower bound $3/2$.

For the upper bound, assume that either v_1 or v_2 is the center and the first ordered constraint is $\mathcal{O}_1$ is $(v_1, v_2, \dots)$, the algorithm works as follows:

1. It adds $\{v_1, v_2\}$ in the first round.
2. For any constraint (including the first) that starts with v_1 and v_2, the algorithm always adds edges of the form (v_1, v_k) or (v_2, v_k) where $k \neq 1, 2$ in such a way that the following two conditions hold:
   - Degree of v_1 and degree of v_2 differ by at most 1.
   - The degree of v_k is 1 for $k \neq 1, 2$
3. Upon seeing a constraint that does not start with v_1 and v_2, which reveals the center of the star, it connects the center to all vertices that are not yet connected to the center.

Since the algorithm adds, at most $n/2 - 1$ edges to the wrong center, this gives us an asymptotic upper bound $3/2$, which matches the lower bound.

□

5.3.3 Paths

In the next two theorems, we give matching lower and upper bounds (in the limit) for path graphs.

Theorem 24. *The **Online Network Construction with Ordered Constraints** problem when the algorithm knows that the optimal solution forms a **path** has an asymptotic lower bound of* 2 *for the competitive ratio.*

Proof. Let us name the vertices $v_1, v_2, v_3, \ldots, v_n$. For $3 \leq i \leq n$, define the **pre-degree** of a vertex v_i to be the number of neighbors v_i has in $\{v_1, v_2, v_3, \ldots, v_{i-1}\}$. Algorithm 5 below is a simple strategy the adversary can take to force $v_3, \ldots, v_n$ to all have pre-degree at least 2. Since any algorithm will add at least $2n-3$ edges, this gives an asymptotic lower bound of 2.

Algorithm 5 Forcing pre-degree to be at least 2

```
Give ordered constraint $\mathcal{O} = (v_1, v_2, v_3, \ldots, v_n)$ to the algorithm;
**for** $i = 3$ to $n$ **do**
  **if** the pre-degree of $v_i$ is at least 2 **then**
    continue;
  **else**
    pick a path on $\{v_1, v_2, v_3, \ldots, v_{i-1}\}$ (say $P_i$) that satisfies all the constraints up to this round (the
    existence of $P_i$ follows by induction) and an endpoint $u$ of the path that is not connected to $v_i$,
    and give the algorithm the constraint $(v_i, u)$;
  **end if**
**end for**
```

Suppose P_i was the path picked in round i (i.e. P_i satisfies all constraints up to round i). Then, P_i along with the edge (v_i, u) is a path that satisfies all constraints up to round $i+1$. Hence by induction, for all i, there is a path that satisfies all constraints given by the adversary up to round i.

$\square$

Theorem 25. *The competitive ratio for the **Online Network Construction with Ordered Constraints** problem when the algorithm knows that the optimal solution forms a **path** has an asymptotic upper bound of* 2*.*

Proof. For our algorithm matching the upper bound, we use the PQ-trees, introduced by Booth and Lueker (70), which keep track all consistent permutations of vertices given contiguous intervals of vertices, since vertices in any ordered constraint form a contiguous interval if the underlying graph is a path. Our analysis is based on ideas from Angluin et al. (17), who also use PQ-trees for analyzing the general problem.[1]

A **PQ-tree** is a tree whose leaf nodes are the vertices and each internal node is either a **p-node** or a **q-node**.

- A **p-node** has a two or more children of any type. The children of a p-node form a contiguous interval that can be in any order.
- A **q-node** has three or more children of any type. The children of a q-node form a contiguous interval, but can only be in the given order or its reverse.

Every time a new interval constraint comes, the tree update itself by identifying any of the eleven patterns, P0, P1,. . . , P6, and Q0, Q1, Q2, Q3, of the arrangement of nodes and replacing it with each correspondent replacement. The update fails when it cannot identify any of the patterns, in which case

[1]Angluin et al. (17) have a small error in their argument because their potential function fails to explicitly consider the number of p-nodes, which creates a problem for some of the PQ-tree updates. We fix this, without affecting their asymptotic bound. For the ordered constraints case, we are also able to obtain a much finer analysis.

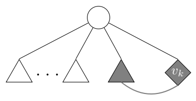
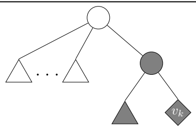
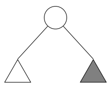
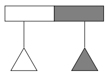
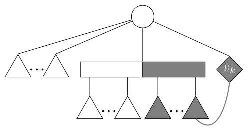
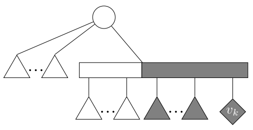
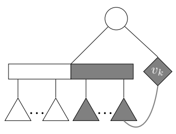
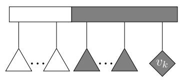
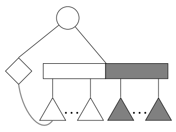
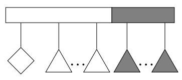
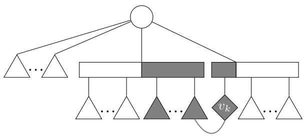
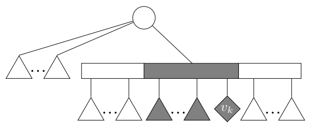
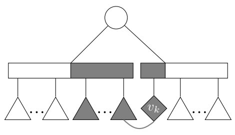
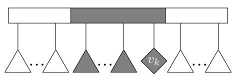
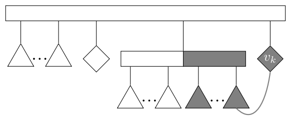
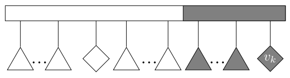
the contiguous intervals fail to produce any consistent permutation. We refer readers to Section 2 of Booth and Lueker (70) for a more detailed description of PQ-trees.

The reason we can use a PQ-tree to guide our algorithm is because of an observation made in Section 5.1 that each ordered constraint $(v_1, v_2, v_3, \ldots, v_{k-1}, v_k)$ is equivalent to $k-1$ connectivity constraints $S_2, \ldots, S_k$, where $S_i = \{v_1, \ldots, v_i\}$. Note that each connectivity constraint corresponds to an interval in the underlying graph. So upon seeing one ordered constraints, we reduce the PQ-tree with the equivalent interval constraints, *in order*. Then what our algorithm does is simply to add edge(s) to the graph every time a pattern is identified and replaced with its replacement, so that the graph satisfies all the seen constraints. Note that to reduce the PQ-tree with one interval constraint, there may be multiple patterns identified and hence multiple edges may be added.

Before running into details of how the patterns determine which edge(s) to add, we note that, without loss of generality, we can assume that the algorithm is in either one of the following two stages.

- The PQ-tree is about to be reduced with $\{v_1, v_2\}$.
- The PQ-tree is about to be reduced with $\{v_1, \ldots, v_k\}$, when the reductions with $\{v_1, v_2\}, \cdots,$ $\{v_1, \ldots, v_{k-1}\}$ have been done.

TABLE II Patterns and their replacements

	Pattern	Replacement
P2		
P3		
P4(1)		
P4(2)		
P5		
P6(1)		
P6(2)		
Q2		

Specific patterns and replacements that appear through the algorithm. P4(1) denotes the case of P4 where the top p-node is retained in the replacement and P4(2) denotes the case where the top p-node is deleted. The same is true for P6. P0, P1, Q0, and Q1 are just relabelling rules, and we have omitted them because no edges need to be added. We use the same shapes to represent p-nodes, q-nodes, and subtrees as in Booth and Lueker's paper (70) for easy reference, and we use diamonds to represent leaf nodes.

Because of the structure of constraints discussed above, we do not encounter all PQ-tree patterns in their full generality, but in the special forms demonstrated in Table II. Based on this, we make three important observations which can be verified by carefully examining how a PQ-tree evolves along with our algorithm.

1. The only p-node that can have more than two children is the root.
2. At least one of the two children of a non-root p-node is a leaf node.
3. For all q-nodes, there must be at least one leaf node in any two adjacent children. Hence, Q3 doesn't appear.

Now we describe how the edges are going to be added. Note that a PQ-tree inherently learns edges that appear in the optimum solution even when those edges are not forced by constraints. Apart from adding edges that are necessary to satisfy the constraints, our algorithm will also add any edge that the PQ-tree inherently learns. For all the patterns except Q2 such that a leaf node v_k is about to be added as a child to a different node, we can add one edge joining v_k to v_{k-1}. For all such patterns except Q2, it is obvious that this would satisfy the current constraint and all inherently learnt edges are also added.

For Q2, the PQ-tree could learn two edges. The first edge is (v_k, v_{k-1}). The second one is an edge between the leftmost subtree of the daughter q-node (call T_l) and the node to its left (call v_l). Based on Observation 3, v_l is a leaf. But based on the algorithm, one of these two edges is already added. Hence, we only need to add one edge when Q2 is applied. For P5, we add the edge as shown in Table II.

In Table III, we show how the terms in the potential function: $\sum_{p \in P} c(p)$, $|P|$, and $|Q|$ change according to the updates.

TABLE III Update of the potential function

	$\sum_{p\in P} c(p)$	$\|P\|$	$\|Q\|$	$-\Delta\Phi$	number of edges added
P2	1	1	0	$-a-b$	1
P3	-2	-1	1	$2a+b-c$	0
P4(1)	-1	0	0	a	1
P4(2)	-2	-1	0	$2a+b$	1
P5	-2	-1	0	$2a+b$	1
P6(1)	-1	0	-1	$a+c$	1
P6(2)	-2	-1	-1	$2a+b+c$	1
Q2	0	0	-1	c	1
Q3	0	0	-2	$2c$	1

Let us denote by P and Q the sets of p-nodes and q-nodes, respectively, and by $c(p)$ the number of children node p has. And let potential function ϕ of a tree T be defined as

$$\phi(T) = a \sum_{p \in P} c(p) + b|P| + c|Q|,$$

where a, b, and c are coefficients to be determined later.

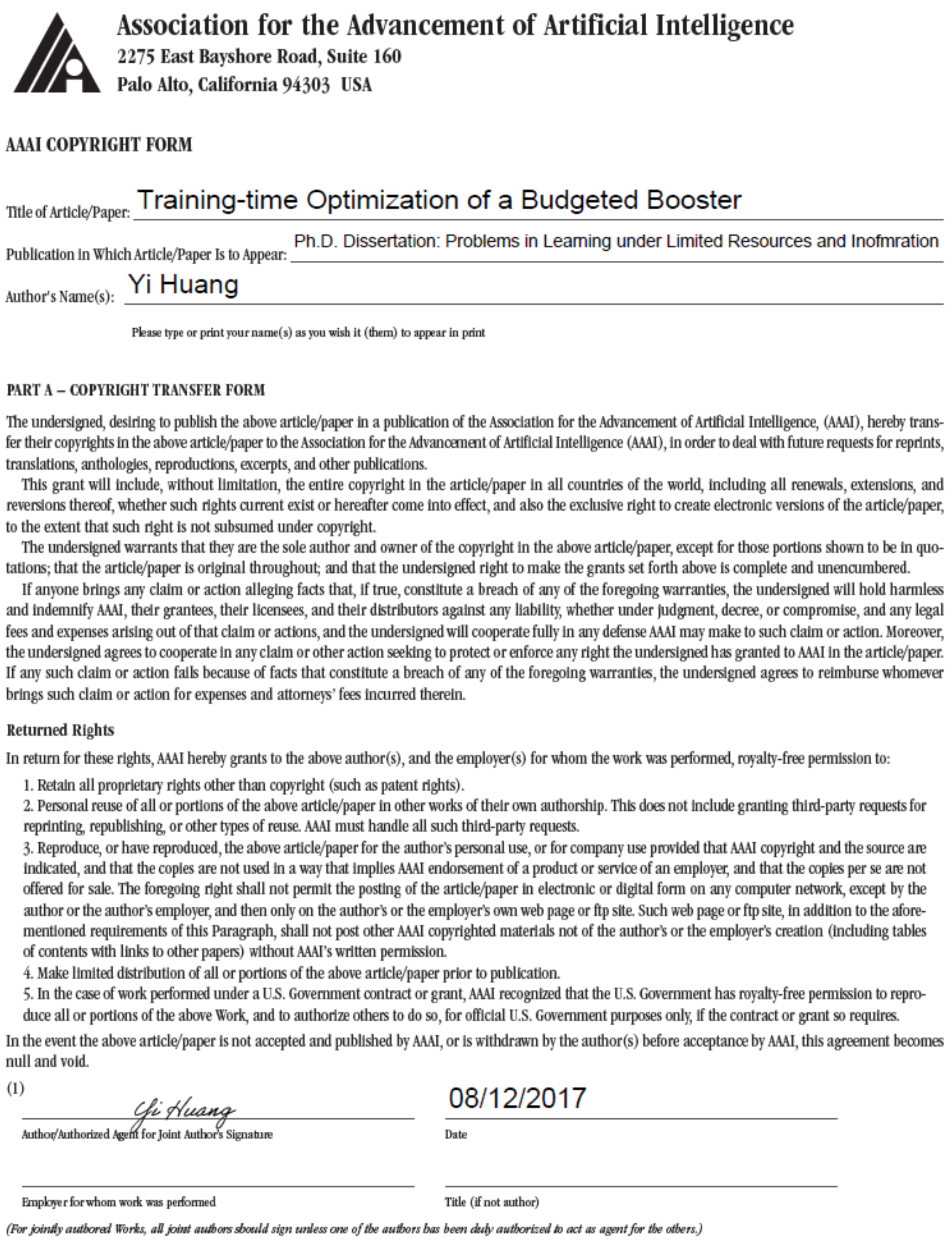
We want to upper bound the number of edges added for each pattern by the drop of potential function. We collect the change in the three terms in the potential function that each replacement causes in Table III, and we can solve a simple linear system to get that choosing $a = 2$, $b = -3$, and $c = 1$ is sufficient. For ease of analysis, we add a dummy vertex v_{n+1} that does not appear in any constraint. Now, the potential function starts at $2n - 1$ (a single p-node with $n + 1$ children) and decreases to 2 when a path is uniquely determined. Hence, the number of edges added by the algorithm is $2n - 3$, which gives the desired asymptotic upper bound.

$\square$

APPENDIX

This appendix contains copyright information.

AAAI



Association for the Advancement of Artificial Intelligence
2275 East Bayshore Road, Suite 160
Palo Alto, California 94303 USA

AAAI COPYRIGHT FORM

Title of Article/Paper: Training-time Optimization of a Budgeted Booster

Publication in Which Article/Paper Is to Appear: Ph.D. Dissertation: Problems in Learning under Limited Resources and Inofmration

Author's Name(s): Yi Huang

Please type or print your name(s) as you wish it (them) to appear in print

PART A – COPYRIGHT TRANSFER FORM

The undersigned, desiring to publish the above article/paper in a publication of the Association for the Advancement of Artificial Intelligence, (AAAI), hereby transfer their copyrights in the above article/paper to the Association for the Advancement of Artificial Intelligence (AAAI), in order to deal with future requests for reprints, translations, anthologies, reproductions, excerpts, and other publications.

This grant will include, without limitation, the entire copyright in the article/paper in all countries of the world, including all renewals, extensions, and reversions thereof, whether such rights current exist or hereafter come into effect, and also the exclusive right to create electronic versions of the article/paper, to the extent that such right is not subsumed under copyright.

The undersigned warrants that they are the sole author and owner of the copyright in the above article/paper, except for those portions shown to be in quotations; that the article/paper is original throughout; and that the undersigned right to make the grants set forth above is complete and unencumbered.

If anyone brings any claim or action alleging facts that, if true, constitute a breach of any of the foregoing warranties, the undersigned will hold harmless and indemnify AAAI, their grantees, their licensees, and their distributors against any liability, whether under judgment, decree, or compromise, and any legal fees and expenses arising out of that claim or actions, and the undersigned will cooperate fully in any defense AAAI may make to such claim or action. Moreover, the undersigned agrees to cooperate in any claim or other action seeking to protect or enforce any right the undersigned has granted to AAAI in the article/paper. If any such claim or action fails because of facts that constitute a breach of any of the foregoing warranties, the undersigned agrees to reimburse whomever brings such claim or action for expenses and attorneys' fees incurred therein.

Returned Rights

In return for these rights, AAAI hereby grants to the above author(s), and the employer(s) for whom the work was performed, royalty-free permission to:

1. Retain all proprietary rights other than copyright (such as patent rights).
2. Personal reuse of all or portions of the above article/paper in other works of their own authorship. This does not include granting third-party requests for reprinting, republishing, or other types of reuse. AAAI must handle all such third-party requests.
3. Reproduce, or have reproduced, the above article/paper for the author's personal use, or for company use provided that AAAI copyright and the source are indicated, and that the copies are not used in a way that implies AAAI endorsement of a product or service of an employer, and that the copies per se are not offered for sale. The foregoing right shall not permit the posting of the article/paper in electronic or digital form on any computer network, except by the author or the author's employer, and then only on the author's or the employer's own web page or ftp site. Such web page or ftp site, in addition to the aforementioned requirements of this Paragraph, shall not post other AAAI copyrighted materials not of the author's or the employer's creation (including tables of contents with links to other papers) without AAAI's written permission.
4. Make limited distribution of all or portions of the above article/paper prior to publication.
5. In the case of work performed under a U.S. Government contract or grant, AAAI recognized that the U.S. Government has royalty-free permission to reproduce all or portions of the above Work, and to authorize others to do so, for official U.S. Government purposes only, if the contract or grant so requires.

In the event the above article/paper is not accepted and published by AAAI, or is withdrawn by the author(s) before acceptance by AAAI, this agreement becomes null and void.

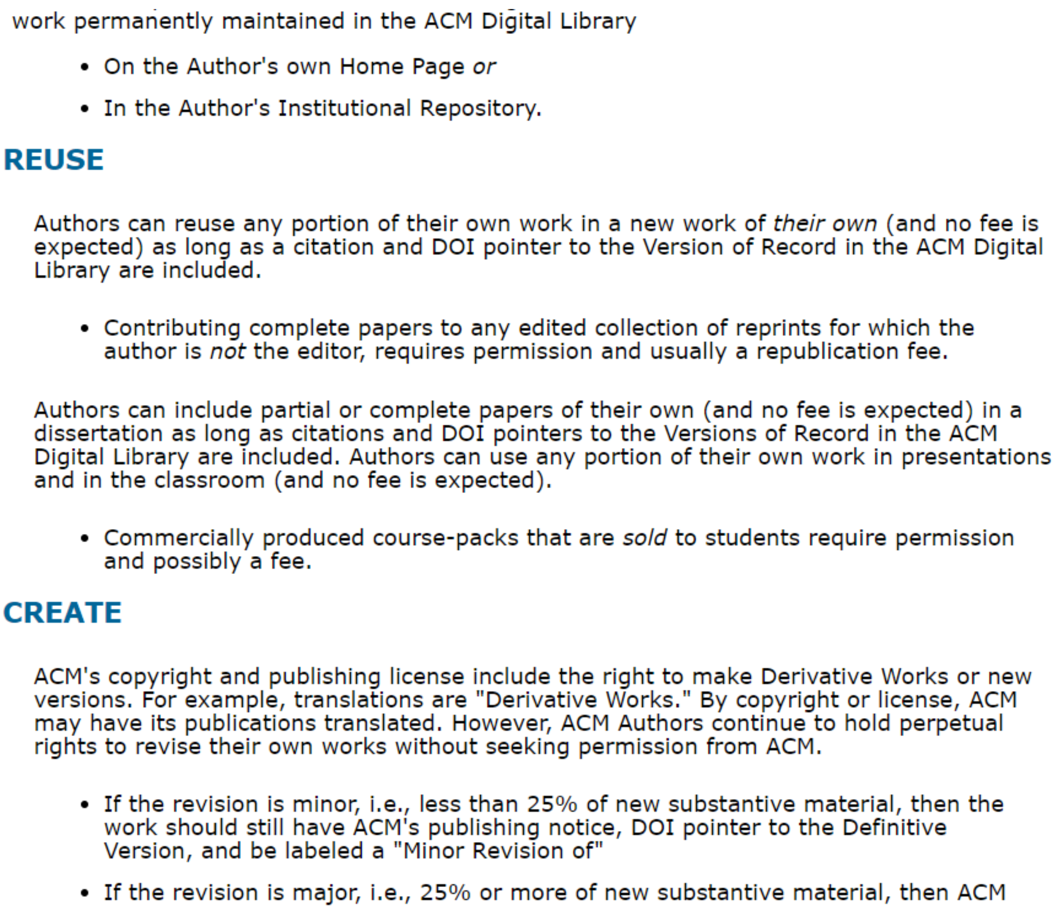
(1)

Yi Huang	08/12/2017
Author/Authorized Agent for Joint Author's Signature	Date
Employer for whom work was performed	Title (if not author)

(For jointly authored Works, all joint authors should sign unless one of the authors has been duly authorized to act as agent for the others.)

ACM

Screen shot from `http://authors.acm.org/main.html`. See the “RESUSE” section.



work permanently maintained in the ACM Digital Library

- On the Author's own Home Page *or*
- In the Author's Institutional Repository.

REUSE

Authors can reuse any portion of their own work in a new work of *their own* (and no fee is expected) as long as a citation and DOI pointer to the Version of Record in the ACM Digital Library are included.

- Contributing complete papers to any edited collection of reprints for which the author is *not* the editor, requires permission and usually a republication fee.

Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included. Authors can use any portion of their own work in presentations and in the classroom (and no fee is expected).

- Commercially produced course-packs that are *sold* to students require permission and possibly a fee.

CREATE

ACM's copyright and publishing license include the right to make Derivative Works or new versions. For example, translations are "Derivative Works." By copyright or license, ACM may have its publications translated. However, ACM Authors continue to hold perpetual rights to revise their own works without seeking permission from ACM.

- If the revision is minor, i.e., less than 25% of new substantive material, then the work should still have ACM's publishing notice, DOI pointer to the Definitive Version, and be labeled a "Minor Revision of"
- If the revision is major, i.e., 25% or more of new substantive material, then ACM

CITED LITERATURE

1. Huang, Y., Powers, B., and Reyzin, L.: Training-time optimization of a budgeted booster. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, pages 3583–3589, 2015.

2. Fish, B., Huang, Y., and Reyzin, L.: Recovering social networks by observing votes. In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016, pages 376–384, 2016.

3. Huang, Y., Janardhanan, M. V., and Reyzin, L.: Network construction with ordered constraints. CoRR, abs/1702.07292, 2017.

4. Reyzin, L.: Boosting on a budget: Sampling for feature-efficient prediction. In ICML, pages 529–536, 2011.

5. Grubb, A. and Bagnell, D.: Speedboost: Anytime prediction with uniform near-optimality. In AISTATS, pages 458–466, 2012.

6. Wald, A.: Sequential Analysis. Wiley, 1947.

7. Globerson, A. and Roweis, S. T.: Nightmare at test time: robust learning by feature deletion. In ICML, pages 353–360, 2006.

8. Conitzer, V.: The maximum likelihood approach to voting on social networks. In Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on, pages 1482–1487. IEEE, 2013.

9. Freund, Y. and Schapire, R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci., 55(1):119–139, 1997.

10. Ben-David, S. and Dichterman, E.: Learning with restricted focus of attention. In COLT, pages 287–296, New York, NY, USA, 1993. ACM.

11. Greiner, R., Grove, A. J., and Roth, D.: Learning cost-sensitive active classifiers. Artif. Intell., 139(2):137–174, 2002.

12. Gao, T. and Koller, D.: Active classification based on value of classifier. In Advances in Neural Information Processing Systems, pages 1062–1070, 2011.

13. Schwing, A. G., Zach, C., Zheng, Y., and Pollefeys, M.: Adaptive random forest–how many "experts" to ask before making a decision? In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pages 1377–1384. IEEE, 2011.

14. He, H., III, H. D., and Eisner, J.: Imitation learning by coaching. In NIPS, pages 3158–3166, 2012.

15. Xu, Z., Weinberger, K., and Chapelle, O.: The greedy miser: Learning under test-time budgets. In ICML, pages 1175–1182. ACM, July 2012.

16. Goles, E. and Olivos, J.: Periodic behaviour of generalized threshold functions. Discrete Mathematics, 30(2):187–189, 1980.

17. Angluin, D., Aspnes, J., and Reyzin, L.: Network construction with subgraph connectivity constraints. Journal of Combinatorial Optimization, 29(2):418–432, 2015.

18. Korach, E. and Stern, M.: The clustering matroid and the optimal clustering tree. Mathematical Programming, 98(1-3):385–414, 2003.

19. Korach, E. and Stern, M.: The complete optimal stars-clustering-tree problem. Discrete Applied Mathematics, 156(4):444–450, 2008.

20. Schapire, R. E.: The strength of weak learnability. Machine learning, 5(2):197–227, 1990.

21. Freund, Y. and Schapire, R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci., 55(1):119–139, 1997.

22. Schapire, R. and Freund, Y.: Boosting: Foundations and Algorithms. Adaptive computation and machine learning. MIT Press, 2012.

23. Mukherjee, I. and Schapire, R. E.: A theory of multiclass boosting. Journal of Machine Learning Research, 14(Feb):437–497, 2013.

24. Drucker, H.: Improving regressors using boosting techniques. In ICML, volume 97, pages 107–115, 1997.

25. Rokach, L. and Maimon, O.: Data mining with decision trees: theory and applications. World scientific, 2014.

26. Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A.: Classification and regression trees. CRC press, 1984.

27. Arora, S. and Barak, B.: Computational complexity: a modern approach. Cambridge University Press, 2009.

28. Vazirani, V. V.: Approximation algorithms. Springer Science & Business Media, 2013.

29. Feige, U.: A threshold of ln n for approximating set cover. Journal of the ACM (JACM), 45(4):634–652, 1998.

30. Chernoff, H.: Sequential Analysis and Optimal Design. SIAM, 1972.

31. Cesa-Bianchi, N., Shalev-Shwartz, S., and Shamir, O.: Efficient learning with partially observed attributes. CoRR, abs/1004.4421, 2010.

32. Pelossof, R., Jones, M., and Ying, Z.: Speeding-up margin based learning via stochastic curtailment. In ICML/COLT Budgeted Learning Workshop, Haifa, Israel, June 25 2010.

33. Sun, P. and Zhou, J.: Saving evaluation time for the decision function in boosting: Representation and reordering base learner. In ICML, 2013.

34. Reyzin, L. and Schapire, R. E.: How boosting the margin can also boost classifier complexity. In ICML, pages 753–760, 2006.

35. Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S.: Boosting the margin: A new explanation for the effectiveness of voting methods. the Annals of Statistics, 26(5):1651–1686, 1998.

36. Amelio, A. and Pizzuti, C.: Analyzing voting behavior in italian parliament: group cohesion and evolution. In Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on, pages 140–146. IEEE, 2012.

37. Jakulin, A., Buntine, W., La Pira, T. M., and Brasher, H.: Analyzing the us senate in 2003: Similarities, clusters, and blocs. Political Analysis, 17(3):291–310, 2009.

38. Macon, K. T., Mucha, P. J., and Porter, M. A.: Community structure in the united nations general assembly. Physica A: Statistical Mechanics and its Applications, 391(1):343–361, 2012.

39. Porter, M. A., Mucha, P. J., Newman, M. E., and Friend, A. J.: Community structure in the united states house of representatives. Physica A: Statistical Mechanics and its Applications, 386(1):414–438, 2007.

40. Waugh, A. S., Pei, L., Fowler, J. H., Mucha, P. J., and Porter, M. A.: Party polarization in congress: A network science approach. arXiv preprint arXiv:0907.3509, 2009.

41. Zhang, Y., Friend, A., Traud, A. L., Porter, M. A., Fowler, J. H., and Mucha, P. J.: Community structure in congressional cosponsorship networks. Physica A: Statistical Mechanics and its Applications, 387(7):1705–1712, 2008.

42. Chen, W., Lakshmanan, L. V., and Castillo, C.: Information and influence propagation in social networks. Synthesis Lectures on Data Management, 5(4):1–177, 2013.

43. Kempe, D., Kleinberg, J., and Tardos, É.: Maximizing the spread of influence through a social network. Theory OF Computing, 11(4):105–147, 2015.

44. Procaccia, A. D., Shah, N., and Sodomka, E.: Ranked voting on social networks. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, pages 2040–2046, 2015.

45. Tsang, A., Doucette, J. A., and Hosseini, H.: Voting with social influence: Using arguments to uncover ground truth. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015, pages 1841–1842, 2015.

46. Frischknecht, S., Keller, B., and Wattenhofer, R.: Convergence in (social) influence networks. In Distributed Computing, pages 433–446. Springer, 2013.

47. Schaul, K.: A quick puzzle to tell whether you know what people are thinking, October 2015. [The Washington Post; posted online 09-October-2015].

48. Grabisch, M. and Rusinowska, A.: A model of influence in a social network. Theory and Decision, 69(1):69–96, 2008.

49. Grandi, U., Lorini, E., and Perrussel, L.: Propositional opinion diffusion. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015, pages 989–997, 2015.

50. Schwind, N., Inoue, K., Bourgne, G., Konieczny, S., and Marquis, P.: Belief revision games. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA., pages 1590–1596, 2015.

51. Andris, C., Lee, D., Hamilton, M. J., Martino, M., Gunning, C. E., and Selden, J. A.: The rise of partisanship and super-cooperators in the US House of Representatives. PLoS ONE, 10(4), 2015.

52. Motwani, R. and Sudan, M.: Computing roots of graphs is hard. Discrete Applied Mathematics, 54(1):81–88, 1994.

53. Penn, M. and Tennenholtz, M.: On multi-object auctions and matching theory: Algorithmic aspects. In Graph Theory, Combinatorics and Algorithms, pages 173–188. Springer, 2005.

54. Newman, M. E. and Girvan, M.: Finding and evaluating community structure in networks. Physical Review E, 69(2):026113, 2004.

55. Chockler, G., Melamed, R., Tock, Y., and Vitenberg, R.: Constructing scalable overlays for pub-sub with many topics. In Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, pages 109–118. ACM, 2007.

56. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., and Naor, J. S.: A general approach to online network optimization problems. ACM Transactions on Algorithms (TALG), 2(4):640–660, 2006.

57. Alon, N., Awerbuch, B., and Azar, Y.: The online set cover problem. In Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, pages 100–105. ACM, 2003.

58. Gupta, A., Krishnaswamy, R., and Ravi, R.: Online and stochastic survivable network design. SIAM Journal on Computing, 41(6):1649–1672, 2012.

59. Moulin, H. and Laigret, F.: Equal-need sharing of a network under connectivity constraints. Games and Economic Behavior, 72(1):314–320, 2011.

60. Angluin, D., Aspnes, J., and Reyzin, L.: Optimally learning social networks with activations and suppressions. In International Conference on Algorithmic Learning Theory, pages 272–286. Springer, 2008.

61. Angluin, D., Aspnes, J., and Reyzin, L.: Inferring social networks from outbreaks. In International Conference on Algorithmic Learning Theory, pages 104–118. Springer, 2010.

62. Gomez-Rodriguez, M., Leskovec, J., and Krause, A.: Inferring networks of diffusion and influence. ACM Transactions on Knowledge Discovery from Data (TKDD), 5(4):21, 2012.

63. Saito, K., Nakano, R., and Kimura, M.: Prediction of information diffusion probabilities for independent cascade model. In International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, pages 67–75. Springer, 2008.

64. Alon, N. and Asodi, V.: Learning a hidden subgraph. SIAM Journal on Discrete Mathematics, 18(4):697–712, 2005.

65. Alon, N., Beigel, R., Kasif, S., Rudich, S., and Sudakov, B.: Learning a hidden matching. SIAM Journal on Computing, 33(2):487–501, 2004.

66. Angluin, D. and Chen, J.: Learning a hidden graph using o (logn) queries per edge. Journal of Computer and System Sciences, 74(4):546–556, 2008.

67. Beigel, R., Alon, N., Kasif, S., Apaydin, M. S., and Fortnow, L.: An optimal procedure for gap closing in whole genome shotgun sequencing. In Proceedings of the fifth annual international conference on Computational biology, pages 22–30. ACM, 2001.

68. Grebinski, V. and Kucherov, G.: Reconstructing a hamiltonian cycle by querying the graph: Application to dna physical mapping. Discrete Applied Mathematics, 88(1):147–165, 1998.

69. Reyzin, L. and Srivastava, N.: Learning and verifying graphs using queries with a focus on edge counting. In International Conference on Algorithmic Learning Theory, pages 285–297. Springer, 2007.

70. Booth, K. S. and Lueker, G. S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. Journal of Computer and System Sciences, 13(3):335–379, 1976.

71. Feige, U.: A threshold of ln n for approximating set cover. Journal of the ACM (JACM), 45(4):634–652, 1998.

72. Raz, R. and Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pages 475–484. ACM, 1997.

73. Buchbinder, N. and Naor, J.: The design of competitive online algorithms via a primal: dual approach. Foundations and Trends® in Theoretical Computer Science, 3(2–3):93–263, 2009.

74. Reyzin, L.: Boosting on a feature budget. In ICML/COLT Budgeted Learning Workshop, Haifa, Israel, June 25 2010.

75. Sutton, R. S. and Barto, A. G.: Reinforcement Learning: An Introduction. MIT Press, 1998.

76. Grove, A. J. and Schuurmans, D.: Boosting in the limit: Maximizing the margin of learned ensembles. In AAAI/IAAI, pages 692–699, 1998.

77. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 58:13–30, 1963.

78. Breiman, L.: Prediction games and arcing classifiers. Neural Computation, 11(7):1493–1517, 1999.

79. Mason, L., Bartlett, P. L., and Baxter, J.: Direct optimization of margins improves generalization in combined classifiers. In NIPS, pages 288–294, 1998.

80. Langford, J., Seeger, M., and Megiddo, N.: An improved predictive accuracy bound for averaging classifiers. In ICML, pages 290–297, 2001.

81. Wang, L., Sugiyama, M., Yang, C., Zhou, Z.-H., and Feng, J.: On the margin explanation of boosting algorithms. In COLT, pages 479–490, 2008.

82. Schapire, R. E.: The boosting approach to machine learning: An overview. In Nonlinear Estimation and Classification. Springer, 2003.

83. Schapire, R. E. and Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297–336, 1999.

84. Breiman, L.: Random forests. Machine Learning, 45(1):5–32, 2001.

85. Winkler, P.: Puzzled delightful graph theory. Communications of the ACM, 51(8):104–104, 2008.

86. Valiant, L. G.: A theory of the learnable. Communications of the ACM, 27(11):1134–1142, 1984.

87. Kearns, M. and Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. Machine Learning: From Theory to Applications, pages 29–49, 1993.

88. Zhu, J., Zou, H., Rosset, S., Hastie, T., et al.: Multi-class adaboost. Statistics and its Interface, 2(3):349–360, 2009.

VITA

EDUCATION

B.Sc., Nankai University, Tianjin, China, 2004

M.S., Chinese Academy of Sciences, Beijing, China, 2008

M.S., Georgia Institute of Technology, Atlanta, US, 2010

TEACHING

Teaching/Research Assistant, *University of Illinois at Chicago,* Chicago, Il, Fall 2012 – Summer 2017

Teaching/Research Assistant, *Geogia Institute of Technology,* Atlanta, GA, Fall 2008 – Spring 2010

PUBLICATIONS

Training-Time Optimization of a Budgeted Booster. With Brian Powers and Lev Reyzin. In Proceedings of the 2015 International Joint Conferences on Artificial Intelligence Organization (IJCAI).

Recovering Social Networks by Observing Votes. With Benjamin Fish and Lev Reyzin. In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS).

Network Construction with Ordered Constraints. With Mano Vikash Janardhanan and Lev Reyzin. *arXiv preprint: 1702.07292*