UNIVERSITY COLLEGE LONDON

DOCTORAL THESIS

Efficient Algorithms for Online Learning over Graphs

Author:
Stephen Ugo PASTERIS

Supervisor:
Dr. Mark HERBSTER

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Department of Computer Science

September 21, 2016

Declaration of Authorship

I, Stephen Ugo PASTERIS, declare that this thesis titled, "Efficient Algorithms for Online Learning over Graphs" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSITY COLLEGE LONDON

Abstract

Department of Computer Science

Doctor of Philosophy

Efficient Algorithms for Online Learning over Graphs

by Stephen Ugo PASTERIS

In this thesis we consider the problem of online learning with labelled graphs, in particular designing algorithms that can perform this problem quickly and with low memory requirements. We consider the tasks of Classification (in which we are asked to predict the labels of vertices) and Similarity Prediction (in which we are asked to predict whether two given vertices have the same label). The first half of the thesis considers non-probabilistic online learning, where there is no probability distribution on the labelling and we bound the number of mistakes of an algorithm by a function of the labelling's complexity (i.e. its “naturalness"), often the cut-size. The second half of the thesis considers probabilistic machine learning in which we have a known probability distribution on the labelling. Before considering probabilistic online learning we first analyse the junction tree algorithm, on which we base our online algorithms, and design a new version of it, superior to the otherwise current state of the art. Explicitly, the novel contributions of this thesis are as follows:

- A new algorithm for online prediction of the labelling of a graph which has better performance than previous algorithms on certain graph and labelling families.
- Two algorithms for online similarity prediction on a graph (a novel problem solved in this thesis). One performs very well whilst the other not so well but which runs exponentially faster.
- A new (better than before, in terms of time and space complexity) state of the art junction tree algorithm, as well as an application of it to the problem of online learning in an Ising model.
- An algorithm that, in linear time, finds the optimal junction tree for online inference in tree-structured Ising models, the resulting online junction tree algorithm being far superior to the previous state of the art.

All claims in this thesis are supported by mathematical proofs.

Acknowledgements

I would like to acknowledge Mark Herbster, Fabio Vitale, Claudio Gentile and Shaona Ghosh, who have all contributed to this thesis.

I would also like to acknowledge the EPSRC Doctoral Training Centre of Financial Computing and Analytics for funding my PhD, and all my family for their support throughout.

Contents

Chapter 1

Introduction

In this thesis we introduce several efficient algorithms for online classification and online similarity prediction on networked data. The computational model used by this thesis is the real random access machine. i.e. we assume that all basic operations such as arithmetic operations and memory reads/writes take constant time, storing any number or pointer takes constant space, and indirect addressing is supported. We now give the preliminary definitions required by the thesis:

Preliminary definitions: The symbol $:=$ is used for definition: e.g. $x := y$ means "x is defined to be equal to y". Given $a \in \mathbb{N}$ we define $\mathbb{N}_a$ to be equal to the set of the first a natural numbers: i.e. the set $\{1, 2, 3, ..., (a-1), a\}$. In our pseudo-code the left arrow, $\leftarrow$, denotes assignment: e.g. $a \leftarrow b$ indicates that the value b is computed and then assigned to the variable a. A graph G is an ordered pair (V, E) of sets where the elements of V are called *vertices* and the elements of E, called *edges*, are unordered pairs of distinct vertices. We will denote an edge, composed of vertices v and w, by (v, w). Given a graph G we define by $\mathcal{V}(G)$ and $\mathcal{E}(G)$ the set of vertices of G (i.e. V, above) and the set of edges of G (i.e. E, above) respectively. We may also use the notation $V(G)$ and $E(G)$ to denote the vertex and edge set of a graph G or even simply V and E where the graph is clear from the context. Sometimes in this thesis, with a slight abuse of notation, by writing $v \in G$ we mean $v \in \mathcal{V}(G)$.

In addition to the above notation, each chapter has its own definitions. Note, however, that the same symbol may mean very different things in different chapters.

1.1 Online Learning on a Graph

The online learning protocol is a game between *nature* and *learner*. The game proceeds in trials $t = 1, 2, 3, ...T$. For every $t \in \{0, 1, 2, ..., T\}$ we have a natural number $\mathcal{M}_t$ which is the *cumulative mistakes* made by the learner up to and including trial t. We initialise with $\mathcal{M}_0 \leftarrow 0$. On trial t the following happens:

1. Nature asks learner a question q_t.
2. Learner gives an answer $\hat{a}_t$.
3. Nature reveals correct answer a_t to learner.
4. If $\hat{a}_t = a_t$ then $\mathcal{M}_t \leftarrow \mathcal{M}_{t-1}$. Otherwise $\mathcal{M}_t \leftarrow \mathcal{M}_{t-1} + 1$

The aim of the learner is to few mistakes: i.e. for $\mathcal{M}_T$ to be small.

In this thesis we focus on online learning over a labelled graph: that is a graph G and a labelling $y : \mathcal{V}(G) \to \mathbb{N}_K$ for some K (note, however, that sometimes in this thesis we will denote a labelling y by vector notation $\boldsymbol{y}$, were the notation y_v is used instead of $y(v)$). Sometimes in this thesis we will restrict to $K = 2$. The graph G is known to the learner a-priori but the labelling y is not. This thesis covers the following two learning tasks:

Classification: The questions are of the form "what is the label of vertex v". Formally: q_t is any vertex $v_t \in \mathcal{V}(G)$ selected freely by nature and $a_t := y(v_t)$.

Similarity Prediction: The questions are of the form "do vertices v and w have the same label". Formally: q_t is any pair of vertices $v_t, w_t \in \mathcal{V}(G)$ selected freely by nature, $a_t := 0$ if $y(v_t) = y(w_t)$ and $a_t := 1$ if $y(v_t) \neq y(w_t)$. In the literature similarity prediction is known also as "semi-supervised clustering".

1.2 Overview of the Thesis

We now give an overview of the contents of the thesis.

1. Chapter 2 : In this chapter we design an online algorithm to classify the vertices of a graph. Underpinning the algorithm is the probability distribution of an Ising model isomorphic to the graph. Each classification is based on predicting the label with maximum marginal probability in the limit of zero-temperature with respect to the labels and vertices seen so far. Computing these classifications is unfortunately based on a $\#P$-complete problem. This motivates us to develop an approximate prediction for which we give a sequential guarantee in the online mistake bound framework. The performance guarantee is comparable to previous upper bounds that are logarithmic in the diameter [16] if the graph is a tree. For a general graph, the algorithm exploits the additional connectivity over a tree to provide a per-cluster bound. The algorithm is efficient, as the cumulative time to sequentially predict all of the vertices of the graph is quadratic in the size of the graph.
   This chapter was written in collaboration with Mark Herbster and is based on our paper entitled "Online Prediction at the Limit of Zero Temperature" and published in NIPS 2015. The experiments were performed by Shaona Gosh.

2. Chapter 3: In this chapter we deal with the online similarity prediction problem. By converting the problem to a linear classification problem on a space of matrices we apply the perceptron and matrix winnow algorithms to give us two algorithms to solve the similarity prediction problem. We then use the concept of a binary support tree (first described in [32] and [17]) to ensure these algorithms have a low mistake bound. Finally we show how the perceptron-based algorithm on the binary support tree can be made exponentially faster.

This chapter is based on our paper "Online similarity prediction of networked data from known and unknown graphs" in COLT 2013. The final section of this chapter is all my own work. The other sections of this chapter were written in collaboration with Mark Herbster and Claudio Gentile.

3. Chapter 4: The junction tree algorithm, which is used to compute all marginals in a multivariate probability distribution stored in a factored form, will serve as the basis for the design of efficient algorithms for online Baysian classification in general Ising models (in Chapter 5). Hence, this chapter is devoted to studying the junction tree algorithm and developing efficient new versions of it. The junction tree algorithm first constructs a tree called a junction tree who's vertices are sets of random variables. The algorithm then performs a generalised version of belief propagation on the junction tree. The Shafer-Shenoy and Hugin architectures are two ways to perform this belief propagation that tradeoff time and space complexities in different ways: Hugin propagation is at least as fast as Shafer-Shenoy propagation and in the cases that we have large vertices of high degree is significantly faster. However, this speed increase comes at the cost of an increased space complexity. This chapter first introduces a simple novel architecture, ARCH-1, which has the best of both worlds: the speed of Hugin propagation and the low space requirements of Shafer-Shenoy propagation. A more complicated novel architecture, ARCH-2, is then introduced which has, up to a factor only linear in the maximum cardinality of any vertex, time and space complexities at least as good as ARCH-1 and in the cases that we have large vertices of high degree is significantly faster than ARCH-1.
   This chapter is all my own work.

4. Chapter 5: In this chapter we first define the general Ising model, show how it comes from a natural process and show that the Bayes classifier minimises the expected number of mistakes in an online learning game with the Ising model. We show how to convert the junction tree algorithm so that it implements efficient online Bayesian classification in general Ising models. We give the online versions of Shafer-Shenoy propagation/ARCH-1, Hugin propagation and the novel architecture ARCH-2. We show that online Shafer-Shenoy is more space efficient than online Hugin but at the expense of a higher time complexity. We then show that the online version of ARCH-2 essentially achieves the best of both worlds: it has, up to a factor linear in the width of the junction tree, the speed of online Hugin propagation and the space efficiency of online Shafer-Shenoy propagation. Finally we develop an algorithm that, for any tree structured Ising model, constructs, in linear time and space, a junction tree that gives optimal time per prediction for online Hugin/ARCH-2. We show that this prediction time is logarithmic in the maximum cardinality of a binary subtree of the tree (that is, a subtree for which every vertex has degree of at most three). The space requirement for Hugin/ARCH-2 with the junction tree is linear in the cardinality of the tree which is optimal for the online junction tree algorithm.
   The final section of this chapter (except the last two subsections, which are all my own work) is based on our paper "Online sum-product

computations over trees" in NIPS 2012 which was written in collaboration with Mark Herbster and Fabio Vitale. I submitted this paper as part of my MRes dissertation in Financial Computing. All other sections in this chapter are all my own work.

Chapter 2

Online Approximate Prediction at the Limit of Zero Temperature in an Ising Model

2.1 Abstract

We design an online algorithm to classify the vertices of a graph. Underpinning the algorithm is the probability distribution of an Ising model isomorphic to the graph. Each classification is based on predicting the label with maximum marginal probability in the limit of zero-temperature with respect to the labels and vertices seen so far. Computing these classifications is unfortunately based on a $\#P$-complete problem. This motivates us to develop an algorithm for which we give a sequential guarantee in the online mistake bound framework. Our algorithm is optimal when the graph is a tree matching the prior results in [16]. For a general graph, the algorithm exploits the additional connectivity over a tree to provide a per-cluster bound. The algorithm is efficient, as the cumulative time to sequentially predict all of the vertices of the graph is quadratic in the size of the graph.

2.2 Introduction

Semi-supervised learning is now a standard methodology in machine learning. A common approach in semi-supervised learning is to build a graph [9] from a given set of labeled and unlabeled data with each datum represented as a vertex. The hope is that the constructed graph will capture either the cluster [18] or manifold [7] structure of the data. Typically, an edge in this graph indicates the expectation that the joined data points are more likely to have the same label. One method to exploit this representation is to use the semi-norm induced by the Laplacian of the graph [61, 7, 60, 54]. A shared idea of the Laplacian semi-norm based approaches is that the smoothness of a boolean labeling of the graph is measured via the *"cut"*, which is just the number of edges that connect disagreeing labels. In practice the semi-norm is then used as a regularizer in which the optimization problem is relaxed from boolean to real values. Our approach also uses the "cut", but unrelaxed, to define an Ising distribution over the vertices of the graph.

Predicting with the vertex marginals of an Ising distribution in the limit of zero temperature was shown to be optimal in the mistake bound model [16, Section 4.1] when the graph is a tree. The exact computation of marginal probabilities in the Ising model is intractable on non-trees [25]. However, in the limit of zero temperature, a rich combinatorial structure called the Picard-Queyranne graph [45] emerges. We exploit this structure to give an algorithm which 1) is optimal on trees, 2) has a quadratic cumulative computational complexity, and 3) has a mistake bound on generic graphs that is stronger than previous bounds in many natural cases.

The paper is organized as follows. In the remainder of this section, we introduce the Ising model and lightly review previous work in the online mistake bound model for predicting the labeling of a graph. In Section 2.3 we review our key technical tool the Picard-Queyranne graph [45] and explain the required notation. In the body of Section 2.4 we provide a mistake bound analysis of our algorithm as well as the intractable `0-Ising` algorithm and then conclude with a detailed comparison to the state of the art. In the appendices we provide proofs as well as preliminary experimental results.

Ising model in the limit zero temperature. In our setting, the parameters of the Ising model are an n-vertex graph $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}))$ and a temperature parameter $\tau > 0$, where $V(\mathcal{G}) = \{1, \ldots, n\}$ denotes the vertex set and $E(\mathcal{G})$ denotes the edge set. Each vertex of this graph may be labeled with one of two states $\{0,1\}$ and thus a labeling of a graph may be denoted by a vector $\boldsymbol{u} \in \{0,1\}^n$ where u_i denotes the label of vertex i. The *cutsize* of a labeling $\boldsymbol{u}$ is defined as $\phi_{\mathcal{G}}(\boldsymbol{u}) := \sum_{(i,j)\in E(\mathcal{G})} |u_i - u_j|$. The Ising probability distribution over labelings of $\mathcal{G}$ is then defined as $p_\tau^{\mathcal{G}}(\boldsymbol{u}) \propto \exp\left(-\frac{1}{\tau}\phi_{\mathcal{G}}(\boldsymbol{u})\right)$ where $\tau > 0$ is the temperature parameter. In our online setting at the beginning of trial $t+1$ we will have already received an *example sequence*, $\mathcal{S}_t$, of t vertex-label pairs $(i_1, y_1), \ldots, (i_t, y_t)$ where pair $(i, y) \in V(\mathcal{G}) \times \{0,1\}$. We use $p_\tau^{\mathcal{G}}(u_v = y|\mathcal{S}_t) := p_\tau^{\mathcal{G}}(u_v = y|u_{i_1} = y_1, \ldots, u_{i_t} = y_t)$ to denote the marginal probability that vertex v has label y given the previously labeled vertices of $\mathcal{S}_t$. For convenience we also define the marginalized *cutsize* $\phi_{\mathcal{G}}(\boldsymbol{u}|\mathcal{S}_t)$ to be equal to $\phi_{\mathcal{G}}(\boldsymbol{u})$ if $u_{i_1} = y_1, \ldots, u_{i_t} = y_t$ and equal to `undefined` otherwise. Our prediction $\hat{y}_{t+1}$ of vertex i_{t+1} is then the label with maximal marginal probability in the limit of zero temperature, thus

$$\hat{y}^{0\mathrm{I}}_{t+1}(i_{t+1}|\mathcal{S}_t) := \operatorname*{argmax}_{y\in\{0,1\}} \lim_{\tau\to 0} p_\tau^{\mathcal{G}}(u_{i_{t+1}} = y|u_{i_1} = y_1, \ldots, u_{i_t} = y_t)\,. \qquad [\texttt{0-Ising}] \tag{2.1}$$

Note the prediction is undefined if the labels are equally probable. In low temperatures the mass of the marginal is dominated by the labelings consistent with $\mathcal{S}_t$ and the proposed label of vertex i_{t+1} of minimal cut; as we approach zero, $\hat{y}_{t+1}$ is the label consistent with the maximum number of labelings of minimal cut. Thus if $k := \min\limits_{\boldsymbol{u}\in\{0,1\}^n} \phi_{\mathcal{G}}(\boldsymbol{u}|\mathcal{S})$ then we have that

$$\hat{y}^{0\mathrm{I}}(v|\mathcal{S}) = \begin{cases} 0 & |\boldsymbol{u} \in \{0,1\}^n : \phi_{\mathcal{G}}(\boldsymbol{u}|(\mathcal{S},(v,0))) = k| > |\boldsymbol{u} \in \{0,1\}^n : \phi_{\mathcal{G}}(\boldsymbol{u}|(\mathcal{S},(v,1))) = k| \\ 1 & |\boldsymbol{u} \in \{0,1\}^n : \phi_{\mathcal{G}}(\boldsymbol{u}|(\mathcal{S},(v,0))) = k| < |\boldsymbol{u} \in \{0,1\}^n : \phi_{\mathcal{G}}(\boldsymbol{u}|(\mathcal{S},(v,1))) = k| \end{cases}.$$

The problem of counting minimum label-consistent cuts was shown to be #P-complete in [46] and further computing $\hat{y}^{0\mathrm{I}}(v|\mathcal{S})$ is also NP-hard (see Section 2.11). In Section 2.3.1 we introduce the Picard-Queyranne graph [45] which captures the combinatorial structure of the set of minimum-cuts. We then use this simplifying structure as a basis to design a heuristic approximation to $\hat{y}^{0\mathrm{I}}(v|\mathcal{S})$ with a mistake bound guarantee.

Predicting the labelling of a graph in the mistake bound model. We prove performance guarantees for our method in the mistake bound model introduced by Littlestone [38]. On the graph this model corresponds to the following game. `Nature` presents a graph $\mathcal{G}$; `Nature` queries a vertex $i_1 \in V(\mathcal{G}) = \mathbb{N}_n$; the `learner` predicts the label of the vertex $\hat{y}_1 \in \{0,1\}$;

nature presents a label y_1; nature queries a vertex i_2; the learner predicts $\hat{y}_2$; and so forth. The learner's goal is to minimize the total number of mistakes $M = |\{t : \hat{y}_t \neq y_t\}|$. If nature is adversarial, the learner will always make a "mistake", but if nature is regular or simple, there is hope that a learner may incur only a few mistakes. Thus, a central goal of online learning is to design algorithms whose total mistakes can be bounded relative to the complexity of nature's labeling. The graph labeling problem has been studied extensively in the online literature. Here we provide a rough discussion of the two main approaches for graph label prediction, and in Section 2.4.3 we provide a more detailed comparison. The first approach is based on the graph Laplacian [32, 28, 29]; it provides bounds that utilize the additional connectivity of non-tree graphs, which are particularly strong when the graph contains uniformly-labeled clusters of small (resistance) diameter. The drawbacks of this approach are that the bounds are weaker on graphs with large diameter and that the computation times are slower. The second approach is to estimate the original graph with an appropriately selected tree or "path" graph [30, 17, 16, 56]; this leads to faster computation times, and bounds that are better on graphs with large diameters. The algorithm treeOpt [16] is optimal on trees. These algorithms may be extended to non-tree graphs by first selecting a spanning tree uniformly at random [17] and then applying the algorithm to the sampled tree. This randomized approach enables expected mistake bounds which exploit the cluster structure in the graph.

The bounds we prove for the NP-hard 0-Ising prediction and our heuristic are most similar to the "small p" bounds proven for the p-seminorm interpolation algorithm [29]. Although these bounds are not strictly comparable, a key strength of our approach is that the new bounds often improve when the graph contains uniformly-labeled clusters of varying diameters. Furthermore, when the graph is a tree we match the optimal bounds of [16]. Finally, the cumulative time required to compute the complete labeling of a graph is quadratic in the size of the graph for our algorithm, while [29] requires the minimization of a non-strongly convex function (on every trial) which is not differentiable when $p \to 1$.

2.3 Preliminaries

An (undirected) graph $\mathcal{G}$ is a pair of sets (V, E) such that E is a set of *unordered* pairs of distinct elements from V. We say that $\mathcal{R}$ is a subgraph $\mathcal{R} \subseteq \mathcal{G}$ iff $V(\mathcal{R}) \subseteq V(\mathcal{G})$ and $E(\mathcal{R}) = \{(i,j) : i,j \in V(\mathcal{R}), (i,j) \in E(\mathcal{G})\}$. Given any subgraph $\mathcal{R} \subseteq G$, we define its *boundary* (or inner border) $\partial_0(\mathcal{R})$, its *neighbourhood* (or exterior border) $\partial_e(\mathcal{R})$ respectively as $\partial_0(\mathcal{R}) := \{j : i \notin V(\mathcal{R}), j \in V(\mathcal{R}), (i,j) \in E(\mathcal{G})\}$, and $\partial_e(\mathcal{R}) := \{i : i \notin V(\mathcal{R}), j \in V(\mathcal{R}), (i,j) \in E(\mathcal{G})\}$, and its exterior edge border $\partial_e^E(\mathcal{R}) := \{(i,j) : i \notin V(\mathcal{R}), j \in V(\mathcal{R}), (i,j) \in E(\mathcal{G})\}$. The length of a subgraph $\mathcal{P}$ is denoted by $|\mathcal{P}| := |E(\mathcal{P})|$ and we denote the diameter of a graph by $D(\mathcal{G})$. A pair of vertices $v, w \in V(\mathcal{G})$ are *κ-connected* if there exist κ edge-disjoint paths connecting them. The *connectivity of a graph*, $\kappa(\mathcal{G})$, is the maximal value of κ such that every pair of points in $\mathcal{G}$ is κ-connected. The *atomic number* $\mathcal{N}_\kappa(\mathcal{G})$ of a graph at connectivity level κ is the minimum cardinality c of a partition of $\mathcal{G}$ into subgraphs $\{\mathcal{R}_1, \ldots, \mathcal{R}_c\}$ such that $\kappa(\mathcal{R}_i) \geq \kappa$ for all $1 \leq i \leq c$.

Our results also require the use of *directed-*, *multi-*, and *quotient-* graphs.

Every undirected graph also defines a directed graph where each undirected edge (i,j) is represented by directed edges (i,j) and (j,i). An *orientation* of an undirected graph is an assignment of a direction to each edge, turning the initial graph into a directed graph. In a *multi-graph* the edge set is now a multi-set and thus there may be multiple edges between two vertices. A *quotient-graph* $\mathbb{G}$ is defined from a graph $\mathcal{G}$ and a partition of its vertex set $\{V_i\}_{i=1}^N$ so that $V(\mathbb{G}) := \{V_i\}_{i=1}^N$ (we often call these vertices *super-vertices* to emphasize that they are sets) and the multiset $E(\mathbb{G}) := \{(I,J) : I,J \in V(\mathbb{G}), I \neq J, i \in I, j \in J, (i,j) \in E(\mathcal{G})\}$. We commonly construct a quotient-graph $\mathbb{G}$ by "merging" a collection of super-vertices, for example, in Figure 2.2 from 2.2a to 2.2b where 6 and 9 are merged to "6/9" and also the five `merges` that transforms 2.2c to 2.2d.

The set of all *label-consistent minimum-cuts* in a graph with respect to an example sequence $\mathcal{S}$ is $\mathcal{U}^*_{\mathcal{G}}(\mathcal{S}) := \operatorname{argmin}_{\boldsymbol{u}\in\{0,1\}^n} \phi_{\mathcal{G}}(\boldsymbol{u}|\mathcal{S})$. The minimum is typically non-unique. For example in Figure 2.2a, the vertex sets $\{v_1,\ldots,v_4\},\{v_5,\ldots,v_{12}\}$ correspond to one label-consistent minimum-cut and $\{v_1,\ldots,v_5,v_7,v_8\},\{v_6,v_9\ldots,v_{12}\}$ to another (the cutsize is 3). The (uncapacitated) *maximum flow* is the number of edge-disjoint paths between a source and target vertex. Thus in Figure 2.2b between vertex "1" and vertex "6/9" there are at most 3 simultaneously edge-disjoint paths; these are also not unique, as one path must pass through either vertices $\langle v_{11}, v_{12}\rangle$ or vertices $\langle v_{11}, v_{10}, v_{12}\rangle$. Figure 2.2c illustrates one such flow $\mathcal{F}$ (just the directed edges). For convenience it is natural to view the maximum flow or the label-consistent minimum-cut as being with respect to only two vertices as in Figure 2.2a transformed to Figure 2.2b so that $\mathcal{H} \leftarrow \texttt{merge}(\mathcal{G},\{v_6,v_9\})$. The "flow" and the "cut" are related by Menger's theorem which states that the minimum-cut with respect to a source and target vertex is equal to the max flow between them. Given a connected graph $\mathcal{H}$ and source and target vertices s,t the Ford-Fulkerson algorithm [22] can find k edge-disjoint paths from s to t in time $O(k|E(\mathcal{H})|)$ where k is the value of the max flow.

2.3.1 The Picard-Queyranne graph

Given a set of labels there may be multiple label-consistent minimum-cuts as well as multiple maximum flows in a graph. The Picard-Queyranne (PQ) graph [45] reduces this multiplicity as far as is possible with respect to the indeterminacy of the maximum flow. The vertices of the PQ-graph are defined as a super-vertex set on a partition of the original graph's vertex set. Two vertices are contained in the same super-vertex iff they have the same label in *every* label-consistent minimum-cut. An edge between two vertices defines an analogous edge between two super-vertices iff that edge is conserved in *every* maximum flow. Furthermore the edges between super-vertices strictly orient the labels in any label-consistent minimum-cut as may be seen in the formal definition that follows.

First we introduce the following useful notations: let $k_{\mathcal{G},\mathcal{S}} := \min\{\phi_{\mathcal{G}}(\boldsymbol{u}|\mathcal{S}) : \boldsymbol{u} \in \{0,1\}^n\}$ denote the minimum-cutsize of $\mathcal{G}$ with respect to $\mathcal{S}$; let $i \overset{\mathcal{S}}{\sim} j$ denote an equivalence relation between vertices in $V(\mathcal{G})$ where $i \overset{\mathcal{S}}{\sim} j$ iff $\forall \boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S}) : u_i = u_j$; and then we define,

Definition 1 ([45]). *The* Picard-Queyranne graph $\mathbb{G}(\mathcal{G},\mathcal{S})$ *is derived from graph $\mathcal{G}$ and non-trivial example sequence $\mathcal{S}$. The graph is an orientation of the quotient graph derived from the partition $\{\perp, I_2, \ldots, I_{N-1}, \top\}$ of $V(\mathcal{G})$ induced by $\overset{\mathcal{S}}{\sim}$. The edge set of $\mathbb{G}$ is constructed of $k_{\mathcal{G},\mathcal{S}}$ edge-disjoint paths starting at source vertex $\perp$ and terminating at target vertex $\top$. A labeling $\boldsymbol{u} \in \{0,1\}^n$ is in $\mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ iff*

1. *$i \in \bot$ implies $u_i = 0$ and $i \in \top$ implies $u_i = 1$*
2. *$i, j \in H$ implies $u_i = u_j$*
3. *$i \in I$, $j \in J$, $(I, J) \in E(\mathbb{G})$, and $u_i = 1$ implies $u_j = 1$*

where $\bot$ and $\top$ are the source and target vertices and $H, I, J \in V(\mathbb{G})$.

As $\mathbb{G}(\mathcal{G}, \mathcal{S})$ is a DAG it naturally defines a partial order $(V(\mathbb{G}), \leq_{\mathbb{G}})$ on the vertex set where $I \leq_{\mathbb{G}} J$ if there exists a path starting at I and ending at J. The least and greatest elements of the partial order are $\bot$ and $\top$. The notation $\uparrow R$ and $\downarrow R$ denote the *up set* and *down set* of R. Given the set $\mathcal{U}^*$ of all label-consistent minimum-cuts then if $\boldsymbol{u} \in \mathcal{U}^*$ there exists an antichain $A \subseteq V(\mathbb{G}) \setminus \{\top\}$ such that $u_i = 0$ when $i \in I \in \downarrow A$ otherwise $u_i = 1$; furthermore for every antichain there exists a label-consistent minimum-cut. The simple structure of $\mathbb{G}(\mathcal{G}, \mathcal{S})$ was utilized by [45] to enable the efficient algorithmic enumeration of minimum-cuts. However, the cardinality of this set of all label-consistent minimum-cuts is potentially exponential in the size of the PQ-graph and the exact computation of the cardinality was later shown #P-complete in [46]. In Figure 2.1 we give the algorithm from [45, 4] to

```
PicardQueyranneGraph(graph: $\mathcal{G}$; example sequence: $\mathcal{S} = (v_k, y_k)_{k=1}^{t}$)
1. $(\mathcal{H}, s, t) \leftarrow \mathrm{SourceTargetMerge}(\mathcal{G}, \mathcal{S})$
2. $\mathcal{F} \leftarrow \mathrm{MaxFlow}(\mathcal{H}, s, t)$
3. $\mathcal{I} \leftarrow (V(\mathcal{I}), E(\mathcal{I}))$ where $V(\mathcal{I}) := V(\mathcal{H})$ and $E(\mathcal{I}) := \{(i,j) : (i,j) \in E(\mathcal{H}), (j,i) \notin \mathcal{F}\}$
4. $\mathbb{G}^0 \leftarrow \mathrm{QuotientGraph}(\mathrm{StronglyConnectedComponents}(\mathcal{I}), \mathcal{H})$
5. $E(\mathbb{G}) \leftarrow E(\mathbb{G}^0)$; $V(\mathbb{G}) \leftarrow V(\mathbb{G}^0)$ except $\bot(\mathbb{G}) \leftarrow \bot(\mathbb{G}^0) \cup \{v_k : k \in \mathbb{N}_t, y_k = 0\}$
                                   and$\top(\mathbb{G}) \leftarrow \top(\mathbb{G}^0) \cup \{v_k : k \in \mathbb{N}_t, y_k = 1\}$
Return: directed graph: $\mathbb{G}$
```

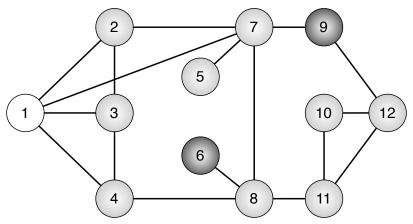
FIGURE 2.1: Computing the Picard-Queyranne graph

compute a PQ-graph. We illustrate the computation in Figure 2.2. The algorithm operates first on $(\mathcal{G}, \mathcal{S})$ (step 1) by "`merging`" all vertices which share the same label in $\mathcal{S}$ to create $\mathcal{H}$. In step 2 a max flow graph $\mathcal{F} \subseteq \mathcal{H}$ is computed by the Ford-fulkerson algorithm. It is well-known in the case of unweighted graphs that a max flow graph $\mathcal{F}$ may be output as a DAG of k edge-disjoint paths where k is the value of the flow. In step 3 all edges in the flow become directed edges creating $\mathcal{I}$. The graph $\mathbb{G}^0$ is then created in step 4 from $\mathcal{I}$ where the strongly connected components become the super-vertices of $\mathbb{G}^0$ and the super-edges correspond to a subset of flow edges from $\mathcal{F}$. Finally, in step 5, we create the PQ-graph $\mathbb{G}$ by "fixing" the source and target vertices so that they also have as elements the original labeled vertices from $\mathcal{S}$ which were merged in step 1. The correctness of the algorithm follows from arguments in [45]; we provide an independent proof in Section 2.6.

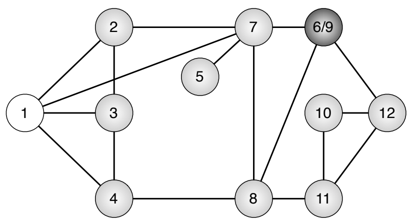
Theorem 2 ([45]). *The algorithm in Figure 2.1 computes the unique Picard-Queyranne graph $\mathbb{G}(\mathcal{G}, \mathcal{S})$ derived from graph $\mathcal{G}$ and non-trivial example sequence $\mathcal{S}$.*

2.4 Mistake Bounds Analysis

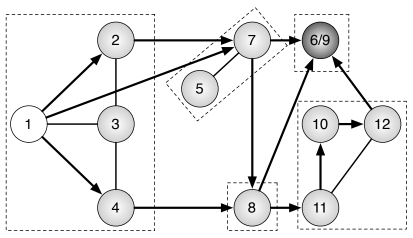
In this section we analyze the mistakes incurred by the intractable `0-Ising` strategy (see (2.1)) and the strategy `longest-path` (see Figure 2.3). Our analysis splits into two parts. Firstly, we show (Section 2.4.1, Theorem 4)



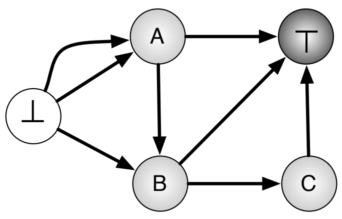
(A) Graph $\mathcal{G}$ and $\mathcal{S} = \langle(v_1, 0), (v_6, 1), (v_9, 1)\rangle$



(B) Graph $\mathcal{H}$ (step 1 in Figure 2.1)



(C) Graph $\mathcal{I}$ (step 3 in Figure 2.1)



(D) PQ Graph $\mathbb{G}$ (step 4 in Figure 2.1)

FIGURE 2.2: Building a Picard-Queyranne graph

for a sufficiently *regular* graph label prediction algorithm, that we may analyze *independently* the mistake bound of each uniformly-labeled cluster (connected subgraph). Secondly, the per-cluster analysis then separates into three cases, the result of which is summarized in Theorem 7. For a given cluster $\mathcal{C}$ when its internal connectivity is larger than the number of edges in the boundary ($\kappa(\mathcal{C}) > |\partial_e^E(\mathcal{C})|$) we will incur no more than one mistake in that cluster. On the other hand for smaller connectivity clusters ($\kappa(\mathcal{C}) \leq |\partial_e^E(\mathcal{C})|$) we incur up to quadratically in mistakes via the edge boundary size. When $\mathcal{C}$ is a tree we incur $\mathcal{O}(|\partial_e^E(\mathcal{C})| \log D(\mathcal{C}))$ mistakes.

The analysis of smaller connectivity clusters separates into two parts. First, a sequence of trials in which the label-consistent minimum-cut does not increase, we call a *PQ-game* (Section 2.4.2) as in essence it is played on a PQ-graph. We give a mistake bound for a PQ-game for the intractable `0-Ising` prediction and a comparable bound for the strategy `longest-path` in Theorem 5. Second, when the label-consistent minimum-cut increases the current PQ-game ends and a new one begins, leading to a sequence of PQ-games. The mistakes incurred over a sequence of PQ-games is addressed in the aforementioned Theorem 7 and finally Section 2.4.3 concludes with a discussion of the combined bounds of Theorems 4 and 7 with respect to other graph label prediction algorithms.

2.4.1 Per-cluster mistake bounds for *regular* graph label prediction algorithms

An algorithm is called *regular* if it is *permutation-invariant*, *label-monotone*, and *Markov*. An algorithm is *permutation-invariant* if the prediction at any time t does not depend on the order of the examples up to time t; *label-monotone* if for every example sequence if we insert an example "between" examples t and $t+1$ with label y then the prediction at time $t+1$ is unchanged or changed to y; and *Markov* with respect to a graph $\mathcal{G}$ if for any disjoint vertex sets P and Q and separating set R then the predictions in P are independent of the labels in Q given the labels of R. A subgraph is *uniformly-labeled* with respect to an example sequence iff the label of each vertex is the same and these labels are consistent with the example sequence. The following definition characterizes the "worst-case" example sequences for regular algorithms with respect to uniformly-labeled clusters.

Definition 3. *Given an online algorithm $\mathcal{A}$ and a uniformly-labeled subgraph $\mathcal{C} \subseteq \mathcal{G}$, then $\mathcal{B}_{\mathcal{A}}(\mathcal{C};\mathcal{G})$ denotes the maximal mistakes made only in $\mathcal{C}$ for the presentation of any permutation of examples in $\partial_e(\mathcal{C})$, each with label y, followed by any permutation of examples in $\mathcal{C}$, each with label $1-y$.*

The following theorem enables us to analyze the mistakes incurred in each uniformly-labeled subgraph $\mathcal{C}$ *independently* of each other and *independently* of the remaining graph structure excepting the subgraph's exterior border $\partial_e(\mathcal{C})$.

Theorem 4 (Proof in Section 2.8)**.** *Given an online permutation-invariant label-monotone Markov algorithm $\mathcal{A}$ and a graph $\mathcal{G}$ which is covered by uniformly-labeled subgraphs $\mathcal{C}_1, \ldots, \mathcal{C}_c$ the mistakes incurred by the algorithm may be bounded by $M \leq \sum_{i=1}^{c} \mathcal{B}_{\mathcal{A}}(\mathcal{C}_i;\mathcal{G})$.*

The above theorem paired with Theorem 7 completes the mistake bound analysis of our algorithms.

2.4.2 PQ-games

Given a PQ-graph $\mathbb{G} = \mathbb{G}(\mathcal{G}, \mathcal{S})$, the derived online PQ-game is played between a `player` and an `adversary`. The aim of the player is to minimize their mistaken predictions; for the adversary it is to maximize the player's mistaken predictions. Thus to play the adversary proposes a vertex $z \in Z \in V(\mathbb{G})$, the player then predicts a label $\hat{y} \in \{0,1\}$, then the adversary returns a label $y \in \{0,1\}$ and either a *mistake* is incurred or not. The only restriction on the adversary is to not return a label which increases the label-consistent minimum-cut. As long as the adversary does not give an example $(z \in \bot, 1)$ or $(z \in \top, 0)$, the label-consistent minimum-cut does not increase no matter the value of y; which also implies the player has a trivial strategy to predict the label of $z \in \bot \cup \top$. After the example is given, we have an updated PQ-graph with new source and target super-vertices as seen in the proposition below.

Proposition 1. *If $\mathbb{G}(\mathcal{G}, \mathcal{S})$ is a PQ-graph and $(z, y=0)$ ($(z, y=1)$) is an example with $z \in Z \in V(\mathbb{G})$ and $z \notin \top$ ($z \notin \bot$) then let $\mathbb{Z} = \downarrow\{Z\}$ ($\mathbb{Z} = \uparrow\{Z\}$) then $\mathbb{G}(\mathcal{G}, \langle \mathcal{S}, (z,y) \rangle) = \texttt{merge}(\mathbb{G}(\mathcal{G}, \mathcal{S}), \mathbb{Z})$.*

Thus given the PQ-graph $\mathbb{G}$ the PQ-game is independent of $\mathcal{G}$ and $\mathcal{S}$, since a "play" $z \in V(\mathcal{G})$ induces a "play" $Z \in V(\mathbb{G})$ (with $z \in Z$).

Mistake bounds for PQ-games. Given a *single* PQ-game, in the following we will discuss the three strategies `fixed-paths`, `0-Ising`, and `longest-path` that the `player` may adopt for which we prove online mistake bounds. The first strategy `fixed-paths` is merely motivational: it can be used to play a *single* PQ-game, but not a sequence. The second strategy `0-Ising` is computationally infeasible. Finally, the `longest-path` strategy is "dynamically" similar to `fixed-paths` but is also permutation-invariant. Common to all our analyses is a *k-path cover* P of PQ-graph $\mathbb{G}$

which is a partitioning of the edge-set of $\mathbb{G}$ into k edge-disjoint directed paths $P := \{p^1, \dots, p^k\}$ from $\perp$ to $\top$. Note that the cover is not necessarily unique; for example, in Figure 2.2d, we have the two unique path covers $P_1 := \{(\perp, A, \top), (\perp, A, B, \top), (\perp, B, C, \top)\}$ and $P_2 := \{(\perp, A, \top), (\perp, A, B, C, \top), (\perp, B, \top)\}$. We denote the set of all path covers as $\mathcal{P}$ and thus we have for Figure 2.2d that $\mathcal{P} := \{P_1, P_2\}$. This cover motivates a simple mistake bound and strategy. Suppose we had a single path of length $|p|$ where the first and last vertex are the "source" and "target" vertices. So the minimum label-consistent cut-size is "1" and a natural strategy is simply to predict with the "nearest-neighbor" revealed label and trivially our mistake bound is $\log |p|$. Generalizing to multiple paths we have the following strategy.

Strategy `fixed-paths`$(\widetilde{P})$**:** Given a PQ-graph choose a path cover $\{\tilde{p}^1, \dots, \tilde{p}^k\} = \widetilde{P} \in \mathcal{P}(\mathbb{G})$. If the path cover is also vertex-disjoint except for the source and target vertex we may directly use the "nearest-neighbor" strategy detailed above, achieving the mistake upper bound $M \leq \sum_{i=1}^{k} \log |\tilde{p}^i|$. Unsurprisingly, in the vertex-disjoint case it is a mistake-bound optimal [38] algorithm. If, however, $\widetilde{P}$ is not vertex-disjoint and we need to predict a vertex V we may select a path in $\widetilde{P}$ containing V and predict with the nearest neighbour and also obtain the bound above. In this case, however, the bound may not be "optimal." Essentially the same technique was used in [24] in a related setting for learning "directed cuts." A limitation of the `fixed-paths` strategy is that it does not seem possible to extend into a strategy that can play a *sequence* of PQ-games and still meet the regularity properties, particularly permutation-invariance as required by Theorem 4.

Strategy `0-Ising`**:** The prediction of the Ising model in the limit of zero temperature (cf. (2.1)), is equivalent to those of the well-known *Halving* algorithm [5, 39] where the hypothesis class $\mathcal{U}^*$ is the set of label-consistent minimum-cuts. The mistake upper bound of the *Halving algorithm* is just $M \leq \log |\mathcal{U}^*|$ where this bound follows from the observation that whenever a mistake is made at least "half" of concepts in $\mathcal{U}^*$ are no longer consistent. We observe that we may upper bound $|\mathcal{U}^*| \leq \operatorname{argmin}_{P \in \mathcal{P}(\mathbb{G})} \prod_{i=1}^{k} |p^i|$ since the product of path lengths from *any* path cover P is an upper bound on the cardinality of $\mathcal{U}^*$ and hence we have the bound in (2.2). And in fact this bound may be a significant improvement over the `fixed-paths` strategy's bound as seen in the following proposition.

Proposition 2 (Proof in Section 2.7)**.** *For every $c \geq 2$ there exists a PQ-graph $\mathbb{G}_c$, with a path cover $P' \in \mathcal{P}(\mathbb{G}_c)$ and a PQ-game example sequence such that the mistakes $M_{\texttt{fixed-paths}(P')} = \Omega(c^2)$, while for all PQ-game example sequences on $\mathbb{G}_c$ the mistakes $M_{\texttt{0-Ising}} = \mathcal{O}(c)$.*

Unfortunately the `0-Ising` strategy has the drawback that counting label-consistent minimum-cuts is #P-complete and computing the prediction (see (2.1)) is NP-hard (see Section 2.11).

Strategy `longest-path`**:** In our search for an efficient and *regular* prediction strategy it seems natural to attempt to "dynamize" the `fixed-paths` approach and predict with a nearest neighbor along a dynamic path. Two such permutation-invariant methods are the `longest-path` and `shortest-path` strategies. The strategy `shortest-path` predicts the label of a super-vertex Z in a PQ-game $\mathbb{G}$ as 0 iff the shortest directed path $(\perp, \dots, Z)$ is shorter than the shortest directed path $(Z, \dots, \top)$. The strategy `longest-path` predicts the label of a super-vertex Z in a PQ-game $\mathbb{G}$ as 0 iff the longest directed path $(\perp, \dots, Z)$ is shorter than the longest directed path $(Z, \dots, \top)$. The strategy `shortest-path` seems to be intuitively favored over `longest-path`

```
Input: Graph: $\mathcal{G}$, Example sequence: $\mathcal{S} = \langle(i_1,0),(i_2,1),(i_3,y_3),\dots,(i_\ell,y_\ell)\rangle \in (\mathbb{N}_n \times \{0,1\})^\ell$
Initialization: $\mathbb{G}_3 = \texttt{PicardQueyranneGraph}(\mathcal{G},\mathcal{S}_2)$
for $t = 3,\dots,\ell$ do
    Receive: $i_t \in \{1,\dots,n\}$
    $I_t = V \in V(\mathbb{G}_t)$ with $i_t \in V$
    Predict (longest-path): $\hat{y}_t = \begin{cases} 0 & |\texttt{longest-path}(\mathbb{G}_t,\bot_t,I_t)| \le |\texttt{longest-path}(\mathbb{G}_t,I_t,\top_t)| \\ 1 & \textbf{otherwise} \end{cases}$
    Predict (0-Ising):   $\hat{y}_t = \hat{y}^{\mathrm{I}0}(i_t|\mathcal{S}_{t-1})$   % as per equation (2.1)
    Receive: $y_t$
    if $(i_t \notin \bot_t$ or $y_t \neq 1)$ and $(i_t \notin \top_t$ or $y_t \neq 0)$ then   % cut unchanged
        $\mathbb{G}_{t+1} = \begin{cases} \texttt{merge}(\mathbb{G}_t,\downarrow\{I_t\}) & y_t = 0 \\ \texttt{merge}(\mathbb{G}_t,\uparrow\{I_t\}) & y_t = 1 \end{cases}$
    else   % cut increases
        $\mathbb{G}_{t+1} = \texttt{PicardQueyranneGraph}(\mathcal{G},\mathcal{S}_t)$
end
```

FIGURE 2.3: `Longest-path` and `0-Ising` online prediction

as it is just the "nearest-neighbor" prediction with respect to the geodesic distance. However, the following proposition shows that it is strictly worse than any `fixed-paths` strategy in the worst case.

Proposition 3 (Proof in Section 2.7). *For every $c \geq 4$ there exists a PQ-graph $\mathbb{G}_c$ and a PQ-game example sequence such that the mistakes $M_{\texttt{shortest-path}} = \Omega(c^2 \log(c))$, while for every path cover $P \in \mathcal{P}(\mathbb{G}_c)$ and for all PQ-game example sequences on $\mathbb{G}_c$ the mistakes $M_{\texttt{fixed-paths}(P)} = \mathcal{O}(c^2)$.*

In contrast, for the strategy `longest-paths` in the proof of Theorem 5 we show that there always exists some retrospective path cover $P_{\texttt{lp}} \in \mathcal{P}(\mathbb{G})$ such that $M_{\texttt{longest-paths}} \leq \sum_{i=1}^{k} \log|p^i_{\texttt{lp}}|$. Computing the "longest-path" has time complexity linear in the number of edges in a DAG.

Summarizing the mistake bounds for the three PQ-game strategies for a single PQ-game we have the following theorem.

Theorem 5 (Proof in Section 2.7). *The mistakes, M, of an online PQ-game for player strategies `fixed-paths`$(\widetilde{P})$, `0-Ising`, and `longest-path` on PQ-graph $\mathbb{G}$ and k-path cover $\widetilde{P} \in \mathcal{P}(\mathbb{G})$ is bounded by*

$$M \leq \begin{cases} \sum_{i=1}^{k} \log|\tilde{p}^i| & \texttt{fixed-paths}(\widetilde{P}) \\ \operatorname{argmin}_{P\in\mathcal{P}(\mathbb{G})} \sum_{i=1}^{k} \log|p^i| & \texttt{0-Ising} \\ \operatorname{argmax}_{P\in\mathcal{P}(\mathbb{G})} \sum_{i=1}^{k} \log|p^i| & \texttt{longest-path} \end{cases}. \tag{2.2}$$

2.4.3 Global analysis of prediction at zero temperature

In Figure 2.3 we summarize the prediction protocol for `0-Ising` and `longest-path`. We claim the regularity properties of our strategies in the following theorem.

Theorem 6 (Proof in Section 2.9). *The strategies `0-Ising` and `longest-path` are permutation-invariant, label-monotone, and Markov.*

The technical hurdle here is to prove that label-monotonicity holds over a sequence of PQ-games. For this we need an analog of Proposition 1 to describe how the PQ-graph changes when the label-consistent minimum-cut increases (see Proposition 4). The application of the following theorem along with Theorem 4 implies we may bound the mistakes of each uniformly-labeled cluster in potentially three ways.

Theorem 7 (Proof in Section 2.8)**.** *Given either the* `0-Ising` *or* `longest-path` *strategy $\mathcal{A}$ the mistakes on uniformly-labeled subgraph $\mathcal{C} \subseteq \mathcal{G}$ are bounded by*

$$\mathcal{B}_{\mathcal{A}}(\mathcal{C};\mathcal{G}) \in \begin{cases} \mathcal{O}(1) & \kappa(\mathcal{C}) > |\partial_e^E(\mathcal{C})| \\ \mathcal{O}\left(|\partial_e^E(\mathcal{C})|(1 + |\partial_e^E(\mathcal{C})| - \kappa(\mathcal{C})) \log N(\mathcal{C})\right) & \kappa(\mathcal{C}) \leq |\partial_e^E(\mathcal{C})| \\ \mathcal{O}(|\partial_e^E(\mathcal{C})| \log D(\mathcal{C})) & \mathcal{C} \text{ is a tree} \end{cases} \tag{2.3}$$

with the atomic number $N(\mathcal{C}) := \mathcal{N}_{|\partial_e^E(\mathcal{C})|+1}(\mathcal{C}) \leq |V(\mathcal{C})|$.

First, if the internal connectivity of the cluster is high we will only make a single mistake in that cluster. Second, if the cluster is a tree then we pay the external connectivity of the cluster $|\partial_e^E(\mathcal{C})|$ times the log of the cluster diameter. Finally, in the remaining case we pay quadratically in the external connectivity and logarithmically in the "atomic number" of the cluster. The atomic number captures the fact that even a poorly connected cluster may have sub-regions of high internal connectivity. **Computational complexity.** If $\mathcal{G}$ is a graph and $\mathcal{S}$ an example sequence with a label-consistent minimum-cut of ϕ then we may implement the `longest-path` strategy so that it has a cumulative computational complexity of $\mathcal{O}(\max(\phi, n)\,|E(\mathcal{G})|)$. This follows because if on a trial the "cut" does not increase we may implement prediction and update in $\mathcal{O}(|E(\mathcal{G})|)$ time. On the other hand if the "cut" increases by ϕ' we pay $\mathcal{O}(\phi'|E(\mathcal{G})|)$ time. To do so we implement an online "Ford-Fulkerson" algorithm [22] which starts from the previous "residual" graph to which it then adds the additional ϕ' flow paths with ϕ' steps of size $\mathcal{O}(|E(\mathcal{G})|)$.

Discussion. There are essentially five dominating mistake bounds for the online graph labeling problem: (I) the bound of `treeOpt` [16] on trees, (II) the bound in expectation of `treeOpt` on a random spanning tree sampled from a graph [16], (III) the bound of `p-seminorm interpolation` [29] tuned for "sparsity" ($p < 2$), (IV) the bound of `p-seminorm interpolation` as tuned to be equivalent to "online label propagation [61]" ($p = 2$), (V) this paper's `longest-path` strategy.

The algorithm `treeOpt` was shown to be optimal on trees. In Section 2.10 we show that `longest-path` also obtains the same optimal bound on trees. Algorithm (II) applies to generic graphs and is obtained from (I) by sampling a random spanning tree (RST). It is not directly comparable to the other algorithms as its bound holds only in *expectation* with respect to the RST.

We use [29, Corollary 10] to compare (V) to (III) and (IV). We introduce the following simplifying notation to compare bounds. Let $\mathcal{C}_1, \ldots, \mathcal{C}_c$ denote uniformly-labeled clusters (connected subgraphs) which cover the graph and set $\kappa_r := \kappa(\mathcal{C}_r)$ and $\phi_r := |\partial_e^E(\mathcal{C}_r)|$. We define $D_{r(i)}$ to be the *wide diameter* at connectivity level i of cluster $\mathcal{C}_r$. The wide diameter $D_{r(i)}$ is the minimum value such that for all pairs of vertices $v, w \in \mathcal{C}_r$ there exists i edge-disjoints of paths from v to w of length *at least* $D_{r(i)}$ in $\mathcal{C}_r$ (and if $i > \kappa_r$ then $D_{r(i)} := +\infty$). Thus $D_{r(1)}$ is the diameter of cluster $\mathcal{C}_r$ and $D_{r(1)} \leq D_{r(2)} \leq \cdots$. Let ϕ denote the minimum label-consistent cutsize and observe that if the cardinality of the cover $|\{\mathcal{C}_1, \ldots, \mathcal{C}_c\}|$ is minimized then we have that $2\phi = \sum_{r=1}^{c} \phi_r$.

Thus using [29, Corollary 10] we have the following upper bounds of (III): $(\phi/\kappa^*)^2 \log D^* + c$ and (IV): $(\phi/\kappa^*)D^* + c$ where $\kappa^* := \min_r \kappa_r$ and $D^* := \max_r D_r(\kappa^*)$. In comparison we have (V): $[\sum_{r=1}^{c} \max(0, \phi_r - \kappa_r + 1)\phi_r \log N_r] + c$ with atomic numbers $N_r := \mathcal{N}_{\phi_r+1}(\mathcal{C}_r)$. To contrast the bounds, consider a double lollipop labeled-graph: first create a lollipop which is a path of $n/4$ vertices attached to a clique of $n/4$ vertices. Label these vertices 1. Second, clone the lollipop except with labels 0. Finally join the two cliques with $n/8$ edges arbitrarily. For (III) and (IV) the bounds are $\mathcal{O}(n)$ independent of the choice of clusters. Whereas an upper bound for (V) is the exponentially smaller $\mathcal{O}(\log n)$ which is obtained by choosing a four cluster cover consisting of the two paths and the two cliques. This emphasizes the generic problem of (III) and (IV): parameters κ^* and D^* are defined by the "worst" clusters; whereas (V) is truly a "per-cluster" bound. We consider the previous "constructed" example to be representative of a generic case where the graph contains clusters of many resistance diameters as well as sparse interconnecting "background" vertices.

On the other hand, there are cases in which (III,IV) improve on (V). For a graph with only small diameter clusters and if the cutsize exceeds the cluster connectivity then (IV) improves on (III,V) given the linear versus quadratic dependence on the cutsize. The log-diameter may be arbitrarily smaller than log-atomic-number ((III) improves on (V)) and also vice-versa. Other subtleties not accounted for in the above comparison include the fact a) the wide diameter is a crude upper bound for resistance diameter (cf. [29, Theorem 1]) and b) the clusters of (III,IV) are not required to be uniformly-labeled. Regarding "a)" replacing "wide" with "resistance" does not change the fact the bound now holds with respect to the worst resistance diameter and the example above is still problematic. Regarding "b)" it is a nice property but we do not know how to exploit this to give an example that significantly improves (III) or (IV) over a slightly more detailed analysis of (V). Finally (III,IV) depend on a correct choice of tunable parameter p.

Thus in summary (V) matches the optimal bound of (I) on trees, and can often improve on (III,IV) when a graph is naturally covered by label-consistent clusters of different diameters. However (III,IV) may improve on (V) in a number of cases including when the log-diameter is significantly smaller than log-atomic-number of the clusters.

Technical Appendices

2.5 Experiments

In this section we present some preliminary experiments that compare the `longest-path` strategy to `treeOpt` [16] and label propagation [61, 7]. The datasets include the four standardized benchmark datasets `USPS 2 vs 3`, `3 vs 8`, `20 newsgroups` and `ISOLET` as well as a constructed dataset `Stripes`. We used our own implementation of `longest-path` and `labelProp`. For the purposes of computational efficiency we ran our experiments in the "batch mode" rather than "online."

We used the benchmark datasets as follows. With the `USPS` datasets we sampled 500 digits from each class. For `ISOLET` we combined "`ISOLET1`" to "`ISOLET5`" giving 3900 in class "0" (letters 'A-M') and 3897 class "1" (letters 'N-Z') examples. While for `20 newsgroups` we combined "comp.*" and "rec.*" creating class "0" with 8124 examples and combining "sci.*" and "talk.*" creating class "1" with 8118 examples. Thus all datasets are nearly balanced between the classes. We then constructed a graph for each dataset by computing a "cost" matrix between all examples (patterns) in the dataset, using the Euclidean metric except on `ISOLET` where we used the "cosine distance." We then constructed both an unweighted minimum spanning tree (MST) and a "3-NN" graph (via the cost matrix) and then "unioned" the edge sets together creating our final graph for each of the datasets. The rationale behind the methodology was based on the common empirical observation that 3-NN graphs are often among the most competitive of the unweighted k-NN graphs. We then added a MST to ensure that the final graph was connected. This produced a relatively sparse graph that reduced the computational burden for all methods and reduced variance by avoiding model selection. Although it was beyond the scope of our limited study, it may be the case that constructed graphs with higher connectivity could potentially lead to higher accuracies.

In Figure 2.4 we report our results. We give the mean accuracy (computed over *all* labels in the graph) and its standard deviation from ten runs. For each "column," and each run of 10, we sampled uniformly $\ell/2$ labels from each class. For the `USPS` datasets we also randomly sampled and built a new graph on each run. Finally on each run as `treeOpt` expects a tree we further sampled a uniform random spanning tree as per [16] from the built graph on each of the 10 runs.

Our obervations are as follows. `LabelProp` performs systematically well across all datasets. `treeOpt` tends to have the weakest performance. Note, however, that `treeOpt` is very computationally efficient and it is natural to run with an ensemble of trees to improve performance; this is discussed and experimentally confirmed in [56]. `Longest-path` is competitive and improves on `labelProp` often. But it has a "failure mode" as seen in the first column for the relatively smaller label sets. We observed that when this occurred we are finding small PQ-graphs corresponding to unbalanced trivial label-consistent minimum-cuts.

We also show results on a constructed dataset to illustrate the potential of the algorithm. `Stripes` is a 60×60 grid graph with toroidal boundary connectivity. Thus each vertex has four neighbors. The problem corresponds to a simple geometric concept of "stripes." We induce the two classes by alternately "coloring" each of the 6 vertical stripes of 10×60 vertices. For this dataset the performance of `longest-path` strongly dominates. We provide a visualization of a typical PQ-graph from a `Stripes` in Figure 2.5.

		$\ell = 8$	$\ell = 16$	$\ell = 32$	$\ell = 64$	$\ell = 128$
USPS 2 vs. 3	`labelProp`	**.980** ± .010	**.982** ± .008	.984 ± .005	**.988** ± .003	**.991** ± .002
	`treeOpt`	.814 ± .055	.885 ± .032	.891 ± .032	.956 ± .013	.959 ± .013
	`longest-path`	.504 ± .001	.940 ± .143	**.987** ± .003	**.988** ± .003	.990 ± .002
		$\ell = 8$	$\ell = 16$	$\ell = 32$	$\ell = 64$	$\ell = 128$
USPS 3 vs. 8	`labelProp`	**.956** ± .009	**.953** ± .007	.961 ± .007	.967 ± .004	.971 ± .004
	`treeOpt`	.797 ± .105	.749 ± .095	.878 ± .013	.935 ± .026	.960 ± .023
	`longest-path`	.505 ± .001	.600 ± .184	**.969** ± .006	**.971** ± .004	**.972** ± .003
		$\ell = 32$	$\ell = 128$	$\ell = 512$	$\ell = 1600$	$\ell = 2048$
ISOLET	`labelProp`	**.661** ± .039	**.764** ± .024	.820 ± .012	.887 ± .006	.899 ± .004
	`treeOpt`	.658 ± .042	.731 ± .012	**.824** ± .008	.888 ± .007	.906 ± .002
	`longest-path`	.524 ± .033	.726 ± .016	.799 ± .016	**.906** ± .007	**.921** ± .006
		$\ell = 800$	$\ell = 1000$	$\ell = 2000$	$\ell = 4000$	$\ell = 6000$
Newsgroups	`labelProp`	**.825** ± .005	**.826** ± .007	**.844** ± .004	**.871** ± .002	**.894** ± .003
	`treeOpt`	.753 ± .006	.758 ± .014	.804 ± .001	.847 ± .002	.878 ± .003
	`longest-path`	.549 ± .004	.798 ± .013	.839 ± .005	.867 ± .003	.890 ± .002
		$\ell = 250$	$\ell = 450$	$\ell = 650$	$\ell = 850$	$\ell = 1050$
Stripes	`labelProp`	.915 ± .013	.948 ± .005	.962 ± .004	.972 ± .004	.978 ± .005
	`treeOpt`	.817 ± .012	.879 ± .017	.909 ± .006	.928 ± .003	.936 ± .006
	`longest-path`	**.921** ± .090	**.998** ± .002	**.997** ± .002	**.994** ± .004	**.993** ± .002

FIGURE 2.4: Experiments

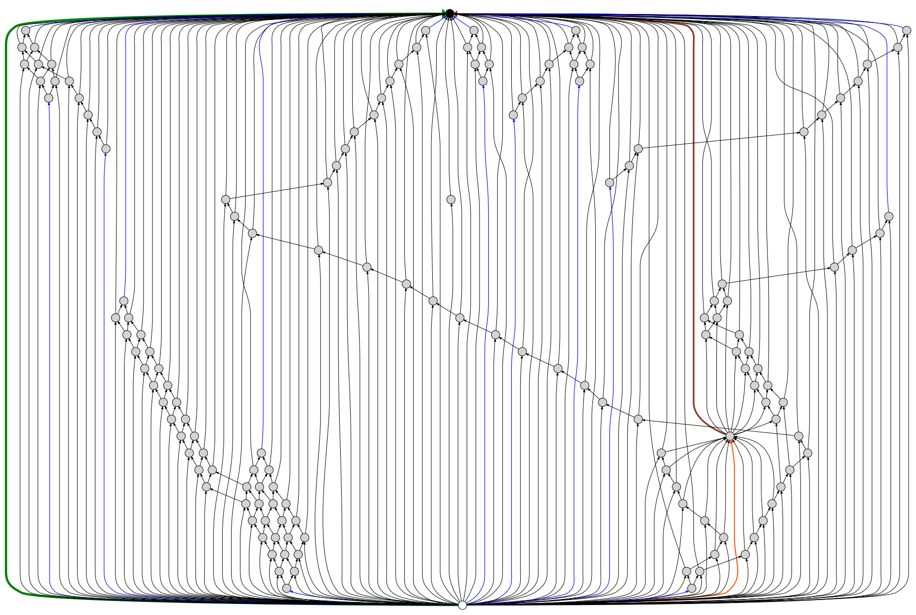


FIGURE 2.5: A PQ-Graph is generated by a `stripes` run ($\ell = 250$). There are 152 super-vertices, the source vertex ($\perp$) is white, the target ($\top$) is black. The value of the flow is 654. Many of the edges encoded represent multi-edges. Of these there are six types: a green edge with a flow of 231, a brown edge with a flow of 7, an orange with a flow of 5, blue with a flow of 2 and the remaining black edges with a flow of 1.

2.6 Properties of the PQ-graph (proof of Theorem 2)

We separate the proof of Theorem 2 into two claims.

Claim 1. *The Picard-Queyranne graph $\mathbb{G}(\mathcal{G}, \mathcal{S})$ of Definition 1 is uniquely defined.*

Claim 2. *The algorithm in Figure 2.1 computes the Picard-Queyranne graph $\mathbb{G}(\mathcal{G}, \mathcal{S})$.*

Proof of Claim 1.

To prove Claim 1 we need to show that $V(\mathbb{G})$ and $E(\mathbb{G})$ are unique. Observe that $V(\mathbb{G})$ is trivially unique as it is the partition on $V(\mathcal{G})$ induced by $\overset{\mathcal{S}}{\sim}$.

To prove $E(\mathbb{G})$ is unique, first recall that $\mathbb{G}$ is an orientation of a quotient graph of $\mathcal{G}$ induced by the partition $(V(\mathbb{G}))$ of $V(\mathcal{G})$; thus up to direction the edges of $\mathbb{G}$ are determined uniquely. Hence, all that is left to prove is that the directions of the edges are uniquely determined. If this is not the case then there exists $\mathbb{G}' = (V(\mathbb{G}), E(\mathbb{G}'))$ and $\mathbb{G}'' = (V(\mathbb{G}), E(\mathbb{G}''))$ satisfying Definition 1 such that there exists $(I, J) \in E(\mathbb{G}')$ and $(J, I) \in E(\mathbb{G}'')$. Then for $i \in I$ and $j \in J$, for all $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ we have $u_i = 1 \Leftrightarrow u_j = 1$ by condition 3 applied to $\mathbb{G}'$ and $\mathbb{G}''$. Thus for arbitrary $i \in I$ and $j \in J$ we have that $u_i = u_j$ which contradicts the fact that I, J are distinct elements in the partition induced by $\overset{\mathcal{S}}{\sim}$. ■

Proof of Claim 2.

In this proof we define $\mathcal{G}, \mathcal{S}, \mathcal{H}, \mathcal{I}$, $\mathbb{G}^0$ and $\mathbb{G}$ as in the algorithm of Figure 2.1. We also define $k := k_{\mathcal{G},\mathcal{S}}$. In step 2 of the algorithm the graph $\mathcal{H}$ is formed by merging the vertices of $\mathcal{G}$ in the example sequence $\mathcal{S}$ which are labeled 0 and 1 to create source vertex s and target vertex t, respectively. In steps 2-4 we then compute the PQ-graph $\mathbb{G}^0(\mathcal{H}, ((s, 0), (t, 1)))$ and then in step 5, we construct $\mathbb{G}(\mathcal{G}, \mathcal{S})$ by "de-merging" vertices s and t. Note that the set of all labelings of $\mathcal{H}$ which satisfy $u_s = 0$ and $u_t = 1$ with minimum-cut is isomorphic to the set of all labelings $\mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ of $\mathcal{G}$ consistent with $\mathcal{S}$ with minimum-cut; and likewise the edge sets $E(\mathcal{H})$ and $E(\mathcal{G})$ are isomorphic. Hence for simplicity in presentation we lightly abuse notation in the following proof at times by identifying $\mathcal{H}$ with $\mathcal{G}$ and $\mathbb{G}^0$ with $\mathbb{G}$.

Lemma 8. *The edge set $E(\mathbb{G})$ computed by the algorithm consists of k edge-disjoint paths from $\perp$ to $\top$*

Proof. We have, from the Ford-Fulkerson algorithm on an unweighted graph, that the flow $\mathcal{F}$ consists of k edge-disjoint directed paths F_1, F_2, ..., F_k from s to t. Take path F_1 and write it as $(s = f_0, f_1, \ldots, f_m = t)$. For every $i < m$ we have $(f_i, f_{i+1}) \in \mathcal{F}$ so $(f_{i+1}, f_i) \notin \mathcal{F}$ and hence $(f_i, f_{i+1}) \in \mathcal{I}$ so F_1 is a directed path in $\mathcal{I}$. For each strongly connected component $H \in V(\mathbb{G})$, let $f^H = \{i \in \mathbb{N}_m : f_i \in H\}$. Suppose we have $i, j, l \in \mathbb{N}_m$ with $i < j < l$ and $i, l \in f^H$ for some $H \in V(\mathbb{G})$. Then since f_i and f_l are in the same strongly connected component (of $\mathcal{I}$) there exists a directed path p in $\mathcal{I}$ from f_l to f_i. Hence we have a path $(f_i, f_{i+1}, \ldots, f_j)$ from f_i to f_j and a path $\langle (f_j, f_{j+1}, \ldots, f_l), p\rangle$ from f_j to f_i, in $\mathcal{I}$. This implies that f_i and f_j are in the same strongly connected component and hence $f_j \in H$ so $j \in f^H$. We can hence write $f^H = \{c : a \leq c < b\}$ for some $a, b \in \mathbb{N}_{m+1}$ with $a \leq b$. This means we can write F_1 as $\langle g^{H_0}, g^{H_1}, \ldots, g^{H_l}\rangle$ for some l and distinct H_i where g^{H_i} is a sequence containing exactly the elements of $\{f_a : a \in f^{H_i}\}$. Since $f_0 = s$ (resp. $f_m = t$), and $f_0 \in f^{H_0}$ (resp. $f_m \in f^{H_l}$) we must have $H_0 = \perp$ (resp. $H_l = \top$). Upon the collapse of $\mathcal{I}$ to $\mathbb{G}$, F_1 hence becomes the path $(\perp = H_0, H_1, \ldots, H_l = \top)$, i.e., F_1 collapses to a path in $\mathbb{G}$ from $\perp$ to $\top$. This happens for each F_i, giving us k edge-disjoint paths in $\mathbb{G}$ from $\perp$ to $\top$. The result is then seen by noting that every directed edge in $\mathbb{G}$ comes from a directed edge in $\mathcal{F}$ (otherwise the edge would be bidirected in $\mathcal{I}$ implying that both vertices on the edge would be in the same strongly connected component (and hence such an edge would disappear)). □

Lemma 9. *Every labelling $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ satisfies the conditions in Definition 1 with respect to $\mathbb{G}$ as computed by the algorithm.*

Proof. Let $\boldsymbol{u}$ be a labelling in $\mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$. First we prove the useful fact that,

$$\text{if } u_i \neq u_j \text{ and } (i, j) \in E(\mathcal{I}) \text{ then } u_i = 0, u_j = 1 \tag{2.4}$$

The above is seen since k edges in $\mathcal{F}$ are cut under $\boldsymbol{u}$ (as $\mathcal{F}$ consists of k edge-disjoint directed path graphs from "s" to "t" with $u_s = 0$ and $u_t = 1$). Thus if there was a cut edge in $E(\mathcal{G}) \setminus E(\mathcal{F})$ the cut of $\boldsymbol{u}$ would be larger than k which is a contradiction. Hence we have (since $u_i \neq u_j$ and $(i,j) \in E(\mathcal{H})$ (as $(i,j) \in E(\mathcal{I})$)) that either $(i,j) \in \mathcal{F}$ or $(j,i) \in \mathcal{F}$. But if $(j,i) \in \mathcal{F}$ then $(i,j) \notin E(\mathcal{I})$ which is a contradiction. Hence we have that $(i,j) \in \mathcal{F}$ so, since $\boldsymbol{u}$ minimises the cut and $u_i \neq u_j$, we must have $u_i = 0$ and $u_j = 1$.

We now prove that $\boldsymbol{u}$ satisfies conditions 1-3 of Definition 1.

Condition 2: Suppose, for contradiction, that i and j are in the same super-vertex $H \in V(\mathbb{G})$ and $u_i \neq u_j$. Then without loss of generality assume $u_i = 1$ and $u_j = 0$. Since i and j are in the same strongly connected component of $\mathcal{I}$ there exists a directed path $(i = v_0, v_1, v_2, \ldots, v_{m-1}, v_m = j)$ in $\mathcal{I}$. Since $u_i \neq u_j$ there exists $a < m$ such that $u_{v_a} \neq u_{v_{a+1}}$ so let b be the minimum such a. Since $u_{v_l} = u_{v_{l+1}}$ for all $l < b$, we have $u_{v_b} = u_{v_0} = u_i = 1$. But, since $(v_b, v_{b+1}) \in E(\mathcal{I})$, and $u_{v_b} \neq u_{v_{b+1}}$, this contradicts (2.4).

Condition 1: We have a vertex $s \in \perp$ with $u_s = 0$ (since $\boldsymbol{u}$ is label-consistent). Hence, if $i \in \perp$ then, by condition 2 (with $H := \perp$) $u_i = u_s = 0$. The same goes for $\top$ (with 1 instead of 0 and t instead of s).

Condition 3: Since $(I, J) \in E(\mathbb{G})$ with $I \neq J$ there exists $i' \in I$ and $j' \in J$ such that there exists an edge (i', j') in $\mathcal{I}$. Hence, if $u_i = 1$ then by condition 2 (with $H := I$), $u_{i'} = 1$ which implies, by (2.4), that $u_{j'} = 1$ and hence, by condition 2 (with $H := J$), $u_j = 1$. □

We now prove the converse of Lemma 9.

Lemma 10. *If a labelling $\boldsymbol{u}$ satisfies the conditions in Definition 1 with respect to $\mathbb{G}$ as computed by the algorithm then we have $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$.*

Proof. By condition 1, $u_s = 0$ and $u_t = 1$ so $\boldsymbol{u}$ is label-consistent. From the proof of Lemma 8 we have that $\mathbb{G}$ is formed of k edge-disjoint paths $P_1, P_2, P_3, \ldots, P_k$ which are the flow paths $F_1, F_2, F_3, \ldots, F_k$ after collapse. Let's consider P_1 and F_1. Let $P_1 = (\perp = H_0, H_1, \ldots, H_l = \top)$ and $F_1 = \langle g^{H_0}, g^{H_1}, \ldots, g^{H_l} \rangle$ for g^{H_i} defined as in the proof of Lemma 8. By condition 2 we have that, for any $a \in \mathbb{N}_l$ there exists $u^{H_a} \in \{0,1\}$ such that for all $i \in H_a$, $u_i = u^{H_a}$. Let $b = \min\{a \in \mathbb{N}_l : u^{H_a} = 1\}$ (note that b exists since $u^{H_l} = 1$ (by condition 1 and since $H_l = \top$) and that $b > 0$ since $u^{H_0} = 0$ (by condition 1 and since $H_0 = \perp$)). Suppose, for contradiction, that $u^{H_c} = 0$ for some $c > b$. Then $d = \min\{a > b : u^{H_a} = 0\}$ is defined. Then we have $u^{H_{d-1}} = 1$ so we have a directed edge (H_{d-1}, H_d) in $\mathbb{G}$ with $u^{H_{d-1}} = 1$ and $u^{H_d} = 0$ which contradicts condition 3. We hence have that $H_a = 1$ for all $a \geq b$. By definition of b we also have that $u^{H_a} = 0$ for all $a < b$. Hence, we have that all elements i of the sequence $\langle g^{H_0}, g^{H_1}, \ldots, g^{H_{b-1}} \rangle$ satisfy $u_i = 0$ and all elements i of the sequence $\langle g^{H_b}, g^{H_{b+1}}, \ldots, g^{H_l} \rangle$ satisfy $u_i = 1$. This means that F_1 has exactly one cut edge (the edge from the final vertex of $g^{H_{b-1}}$ to the first vertex of g^{H_b}). The same argument for all of the k edge-disjoint paths $P_1, P_2, ..P_k$ implies that every flow path has exactly one cut edge which gives us exactly k cut edges in $\mathcal{F}$. Suppose now that we have an edge (i,j) of $\mathcal{G}$ such that $(i,j), (j,i) \notin \mathcal{F}$. Then (i,j) (resp. (j,i)) is a path from i to j (resp. j to i) in $\mathcal{I}$. This implies that i and j are in the same strongly connected component of $\mathcal{I}$ and hence by condition 2, $u_i = u_j$. This means that all the edges that aren't on flow paths are not cut and hence that there are exactly k cut edges in $\mathcal{G}$. □

Lemma 11. *If $\mathbb{G}$ is the graph produced by the algorithm then the vertex set $V(\mathbb{G})$ is the partition induced by $\overset{\mathcal{S}}{\sim}$ on $V(\mathcal{G})$.*

Proof. Since from Lemma 9 we have shown for $V(\mathbb{G})$ as computed by the algorithm the condition

$$\forall \boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S}) : i, j \in H \in V(\mathbb{G}) \Rightarrow u_i = u_j \,,$$

we now need to show,

$$\forall i, j : i \in H \in V(\mathbb{G}), j \notin H \Rightarrow \exists \boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S}) : u_i \neq u_j \,.$$

Thus given $i \in H \in V(\mathbb{G})$, $j \in H' \in V(\mathbb{G})$ and $H \neq H'$ we now show there exists a labeling $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ such that $u_i \neq u_j$.

Assume $H' \not\leq_{\mathbb{G}} H$ (else swap H and H'). Let $\mathbb{D} := \downarrow\{H\}$ then for all $i \in \bigcup\mathbb{D}$ set $u_i := 0$ and for all $i \in \bigcup(V(\mathbb{G}) \setminus \mathbb{D})$ set $u_i := 1$. We have (since $H \in \mathbb{D}$) that $\boldsymbol{u}$ labels all vertices in H as 0 and (since $H' \notin \mathbb{D}$) that $\boldsymbol{u}$ labels all vertices in H' as 1. Hence $\boldsymbol{u}$ labels H and H' differently. Hence, all that is required to prove now is that $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ which, by Lemma 10, is proved by showing that $\boldsymbol{u}$ satisfies the conditions 1-3 in Definition 1 which we now show.

Condition 1: We have that $\perp \in \mathbb{D}$ and hence all vertices $i \in \perp$ are labelled 0 by $\boldsymbol{u}$. We also have that $\top \notin \mathbb{D}$ (as $H' \leq_{\mathbb{G}} \top$ implies $H \neq \top$) and hence all vertices in $\top$ are labelled 1 by $\boldsymbol{u}$.

Condition 2: It is clear from the definition of $\boldsymbol{u}$ that, given $H \in V(\mathbb{G})$, all vertices i in H have the same label u_i since if $H \in \mathbb{D}$ then all vertices in H are labelled 0 and if $H \notin \mathbb{D}$ then all vertices in H are labelled 1.

Condition 3: Suppose we have $(I, J) \in E(\mathbb{G})$ and $i \in I$ with $u_i = 1$. We now prove that for all $j \in J$, $u_j = 1$ which shows that $\boldsymbol{u}$ satisfies condition 3. To prove this assume the converse: that $u_j = 0$. Then we must have that $J \in \mathbb{D}$ so there exists a path P in $\mathbb{G}$ from J to H. Hence we have a path $\langle (I, J), P\rangle$ from I to H in $\mathbb{G}$ which implies that $I \in \mathbb{D}$ and hence all vertices in I, and hence i, are labelled 0 by $\boldsymbol{u}$ which is a contradiction. □

Lemmas 8-11 show that $\mathbb{G}$ satisfies Claim 2. ■

2.7 PQ-game proofs (Propositions 1,2,3 and Theorem 5)

2.7.1 Proof of Proposition 1

Proof. By symmetry we can, without loss of generality, assume that $y = 1$. Let $\perp'$ and $\top'$ be the source and target vertices of $\mathbb{G}(\mathcal{G}, \langle \mathcal{S}, (z, y)\rangle)$. Given a labelling $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$, define $\boldsymbol{u}'$ to be the labelling of $\mathcal{G}$ such that for all $v \in \bigcup\uparrow\{Z\}$ we have $u'_v := 1$ and for all $v \notin \bigcup\uparrow\{Z\}$ we have $u'_v := u_v$. We now show that given any labelling $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ we have that the cutsize of $\boldsymbol{u}'$ is equal to $k_{\mathcal{G},\mathcal{S}}$ and hence (since $u'_z = 1$) that $\boldsymbol{u}' \in \mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z, 1)\rangle)$. To show this split $\mathbb{G}(\mathcal{G}, \mathcal{S})$ into $k_{\mathcal{G},\mathcal{S}}$ edge-disjoint directed paths $p_1, p_2, ..., p_{k_{\mathcal{G},\mathcal{S}}}$. Note that, since $\boldsymbol{u}$ has a cutsize of $k_{\mathcal{G},\mathcal{S}}$, each path p_i has a single cut under $\boldsymbol{u}$. We can write p_i as $(\perp, Y_1, Y_2, ..., Y_m, X_1, X_2, ..., X'_m = \top)$ where each Y_j is not in $\uparrow\{Z\}$ and each X_j is in $\uparrow\{Z\}$. Hence, it is easy to see that since p_i has a single cut under $\boldsymbol{u}$ it also has a single cut under $\boldsymbol{u}'$. $\boldsymbol{u}'$ hence induces a cutsize of $k_{\mathcal{G},\mathcal{S}}$ in $\mathbb{G}(\mathcal{G}, \mathcal{S})$ so $\boldsymbol{u}'$ has a cutsize of $k_{\mathcal{G},\mathcal{S}}$. Note that since (by the above) $k_{\mathcal{G},\langle \mathcal{S},(z,y)\rangle} = k_{\mathcal{G},\mathcal{S}}$ we have $\mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z, 1)\rangle) = \{\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S}) : u_z = 1\}$.

Given $X \in V(\mathbb{G}(\mathcal{G}, \mathcal{S}))$ and $x, y \in X$ we have, for all $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z, 1)\rangle)$, $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ and hence, by Item 1 of Definition 1 we have $u_x = u_y$. This implies that x and y are in the same super-vertex of $\mathbb{G}(\mathcal{G}, \langle \mathcal{S}, (z, y)\rangle)$. Hence, given $X \in V(\mathbb{G}(\mathcal{G}, \mathcal{S}))$, we have that X is a subset of some super-vertex, in $\mathbb{G}(\mathcal{G}, \langle \mathcal{S}, (z, y)\rangle)$. Given $X \in V(\mathbb{G}(\mathcal{G}, \mathcal{S}))$ define $\bar{X}$ to be the super-vertex in $\mathbb{G}(\mathcal{G}, \langle \mathcal{S}, (z, y)\rangle)$ that contains X as a subset.

We now show that for every $X \in \uparrow\{Z\}$ we have $\bar{X} = \top'$. Suppose we have $X \in \uparrow\{Z\}$. Then let $(Z = Y_0, Y_1, ..., Y_m = X)$ be a directed path in $\mathbb{G}(\mathcal{G}, \mathcal{S})$. For all i choose x_i to be an arbitrary vertex in Y_i. Let $\boldsymbol{u}$ be an arbitrary labelling in $\mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z, 1)\rangle)$. Then since $x_0 \in Z$ we have $u_{x_0} = 1$ and hence, by induction on i using Item 3 of Definition 1 (and noting that $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$) we have $u_{x_i} = 1$. This implies that $u_{x_m} = 1$. We have just shown that for every labelling $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z, 1)\rangle)$, $\boldsymbol{u}$ labels all vertices in X as 1 and hence we have that $X \subseteq \top'$ which implies that $\bar{X} = \top'$.

We now show that for every $X \in V(\mathbb{G}(\mathcal{G}, \mathcal{S})) \setminus \uparrow\{Z\}$ we have $\bar{X} \neq \top'$. To see this let x be an arbitrary member of X. Then since $X \neq \top$ choose a labelling $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ such that $u_x = 0$. We then have $u'_x = 0$ so since (by the above) $\boldsymbol{u}' \in \mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z, 1)\rangle)$ we have $x \notin \top'$. Hence $X \not\subseteq \top'$ so we have $\bar{X} \neq \top'$.

We now show that for every $X, Y \in V(\mathbb{G}(\mathcal{G}, \mathcal{S})) \setminus \uparrow\{Z\}$ we have that $\bar{X} \neq \bar{Y}$. To see this choose $x \in X$ and $y \in Y$. Since x and y are in different super-vertices in $\mathbb{G}(\mathcal{G},\mathcal{S})$ we can choose $\boldsymbol{u} \in \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$ such that $u_x \neq u_y$. We then have $u'_x = u_x \neq u_y = u'_y$ so, since (by the above) $\boldsymbol{u}' \in \mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z,1)\rangle)$, we have that x and y are in different super-vertices of $\mathbb{G}(\mathcal{G}, \langle \mathcal{S}, (z,y)\rangle)$. It follows that $\bar{X} \neq \bar{Y}$.

Combining the above results we get the following: for every $X \in \uparrow\{Z\}$ we have $\bar{X} = \top'$ and for every $X \in V(\mathbb{G}(\mathcal{G},\mathcal{S})) \setminus \uparrow\{Z\}$ we have $\bar{X} = X$. By Item 3 of Definition 1 (and since $\mathcal{U}^*_{\mathcal{G}}(\langle \mathcal{S}, (z,1)\rangle) \subseteq \mathcal{U}^*_{\mathcal{G}}(\mathcal{S})$) we have that the directions of the edges in $\mathbb{G}(\mathcal{G}, \langle \mathcal{S}, (z,y)\rangle)$ are inherited from $\mathbb{G}(\mathcal{G},\mathcal{S})$. This completes the proof. □

2.7.2 Proof of Propositions 2 and 3

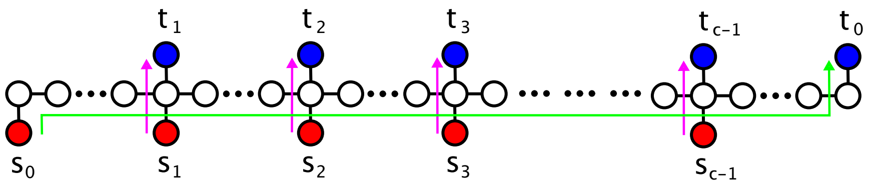
Proof of Proposition 2. The comb PQ-graph $\mathbb{G}_{c,r}$ is generated from the labeled graph $\mathcal{G}$ with $cr + 2c - 1$ vertices, and $2c$ labels. We initially have a path,

$$p^0 := (s_0, v_0, v_1, \ldots, v_{cr-1}, v_{cr}, t_0), ,$$

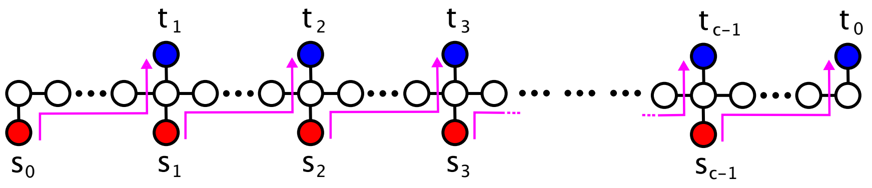
of $cr+3$ vertices and then every r vertices (from v_r to $v_{(c-1)r}$) we have a bottom vertex s_i and a top vertex t_i forming a three-vertex path $p^i := (s_i, v_{ir}, t_i)$ for $i \in \mathbb{N}_{c-1}$ that intersects p^0. Thus if $\{s_0, \ldots, s_{c-1}\}$ are each labeled "0" and $\{t_0, \ldots, t_{c-1}\}$ are each labeled "1"; then the PQ-graph $\mathbb{G}_{c,r}$ will have $cr + 3$ vertices with all source "s"-vertices "glued" together to form $\perp$ and likewise the "t"-vertices form $\top$ (Notationally it is still convenient to refer to these "s" and "t" vertices within "$\perp$" and "$\top$" as they now correspond to edges leaving the super-vertex). Thus $P := \{p^0, \ldots, p^{c-1}\}$ is one path cover (see Figure 2.6a). However, we also have another "zig-zag" path cover (see Figure 2.6b)

$$P' = \{(s_0, \ldots, t_1), (s_1, \ldots, t_2), \ldots, (s_{c-2}, \ldots, t_{c-1}), (s_{c-1}, \ldots, t_0)\}.$$

Observe that path cover P has one path of length $cr + 2$ and $c - 1$ paths of length 2 whereas P' has c paths of length $r + 2$; thus the path lengths are very unbalanced in P as opposed to P' which leads to the following mistake bounds $M_{\texttt{0-Ising}} \leq O(\log r + \log c + (c-1))$ and $M_{\texttt{fixed-paths}(P')} \leq O(c \log r)$.



(A) Path cover P of $\mathbb{G}_{c,r}$



(B) Path cover P' of $\mathbb{G}_{c,r}$

FIGURE 2.6: Two path covers of the comb PQ-graph

Now if we assume the true labeling of $\mathcal{G}$ is all "0" except at the "t"-vertices then an adversary may also force $\texttt{fixed-paths}(P')$ to incur $\Omega(c \log r)$ mistakes by forcing $\Omega(\log r)$ mistakes in first path $(s_0, \ldots, t_1)$ (e.g. by first selecting $v_{r/2+1}$ then $v_{3r/4+1}$ then $v_{7r/8+1}$ and so on) and then $\Omega(\log r)$ mistakes in the second path $(s_1, \ldots, t_2)$ and so on. Thus comparing the two algorithms by setting $r := 2^c$ we have that $M_{\texttt{0-Ising}} \leq O(c)$ and $M_{\texttt{fixed-paths}(P')} \geq \Omega(c^2)$. □

Proof of Proposition 3. We consider a generalization of the (c,r)-comb to the (b,c,r)-grid, with $b, c < r$. The (b,c,r)-grid is essentially is a "stacking" of $b + 1$ of the

(c, r)-combs. More precisely we have $b+1$ directed path graphs of the form

$$p^{i,\cdot} := (s_{i,.}, v_{i,0}, v_{i,1}, \dots, v_{i,cr-1}, v_{i,cr}, t_{i,.})$$

for $i \in \{0, 1, \dots, b\}$ each with $cr + 3$ vertices. We then create a grid intersecting these $b+1$ paths with $c-1$ paths of the form

$$p^{\cdot,j} := (s_{.,j}, v_{1,jr}, v_{2,jr}, \dots, v_{b,jr}, t_{.,j})$$

for $j \in \mathbb{N}_{c-1}$, so that the grid has $(b+1)(cr+3) + 2(c-1)$ vertices and a label-consistent minimum-cut size of $b+c$. We assume the true labeling of each vertex is "0" except the "t" vertices. For the following we define the path segments

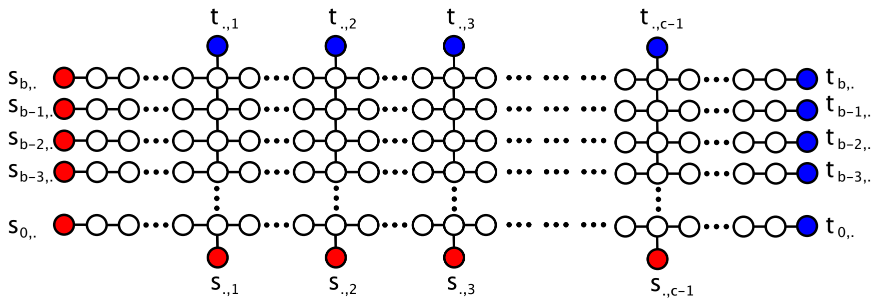


FIGURE 2.7: The grid PQ-graph

$z(i,j) := (v_{i,jr}, v_{i,jr+1}, \dots, v_{i,(j+1)r-1})$ for $i \in \{0, 1, \dots, b\}$ and $j \in \{0, 1, \dots, c-1\}$. We now describe an adversarial strategy for the shortest path heuristic: First force, $\Omega(\log(r-b))$ mistakes on $z(\frac{b}{2}+1, 0)$ by first selecting $v_{\frac{b}{2}+1,\frac{r+b}{2}+1}$, then selecting $v_{\frac{b}{2}+1,\frac{3(r+b)}{4}+1}$, then selecting $v_{\frac{b}{2}+1,\frac{7(r+b)}{8}+1}$ and so on. In the same way then force $\Omega(\log(r-b))$ mistakes on $z(\frac{b}{2}+1, j)$ for every $1 \leq j \leq c-1$ in turn. We have now forced $\Omega(c\log(r-b))$ mistakes on $p^{\frac{b}{2}+1,\cdot}$. In the same way then force $\Omega(c\log(r-b))$ mistakes on path $p^{\frac{3b}{4}+1,\cdot}$, then $\Omega(c\log(r-b))$ mistakes on path $p^{\frac{7b}{8}+1,\cdot}$ and so on. This gives us a total of $\Omega(c\log(b)\log(r-b))$ mistakes. Thus, if we set $b = c, r = 2^c$ we have that $M_{\texttt{shortest-path}} = \Omega(c\log(c)\log(2^c - c))$. Noting that $\Omega(\log(2^c - c)) = \Omega(c)$ we hence have that $M_{\texttt{shortest-path}} = \Omega(c^2\log(c))$.

On the other hand, since there are $O(bcr)$ vertices in the grid and the grid has $b+c$ edge disjoint paths from source to sink we must have that, for any path cover P, $M_{\texttt{fixed-paths}(P)} = O((b+c)\log bcr)$. Thus if we set $b = c, r = 2^c$ we have that $M_{\texttt{fixed-paths}(P)} = O(c^2)$. Hence we have $M_{\texttt{shortest-path}} = \Omega(c^2\log(c))$ and $M_{\texttt{fixed-paths}(P)} = O(c^2)$. $\square$

2.7.3 Proof of Theorem 5

Proof of Theorem 5. The bounds for strategies `fixed-paths` and `0-Ising` are straightforward (see discussion in Section 2.4.2) and we focus on the proof of (2.2) for the `longest-path` strategy.

Let $(\mathbb{G}_1, \dots, \mathbb{G}_T)$ be the sequence of PQ-graphs in a PQ-game of cutsize k (where $\mathbb{G}_T$ is the final PQ-graph of cutsize k and hence the mistake made in this graph is not counted towards the mistakes of the PQ-game). Recall that, for any $t \leq T$, the edge set $E(\mathbb{G}_t)$ may be partitioned (non-uniquely) into k edge-disjoint directed paths. For all $t \leq T$ we will constuct k edge-disjoint directed paths $\{p_t^1, p_t^2, ..., p_t^k\}$. Note that we are treating each path as a set of edges rather than vertices so that $|p_t^i|$ is the length of the ith path in $\mathbb{G}_t$. Let M_t denote the number of mistakes ("backwards cumulative") of `longest-path` made in the sequence $(\mathbb{G}_t, \mathbb{G}_{t+1}, \dots, \mathbb{G}_T)$. We prove by a "reverse" induction (i.e., from $t = T$ to $t = 1$) that for all t there exist k edge-disjoint directed paths $\{p_t^1, p_t^2, ..., p_t^k\}$ from $\perp$ to $\top$ in $\mathbb{G}_t$, and numbers ("mistakes on a path") $\{M_t^1, M_t^2, ..., M_t^k\}$ with $M_t^i \leq \log|p_t^i|$ for $1 \leq i \leq k$ such that the "backwards cumulative mistakes" is $M_t = \sum_{i=1}^k M_t^i$.

We now consider the base case of our induction: for $t = T$ we arbitrarily choose $\{p_T^1, p_T^2, ..., p_T^k\}$ to be an arbitrary set of k edge-disjoint paths from $\perp$ to $\top$ in $\mathbb{G}_T$. Let $M_T^1 = \cdots = M_T^k = 0$. We clearly have $M_T = 0 = \sum_{i=1}^k M_T^i$ and that $M_T^i = 0 \le \log |p_T^i|$ for $1 \le i \le k$.

Suppose now that the inductive hypothesis holds for some $t > 1$. We proceed to show that it holds for $t-1$. On trial $t-1$ we receive example (Z_{t-1}, y_{t-1}). If $Z_{t-1} \in \{\perp, \top\}$ then since we are "within" a PQ-game we have not made a mistake, so the inductive hypothesis holds trivially with $p_{t-1}^i := p_t^i$ and $M_{t-1}^i := M_t^i$ for $1 \le i \le k$. Suppose instead that $Z_{t-1} \notin \{\perp, \top\}$. Without loss of generality assume that the label $y_{t-1} = 1$. Define $\hat{p}_t^i$ to be equal to p_t^i except that the "final" edge is removed i.e., $\hat{p}_t^i := p_t^i \cap \{(I, J) \in E(\mathbb{G}_t) : J \neq \top\}$. Since (by Proposition 1) $\mathbb{G}_t = \texttt{merge}(\mathbb{G}_{t-1}, \uparrow\{Z_{t-1}\})$ observe that $\hat{p}_t^i \subset E(\mathbb{G}_{t-1})$. By construction observe that since $Z_{t-1} \notin V(\mathbb{G}_t)$, it is in no path $\hat{p}_t^i$ however there is at least one vertex $Z'_{t-1} \in V(\mathbb{G}_{t-1})$ and a i' such that $\hat{p}_t^{i'} \cup \{(Z'_{t-1}, Z_{t-1})\}$ is a directed path in $\mathbb{G}_{t-1}$. Define $r_{t-1}^{\perp}$ to be the longest directed path in $\mathbb{G}_{t-1}$ from $\perp$ to Z_{t-1} and $r_{t-1}^{\top}$ to be the longest directed path in $\mathbb{G}_{t-1}$ from Z_{t-1} to $\top$. Now define path $p_{t-1}^{i'} := \hat{p}_t^{i'} \cup \{(Z'_{t-1}, Z_{t-1})\} \cup r_{t-1}^{\top}$. Finally select an arbitrary edge-disjoint extensions of the paths $\{\hat{p}_t^1, \ldots, \hat{p}_t^{i'-1}, \hat{p}_t^{i'+1}, \ldots, \hat{p}_t^k\}$ to paths $\{p_{t-1}^1, \ldots, p_{t-1}^{i'-1}, p_{t-1}^{i'+1}, \ldots, p_{t-1}^k\}$ so each is a path from $\perp$ to $\top$, in $\mathbb{G}_{t-1}$.

Now consider the sub-case where `longest-path` incurred a mistake in $\mathbb{G}_{t-1}$. Then we have $M_{t-1} = M_t + 1$ so choose $M_{t-1}^{i'} := 1 + M_t^{i'}$ and choose, for all $i \neq i'$, $M_{t-1}^i := M_t^i$. By the inductive hypothesis we hence have that $M_{t-1} = 1 + M_t = 1 + \sum_{i=1}^k M_t^i = \sum_{i=1}^t M_{t-1}^i$. We predicted Z_{t-1} to be 0 so we hence have that $|r_{t-1}^{\perp}| \le |r_{t-1}^{\top}|$. Since $p_{t-1}^{i'} \setminus r_{t-1}^{\top}$ is a directed path in $\mathbb{G}_{t-1}$ from $\perp$ to Z_{t-1} we have $|p_{t-1}^{i'} \setminus r_{t-1}^{\top}| \le |r_{t-1}^{\perp}|$ and we hence have that $|p_{t-1}^{i'} \setminus r_{t-1}^{\top}| \le |r_{t-1}^{\top}|$. Hence we have, by the inductive hypothesis, that $|p_{t-1}^{i'}| = |p_{t-1}^{i'} \setminus r_{t-1}^{\top}| + |r_{t-1}^{\top}| \ge 2|p_{t-1}^{i'} \setminus r_{t-1}^{\top}| = 2|p_t^{i'}| \ge 2 \times 2^{M_t^{i'}} = 2^{1+M_t^{i'}} = 2^{M_{t-1}^{i'}}$. Since, $|p_{t-1}^i| \ge |p_t^i|$ for all i, we have, by the inductive hypothesis, that, for all $i \neq i'$, $|p_{t-1}^i| \ge |p_t^i| \ge 2^{M_t^i} = 2^{M_{t-1}^i}$. And thus the sub-case with a mistake is shown.

Now consider the sub-case where we didn't make a mistake in $\mathbb{G}_{t-1}$. Then we have that $M_{t-1} = M_t$ so choose, for all i, $M_{t-1}^i := M_t^i$ and we have, by the inductive hypothesis, $M_{t-1} = M_t = \sum_{i=1}^k M_t^i = \sum_{i=1}^k M_{t-1}^i$. Since, for all i, $|p_{t-1}^i| \ge |p_t^i|$, we have, by the inductive hypothesis, that $|p_{t-1}^i| \ge |p_t^i| \ge 2^{M_t^i} = 2^{M_{t-1}^i}$. And thus the sub-case without a mistake is shown.

We conclude observing that the cumulative mistakes from trials 1 to T of the PQ-game is $M_1 = \sum_{i=1}^k M_1^i \le \sum_{i=1}^k \log |p_1^i|$ □

2.8 Global mistake analysis (proofs of Theorems 4 and 7)

2.8.1 Proof of Theorem 4

Suppose we have a uniformly-labeled subgraph $\mathcal{C}$.

A *covering sequence* is an example sequence (i.e. a sequence of pairs (v, y) where y is the label of vertex v) that contains every vertex in $\mathcal{G}$ and the label of any vertex in $\mathcal{C}$ is, without loss of generality, equal to 1.

When, given a covering sequence, $\mathcal{R}$, we say "mistakes made in $\mathcal{C}$ with $\mathcal{R}$" we mean "mistakes made in $\mathcal{C}$ when algorithm $\mathcal{A}$ is run on $\mathcal{R}$". We also say, for a vertex w, "w is predicted 0 (resp. 1) with $\mathcal{R}$" when we mean "when algorithm $\mathcal{A}$ is run on $\mathcal{R}$ the label of w is predicted, by $\mathcal{A}$, to be 0 (resp. 1).

Lemma 12. *Suppose we have a covering sequence $\mathcal{R} = \langle \mathcal{S}, (v, 1), \mathcal{T} \rangle$ for example sequences $\mathcal{S}$ and $\mathcal{T}$ where $v \notin \mathcal{C}$. Let $\mathcal{R}' := \langle \mathcal{S}, \mathcal{T}, (v, 0) \rangle$. Then the number of mistakes made in $\mathcal{C}$ with $\mathcal{R}'$ is at least the number of mistakes made in $\mathcal{C}$ with $\mathcal{R}$.*

Proof. All we need to show is that given some $w \in \mathcal{C}$ in which a mistake is made on w (i.e. w is predicted 0) with $\mathcal{R}$, then a mistake is made (i.e. w is predicted 0) with $\mathcal{R}'$. This is clearly true if $(w, 1)$ is in $\mathcal{S}$ (the algorithms are identical on $\mathcal{S}$) so assume otherwise. We must then have that $(w, 1)$ is in $\mathcal{T}$. If w is predicted 1 with $\mathcal{R}'$ then by monotonicity and permutation invariance, w must be predicted 1 with $\mathcal{R}$ which is a contradiction. We must hence have that w is predicted 0 with $\mathcal{R}'$. □

Lemma 13. *Suppose we have a covering sequence $\mathcal{R} = \langle \mathcal{S}, (v, 0), \mathcal{T} \rangle$ for example sequences $\mathcal{S}$ and $\mathcal{T}$ where $v \notin \mathcal{C}$. Let $\mathcal{R}' := \langle (v, 0), \mathcal{S}, \mathcal{T} \rangle$. Then the number of mistakes made in $\mathcal{C}$ with $\mathcal{R}'$ is at least the number of mistakes made in $\mathcal{C}$ with $\mathcal{R}$.*

Proof. All we need to show is that given some $w \in \mathcal{C}$ in which a mistake is made on w (i.e. w is predicted 0) with $\mathcal{R}$, then a mistake is made (i.e. w is predicted 0) with $\mathcal{R}'$. This is clearly true if $(w, 1)$ is in $\mathcal{T}$ (the algorithms are identical on $\mathcal{T}$ by permutation invariance) so assume otherwise. We must then have that $(w, 1)$ is in $\mathcal{S}$. Since w is predicted 0 with $\mathcal{R}$, by monotonicity and permutation invariance w must be predicted 0 with $\mathcal{R}'$. □

Lemma 14. *Given a covering sequence $\mathcal{R}$, there exists sequences $\mathcal{U}$ and $\mathcal{V}$ such that the elements of $\mathcal{U}$ are $\{(v, 0) : v \notin \mathcal{C}\}$ and the elements of $\mathcal{V}$ are $\{(v, 1) : v \in \mathcal{C}\}$ and at least as many mistakes are made in $\mathcal{C}$ with $\langle \mathcal{U}, \mathcal{V} \rangle$ as are made with $\mathcal{R}$.*

Proof. Repeatedly use Lemma 12 on $\mathcal{R}$ to form a covering sequence $\mathcal{R}'$ which makes at least as many mistakes in $\mathcal{C}$ as $\mathcal{R}$ and in which for every $v \notin \mathcal{C}$, the label of v is 0. Next, repeatedly use Lemma 13 on $\mathcal{R}'$ to form the covering sequence $\langle \mathcal{U}, \mathcal{V} \rangle$ in the lemma. □

Suppose then that $\mathcal{R}$ is our true label sequence. Then find sequences $\mathcal{U}$ and $\mathcal{V}$ as in Lemma 14. The number of mistakes made in $\mathcal{C}$ with sequence $\mathcal{R}$ is then no more than the number of mistakes made in $\mathcal{C}$ with sequence $\langle \mathcal{U}, \mathcal{V} \rangle$ which, by the Markov property, is bounded above by $\mathcal{B}_{\mathcal{A}}(\mathcal{C}; \mathcal{G})$. This completes the proof of Theorem 4. ■

2.8.2 Proof Theorem 7

The proof of Theorem 7 separates into three cases: high-connectivity clusters, low-connectivity clusters, and a tree cluster. In each case we assume that we have received the label of each vertex in $\partial_e(\mathcal{C})$ and we then upper bound the number of mistakes in $\mathcal{C}$. Without loss of generality assume that each vertex of $\mathcal{C}$ is labelled 1 and each vertex of $\partial_e(\mathcal{C})$ is labelled 0.

CASE 1 : If $\kappa(\mathcal{C}) > |\partial_e^E(\mathcal{C})|$ then $\mathcal{B}_{\mathcal{A}}(\mathcal{C}; \mathcal{G}) \in \mathcal{O}(1)$.

Proof. Suppose we have made a single mistake in cluster $\mathcal{C}$. Then we have received the true label y_v (equal to 1) for some vertex $v \in \mathcal{C}$. Let $\boldsymbol{u}$ be a consistent (with the observed labels) labelling of $\mathcal{C} \cup \partial_e(\mathcal{C})$ that minimises the cut. Note that we have $u_v = 1$ and for all $w \in \partial_e(\mathcal{C})$ we have $u_w = 0$. So if $u_z = 1$ for all $z \in \mathcal{C}$ then $\boldsymbol{u}$ has a cut of size $|\partial_e^E(\mathcal{C})|$. If there exists a vertex $z \in \mathcal{C}$ with $u_z = 0$ then, since there are at least $\kappa(\mathcal{C})$ edge disjoint paths between z and v, we have that $\boldsymbol{u}$ has a cutsize of at least $\kappa(\mathcal{C})$. So since $\kappa(\mathcal{C}) > |\partial_e^E(\mathcal{C})|$ and $\boldsymbol{u}$ minimises the cut we must have that $u_z = 1$ for all $z \in \mathcal{C}$. Hence, since the the next prediction in $\mathcal{C}$ is consistent with a labelling of minimum-cut, it will predict the label as 1 so will not be a mistake. We can hence make at most one mistake in $\mathcal{C}$. □

CASE 2 : If $\kappa(\mathcal{C}) \leq |\partial_e^E(\mathcal{C})|$ then $\mathcal{B}_{\mathcal{A}}(\mathcal{C}; \mathcal{G}) \in \mathcal{O}(|\partial_e^E(\mathcal{C})|(1 + |\partial_e^E(\mathcal{C})| - \kappa(\mathcal{C})) \log \mathcal{N}_{|\partial_e^E(\mathcal{C})|+1})$.

Proof. We consider the sequence of PQ-games after we have made a single mistake. We first bound the cutsize of the first PQ-game. Since we have made a single mistake we have received the true label y_v (equal to 1) for some vertex $v \in \mathcal{C}$. Let $\boldsymbol{u}$ be a consistent (with the observed labels) labelling of $\mathcal{C} \cup \partial_e(\mathcal{C})$ that minimises the cut. Note that we have $u_v = 1$ and for all $w \in \partial_e(\mathcal{C})$ we have $u_w = 0$. So if $u_z = 1$ for all $z \in \mathcal{C}$ then $\boldsymbol{u}$ has a cut of size $|\partial_e^E(\mathcal{C})|$. If there exists a vertex z with $u_z = 0$

then, since there are at least $\kappa(\mathcal{C})$ edge disjoint paths between z and v, we have that $\boldsymbol{u}$ has a cut of size at least $\kappa(\mathcal{C})$. Hence, since $\kappa(\mathcal{C}) \leq |\partial_e^E(\mathcal{C})|$ we have that $\boldsymbol{u}$ has a cutsize of at least $\kappa(\mathcal{C})$. Hence, the cutsize of the first PQ-game is at least $\kappa(\mathcal{C})$.

Since the true cutsize is $|\partial_e^E(\mathcal{C})|$ we hence have that up to $|\partial_e^E(\mathcal{C})| - \kappa(\mathcal{C}) + 1$ PQ-games are played and the cutsize of each PQ-game is no greater than $|\partial_e^E(\mathcal{C})|$. We now bound the number of mistakes made in each PQ-game. To do this we first bound the number of super-vertices in any of the PQ-graphs. Suppose we have a set X of vertices in $\mathcal{C}$ with connectivity greater than $|\partial_e^E(\mathcal{C})|$. Then suppose $\boldsymbol{u}$ is a consistent (with the observed labels, at any point in the algorithm) labelling of $\mathcal{C} \cup \partial_e(\mathcal{C})$ that minimizes the cut. We must have that $\boldsymbol{u}$ has a cutsize equal to the cutsize of a PQ-game and hence has a cutsize no greater than $|\partial_e^E(\mathcal{C})|$. Suppose, for contradiction, that there exist vertices $x, y \in X$ such that $u_x \neq u_y$. Then since there are at over $|\partial_e^E(\mathcal{C})|$ edge-disjoint paths from x to y we have that $\boldsymbol{u}$ has a cutsize greater than $|\partial_e^E(\mathcal{C})|$ which is a contradiction. We have just shown that for any consistent (with the observed labels) labelling, $\boldsymbol{u}$, of $\mathcal{C} \cup \partial_e(\mathcal{C})$ that minimizes the cut we have that u_x is identical for all $x \in X$. By definition of the PQ-graph this means that X is a subset of some super-vertex of the PQ-graph. Hence, we have that any PQ-graph in the algorithm has at most $1 + \mathcal{N}_{|\partial_e^E(\mathcal{C})|+1}(\mathcal{C})$ super-vertices (the "$+1$" corresponds to the super-vertex formed from $\partial_e(\mathcal{C})$).

Now we can apply Theorem 5 to sum the bounds of each PQ-game where we upper bound k for each game by $|\partial_e^E(\mathcal{C})|$ and the path length by $\mathcal{N}_{|\partial_e^E(\mathcal{C})|+1}(\mathcal{C})$. Since there are at most $|\partial_e^E(\mathcal{C})| - \kappa(\mathcal{C}) + 1$ PQ-games this gives us a maximum of at most $\mathcal{O}(|\partial_e^E(\mathcal{C})|(1 + |\partial_e^E(\mathcal{C})| - \kappa(\mathcal{C})) \log \mathcal{N}_{|\partial_e^E(\mathcal{C})|+1})$ mistakes inside the PQ-games. Finally, note that there are at most $|\partial_e^E(\mathcal{C})| - \kappa(\mathcal{C}) + 1$ mistakes between PQ-games. The result follows. □

CASE 3 : If $\mathcal{C}$ is a tree then $\mathcal{B}_{\mathcal{A}}(\mathcal{C};\mathcal{G}) \in \mathcal{O}(|\partial_e^E(\mathcal{C})| \log D(\mathcal{C}))$.

Theorem 15. *Given a tree structured subgraph, $\mathcal{C}$, of $\mathcal{G}$ we have $\mathcal{B}_{\mathcal{A}}(\mathcal{C};\mathcal{G}) \in \mathcal{O}(|\partial_e^E(\mathcal{C})| \log_2(D(\mathcal{C})))$*

Proof of Theorem 15

Suppose $k := \partial_e^E(\mathcal{C})$. For every vertex $v \in \mathcal{C}$ let $\eta(v)$ be the number of neighbours of v that are not in $\mathcal{C}$. Then consider the tree $\mathcal{T}$ which is formed from $\mathcal{C}$ by adding, to each vertex v, $\eta(v)$ vertices. Label $\mathcal{T}$ as follows: if $v \in \mathcal{C}$ label v as 1 and if $v \notin \mathcal{C}$ label v as 0. Then $\mathcal{B}_{\mathcal{A}}(\mathcal{C};\mathcal{G})$ is upper-bounded by the maximum number of mistakes made in $\mathcal{T}$ with any permutation of the vertices. Note that the cutsize of the labelling of $\mathcal{T}$ is equal to k so by Section 2.10, the number of mistakes made in $\mathcal{T}$ is upper-bounded by $\mathcal{O}\left(\mathrm{LB}(\mathcal{T}, k)\right)$ which is upper bounded by $\mathcal{O}(k \log(D(\mathcal{T})))$. The result follows since $D(\mathcal{T}) \leq D(\mathcal{C}) + 2$. □

2.9 Regularity properties of `longest-path` and `0-Ising` (proof of Theorem 6)

The proof of all properties except for the label-monotonicity of `longest-path` are straightforward.

2.9.1 Proof that `longest-path` is label-monotone

In this proof we use the more explicit notation $(v \rightarrow w)$ for a directed edge from v to w.

Let $\mathcal{S}$ be an example sequence. Let z be a vertex of $\mathcal{G}$ that is not contained in $\mathcal{S}$. Define $\mathcal{S}' := \langle \mathcal{S}, (z, 0) \rangle$ (note that, in what follows, we don't lose generality by assuming that z is labelled 0 by the symmetry over switching the labels 0 and 1 on all vertices). Let $(\mathcal{H}, s, t)$ (resp. $(\mathcal{H}', s', t')$) be the result of step 1 of the PQ-graph construction algorithm (see Figure 2.1) when run on sequence $\mathcal{S}$ (resp. $\mathcal{S}'$). Note that $\mathcal{H}'$ is identical to $\mathcal{H}$ except that the vertices s and z are merged into a single vertex s'. Let $\mathbb{G}$ (resp. $\mathbb{G}'$) be the graph formed at step 4 of the PQ-graph construction algorithm (see Figure 2.1) when run on sequence $\mathcal{S}$ (resp. $\mathcal{S}'$). Let $\bot$ and $\top$ (resp. $\bot'$ and $\top'$) be the source and target super-vertices of $\mathbb{G}$ (resp. $\mathbb{G}'$) respectively. Given $v \in V(\mathcal{H})$ (resp. $v \in V(\mathcal{H}')$) define $\Psi(v)$ (resp. $\Psi'(v)$) to be the super-vertex in $\mathbb{G}$ (resp. $\mathbb{G}'$) that contains v.

To prove label monotonicity we need to show that given $h \in V(\mathcal{H}') \setminus \{s', t'\}$, if the label of h is predicted 0 in $\mathbb{G}$ then it is predicted 0 in $\mathbb{G}'$. Let ϕ and ψ be the longest directed paths in $\mathbb{G}$ from $\bot$ to $\Psi(h)$ and from $\Psi(h)$ to $\top$ respectively. Let ϕ' and ψ' be the longest directed paths in $\mathbb{G}'$ from $\bot'$ to $\Psi'(h)$ and from $\Psi'(h)$ to $\top'$ respectively. If $\Psi'(h) = \bot'$ then we are done since then the label of h is predicted 0 in $\mathbb{G}'$, so assume otherwise. Since the label of h is 0 in $\mathbb{G}$ we also have that $h \notin \top$.

We need the following proposition about the graphs $\mathbb{G}$ and $\mathbb{G}'$. We will prove the proposition later in the section.

Proposition 4. *We have the following results:*

if $X, Y \in V(\mathbb{G}'), X \subseteq \top$ *and* $(X \to Y) \in E(\mathbb{G}')$ *then* $Y \subseteq \top$ **[edge creation]** (a)

if $v \in V(\mathcal{H}) \setminus (\top \cup \{s,z\})$ *then* $\Psi'(v) = \bot'$ *or* $\Psi'(v) = \Psi(v)$ **[vertex collapse/conservation]** (b)

if $v, w \in V(\mathcal{H}) \setminus (\top \cup \{s,z\})$ *and* $\Psi'(v) \neq \bot'$ *then* **[edge conservation]**
$$(\Psi'(v) \to \Psi'(w)) \in E(\mathbb{G}') \Leftrightarrow (\Psi(v) \to \Psi(w)) \in E(\mathbb{G}) \quad \text{(c)}$$

Lemma 16. $|\phi'| \leq |\phi|$

Proof. Since, $\Psi'(h) \neq \bot'$ write ϕ' as $(\bot', X_1, X_2, ..., X_m = \Psi'(h))$. If, for some i, we have $X_i \subseteq \top$ then by Proposition 4 Item (a) (by induction through $X_i, X_{i+1}, ..., X_m = \Psi'(h)$ using inductive hypothesis $X_j \subseteq \top$) we would have $\Psi'(h) \subseteq \top$ which would imply that $h \in \top$ which is a contradiction. Also, if for some i we have $s' \in X_i$, we would have $X_i = \bot'$ which is a contradiction. Hence, for every i, we have some vertex $x_i \in \mathcal{H} \setminus (\top \cup \{s, z\})$ such that $X_i = \Psi'(x_i)$. We hence have a path $(\Psi'(x_1), \Psi'(x_2), ..., \Psi'(x_m))$ in $\mathbb{G}'$ where, for all i, $x_i \in \mathcal{H} \setminus (\top \cup \{s, z\})$ and $\Psi'(x_i) \neq \bot'$. By Proposition 4 Item (b) we have that for each i, $\Psi'(x_i) = \Psi(x_i)$ (and since $h \in \mathcal{H} \setminus (\top \cup \{s, z\})$, we have $\Psi'(x_m) = \Psi'(h) = \Psi(h)$) so by Proposition 4 Item (c) $(\Psi(x_1), \Psi(x_2), ..., \Psi(x_m) = \Psi(h))$ is a directed path in $\mathbb{G}$. Since $\Psi(x_1) = \Psi'(x_1)$ and (since $\Psi'(x_1)$ is a subset of $\mathcal{H}'$) $s \notin \Psi'(x_1)$ we must have $\Psi(x_1) = \Psi'(x_1) \neq \Psi(s) = \bot$. We can hence continue (in $\mathbb{G}$) the path $(\Psi(x_1), \Psi(x_2), ..., \Psi(x_m) = \Psi(h))$ back to $\bot$ giving us, for some $m' \geq 0$ a directed path $(\bot, Y_1, Y_2, ...Y_{m'}, \Psi(x_1), \Psi(x_2), ..., \Psi(x_m) = \Psi(h))$ in $\mathbb{G}$. We hence have constructed a directed path in $\mathbb{G}$, from $\bot$ to $\Psi(h)$ that is at least as long as ϕ' which proves the result. □

Lemma 17. $|\psi| \leq |\psi'|$

Proof. Since $h \notin \top$ and hence $\Psi(h) \neq \top$ write ψ as $(\Psi(h) = X_1, X_2, ..., X_m, \top)$. Let $x_1 := h$ and for $i > 2$ let x_i be an arbitrary member of X_i. Since $h \notin \top$ and $h \neq s, z$ we have $x_1 \in \mathcal{H} \setminus (\top \cup \{s, z\})$ For $i > 2$ we know (since there is no edge in $\mathbb{G}$ that goes into $\bot$) that $X_i \neq \bot$ and hence $s \notin X_i$. Since $X_i \neq \top$ we then have that $x_i \in \mathcal{H} \setminus (\top \cup s)$. Hence, for all i we have $x_i \in \mathcal{H} \setminus (\top \cup s)$. Suppose, for contradiction, that, for some i, $x_i = z$. Then $z \in X_i \neq \top$ so $z \notin \top$. Hence, by Proposition 1 and since $\Psi(h)$ is downstream of X_i we have that $h \in \bot'$ which contradicts the assumptions of h. Hence we have that, for all i, $x_i \neq z$ and hence $x_i \in \mathcal{H} \setminus (\top \cup \{s, z\})$.

Note that $(\Psi(x_1), \Psi(x_2), ..., \Psi(x_m))$ is a directed path in $\mathbb{G}$ (as it is a subpath of ψ). We now prove the following by induction on i:

1. $\Psi'(x_i) \neq \bot'$
2. $\Psi'(x_i) = \Psi(x_i)$
3. $(\Psi'(x_i) \to \Psi'(x_{i+1}))$ is an edge in $\mathbb{G}'$ (for $i \neq m$)

Note that items 2 and 3 can be proved from Item 1 as follows: Since $x_i \in \mathcal{H} \setminus \{\top \cup \{s, z\}\}$ and $\Psi'(x_i) \neq \bot'$ then by Proposition 4 Item (b) we have $\Psi'(x_i) = \Psi(x_i)$.

Since $x_i, x_{i+1} \in \mathcal{H} \setminus \{\top \cup \{s, z\}\}$ and $\Psi'(x_i) \neq \bot'$ then by Proposition 4 Item (c) we have that $(\Psi'(x_i) \to \Psi'(x_{i+1}))$ is an edge in $\mathbb{G}'$.

We now prove Item 1 from the inductive hypothesis: For $i = 1$ we have (since $h \notin \bot'$) that $\Psi'(x_1) = \Psi'(h) \neq \bot'$. For $i > 1$ we have, from the inductive hypothesis, that $(\Psi'(x_{i-1}) \to \Psi'(x_i))$ is an edge in $\mathbb{G}'$. But no edge in $\mathbb{G}'$ goes into $\bot'$, so $\Psi'(x_i) \neq \bot'$. This completes the inductive proof of the above items.

We hence have that $(\Psi'(x_1), \Psi'(x_2), ..., \Psi'(x_m))$ is a directed path in $\mathbb{G}'$. Note that since $x_m \notin \top$ we have $t' = t \notin \Psi(x_m)$ so (by Item 2 above) we have $t' \notin \Psi(x_m) = \Psi'(x_m)$ and hence $\Psi'(x_m) \neq \top'$. So we can extend (in $\mathbb{G}'$) the path $(\Psi'(x_1), \Psi'(x_2), ..., \Psi'(x_m))$ to a path
$(\Psi'(h) = \Psi'(x_1), \Psi'(x_2), ..., \Psi'(x_m), Y_1, Y_2, Y_{m'}, \top')$ for some $m' \geq 0$. We have now constructed a path in $\mathbb{G}'$ from $\Psi'(h)$ to $\top'$ that is at least as long as $|\psi|$ which proves the result. □

Since the label of h was predicted as 0 in $\mathcal{H}$ we have that $|\phi| \leq |\psi|$. Hence, by Lemmas 16 and 17 we have that $|\phi'| \leq |\phi| \leq |\psi| \leq |\psi'|$. So $|\phi'| \leq |\psi'|$ implying that the label of h is still predicted as 0 in $\mathcal{H}'$. ■

Proof of Proposition 4

By Proposition 1, Proposition 4 clearly holds if $z \notin \top$ so assume otherwise. Let K (resp. K') be the cut-size of a label-consistent minimum-cut of $\mathcal{H}$ (resp. $\mathcal{H}'$). Let $B = \{(x \to y) : (x, y) \in \mathcal{H}, x \notin \top, y \in \top\}$. Given a flow in a graph, we define the *size* of the flow to be the number of edge disjoint paths in it.

We now construct the flow $\mathcal{F}$ (in step 2 of the PQ-graph construction algorithm for $\mathbb{G}$) from s to t in $\mathcal{H}$ of size K, and the flow $\mathcal{F}'$ (in step 2 of the PQ-graph construction algorithm for $\mathbb{G}'$) from s' to t' in $\mathcal{H}'$ of size K' as follows (note that we will refer to the objects in this algorithm later):

Algorithm 18.

1. *Convert $\mathcal{H}$ to a graph $\mathcal{H}''$ by adding a set A of K' edges between s and z.*
2. *By running the Ford-Fulkerson algorithm on $\mathcal{H}$ construct a flow, $\mathcal{F}$, of size K in $\mathcal{H}''$ from s to t such that none of the edges in A are contained in the flow. Note that this is the first K steps in an instance of the Ford-Fulkerson algorithm on $\mathcal{H}''$. Note also that $\mathcal{F}$ is a flow of size K in $\mathcal{H}$.*
3. *Continue (from stage 2) the Ford-Fulkerson algorithm on $\mathcal{H}''$ to get a flow, $\mathcal{F}''$, of size K' from s to t in $\mathcal{H}''$.*
4. *Set $\mathcal{F}'''$ equal to $\mathcal{F}''$. Repeat the following until there is no directed path in $\mathcal{F}'''$ from s to z that does not contain an edge in A:*
    (a) *Choose a directed path in $\mathcal{F}'''$ from s to z that does not contain an edge in A. Remove this path from $\mathcal{F}'''$ and add to $\mathcal{F}'''$ an edge in A (directed from s to z). Note that $\mathcal{F}'''$ is still a valid flow of size K'.*
5. *Merge the vertices s and z to get, from $\mathcal{F}'''$, a flow, $\mathcal{F}'$, of size K' from s' to t' in $\mathcal{H}'$.*

We now let $\mathcal{I}$ and $\mathcal{I}'$ be the graphs formed in step 3 of the PQ-graph construction algorithm for $\mathbb{G}$ (given the maximum flow $\mathcal{F}$) and $\mathbb{G}'$ (given the maximum flow $\mathcal{F}'$) respectively.

Lemma 19. *For all $x, y \in \mathcal{H}$, if $(x \to y) \in B$ then we have $(x \to y) \in \mathcal{F}$.*

Proof. Assume we have some $x, y \in \mathcal{H}$, with $(x \to y) \in B$. Suppose, for contradiction, that both $(x \to y)$ and $(y \to x)$ are not in $\mathcal{F}$. Then we have that $(x \to y)$ and $(y \to x)$ are both in $\mathcal{I}$ implying that $\Psi(x) = \Psi(y)$. Then since $y \in \top$ we have $\Psi(x) = \Psi(y) = \top$, which contradicts the fact that $x \notin \top$. We hence have that either $(x \to y)$ or $(y \to x)$ are in $\mathcal{F}$. Suppose, now, for contradiction, that $(y \to x) \in \mathcal{F}$. Then it is a result of the Ford-Fulkerson algorithm that there exists a directed path p, in $\mathcal{F}$, from y to t that goes through x. Let $p = (y = v_1, v_2, ...v_m = t)$. Since, for all

i we have $(v_i \to v_{i+1}) \in \mathcal{F}$, it is the case that $(v_{i+1} \to v_i) \notin \mathcal{F}$ so $(v_i \to v_{i+1}) \in \mathcal{I}$. Hence p is a path in $\mathcal{I}$ from t to y that goes through x. Since $y \in \top$ and hence $\Psi(y) = \top = \Psi(t)$ we have a directed path in $\mathcal{I}$ from y to t. Putting these together we hence have a directed cycle in $\mathcal{I}$ that contains t and x. So t and x are in the same strongly connected component of $\mathcal{I}$ and hence $\Psi(x) = \Psi(t) = \top$. This contradicts the fact that $x \notin \top$. We hence have that $(y \to x) \notin \mathcal{F}$ which, by the above, implies that $(x \to y) \in \mathcal{F}$. □

Definition 20. *For $0 \leq a \leq (K' - K)$, let $\mathcal{F}_a$ be the flow during step a of stage 3 of algorithm 18 (i.e. $\mathcal{F}_0 = \mathcal{F}$, $\mathcal{F}_{K'-K} = \mathcal{F}''$, and for all a, the size of $\mathcal{F}_{a+1}$ is one more than the size of $\mathcal{F}_a$). For every $0 \leq a \leq (K'-K)$ define $C_a := \mathcal{F}_a \setminus (\{(x \to y) : x, y \in \top\} \cup A)$.*

Lemma 21. *For all $0 \leq a \leq (K' - K)$ we have $C_a = C_0$.*

Proof. We prove by induction on a. The inductive hypothesis clearly holds for $a = 0$.

Now suppose the inductive hypothesis holds for some a. We now show that it also holds for $a + 1$:

First note that by Lemma 19 every directed edge in B is contained in C_0. Now, let p be the path (from s to t) that is found in the Ford-Fulkerson algorithm when the flow goes from $\mathcal{F}_a$ to $\mathcal{F}_{a+1}$. Then let $(s, z = x_0, x_1, x_2, ...x_m = t) := p$ (where the edge (s, z) is in A)

Suppose, for contradiction, that for some $i \leq m$, $x_i \notin \top$. Then let $j := \min\{i : x_i \notin \top\}$. Let $k := \min\{i > j : x_i \in \top\}$ which is defined since $x_m = t \in \top$. We have $(x_{k-1} \to x_k) \in B$ so $(x_{k-1} \to x_k) \in C_0$ and hence by the inductive hypothesis $(x_{k-1} \to x_k) \in C_a$ so $(x_{k-1} \to x_k) \in \mathcal{F}_a$ which contradicts the fact that p is the path found by the Ford-Fulkerson algorithm.

Hence, all the edges in p are in $\{(x \to y) : x, y \in \top\} \cup A$ and hence, by considering the Ford-Fulkerson algorithm, we have $C_{a+1} = C_a$. Hence, by the inductive hypothesis we have $C_{a+1} = C_0$. □

Definition 22. *Let $J := \mathcal{F} \setminus \{(x \to y) : x, y \in \top\}$ and $J' := \mathcal{F}' \setminus \{(x \to y) : x, y \in \top\}$.*

Lemma 23. *We have the following results:*

1. *Given $(x \to y) \in J'$, either $(x \to y) \in J$ or $x = s'$.*
2. *Given $(x \to y) \in J$, either $(x \to y) \in J'$ or $x, y \in \bot'$ or $x \in \{s, z\}$ or $y \in \{s, z\}$.*

Proof. Let $J'' := \mathcal{F}'' \setminus (\{(x \to y) : x, y \in \top\} \cup A)$. Note that $J = C_0$ and $J'' = C_{K'-K}$ so by Lemma 21 we have that $J'' = J$.

Suppose we have $(x \to y) \in J'$ with $x \neq s'$. Then we automatically have that $(x \to y) \in \mathcal{F}'''$ at the start of stage 5 of algorithm 18. On each step in stage 4 of algorithm 18 the only edges added to $\mathcal{F}'''$ are those in A so since (because $x, y \in \mathcal{H}'$ so $x, y \neq s$) $(x, y) \notin A$ we have that $(x \to y) \in \mathcal{F}''$. Since $(x \to y) \notin \{(v \to w) : v, w \in \top\} \cup A$ we hence have $(x \to y) \in J''$ which implies, by the above, that $(x \to y) \in J$. This proves Item 1 of the lemma.

Suppose we have some $(x \to y) \in J$ with $(x \to y) \notin J'$, $x, y \notin \{s, z\}$. Since $(x \to y) \in J$ we have (since, by the above, $J = J''$) that $(x \to y) \in J''$. Since $(x \to y) \in J''$ we have $(x \to y) \notin \{(v \to w) : v, w \in \top\}$ and hence, since $(x \to y) \notin J'$, $(x \to y) \notin \mathcal{F}'$. Since $(x \to y) \in J''$ we must have $(x \to y) \in \mathcal{F}''$. So $(x \to y) \in \mathcal{F}''$ and $(x \to y) \notin \mathcal{F}'$ and hence (since $x, y \notin \{s, z\}$) it must be the case that $(x \to y)$ was removed (from $\mathcal{F}'''$) during stage 4 of algorithm 18. Let $(s = v_1, v_2, ..., v_m = z)$ be the directed path in $\mathcal{F}''$ that contains $(x \to y)$ and was removed during stage 4 of algorithm 18. Since this path is removed and s and z are merged into s' in forming $\mathcal{F}'$ we have that no edge in the cycle (in $\mathcal{H}'$) $(s', v_2, v_3, ..., v_{m-1}, s')$ is in $\mathcal{F}'$. Hence we have that $(s', v_{m-1}, v_{m-1}, ..., v_2, s')$ is a directed cycle in $\mathcal{I}'$ so all vertices in this cycle belong to the same strongly connected component of $\mathcal{I}'$. This implies that for all i we have $\Psi'(v_i) = \Psi'(s') = \bot'$. Since $x, y \notin \{s, z\}$ we have that $x = v_i$ and $y = v_{i+1}$ for some $1 < i < m - 1$. Hence we have that $x, y \in \bot'$ which completes the proof of item 2 of the lemma. □

Lemma 24. *Given some $v \in \top$ with $v \neq z$ we either have $\Psi'(v) = \perp'$ or $\Psi'(v) \subseteq \top$.*

Proof. Suppose we assume the converse: that there exists some $v \in \top \setminus \{z\}$ with $\Psi'(v) \neq \perp'$ and $\Psi'(v) \nsubseteq \top$. Then choose some $x \in \mathcal{H}' \setminus \top$ such that $x \in \Psi'(v)$. Since x and v are in the same strongly connected component in $\mathcal{I}'$ there exists (in $\mathcal{I}'$) a directed path $p := \{v = x_0, x_1, x_2, ..., x_m = x\}$ such that each x_i is in $\Psi'(v)$. Let $i = \min\{j : x_j \notin \top\}$ which exists since $x_m \notin \top$. Note that since $x_0 = v \in \top$ we have $i > 0$ so x_{i-1} exists. Since $\Psi'(v) \neq \perp'$ we know $x_i \neq s'$ (else $s' \in \Psi'(v)$ and hence $\Psi'(v)$ would equal $\perp'$). We hence have that $(x_i \to x_{i-1}) \in B$ so we know, from Lemma 19 that $(x_i \to x_{i-1})$ is in J.

Since $(x_{i-1} \to x_i)$ is a directed edge in p, and hence in $\mathcal{I}'$, we know that $(x_i \to x_{i-1}) \notin \mathcal{F}'$ so we have that $(x_i \to x_{i-1}) \notin J'$ which implies by Lemma 23 Item 2 (since, by the above, $(x_i \to x_{i-1})$ is in J and (since $x_i, x_{i-1} \in \mathcal{H}'$) $x_i, x_{i-1} \notin \{s, z\}$) that $x_i \in \perp'$. Since $x_i \in \Psi'(v)$ this implies that $\Psi'(v) = \perp'$ which is a contradiction. □

Lemma 25. *Given some $X, Y \in \mathbb{G}'$ with $X \subseteq \top$, if there is an edge in $\mathbb{G}'$ from X to Y, then we have $Y \subseteq \top$.*

Proof. Suppose the converse: that there exists some $X, Y \in \mathbb{G}'$ with $X \subseteq \top, Y \nsubseteq \top$ and an edge in $\mathbb{G}'$ from X to Y.

Note first that since $X \subseteq \top$ we have $s' \notin X$ and hence $X \neq \perp'$. Since there is an edge in $\mathbb{G}'$ from X to Y and no edge goes into $\perp'$ we have $Y \neq \perp'$.

Since there is an edge in $\mathbb{G}'$ from X to Y choose $x \in X$ and $y \in Y$ such that there is an edge in $\mathcal{F}'$ from x to y

Since $Y \nsubseteq \top$ and $Y \neq \perp'$ and (since $y \in \mathcal{H}'$) $y \neq z$ we must have, by Lemma 24, that $y \notin \top$. Hence we have that $(x \to y) \in J'$. Since $Y \neq \perp'$ we also have that $y \neq s'$. We hence have that $(y \to x) \in B$ so, by Lemma 19, we have that $(y \to x) \in J$. By Lemma 23 Item 1 we hence have a contradiction (since $(y \to x) \in J$ implies $(x \to y) \notin J$ and we have $(x \to y) \in J'$ and $x \neq s'$). □

Lemma 26. *Given a vertex v such that $(v \to s) \in \mathcal{I}$ and $v \notin \top$ we have that $(v \to s') \in \mathcal{I}'$.*

Proof. Suppose the converse: that there exists a vertex v such that $(v \to s) \in \mathcal{I}$, $v \notin \top$ and $(v \to s') \notin \mathcal{I}'$. Since $(v \to s') \notin \mathcal{I}'$ we have $(s' \to v) \in \mathcal{F}'$. Hence, by considering Stage 5 of Algorithm 18 we must have that either $(s \to v) \in \mathcal{F}'''$ or $(z \to v) \in \mathcal{F}'''$. Since $v \notin \top$ we have $v \neq z$ so (since $v \neq s$ (as there is an edge in $\mathcal{I}$ from s to v)) we have $(s, v), (z, v) \notin A$. Hence, since during Stage 4 of Algorithm 18 the only edges added to the flow are those in A, we must have that either $(s \to v) \in \mathcal{F}''$ or $(z \to v) \in \mathcal{F}''$.

Assume, for contradiction, that $(z \to v) \in \mathcal{F}''$. Then $(z, v) \in E(\mathcal{H}'')$ so since, by the above, $(z, v) \notin A$ (and, since $z, v \in \mathcal{H}$ we have $z, v \neq s'$) we have that $(z, v) \in E(\mathcal{H})$. Since $z \in \top$ and $v \notin \top$ we have that $(v \to z) \in B$ and hence, by Lemma 19, we have that $(v \to z) \in C_0$ so, by Lemma 21, $(v \to z) \in C_{K'-K}$ which implies that $(v \to z) \in \mathcal{F}''$ and hence that $(z \to v) \notin \mathcal{F}''$ which is a contradiction.

We hence have that $(s \to v) \in \mathcal{F}''$ which implies, since $v \notin \top$, that $(s \to v) \in C_{K'-K}$ so by Lemma 21 we have $(s \to v) \in C_0$ which implies that $(s \to v) \in \mathcal{F}$. This implies that $(v \to s) \notin \mathcal{I}$ which is a contradiction. This completes the proof. □

Lemma 27. *Given some $v \in \mathcal{H} \setminus (\top \cup \{s\})$ with $\Psi'(v) \neq \perp'$ or $\Psi(v) \neq \perp$, we have $\Psi(v) \subseteq \Psi'(v)$.*

Proof. We shall prove that $\Psi(v)$ is strongly connected in $\mathcal{I}'$ which directly implies the result.

We first show that s and z are not contained in $\Psi(v)$. Since $v \notin \top$ we have $\Psi(v) \neq \top = \Psi(z)$ which implies that $z \notin \Psi(v)$. Suppose now, for contradiction, that $s \in \Psi(v)$. Since $v \neq s$ we then have a directed paths $(s, x_1, x_2, ..., x_m := v)$ and $(v = y_1, y_2, ..., y_{m'}, s)$ in $\mathcal{I}$ such that, for all i, $x_i, y_i \in \Psi(v)$. Note that since $z \notin \Psi(v)$ none of the x_i or y_i are equal to z. Since (s, x_1) is an edge in $\mathcal{H}$ we must then have that (s', x_1) is an edge in $\mathcal{H}'$ and since no edge of $\mathcal{F}'$ goes into s' we must have that $(x_1 \to s') \notin \mathcal{F}'$ implying that $(s' \to x_1) \in \mathcal{I}'$. Since $(y_{m'} \to s) \in \mathcal{I}$ and (since $\Psi(y_{m'}) = \Psi(v) \neq \top$) $y_{m'} \notin \top$ we have, by Lemma 26 that $(y_{m'} \to s') \in \mathcal{I}'$. Since, for all i, $(x_i \to x_{i+1}) \in \mathcal{I}$ we have that $(x_{i+1} \to x_i) \notin \mathcal{F}$ hence $(x_{i+1} \to x_i) \notin J$. If it was true that $(x_{i+1} \to x_i) \in \mathcal{F}'$ then since $x_i \notin \top$ (as $v \notin \top$ implies that $\Psi(x_i) = \Psi(v) \neq \top$) we would have that $(x_{i+1} \to x_i) \in J'$ so since $x_{i+1}, x_i \neq s'$ (as both are in $\mathcal{H}$) we would have, by Lemma 23 Item 1, that $(x_{i+1} \to x_i) \in J$ which is a contradiction. Hence we have that $(x_{i+1} \to x_i) \notin \mathcal{F}'$ so $(x_i \to x_{i+1}) \in \mathcal{I}'$. Similarly we have $(y_i \to y_{i+1}) \in \mathcal{I}'$ for all i. We hence have that $(s', x_1, x_2, ...x_m = v)$ and $(v = y_1, y_2, ..., y_{m'}, s')$ are directed paths in $\mathcal{I}'$ so we have that v and s' are in the same strongly connected component of $\mathcal{I}'$. So $\Psi'(v) = \Psi(s') = \bot'$ and hence $v \in \bot'$. Since $s \in \Psi(v)$ we also have that $\Psi(v) = \bot$ which is a contradiction.

We hence have that $s, z \notin \Psi(v)$ so $\Psi(v) \subseteq V(\mathcal{H}')$. Suppose that we have vertices $x, y \in \Psi(v)$. We now show that there exists a directed path in $\mathcal{I}'$ from x to y which proves that $\Psi(v)$ is strongly connected in $\mathcal{I}'$:

Since $\Psi(v)$ is strongly connected in $\mathcal{I}$ there exists a directed path p from x to y in $\mathcal{I}$ such that every vertex in p is in $\Psi(v)$. Since $\Psi(v) \neq \top$, p is a path in $V(\mathcal{H}) \setminus \top$. Hence, if some directed edge $(x' \to y')$ is in p and not in $\mathcal{I}'$ then (since, by definition of $\mathcal{I}'$, $(y' \to x') \in \mathcal{F}'$) we have (since $x' \in \Psi(v) \neq \top$ and hence $x' \notin \top$) that $(y' \to x') \in J'$ which implies, by Lemma 23 Item 1 (since $y' \in \mathcal{H}$ so $y' \neq s'$) that $(y' \to x') \in J$, and hence $(y' \to x') \in \mathcal{F}$, which implies that $(x' \to y')$ is not in $\mathcal{I}$ which is a contradiction. Hence, p is a directed path in $\mathcal{I}'$. This completes the proof that $\Psi(v)$ is strongly connected in $\mathcal{I}'$. The result follows. □

Lemma 28. *Given some $v \in \mathcal{H} \setminus (\top \cup \{s\})$ we either have $\Psi'(v) = \bot'$ or $\Psi'(v) = \Psi(v)$.*

Proof. Suppose the converse: that there exists some $v \in \mathcal{H} \setminus (\top \cup \{s\})$ with $\Psi'(v) \neq \bot'$ and $\Psi'(v) \neq \Psi(v)$. Since $\Psi'(v) \neq \bot'$, we have, by Lemma 27, that $\Psi(v) \subseteq \Psi'(v)$. Since $\Psi(v) \neq \Psi'(v)$ we hence can choose some $x \in \Psi'(v) \setminus \Psi(v)$. Since $x \in \Psi'(v)$ we have a directed path (in $\mathcal{I}'$), p (resp. q) in $\Psi'(v)$ from v to x (resp. x to v).

Suppose, for contradiction, that there exists some $v' \in \Psi'(v)$ with $v' \in \top$. By the above we have $\Psi'(v') = \Psi'(v) \neq \bot'$. Note also that since $v' \in \mathcal{H}'$ we have $v' \neq z$. Hence, by Lemma 24 we must have that $\Psi'(v') \subset \top$ which implies (since $\Psi'(v) = \Psi'(v')$) that $\Psi'(v) \subset \top$ which contradicts the fact that $v \in \Psi'(v)$. Hence we have that no element of $\top$ is contained in $\Psi'(v)$.

Since $\Psi'(v) \neq \bot'$ no element of $\Psi'(v)$ is equal to s'. We hence have that the path p contains only vertices in $\mathcal{H}' \setminus (\top \cup \{s'\})$. Hence, if some directed edge $(x' \to y')$ is in p and not in $\mathcal{I}$ then (since, by definition of $\mathcal{I}$, $(y' \to x') \in \mathcal{F}$) we have $(y' \to x') \in J$ which implies, by Lemma 23 Item 2 that $(y' \to x') \in J'$ (because else, by Lemma 23 Item 2, $y', x' \in \bot'$ (since $y', x' \in \mathcal{H}'$ and hence $y', x' \notin \{s, z\}$) which is a contradiction since $y', x' \in \Psi'(v) \neq \bot'$) and hence $(y' \to x') \in \mathcal{F}'$, which implies that $(x' \to y')$ is not in $\mathcal{I}'$ which is a contradiction. Hence, p is a directed path in $\mathcal{I}$. Similarly q is a directed path in $\mathcal{I}$. This implies that v and x are in the same strongly connected component of $\mathcal{I}$. Hence $\Psi(v) = \Psi(x)$ so $x \in \Psi(v)$ which is a contradiction. □

Lemma 29. *Given some $v, w \in \mathcal{H} \setminus (\top \cup \{s\})$ in which $\Psi'(v) \neq \bot'$ then the existence of an edge in $\mathbb{G}'$ from $\Psi'(v)$ to $\Psi'(w)$ implies the existence of an edge in $\mathbb{G}$ from $\Psi(v)$ to $\Psi(w)$.*

Proof. Note first that since there is an edge in $\mathbb{G}'$ going into $\Psi'(w)$ we must have $\Psi'(w) \neq \bot'$. By Lemma 28 we then have that $\Psi'(v) = \Psi(v)$ and $\Psi'(w) = \Psi(w)$. Since there is an edge in $\mathbb{G}'$ from $\Psi'(v)$ to $\Psi'(w)$ there exist vertices $x \in \Psi'(v)$ and $y \in \Psi'(w)$ such that $(x \to y) \in \mathcal{F}'$. Since $\Psi'(v) \neq \bot'$ we have $x \neq s'$. Since $\Psi'(x) = \Psi'(v) \neq \bot'$ and (as $v \in \Psi'(v) = \Psi'(x)$ and $v \notin \top$) $\Psi'(x) \not\subseteq \top$ we have, by Lemma 24, that $x \notin \top$. We hence have that $(x \to y) \in J'$ and that $x \neq s'$ so, by Lemma 23 Item 1, we have $(x \to y) \in J$ and hence there is an edge from x to y in $\mathcal{F}$. Since $\Psi'(v) = \Psi(v)$ and $\Psi'(w) = \Psi(w)$, we hence obtain the result (since there is an edge in $\mathcal{F}$ from a vertex in $\Psi(v)$ to a vertex in $\Psi(w)$). □

Lemma 30. *Given some $v, w \in \mathcal{H} \setminus (\top \cup \{s\})$ with $\Psi'(v) \neq \perp'$ then the existence of an edge in $\mathbb{G}$ from $\Psi(v)$ to $\Psi(w)$ implies the existence of an edge in $\mathbb{G}'$ from $\Psi'(v)$ to $\Psi'(w)$.*

Proof. By Lemma 28 we have that $\Psi'(v) = \Psi(v)$. Since there is an edge in $\mathbb{G}$ from $\Psi(v)$ to $\Psi(w)$ there exist vertices $x \in \Psi(v)$ and $y \in \Psi(w)$ such that $(x \to y) \in \mathcal{F}$. Since $\Psi(v) = \Psi'(v)$ and $s \notin \Psi'(v)$ we have $x \neq s$. Since there is no edge in $\mathcal{F}$ that goes into s we must have $y \neq s$. If x was in $\top$ then we would have $\Psi(v) = \Psi(x) = \top$ which contradicts the fact that $v \notin \top$. Similarly $y \notin \top$. We hence have that $x, y \neq z$. Hence, $(x \to y) \in J$ and $x, y \notin \{s, z\}$ so by Lemma 23 Item 2 we either have that $(x \to y) \in J'$ or that $x, y \in \perp'$. But if $x \in \perp'$, then since $x \in \Psi'(v)$ (since, by the above, $\Psi'(v) = \Psi(v)$) we have that $\Psi'(v) = \perp'$ which is a contradiction. So $(x \to y) \in J'$ and hence $(x \to y) \in \mathcal{F}'$. Since there is an edge in $\mathbb{G}$ that goes into $\Psi(w)$ we have that $\Psi(w) \neq \perp$ so by Lemma 27 we have that $\Psi(w) \subseteq \Psi'(w)$ so $y \in \Psi'(w)$.
Suppose, for contradiction, that $\Psi'(v) = \Psi'(w)$. We know that $\Psi'(v) \neq \perp'$ and hence that $\Psi'(w) \neq \perp'$. By Lemma 28 we hence have that $\Psi(w) = \Psi'(w) = \Psi'(v) = \Psi(v)$ which is a contradiction. We hence have that $\Psi'(v) \neq \Psi'(w)$.
We hence have (since $x \in \Psi(v) = \Psi'(v)$ and $y \in \Psi'(w)$ and $(x \to y) \in \mathcal{F}'$) that there is an edge in $\mathbb{G}'$ from $\Psi'(v)$ to $\Psi'(w)$. □

We have now proved Proposition 4: Item (a) is Lemma 25, Item (b) is Lemma 28 and Item (c) comes directly from lemmas 29 and 30. ■

2.10 Proof of Optimality for Trees

In this section we prove that `0-Ising` and `longest-path` are optimal graph label prediction algorithms on trees in the sense of [16, Theorem 1]. We note that the `0-Ising` strategy when restricted to a tree was already proved optimal in [16] where it was called "`Halving`." Our proof of optimality of `longest-path` uses much of "proof technology" from [16] so for the convenience of the reader in the next subsection we recall their notation and definitions.

2.10.1 Ingredients from [16, Section 2]

Given a set L of edge-disjoint paths contained in a tree T , we say that $l \in L$ is a grafted path if one of the two terminal vertices of l is also an internal vertex of another path $l' \in L$. This shared vertex is called the graft vertex of l. We say that L is a connected blanket if:

1. The union of all paths in L forms a (connected) tree.
2. Every vertex in this (connected) tree can be an internal vertex of at most one such path.
3. Every grafted path in L shares with the remaining paths in L no vertices but the graft.

Finally, L is a blanket if it is either a connected blanket or it has been obtained by a connected blanket after removing one or more of its paths. The size of a blanket L is the number of its paths $|L|$. Note that a blanket need not include all edges of the original tree T . Also, observe that for any size $K < n$, a size-K blanket over a tree T always exists: take L to be any set of K distinct edges in T; then no paths of L have internal vertices and the blanket property trivially holds. On the other hand, a given tree T clearly admits many size-K blankets. Let $\mathcal{L}(T, K)$ be the set of all size-K blankets over T , and define the function LB ("lower bound") as follows:

$$\mathrm{LB}(T, K) := \max_{L \in \mathcal{L}(T,K)} \sum_{l \in L} \lfloor \log_2(|l|) \rfloor \tag{2.5}$$

where $|l|$ is the number of vertices in l. We state the lower bound for any graph label prediction algorithm proved in [16].

Theorem 31. *[16, Theorem 1], Given a tree T and a number $K \in \mathbb{N}$, then for any online prediction algorithm $\mathcal{A}$ there exists a $\{0,1\}$ labelling, ν, of T with cutsize at most K, on which algorithm $\mathcal{A}$ makes at least $\mathrm{LB}(T,K)$ mistakes.*

2.10.2 Proof of Optimality

Let X be the set of cut edges of T and let $K := |X|$. A subtree Q is called a 2-tree (resp. 1-tree) if it is a subtree of T with an inner boundary of two vertices (resp. one vertex) and all vertices in its inner boundary are leaves of it. Given a 1-tree or 2-tree Q we let Q° be equal to Q minus its inner boundary. We have the following lemma.

Lemma 32. *We can find a set $\mathbb{S}$, of 1-trees, and a set $\mathbb{T}$, of 1-trees and 2-trees , which satisfy the following.*

1. *Any 1-tree in $\mathbb{T}$ has a single edge and that edge is cut.*
2. *The trees in $\mathbb{S} \cup \mathbb{T}$ are edge-disjoint.*
3. *The union of the edges of the trees in $\mathbb{S} \cup \mathbb{T}$ is equal to the edge set of T.*
4. *$|\mathbb{T}| \leq 3K$*
5. *For every edge (v,w) in X we have a tree in $\mathbb{T}$ which has (v,w) as a single edge. Note that this implies that $|\mathbb{T}| \geq K$*
6. *Given a tree $Q \in \mathbb{S}$, where $\partial_0(Q) = \{v\}$ for some $v \in T$, then we have $v \in \partial_0(R)$ for some $R \in \mathbb{T}$.*
7. *For any tree $Q \in \mathbb{T} \cup \mathbb{S}$ we have that Q° is identically labelled (i.e. there is no edge of Q° that is in X.)*
8. *For any tree $Q \in \mathbb{S}$ we have that Q is identically labelled (i.e. there is no edge of Q that is in X.)*
9. *For any trees $R, Q \in \mathbb{S} \cup \mathbb{T}$ we have that no vertex in the inner boundary of Q is in R°.*

Proof. Note first that in the following proof we may create subtrees containing a single vertex - such trees can be discarded. We prove by induction on K. For the base case $K = 1$ let $\{(v,w)\} := X$. Let Q be the tree containing the single edge (v,w) (Note that Q° has at most one vertex and is hence identically labelled). Let $\mathbb{A}$ be equal to the set of 1-trees with inner boundary $\{v\}$ or $\{w\}$ such that the trees in $\mathbb{A} \cup \{Q\}$ are edge-disjoint and the union of edges of the trees in $\mathbb{A} \cup \{Q\}$ is equal to the edge set of T. We then have $\mathbb{S} := \mathbb{A}$ and $\mathbb{T} := \{Q\}$. It is easy to check that all the statements of the lemma hold in this case.

Suppose that the inductive hypothesis holds for $K = \kappa$. We now consider the case that $K = \kappa + 1$: In this case choose an edge $e \in X$ and define $X' = X \setminus \{e\}$. Since $|X'| = \kappa$ we can find, by the inductive hypothesis, sets $\mathbb{S}'$ and $\mathbb{T}'$ to be equal to $\mathbb{S}$ and $\mathbb{T}$ (respectively) in the lemma if the set of cut edges was equal to X' (instead of X). Let S be the (unique) tree in $\mathbb{S}' \cup \mathbb{T}'$ that contains the edge e. Note that if S was a 1-tree in $\mathbb{T}'$ then by Lemma 32 Item 1 we would have that S had a single edge and that edge would be in X' and hence not equal to e which would be a contradiction. Hence, if S is in $\mathbb{T}'$ then it has two inner boundary vertices. We hence have three cases.

1. $S \in \mathbb{S}'$: In this case let $(v,w) := e$ where v is closer than w to the inner-boundary vertex of S. Let S' be the maximal subtree of S with leaf v that does not contain w. Let Q be the tree containing the single edge (v,w) (Note that Q° has at most one vertex and is hence identically labelled). Let $\mathbb{A}$ be equal to the set of 1-trees with inner boundary $\{v\}$ or $\{w\}$ such that the trees in $\mathbb{A} \cup \{S', Q\}$ are edge-disjoint and the union of edges of the trees in $\mathbb{A} \cup \{S', Q\}$ is equal to the edge set of S. Let $\mathbb{T} := \mathbb{T}' \cup \{S', Q\}$ (noting that by the inductive hypothesis we have $|\mathbb{T}| = 2 + |\mathbb{T}'| \leq 2 + 3\kappa < 3(\kappa+1) = 3K$) and $\mathbb{S} := (\mathbb{S}' \setminus \{S\}) \cup \mathbb{A}$. By the inductive hypothesis (i.e. the conditions on $\mathbb{S}'$ and $\mathbb{T}'$) it is easy to check that all the statements of the lemma hold in this case.

2. $S \in \mathbb{T}'$ and e is on the path between the inner boundary vertices of S: In this case let $(v, w) := e$. Let S' (resp. S'') be the maximal subtree of S with leaf v (resp. w) that does not contain w (resp. v). Let Q be the tree containing the single edge (v, w) (Note that Q° has at most one vertex and is hence identically labelled). Let $\mathbb{A}$ be equal to the set of 1-trees with inner boundary $\{v\}$ or $\{w\}$ such that the trees in $\mathbb{A} \cup \{S', S'', Q\}$ are edge-disjoint and the union of edges of the trees in $\mathbb{A} \cup \{S', S'', Q\}$ is equal to the edge set of S. Let $\mathbb{T} := (\mathbb{T}' \setminus \{S\}) \cup \{S'', S', Q\}$ (noting that by the inductive hypothesis we have $|\mathbb{T}| := 2 + |\mathbb{T}'| \leq 2 + 3\kappa < 3(\kappa + 1) = 3K$) and $\mathbb{S} := \mathbb{S}' \cup \mathbb{A}$. By the inductive hypothesis (i.e. the conditions on $\mathbb{S}'$ and $\mathbb{T}'$) it is easy to check that all the statements of the lemma hold in this case.
3. $S \in \mathbb{T}'$ and e is not on the path between the inner boundary vertices of S: In this case let $\{x, y\}$ be the inner boundary of S and let $(v, w) := e$. Let z be the vertex where the path from v to y first meets the path from x to y. Without loss of generality let v be closer to z than w is. Let S' (resp. S'') be the maximal subtree of S with x and z (resp. y and z) as leaves. Let R be the maximal subtree of S that has leaves z and v. Let Q be the tree containing the single edge (v, w) (Note that Q° has at most one vertex and is hence identically labelled). Let $\mathbb{A}$ be equal to the set of 1-trees with inner boundary $\{v\}$ or $\{w\}$ or $\{z\}$ such that the trees in $\mathbb{A} \cup \{S', S'', Q, R\}$ are edge-disjoint and the union of edges of the trees in $\mathbb{A} \cup \{S', S'', Q, R\}$ is equal to the edge set of S. Let $\mathbb{T} := (\mathbb{T}' \setminus \{S\}) \cup \{S'', S', Q, R\}$ (noting that by the inductive hypothesis we have $|\mathbb{T}| := 3 + |\mathbb{T}'| \leq 3 + 3\kappa = 3(\kappa + 1) = 3K$) and $\mathbb{S} := \mathbb{S}' \cup \mathbb{A}$. By the inductive hypothesis (i.e. the conditions on $\mathbb{S}'$ and $\mathbb{T}'$) it is easy to check that all the statements of the lemma hold in this case.

□

Let $\mathbb{S}$ and $\mathbb{T}$ be as in the above lemma. Let $J := \bigcup\{\partial_0(S) : S \in \mathbb{S} \cup \mathbb{T}\}$. Given $v \in J$ let $\mathbb{S}(v)$ be the set of trees in $\mathbb{S}$ that have an inner boundary of $\{v\}$ and let $\mathbb{T}(v)$ be the set of trees in $\mathbb{T}$ that have v in their inner boundary.

Lemma 33. *We have $|J| \leq 6K$*

Proof. By Lemma 32 Item 6 we have that any vertex in J is in the inner boundary of a tree in $\mathbb{T}$ and there are (as for all $Q \in \mathbb{T}$ we have that Q has an inner boundary of cardinality of at most two) at most $2|\mathbb{T}|$ such vertices. The result then follows by Lemma 32 Item 4 □

Lemma 34. *Suppose we have some $v \in J$. Then after we have received at least one example in R° for α of the trees $R \in \mathbb{S}(v)$, where $\alpha := |\mathbb{T}(v)| + 1$, we will no longer make any mistakes on any of the trees in $\mathbb{S}(v)$.*

Proof. Let y be the true labelling of T and let $y(v)$ denote the label of vertex v. Without loss of generality assume that $y(v) = 1$. We first note that by Lemma 32 Item 8 we have, for all $R \in \mathbb{S}(v)$, that every vertex a in R satisfies $y(a) = 1$ (since $v \in R$).

Suppose we have received at least one label in R° for α of the trees $R \in \mathbb{S}(v)$, where $\alpha := |\mathbb{T}(v)| + 1$. Then let $\mathbf{u}$ be a consistent (with the observed examples) labelling of T that minimises the cut. Given some $Q \in \mathbb{T}(v)$, define z_Q to be the vertex in Q that is adjacent to v. Let $C := \{(v, z_Q) : Q \in \mathbb{T}(v)\} \cup \bigcup\{E(R) : R \in \mathbb{S}(v)\}$. Then the restriction of $\mathbf{u}$ to the vertices in the edges in C minimises the cutsize in C given the observed examples and the labels $\{u_{z_Q} : Q \in \mathbb{T}(v)\}$.

By labelling all the vertices in $\bigcup\{V(R) : R \in \mathbb{S}(v)\}$ 1 we get a cut in C of size no greater than $|\mathbb{T}(v)|$. But if $u_v = 0$ we get a cut in each of the trees in $\mathbb{S}(v)$ for which we have observed a label in, giving us a cut in C of size at least $|\mathbb{T}(v)| + 1$. To minimise the cutsize in C we must hence have that $u_v = 1$.

Hence, given a tree $R \in \mathbb{S}(v)$, the restriction of $\mathbf{u}$ to R minimises the cutsize in R given the observed labels and conditioned on $u_v = 1$. This cutsize is 0 if and only if all vertices in R are labelled 1. Hence, $u_w = 1$ for all vertices $w \in V(R)$, so no mistake will be made in R. □

Let $\mathbb{U}$ be the set of trees $R \in \mathbb{S}$ in which a mistake is made in R°.

Lemma 35. *We have* $|\mathbb{U}| \leq 12K$

Proof. By Lemma 34 we have that:

$$\mathbb{U} \leq \sum_{v \in J} (|\mathbb{T}(v)| + 1) = |J| + \sum_{v \in J} |\mathbb{T}(v)| = |J| + 2|\mathbb{T}| \tag{2.6}$$

where the last equality comes from the fact that for each tree S in $\mathbb{T}$ we have that S appears in at most two of the sets in $\{\mathbb{T}(v) : v \in J\}$ as it has at most two vertices in its inner boundary. By Lemma 33 and Lemma 32 Item 4 the result follows. □

Lemma 36. *Given a tree $S \in \mathbb{T} \cup \mathbb{U}$, the number of mistakes made in S° by the* `longest-path` *and* `0-Ising` *strategies is bounded above by* $3 \log_2(D(S) + 1)) + 5$*.*

Proof. Let y be the true labelling of T. Define a labelling, $\hat{y}$, of S to be as follows. For $v \in \partial_0(S)$ we have $\hat{y}(v) := 0$. For $v \in S \setminus \partial_0(S)$ we have $\hat{y}(v) := 1$. Define the *full-algorithm* to be the algorithm run on T with labelling y. Define the *sub-algorithm* to be the algorithm run on S with labelling $\hat{y}$.

Since, by Lemma 32 Item 7, S° is identically labelled, by Theorem 4 the number of mistakes made by the full-algorithm in S° is no greater than the maximum number of mistakes, M, that the sub-algorithm makes in $S \setminus \partial_0(S)$ after it has received the labels on $\partial_0(S)$. We hence consider the sub-algorithm.

Since the cutsize of $\hat{y}$ is no greater than 2, $M \leq M_1 + M_2 + 3$ where M_i is the number of mistakes made by the sub-algorithm in the PQ game of cutsize i. Let $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) be the PQ graph at the start of the PQ game at cutsize 1 (resp. cutsize 2 (in the case that S is a 2-tree)). By Theorem 5 there exists 1 (resp. 2) edge disjoint paths p (resp. p, q) in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) such that $M_1 \leq 1 + \log_2(|p|)$ (resp. $M_2 \leq 1 + \log_2(|p|) + \log_2(|q|)$). But $|p| \leq D(S) + 1$ (resp. $|p|, |q| \leq D(S) + 1$). We hence have that $M_1 \leq 1 + \log_2(D(S) + 1)$ (resp. $M_2 \leq 1 + 2\log_2(D(S) + 1)$)

We hence have that $M \leq 3\log_2(D(S)) + 1) + 5$ which, by the above, is an upper bound on the number of mistakes made (by the full algorithm) in S°. □

Lemma 37. *The number of mistakes, $\mathcal{M}$, made by the* `longest-path` *and* `0-Ising` *strategies are bounded above by:*

$$\mathcal{M} \leq \sum_{S \in \mathbb{T} \cup \mathbb{U}} 14 \log_2(D(S) + 1)) \tag{2.7}$$

Proof. Given a tree $S \in \mathbb{S} \cup \mathbb{T}$ let $M(S)$ be the number of mistakes made in S°. By Lemma 32 Item 3 and the definition of J every vertex in T is either in J or in S° for some $S \in \mathbb{S} \cup \mathbb{T}$ and hence the number of mistakes made in T is upper bounded by:

$$\mathcal{M} \leq |J| + \sum_{S \in \mathbb{T} \cup \mathbb{S}} M(S) \tag{2.8}$$

$$= |J| + \sum_{S \in \mathbb{T} \cup \mathbb{U}} M(S) \tag{2.9}$$

$$\leq |J| + \sum_{S \in \mathbb{T} \cup \mathbb{U}} (3 \log_2(D(S) + 1)) + 5) \tag{2.10}$$

$$\leq 6K + \sum_{S \in \mathbb{T} \cup \mathbb{U}} (3 \log_2(D(S) + 1)) + 5) \tag{2.11}$$

$$\leq \sum_{S \in \mathbb{T} \cup \mathbb{U}} (3 \log_2(D(S) + 1)) + 11) \tag{2.12}$$

$$\leq \sum_{S \in \mathbb{T} \cup \mathbb{U}} 14 \log_2(D(S) + 1)) \tag{2.13}$$

where Equation 2.9 comes from the definition of $\mathbb{U}$ (i.e. for all trees $S \in \mathbb{S} \setminus \mathbb{U}$ we have $M(S) = 0$), Equation 2.10 comes from Lemma 36, Equation 2.11 comes from Lemma 33 and Equation 2.12 comes from Lemma 32 Item 5. □

We now define the following paths.

Definition 38. *For any 1-tree $S \in \mathbb{U} \cup \mathbb{T}$ define $\rho(S)$ to be a path in S containing its inner boundary vertex that has maximum length. For any 2-tree $S \in \mathbb{T}$ let $\lambda(S)$ be the path between the inner boundary vertices of S. Let $\lambda'(S)$ be a path in S that has a leaf in $\lambda(S)$ and is edge disjoint from $\lambda(S)$ that has maximum length. Let $\rho(S) = \operatorname{argmax}_{p \in \{\lambda(S), \lambda'(S)\}} |p|$.*

For the following we define the constant $\alpha := \log_2(4/3)/\log_2(2)$.

Lemma 39. *For any tree $S \in \mathbb{U} \cup \mathbb{T}$ we have $\log_2(|\rho(S)|) \geq \log_2(\frac{1}{3}D(S) + 1)$ which is bounded below by $\alpha \log_2(D(S) + 1)$.*

Proof. Direct from the definition of $\rho(S)$. □

Lemma 40. *$\{\rho(S) : S \in \mathbb{U} \cup \mathbb{T}\}$ is a blanket of cardinality at most $15K$.*

Proof. We have, by Lemma 32 items 2, 3, 6 and 9, that $\{\rho(S) : S \in \mathbb{U}\} \cup \{\lambda(S) : S \in \mathbb{T}\} \cup \{\lambda'(S) : S \in \mathbb{T}\}$ is a connected blanket. So since $\{\rho(S) : S \in \mathbb{U} \cup \mathbb{T}\}$ is a subset of $\{\rho(S) : S \in \mathbb{U}\} \cup \{\lambda(S) : S \in \mathbb{T}\} \cup \{\lambda'(S) : S \in \mathbb{T}\}$ it is a blanket. The cardinality of $\{\rho(S) : S \in \mathbb{U} \cup \mathbb{T}\}$ follows from Lemma 32 Item 4 and Lemma 35. □

We now define the following blanket.

Definition 41. *For $i \in \mathbb{N}_K$ inductively define:*

$$S_i = \operatorname{argmax}_{S \in (\mathbb{U} \cup \mathbb{T}) \setminus \{S_j : j < i\}} (|\rho(S)|) \tag{2.14}$$

and define $\mathcal{B} := \{\rho(S_i) : i \in \mathbb{N}_K\}$.

Lemma 42. *$\mathcal{B}$ is a blanket of size K which satisfies:*

$$\sum_{p \in \mathcal{B}} \log_2(|p|) \geq \frac{\alpha}{15} \sum_{S \in \mathbb{U} \cup \mathbb{T}} \log_2(D(S) + 1)\,. \tag{2.15}$$

Proof. By Lemma 40 $\mathcal{B}$ is a subset of a blanket and is hence a blanket. Since $\mathcal{B}$ has K elements it has, by Lemma 40 at least $\frac{1}{15}$th of the elements of $\{\rho(S) : S \in \mathbb{U} \cup \mathbb{T}\}$. So, since in forming $\mathcal{B}$ we picked the paths of greatest cardinality, we must have that $\sum_{p \in \mathcal{B}} \log_2(|p|) \geq \frac{1}{15} \sum_{p \in \{\rho(S) : S \in \mathbb{U} \cup \mathbb{T}\}} \log_2(|p|) = \frac{1}{15} \sum_{S \in \mathbb{U} \cup \mathbb{T}} |\rho(S)|$ which, by Lemma 39, is at least $\frac{\alpha}{15} \sum_{S \in \mathbb{U} \cup \mathbb{T}} \log_2(D(S) + 1)$. □

Theorem 43. *The number of mistakes $\mathcal{M}$ incurred by the* `longest-path` *and* `0-Ising` *strategies on a tree are bounded above by:*

$$\frac{210}{\alpha} \sum_{p \in \mathcal{B}} \log_2(|p|) \tag{2.16}$$

where $\alpha := \log_2(4/3)/\log_2(2)$. So since $\mathcal{B}$ is a blanket of size K the algorithm is, up to a constant factor, optimal (by Theorem 31).

Proof. Direct from Lemmas 37 and 42. □

2.11 Computing the predictions of the `0-Ising` strategy is NP-hard

Theorem 44. *Computing the predictions of the* `0-Ising` *strategy (see equation (2.1)) is NP-hard.*

2.11.1 Proof of Theorem 44

NB: Whenever we mention "graph" in this section we mean a graph with source (label "0") and target (label "1") vertices.

Given a graph $\mathcal{J}$, let $\perp(\mathcal{J})$ and $\top(\mathcal{J})$ be the source and target vertices of $\mathcal{J}$ respectively. We define a *label consistent labeling* of $\mathcal{J}$ to be a labeling $\boldsymbol{u}$ of $\mathcal{J}$ such that $u_{\perp(\mathcal{J})} = 0$ and $u_{\top(\mathcal{J})} = 1$. We define the *cutsize* of $\mathcal{J}$ to be the minimum cutsize of a label consistent labelling of $\mathcal{J}$. We define $\mathcal{Z}(\mathcal{J})$ to be the number of label consistent labellings of $\mathcal{J}$ that have cutsize equal to the cutsize of $\mathcal{J}$.

In this proof we assume that we have an oracle (i.e. a black-box that takes constant time) $\mathbf{test}(\cdot,\cdot)$ that, given an input graph $\mathcal{J}$, of cutsize K, and an input vertex $z \in V(\mathcal{J})$, outputs "0" if there are fewer label consistent labelings of cutsize K that label z as "0" than those that label z as "1" and outputs "1" otherwise. From $\mathbf{test}(\cdot,\cdot)$ we will construct a polynomial time algorithm for counting the number, $\mathcal{Z}(\mathcal{G})$, of label-consistent minimum cuts in a graph $\mathcal{G}$. Since the task of counting the number of label-consistent minimum cuts is #P-hard [46], we hence have that $\mathbf{test}(\cdot,\cdot)$ (i.e. computing equation (2.1)) is NP-hard. Let n be the number of vertices in $\mathcal{G}$.

Definition 45. *Given a graph $\mathcal{C}$ with cutsize $L \leq n$ we define the graph $\mathcal{C}^*$ as follows:*

1. *There exists a set $X \subseteq V(\mathcal{C}^*)$ of $n+1-L$ vertices such that $X \cap V(\mathcal{C}) = \emptyset$ and $V(\mathcal{C}^*) = X \cup V(\mathcal{C})$*
2. *$\perp(\mathcal{C}^*) = \perp(\mathcal{C})$ and $\top(\mathcal{C}^*) = \top(\mathcal{C})$*
3. *$E(\mathcal{C}^*) = E(\mathcal{C}) \cup \{(\perp(\mathcal{C}), x) : x \in X\} \cup \{(x,y) : x,y \in X, x \neq y\} \cup \{(x, \top(\mathcal{C})) : x \in X\}$*

Lemma 46. *Given a graph $\mathcal{C}$ with cutsize $L \leq n$ the graph $\mathcal{C}^*$ has cutsize $n+1$ and $\mathcal{Z}(\mathcal{C}^*) = 2\mathcal{Z}(\mathcal{C})$.*

Proof. Since none of the vertices on the edges of $\{(\perp(\mathcal{C}), x) : x \in X\} \cup \{(x,y) : x,y \in X, x \neq y\} \cup \{(x, \top(\mathcal{C})) : x \in X\}$ are in $V(\mathcal{C}) \setminus \{\perp(\mathcal{C}), \top(\mathcal{C})\}$ we have that for any min-cut (and label consistent) labelling of $\mathcal{C}^*$, the restriction of that labelling onto $\mathcal{C}$ has cutsize L. So suppose we have a (label consistent) labelling $\boldsymbol{u}$ of $\mathcal{C}$ with cutsize L. We now extent to a labelling $\boldsymbol{u}'$ of $\mathcal{C}^*$.

If $u'_x := 1$ for every $x \in X$ we clearly have a cutsize of $n+1$ (L cuts in $E(\mathcal{C})$, $n+1-L$ cuts in $\{(\perp(\mathcal{C}), x) : x \in X\}$ and no cuts in $\{(x,y) : x,y \in X, x \neq y\} \cup \{(x, \top(\mathcal{C})) : x \in X\}$). If $u'_x := 0$ for every $x \in X$ we also have a cutsize of $n+1$ (L cuts in $E(\mathcal{C})$, $n+1-L$ cuts in $\{(x, \top(\mathcal{C})) : x \in X\}$ and no cuts in $\{(\perp(\mathcal{C}), x) : x \in X\} \cup \{(x,y) : x,y \in X, x \neq y\}$).

Now suppose that the above two conditions don't hold: i.e. that we have vertices $x, y \in X$ with $u_x := 0$ and $u_y := 1$. Then we have at least one cut in $\{(x,y) : x,y \in X, x \neq y\}$. Let $X_1 := \{x \in X : u_x = 1\}$ and let $X_0 := \{x \in X : u_x = 0\}$. Then we have $|X_1|$ cuts in $\{(\perp(\mathcal{C}), x) : x \in X\}$ and $|X_0|$ cuts in $\{(x, \top(\mathcal{C})) : x \in X\}$ giving a total of at least $n+1-L$ cuts in the union of these two sets. Adding the L cuts in $V(\mathcal{C})$ gives us a total of over $n+1$ cuts.

So the cutsize of $\mathcal{C}^*$ is $n+1$ and moreover every (label consistent) labelling $\boldsymbol{u}$ of $\mathcal{C}$ of cutsize L extends to exactly two labellings of $\mathcal{C}^*$ of minimum cutsize. Hence we have that $\mathcal{Z}(\mathcal{C}^*) = 2\mathcal{Z}(\mathcal{C})$. □

Definition 47. *Given two graphs $\mathcal{C}$ and $\mathcal{D}$, define the* merger graph, *$[\mathcal{C}, \mathcal{D}]$, as follows:*

1. *The structure of $[\mathcal{C}, \mathcal{D}]$ is the graphs $\mathcal{C}$ and $\mathcal{D}$ with $\top(\mathcal{C})$ and $\perp(\mathcal{D})$ merged into a single vertex. i.e. we have a new vertex z such that $V([\mathcal{C}, \mathcal{D}]) := (V(\mathcal{C}) \setminus \{\top(\mathcal{C})\}) \cup (V(\mathcal{D}) \setminus \{\perp(\mathcal{D})\}) \cup \{z\}$ and $E([\mathcal{C}, \mathcal{D}]) = (E(\mathcal{C}) \setminus \{(v, \top(\mathcal{C})) : v \in V(\mathcal{C})\}) \cup \{(v, z) : (v, \top(\mathcal{C})) \in E(\mathcal{C})\} \cup (E(\mathcal{D}) \setminus \{(\perp(\mathcal{D}), v) : v \in V(\mathcal{D})\}) \cup \{(z, v) : (\perp(\mathcal{D}), v) \in E(\mathcal{D})\}$.*
2. *$\perp([\mathcal{C}, \mathcal{D}]) := \perp(\mathcal{C})$*
3. *$\top([\mathcal{C}, \mathcal{D}]) := \top(\mathcal{D})$*

Lemma 48. *Given graphs $\mathcal{C}$ and $\mathcal{D}$ of cutsize $n+1$ we have the following:*

1. *$[\mathcal{C},\mathcal{D}]$ has cutsize $n+1$*
2. *$\mathcal{Z}([\mathcal{C},\mathcal{D}]) = \mathcal{Z}(\mathcal{C}) + \mathcal{Z}(\mathcal{D})$.*
3. *Given that z is the vertex formed from the merger of $\top(\mathcal{C})$ and $\bot(\mathcal{D})$ then the output of $\mathbf{test}(z,[\mathcal{C},\mathcal{D}])$ is equal to 0 if $\mathcal{Z}(\mathcal{C}) < \mathcal{Z}(\mathcal{D})$ and equal to 1 otherwise.*

Proof. Let z be the vertex formed from the merger of $\top(\mathcal{C})$ and $\bot(\mathcal{D})$. Suppose we have a (label consistent) labelling, $\boldsymbol{u}$, of $[\mathcal{C},\mathcal{D}]$. If $u_z = 0$ (resp. $u_z = 1$) then there are at least $n+1$ cuts in the edges $(E(\mathcal{D})\setminus\{(\bot(\mathcal{D}),v) : v \in V(\mathcal{D})\})\cup\{(z,v) : (\bot(\mathcal{D}),v) \in E(\mathcal{D})\}$ (resp. $(E(\mathcal{C}) \setminus \{(v,\top(\mathcal{C})) : v \in V(\mathcal{C})\}) \cup \{(v,z) : (v,\top(\mathcal{C})) \in E(\mathcal{C})\}$). Hence, we must have that $[\mathcal{C},\mathcal{D}]$ has a cutsize of at least $n+1$ and furthermore that $\boldsymbol{u}$ has cutsize $n+1$ if and only if $u_x = 0$ for every $x \in V(\mathcal{C}) \setminus \{\top(\mathcal{C})\}$ (resp. $u_x = 1$ for every $x \in V(\mathcal{D}) \setminus \{\bot(\mathcal{D})\}$). So since there are $\mathcal{Z}(\mathcal{D})$ (resp. $\mathcal{Z}(\mathcal{C})$) labellings of $\{z\}\cup V(\mathcal{D})\setminus\{\bot(\mathcal{D})\}$ (resp. $\{z\}\cup V(\mathcal{C})\setminus\{\top(\mathcal{C})\}$) that label z as 0 and $\top(\mathcal{D})$ as 1 (resp. label z as 1 and $\bot(\mathcal{C})$ as 0) and have $n+1$ cuts in the edges $(E(\mathcal{D})\setminus\{(\bot(\mathcal{D}),v) : v \in V(\mathcal{D})\})\cup\{(z,v) : (\bot(\mathcal{D}),v) \in E(\mathcal{D})\}$ (resp. $(E(\mathcal{C})\setminus\{(v,\top(\mathcal{C})) : v \in V(\mathcal{C})\})\cup\{(v,z) : (v,\top(\mathcal{C})) \in E(\mathcal{C})\}$) we have that there are exactly $\mathcal{Z}(\mathcal{D})$ (resp. $\mathcal{Z}(\mathcal{C})$) label consistent labellings of $[\mathcal{C},\mathcal{D}]$ that label z as 0 (resp. z as 1) and have cutsize $n+1$. All the items of the lemma follow. □

Definition 49. *Given some $\alpha \in \mathbb{N}_n$ we define $\mathcal{Q}(\alpha)$ as follows:*

1. *There exists a set A of α vertices such that $\bot(\mathcal{Q}(\alpha)), \top(\mathcal{Q}(\alpha)) \notin A$ and $V(\mathcal{Q}(\alpha)) = \{\bot(\mathcal{Q}(\alpha)), \top(\mathcal{Q}(\alpha))\} \cup A$.*
2. *$E(\mathcal{Q}(\alpha)) = \{(\bot(\mathcal{Q}(\alpha)),v) : v \in A\} \cup \{(v,\top(\mathcal{Q}(\alpha))) : v \in A\}$*

Given, for some l, the sequence $(\alpha_0,\alpha_1,...,\alpha_l)$ with $\alpha_i < \alpha_{i+1} < n$ we now define a graph $\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l)$ that has a label consistent minimum cutsize of $n+1$ and such that $\mathcal{Z}(\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l)) = 2\sum_{i=0}^{l} 2^{\alpha_l}$

Definition 50. *Given, for some l, the sequence $(\alpha_0,\alpha_1,...,\alpha_l)$ with $\alpha_i < \alpha_{i+1} < n$ we inductively define $\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l)$ as follows:*

1. *$\mathfrak{B}(\alpha_0) = \mathcal{Q}(\alpha_0)^*$*
2. *$\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l) := [\mathcal{Q}(\alpha_0)^*, \mathfrak{B}(\alpha_1,\alpha_2...,\alpha_l)]$*

Lemma 51. *Given, for some l, the sequence $(\alpha_0,\alpha_1,...,\alpha_l)$ with $\alpha_i < \alpha_{i+1} \leq n$ we have that $\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l)$ has a cutsize of $n+1$ and $\mathcal{Z}(\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l)) := 2\sum_{i=0}^{l} 2^{\alpha_l}$.*

Proof. Noting that $\mathcal{Z}(\mathcal{Q}(\alpha_0)) = 2^{\alpha_0}$ and hence, by Lemma 46 $\mathcal{Z}(\mathcal{Q}(\alpha_0)^*) = 2 \cdot 2^{\alpha_0}$ and $\mathcal{Q}(\alpha_0)^*$ has cutsize $n+1$, the proof is direct by induction on l using items 1 and 2 of Lemma 48. □

Lemma 52. $\mathcal{Z}(\mathcal{G}) \leq 2^n$

Proof. Note that there are 2^n labellings of $\mathcal{G}$ which implies the result. □

The following algorithm calculates $\mathcal{Z}(\mathcal{G})$ (unless $\mathcal{Z}(\mathcal{G}) = 1$ in which case computing $\mathcal{Z}(\mathcal{G})$ is done by running $\mathbf{test}([\mathfrak{B}(0),\mathcal{G}^*],z)$ (where z is the vertex formed from the merger of $\top(\mathfrak{B}(0))$ and $\bot(\mathcal{G}^*)$)).

Algorithm 53. *Throughout the algorithm we maintain a graph $\mathcal{J}$, which is equal to $\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l)$ for some l and sequence $(\alpha_0,\alpha_1,...,\alpha_l)$. $\mathcal{J}$ is initialised to be equal to $\mathfrak{B}(n)$. The algorithm loops over the following:*

1. *Let $\mathfrak{B}(\alpha_0,\alpha_1,...,\alpha_l) := \mathcal{J}$. Construct the graph $[\mathcal{J},\mathcal{G}^*]$. Let z be the vertex in $[\mathcal{J},\mathcal{G}^*]$ formed from the merger of $\top(\mathcal{J})$ and $\bot(\mathcal{G}^*)$.*
2. *Run $\mathbf{test}([\mathcal{J},\mathcal{G}^*],z)$. If the output is 1 then set $\mathcal{J} \leftarrow \mathfrak{B}(\alpha_0-1,\alpha_1,\alpha_2,...,\alpha_l)$. If the output is 0 then run the following:*

(a) *Construct the graph* $[[\mathfrak{B}(0),\mathcal{J}],\mathcal{G}^*]$. *Let* z' *be the vertex in* $[[\mathfrak{B}(0),\mathcal{J}],\mathcal{G}^*]$ *formed from the merger of* $\top([\mathfrak{B}(0),\mathcal{J}])$ *and* $\bot(\mathcal{G}^*)$.

(b) *Run* $\mathbf{test}([[\mathfrak{B}(0),\mathcal{J}],\mathcal{G}^*],z')$. *If the output is* 1 *then the algorithm terminates outputting* $\mathcal{Z}(\mathcal{G})\leftarrow 1+\sum_{i=0}^{l}2^{\alpha_i}$. *If, instead, the output is* 0 *we set* $\mathcal{J}\leftarrow\mathfrak{B}(\alpha_0-1,\alpha_0,\alpha_1,...,\alpha_l)$

Theorem 54. *Given an oracle* $\mathbf{test}(\cdot,\cdot)$, *Algorithm 53 outputs* $\mathcal{Z}(\mathcal{G})$ *in polynomial time.*

Proof. Note first that by Lemma 46 and Lemma 48 Item 1 all graphs, $\mathcal{C}$, in the algorithm satisfy $\mathcal{Z}(\mathcal{C})=n+1$.

Since, by Lemma 52, we have that $\mathcal{Z}(\mathcal{G})\leq 2^n$ we can find, for some $l,\beta_0,\beta_1,...,\beta_l\in\mathbb{Z}$ such that $\beta_1\geq 0$, $\beta_l<n$, for all i we have $\beta_i<\beta_{i+1}$ and $\mathcal{Z}(\mathcal{G})=1+\sum_{i=q}^{l}2^{\beta_i}$. By Lemma 46 we have $\mathcal{Z}(\mathcal{G}^*)=2\mathcal{Z}(\mathcal{G})=2\left(1+\sum_{i=q}^{l}2^{\beta_i}\right)$. Let $\mathcal{J}_t$ be the graph $\mathcal{J}$ at the start of the t^{th} loop.

We prove, by reverse induction (i.e. from l to 0) on $j\leq l$ that there exists a t such that $\mathcal{J}_t=\mathfrak{B}(\beta_j,\beta_{j+1},...,\beta_l)$. We first show the base case: that there is some time t such that $\mathcal{J}_t=\beta_l$. To see this suppose at some time t' we have that $\mathcal{J}_{t'}=\mathfrak{B}(\beta')$ for some $\beta'>\beta_l$. We consider the $(t')^{\text{th}}$ loop. By Lemma 48 Item 3 and since $\mathcal{Z}(\mathcal{G}^*)\leq 2\cdot2\cdot2^{\beta_l}\leq 2\cdot2^{\beta'}=\mathcal{Z}(\mathcal{J}_{t'})$ (where the last equality comes from Lemma 51), the result of $\mathbf{test}([\mathcal{J}_{t'},\mathcal{G}^*],z)$ is 1. We hence have that $\mathcal{J}_{t'+1}=\mathfrak{B}(\beta'-1)$. Hence, since $\mathcal{J}_1=\mathfrak{B}(n)$ and $n>\beta_l$ we have that $\mathcal{J}_{1+n-\beta_l}=\mathfrak{B}(\beta_l)$. We have hence proved that the inductive hypothesis holds for $j=l$ so now suppose it holds for some $0<j\leq l$. We now show that it holds for $j-1$. Since it holds for j choose t'' such that $\mathcal{J}_{t''}=\mathfrak{B}(\beta_j,\beta_{j+1},...,\beta_l)$. We consider the $(t'')^{\text{th}}$ loop. By Lemma 48 Item 3 and since $\mathcal{Z}(\mathcal{G}^*)\geq 2\left(1+\beta_0+\sum_{i=j}^{l}2^{\beta_i}\right)>2\sum_{i=j}^{l}2^{\beta_i}=\mathcal{Z}(\mathcal{J}_{t''})$ (where the last equality comes from Lemma 51), the result of $\mathbf{test}([\mathcal{J}_{t'},\mathcal{G}^*],z)$ is 0. By Lemma 48 Item 3 and since $\mathcal{Z}(\mathcal{G}^*)\geq 2\left(1+\beta_0+\sum_{i=j}^{l}2^{\beta_i}\right)>2\left(1+\sum_{i=j}^{l}2^{\beta_i}\right)=\mathcal{Z}([\mathfrak{B}(0),\mathcal{J}_{t''}])$ (where the last equality comes from Lemma 51), the result of $\mathbf{test}([[\mathfrak{B}(0),\mathcal{J}_{t''}],\mathcal{G}^*],z')$ is 0. We hence have that $\mathcal{J}_{t''+1}=\mathfrak{B}(\beta_j-1,\beta_j,\beta_{j+1},...,\beta_l)$. Note now that if $\beta_j-1=\beta_{j-1}$ we are done. Else we have (as $\beta_{j-1}<\beta_j$) that $\beta_j-1>\beta_{j-1}$. Now suppose we have some t' with $\mathcal{J}_{t'}=\mathfrak{B}(\beta',\beta_j,\beta_{j+1},...,\beta_l)$ for some $\beta'>\beta_{j-1}$. We consider the $(t')^{\text{th}}$ loop. By Lemma 48 Item 3 and since $\mathcal{Z}(\mathcal{G}^*)\leq 2\left(2^{\beta_{j-1}}+\sum_{i=j-1}^{l}2^{\beta_i}\right)=2\left(2\cdot2^{\beta_{j-1}}+\sum_{i=j-1}^{l}2^{\beta_i}\right)\leq 2\left(2^{\beta'}+\sum_{i=j-1}^{l}2^{\beta_i}\right)=\mathcal{Z}(\mathcal{J}_{t'})$ (where the last equality comes from Lemma 51), the result of $\mathbf{test}([\mathcal{J}_{t'},\mathcal{G}^*],z)$ is 1. We hence have that $\mathcal{J}_{t'+1}=\mathfrak{B}(\beta'-1,\beta_j,\beta_{j+1},...\beta_l)$. Hence, since $\mathcal{J}_{t''+1}=\mathfrak{B}(\beta_j-1,\beta_j,\beta_{j+1},...,\beta_l)$ and $\beta_j-1\geq\beta_{j-1}$ we have that $\mathcal{J}_{t''+\beta_j-\beta_{j-1}}=\mathfrak{B}(\beta_{j-1},\beta_j,\beta_{j+1},...,\beta_l)$. This completes the proof of the inductive hypothesis.

By the above we have that there exists a time t such that $\mathcal{J}_t=\mathfrak{B}(\beta_0,\beta_1,...,\beta_l)$. We now show that the algorithm outputs at time t. By Lemma 48 Item 3 and since $\mathcal{Z}(\mathcal{G}^*)=2\left(1+\sum_{i=0}^{l}2^{\beta_i}\right)>2\sum_{i=0}^{l}2^{\beta_i}=\mathcal{Z}(\mathcal{J}_t)$ (where the last equality comes from Lemma 51), the result of $\mathbf{test}([\mathcal{J}_{t'},\mathcal{G}^*],z)$ is 0. By Lemma 48 Item 3 and since $\mathcal{Z}(\mathcal{G}^*)=2\left(1+\sum_{i=0}^{l}2^{\beta_i}\right)=\mathcal{Z}([\mathfrak{B}(0),\mathcal{J}_{t''}])$ (where the last equality comes from Lemma 51), the result of $\mathbf{test}([[\mathfrak{B}(0),\mathcal{J}_{t''}],\mathcal{G}^*],z)$ is 1. The algorithm hence outputs at time t with output $1+\sum_{i=0}^{l}2^{\beta_i}$ which is equal to $\mathcal{Z}(\mathcal{G})$. The algorithm hence outputs correctly.

We now show that given an oracle $\mathbf{test}(\cdot,\cdot)$, Algorithm 53 runs in in polynomial time. Note first that it is clear that each loop takes polynomial time. Hence, all that is required to show is that there is a polynomial number of loops. Let $\mathfrak{B}(\beta_0^t,\beta_1^t,...,\beta_{l^t})\coloneqq\mathcal{J}_t$. It is clear that for all t we have $\beta_0^{t+1}=\beta_0^t-1$ and hence, since $\beta_0^1=n$ we have at most $n+1$ loops. This completes the proof. □

Since, with an oracle $\mathbf{test}(\cdot,\cdot)$, Algorithm 53 solves a #P-hard problem in polynomial time we must have that $\mathbf{test}(\cdot,\cdot)$ is NP-hard.

Chapter 3

Online Similarity Prediction

3.1 Abstract

In this chapter we deal with the online similarity prediction problem. By converting the problem to a linear classification problem on a space of matrices we apply the perceptron and matrix winnow algorithms to give us two algorithms to solve the similarity prediction problem. We then use the concept of a binary support tree (first described in [32] and [17]) to ensure these algorithms have a low mistake bound. Finally we show how the perceptron-based algorithm on the binary support tree can be made exponentially faster.
This chapter is based on our paper "Online similarity prediction of networked data from known and unknown graphs" in COLT 2013. The final section of this chapter is all my own work. The other sections of this chapter were written in collaboration with Mark Herbster and Claudio Gentile.

3.2 Related Work

This chapter lies at the intersection between online learning on graphs and matrix/metric learning. Both fields include a substantial amount of work, so we can hardly do it justice here. Below we outline some of the main contributions in matrix/metric learning, with a special emphasis on those we believe are most related to this chapter.

Similarity prediction on graphs can be seen as a special case of matrix learning. Relevant works on this subject include [55, 57, 15, 33] – see also [27] for recent usage in the context of online cut prediction. In all these papers, special care is put into designing appropriate regularization terms driving the online optimization problem, the focus typically being on spectral sparseness. When operating on graph structures with Laplacian-based regularization, these algorithms achieve mistake bounds depending on functions of the cut-size of the labeled graph.Yet, in the absence of further efforts, their scaling properties make them inappropriate to practical usage in large networks. *Metric* learning is also relevant to this chapter. Metric learning is a special case of matrix learning where the matrix is positive semi-definite. Relevant references include [51, 19, 41, 59, 14]. Some of these papers also contain generalization bound arguments. Yet, no specific concerns are cast on networked data frameworks. Related to our bidirectional reduction from class prediction to similarity prediction is the thread of papers on kernels on pairs (e.g., [6, 41, 37, 11]), where kernels over pairs of objects are constructed as a way to measure the "distance" between the two referenced pairs. The idea is then to combine with any standard kernel algorithm. The so-called matrix completion task (specifically, the recent reference [35]) is also related to our work. In that paper, the authors introduce a matrix recovery method working in noisy environments, which incorporates both a low-rank and a Laplacian-regularization term. The problem of recovery of low-rank matrices has extensively been studied in the recent statistical literature (e.g., [12, 13, 26, 48, 42, 34], and references therein), the main concern being bounding the recovery error rate, but disregarding the computational aspects of the selected estimators. Moreover, the way they typically measure error rate is not easily comparable to online mistake bounds. Finally, the literature on semisupervised clustering/clustering with side information ([8, 21] – see also [47] for a recent reference on spectral approaches to clustering) is related to this chapter, since the similarity feedback can be interpreted as a must-link/cannot-link feedback. Nonetheless, their formal statements are fairly different from ours.

To summarize, whereas we are motivationally close to [35], from a technical viewpoint, we are perhaps closer to [51, 55, 57, 15, 27, 33], as well as to the literature on online learning on graphs.

3.3 Introduction and Overview

3.3.1 Problem

We have a graph G with $\mathcal{V}(G) = \mathbb{N}_n$ and a vector $\boldsymbol{y} : \mathbb{N}_n \to \mathbb{N}_K$, y_i being the *label* of vertex i.

The learner is given the graph G a-priori but does not know the vector $\boldsymbol{y}$. On trial t the learner is given a pair of vertices (i_t, j_t). We define $z_t = 0$ if $y_{i_t} = y_{j_t}$ and $z_t = 1$ otherwise. The learner is asked to give $\hat{z}_t$, its prediction of z_t. After giving the prediction, $\hat{z}_t$, the learner receives the true value of z_t.

3.3.2 Definitions

We define $\boldsymbol{e}_i$ to the i-th basis vector of $\mathbb{R}^n$.
We define L to be the laplacian of G. That is, the $n \times n$ matrix with L_{ii} equal to the degree of vertex i and, for $i \neq j$, $L_{ij} := -1$ (resp. $L_{ij} := 0$) if (i, j) is an edge of G (resp. not an edge of G)).
We define Ψ as an $n \times n$ matrix such that $\Psi^T \Psi = L$.
Given a matrix A we denote by A^+ the pseudo-inverse of A.
For $k \in \mathbb{N}_K$ we define the *class k cut*, Φ^G_k, to be the set of edges $(i, j) \in \mathcal{E}(G)$ for which $y_i = k$ and $y_j \neq k$ or for which $y_i \neq k$ and $y_j = k$. We define the *total cut*, Φ^G to be equal to $\bigcup_{k=1}^K \Phi^G_k$. Note that $|\Phi^G| = \frac{1}{2} \sum_{k=1}^K |\Phi^G_k|$
Given vertices $i, j \in \mathcal{V}(G)$ we define $R^G_{i,j}$ to be the *effective resistance* between vertices i and j. We define R^G to be the *resistance diameter* of G, that is, $\max_{i,j \in \mathcal{V}(G)} R^G_{i,j}$.
We also define the *class k resistance weighted cutsize*, ϕ^G_k, to be:

$$\phi^G_k := \sum_{(i,j) \in \Phi^G_k} R^G_{i,j} \tag{3.1}$$

3.3.3 Overview

In section 3.4 we transform the problem to a linear classification problem in the Hilbert space of $n \times n$ matrices with the Frobenius inner-product. We then run the perceptron and matrix winnow algorithms in this space which gives us the following mistake bounds:

Theorem 55. *If we run the perceptron based algorithm with graph G we get a mistake bound of:*

$$M^{\mathrm{P}} \in \mathcal{O}\left((R^G)^2 \sum_{k=1}^{k} |\Phi^G_k|^2 \right) . \tag{3.2}$$

If we run the matrix winnow based algorithm with graph G we get a mistake bound of:

$$M^{\mathrm{W}} \in \mathcal{O}\left(R^G \log(n) \sum_{k=1}^{k} |\Phi^G_k| \right) . \tag{3.3}$$

In section 3.5 we show how to construct a *binary support tree*, B, of G, who's leaves are the vertices of G (first defined in [32] and [17]). Running the previous algorithms on B, instead of on G, gives us the following expected mistake bounds:

Theorem 56. *If we run the perceptron based algorithm with tree B we get an expected mistake bound of:*

$$\mathbb{E}\left(M^{\mathrm{P}}\right) \in \mathcal{O}\left(\log^4(n)\sum_{k=1}^{k}(\phi_k^G)^2\right) . \tag{3.4}$$

If we run the matrix winnow based algorithm with tree B we get an expected mistake bound of:

$$\mathbb{E}\left(M^{\mathrm{W}}\right) \in \mathcal{O}\left(\log^3(n)\sum_{k=1}^{k}\phi_k^G\right) . \tag{3.5}$$

A severe drawback of these algorithms (when computed naively) is that they take polynomial time per round ($\mathcal{O}(n^2)$ for the perceptron and $\mathcal{O}(n^3)$ for matrix winnow) and initialisation requires taking a pseudo-inverse (typically cubic time). Hence, in section 3.6 we give an implementation of the perceptron algorithm that requires only $\mathcal{O}(\log^2(n))$ time per round, only $\mathcal{O}(n)$ initialisation time and a constant consecutive space requirement (i.e. we don't store arrays but linked data-structures).

3.4 Similarity Prediction via Linear Classification

3.4.1 The Algorithms

In Algorithm 1 we give a simple application of the Matrix Winnow (superscript "w") and Perceptron (superscript "p") algorithms to similarity prediction on graphs. The algorithms work by the reduction of the similarity problem following linear classification problem:

We work in the Hilbert space of $n \times n$ matrices with inner product given by $\langle A, B\rangle := \mathrm{TR}(A^T B)$. The pair of vertices (i_t, j_t) are converted to an element X_t of this space given by the following equation (common to the metric learning literature):

$$X_t \leftarrow (\Psi^+)^\top(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^\top\Psi^+ \tag{3.6}$$

and in the case of matrix winnow X_t is normalised to have trace 1. The algorithms are then the perceptron and matrix winnow algorithms run on the instances X_t.

There is a tuning issue related to Matrix Winnow, since the threshold $\widehat{\theta}^{\mathrm{W}}$ depends on the (unknown) cut-size. To solve this we use the *doubling trick* which is a standard technique. We first define $\omega := \frac{\eta}{(e^{\eta}-e^{-\eta})R^G}$. Let $\mu := 4R^G \log n$. We use the result, proved in Section 3.4.3, that with the correct tuning the matrix winnow algorithm makes no more than $\mu|\Phi^G|$ mistakes. The doubling trick works as follows: We define the algorithm $\mathcal{A}_\tau(i)$ (for a time τ and natural number i) as follows:

1. Set $W_\tau \leftarrow \frac{1}{m}I$ (i.e. the matrix winnow algorithm is restarted)
2. Run the matrix winnow algorithm on $((i_\tau, j_\tau), (i_{\tau+1}, j_{\tau+1}), (i_{\tau+2}, j_{\tau+2}), (i_{\tau+3}, j_{\tau+3}), \ldots)$ with $\widehat{\theta}^{\mathrm{W}} = \omega/2^i$ until it has made $2^{i+1}\mu$ mistakes.
3. Let τ' be the time at which $\mathcal{A}_\tau(i)$ has made $2^{i+1}\mu$ mistakes.
4. Run $\mathcal{A}_{\tau'}(i+1)$.

We now show that the algorithm $\mathcal{A}_1(0)$ makes $\mathcal{O}(|\Phi^G|\, R^G\, \log n)$ mistakes: To prove this note that if the algorithm $\mathcal{A}_\tau(\lceil\log(|\Phi^G|)\rceil)$ is called at some time τ then since, during this algorithm $\widehat{\theta}^{\mathrm{W}} \leq \omega/R^G$ it makes no more than $2^{(\lceil\log(|\Phi^G|)\rceil)}\mu$ mistakes and hence $\mathcal{A}_{\tau'}(\lceil\log(|\Phi^G|)\rceil+1)$ is never called (for any τ'). Since $\mathcal{A}_\tau(i)$ makes $2^{i+1}\mu$

Algorithm 1: Perceptron and Matrix Winnow algorithms on a graph

```
**Input:** Graph $G$ with $\mathcal{V}(G) = \mathbb{N}_n$ and Laplacian $L = \Psi^\top \Psi$,
**Parameters:** Perceptron threshold $\widehat{\theta}^{\mathrm{p}} = (R^G)^2$; Winnow threshold
$\widehat{\theta}^{\mathrm{w}} = \frac{\eta}{e^\eta - e^{-\eta}} \frac{1}{R^G |\Phi^G|}$, Winnow learning rate $\eta = 1.28$;
**Initialization:** $W_0^{\mathrm{p}} = \mathbf{0} \in \mathbb{R}^{m \times m}$; $W_0^{\mathrm{w}} = \frac{1}{m} I \in \mathbb{R}^{m \times m}$;
**For** $t = 1, 2, \ldots, T$ :
    • Get pair of vertices $(i_t, j_t) \in \mathcal{V}(G)^2$, and construct similarity instances,
      $X_t^{\mathrm{p}} \leftarrow (\Psi^+)^\top (e_{i_t} - e_{j_t})(e_{i_t} - e_{j_t})^\top \Psi^+$; $X_t^{\mathrm{w}} \leftarrow \frac{(\Psi^+)^\top (e_{i_t} - e_{j_t})(e_{i_t} - e_{j_t})^\top \Psi^+}{(e_{i_t} - e_{j_t})^\top L^+ (e_{i_t} - e_{j_t})}$;   (3.13)
    • Predict: $\hat{z}_t^{\mathrm{p}} \leftarrow [\mathrm{TR}((W_{t-1}^{\mathrm{p}})^\top X_t^{\mathrm{p}}) > \widehat{\theta}^{\mathrm{p}}]$;
      $\hat{z}_t^{\mathrm{w}} \leftarrow [\mathrm{TR}((W_{t-1}^{\mathrm{w}})^\top X_t^{\mathrm{w}}) > \widehat{\theta}^{\mathrm{w}}]$;
    • Observe $z_t \in \{0, 1\}$ and, if mistake ($z_t \neq \hat{z}_t$), update
      $W_t^{\mathrm{p}} \leftarrow W_{t-1}^{\mathrm{p}} + (z_t - \hat{z}_t^{\mathrm{p}})\, X_t^{\mathrm{p}}$;   $\log W_t^{\mathrm{w}} \leftarrow \log W_{t-1}^{\mathrm{w}} + \eta\, (z_t - \hat{z}_t^{\mathrm{w}})\, X_t^{\mathrm{w}}$ .
```

mistakes (until $\mathcal{A}_{\tau'}(i+1)$ is called) we hence have that the overall algorithm makes no more than m mistakes where:

$$
\begin{aligned}
m &= \sum_{i=1}^{\lceil \log(|\Phi^G|) \rceil} 2^{i+1} \mu && (3.7)\\
&\le 2^{\lceil \log(|\Phi^G|) \rceil + 2} \mu && (3.8)\\
&\le 2^{\lfloor \log(|\Phi^G|) \rfloor + 3} \mu && (3.9)\\
&= 8 \cdot 2^{\lfloor \log(|\Phi^G|) \rfloor} \mu && (3.10)\\
&\le 8 |\Phi^G| \mu && (3.11)\\
&= 32 |\Phi^G| R^G \log n && (3.12)
\end{aligned}
$$

3.4.2 Perceptron Mistake Bound

In this subsection and the next we prove theorem 55. We often use, in this subsection and the next, the result that given an $n \times m$ matrix A and an $m \times n$ matrix B we have $\mathrm{TR}(AB) = \mathrm{TR}(BA)$.

We start off with the Matrix Perceptron bound. We define $X_t := (\Psi^+)^\top (e_{i_t} - e_{j_t})(e_{i_t} - e_{j_t})^\top \Psi^+$ (i.e. $X_t := X^{\mathrm{p}}$) and let $\langle A, B \rangle$ be a shorthand for the inner product $\mathrm{TR}(A^T B)$. For brevity we define, in this subsection, $W_t := W_t^{\mathrm{p}}$ and $\widehat{\theta} := \widehat{\theta}^{\mathrm{p}}$
We define the separator, U as follows:

Definition 57. *Define K vectors $u_1, \ldots, u_K \in \mathbb{R}^n$ such that the i-th component, $u_{k,i}$, of u_k is equal to 1 is $y_i = k$ (where y_i is the label of the i-th vertex of G) and equal to 0 otherwise. Define $U := \Psi \left(\sum_{k=1}^K u_k u_k^\top \right) \Psi^\top$*

We first bound the norms of the instances:

Lemma 58. $\langle X_t, X_t \rangle \le (R^G)^2$

Proof. We can write

$$\begin{aligned}
\langle X_t, X_t\rangle &= \textsc{tr}((\Psi^+)^T(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+(\Psi^+)^T(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+)\\
&= \textsc{tr}((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+(\Psi^+)^T(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+(\Psi^+)^T(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\
&= ((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))^2\\
&= (R^G_{i_t,j_t})^2\\
&\le (R^G)^2\ .
\end{aligned}$$

□

We now prove the linear separability of sequence $(X_1, z_1), (X_2, z_2), \ldots$ w.r.t. U.

Lemma 59. *If $z_t = 0$ then $\langle U, X_t\rangle = 0$. If $z_t = 1$ then $\langle U, X_t\rangle = 2$.*

Proof. We can write

$$\begin{aligned}
\langle U, X_t\rangle &= \textsc{tr}(U^T X_t)\\
&= \sum_{k=1}^{K}\textsc{tr}(\Psi\boldsymbol{u}_k\boldsymbol{u}_k^T\Psi^T(\Psi^+)^T(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+)\\
&= \sum_{k=1}^{K}\textsc{tr}((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+\Psi\boldsymbol{u}_k\boldsymbol{u}_k^T\Psi^T(\Psi^+)^T(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\
&= \sum_{k=1}^{K}((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+\Psi\boldsymbol{u}_k)^2\ .
\end{aligned}$$

By definition of pseudoinverse, $\Psi(\Psi^+\Psi\boldsymbol{u}_k) = (\Psi\Psi^+\Psi)\boldsymbol{u}_k = \Psi\boldsymbol{u}_k$ for all $k = 1,\ldots,K$. Hence we have $\Psi(\Psi^+\Psi - I)\boldsymbol{u}_k = 0$ so $(\Psi^+\Psi - I)\boldsymbol{u}_k = c\mathbf{1}$ for some $c \in \mathbb{R}$ (noting that multiples of $\mathbf{1}$ are the only solution to the equation $\Psi x = 0$) . Hence, $\Psi^+\Psi\boldsymbol{u}_k = \boldsymbol{u}_k + c\mathbf{1}$ for some $c \in \mathbb{R}$. We therefore have that $(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^T\Psi^+\Psi\boldsymbol{u}_k = u_{k,i_t} - u_{k,j_t}$, i.e.,

$$\langle U, X_t\rangle = \sum_{k=1}^{K}(u_{k,i_t} - u_{k,j_t})^2\ .$$

Now, if $z_t = 0$ (i.e., $y_{i_t} = y_{j_t}$) then for all k we have $u_{k,i_t} - u_{k,j_t} = 0$, so that $\langle U, X_t\rangle = 0$. On the other hand, if $z_t = 1$ (i.e., $y_{i_t} \neq y_{j_t}$) then there exist distinct $a, b \in \{1,\ldots,K\}$ such that $|u_{a,i_t} - u_{a,j_t}| = |u_{b,i_t} - u_{b,j_t}| = 1$, and for all other $k \neq a, b$ we have $u_{k,i_t} - u_{k,j_t} = 0$. So, in this case $\langle U, X_t\rangle = 2$. □

Finally, we bound the norm of the separator, U:

Lemma 60. $\langle U, U\rangle \le 2\sum_{k=1}^{K}|\Phi^G_k|^2$

Proof. Let $\Phi^G_{a,b} := \{(i,j) \in \mathcal{E}(G) \,:\, y_i = a,\, y_j = b\}$. We have:

$$\begin{aligned}
\langle U, U\rangle &= \mathrm{TR}(U^T U)\\
&= \mathrm{TR}\left(\left(\sum_{a=1}^{K} \Psi \boldsymbol{u}_a \boldsymbol{u}_a^T \Psi^T\right)\left(\sum_{b=1}^{K} \Psi \boldsymbol{u}_b \boldsymbol{u}_b^T \Psi^T\right)\right)\\
&= \sum_{a=1}^{K}\sum_{b=1}^{K} \mathrm{TR}(\Psi \boldsymbol{u}_a \boldsymbol{u}_a^T \Psi^T \Psi \boldsymbol{u}_b \boldsymbol{u}_b^T \Psi^T)\\
&= \sum_{a=1}^{K}\sum_{b=1}^{K} \mathrm{TR}(\boldsymbol{u}_b^T \Psi^T \Psi \boldsymbol{u}_a \boldsymbol{u}_a^T \Psi^T \Psi \boldsymbol{u}_a)\\
&= \sum_{a=1}^{K}\sum_{b=1}^{K} (\boldsymbol{u}_b^T \Psi^T \Psi \boldsymbol{u}_a)^2\\
&= \sum_{a=1}^{K}\sum_{b=1}^{K} (\boldsymbol{u}_b^T L \boldsymbol{u}_a)^2\\
&= \sum_{a=1}^{K}\left((\boldsymbol{u}_a^T L \boldsymbol{u}_a)^2 + \sum_{b\neq a} (\boldsymbol{u}_b^T L \boldsymbol{u}_a)^2\right)\\
&= \sum_{a=1}^{K}\left((|\Phi^G_a|)^2 + \sum_{b\neq a} (-|\Phi^G_{a,b}|)^2\right)\\
&= \sum_{a=1}^{K}\left(|\Phi^G_a|^2 + \sum_{b\neq a} |\Phi^G_{a,b}|^2\right)
\end{aligned}$$

So, noticing that $\sum_{b\,:\,b\neq a} |\Phi^G_{a,b}| = |\Phi^G_a|$ and hence that $\sum_{b\,:\,b\neq a} |\Phi^G_{a,b}|^2 \leq |\Phi^G_a|^2$, we get the result. □

With the above handy, we now follow the standard analysis of the Perceptron algorithm with non-zero threshold to derive the mistake bound M^{P}.

We denote by $\mathcal{M}^0$ the set of mistaken trials with $z_t = 1$ and $\hat{z}_t = 0$, and by $\mathcal{M}^1$ the set of mistaken trials with $z_t = 0$ and $\hat{z}_t = 1$, with $|\mathcal{M}| = |\mathcal{M}^0| + |\mathcal{M}^1|$. We have

$$\langle W_t, W_t\rangle = \langle W_{t-1}, W_{t-1}\rangle + \langle X_t, X_t\rangle + 2\,(z_t - \hat{z}_t)\,\langle W_{t-1}, X_t\rangle,$$

where

$$(z_t - \hat{z}_t)\,\langle W_{t-1}, X_t\rangle \leq \begin{cases} \widehat{\theta} & \text{if } t \in \mathcal{M}^0 \\ -\widehat{\theta} & \text{if } t \in \mathcal{M}^1\,. \end{cases}$$

Hence, $\langle W_T, W_T\rangle \leq \max_t \langle X_t, X_t\rangle\,|\mathcal{M}| + 2\,\widehat{\theta}\,|\mathcal{M}^0| - 2\,\widehat{\theta}\,|\mathcal{M}^1|$ so, by lemma 58, $\langle W_T, W_T\rangle \leq (R^G)^2 |\mathcal{M}| + 2\,\widehat{\theta}\,|\mathcal{M}^0| - 2\,\widehat{\theta}\,|\mathcal{M}^1|$.
Moreover,

$$\langle U, W_t\rangle = \langle U, W_{t-1} + (z_t - \hat{z}_t)\,X_t\rangle = \langle U, W_{t-1}\rangle + (z_t - \hat{z}_t)\,\langle U, X_t\rangle,$$

where, by lemma 59:

$$(z_t - \hat{z}_t)\,\langle U, X_t\rangle = \begin{cases} 2 & \text{if } t \in \mathcal{M}^0 \\ 0 & \text{if } t \in \mathcal{M}^1\,, \end{cases}$$

Hence $\langle U, W_T\rangle = 2\,|\mathcal{M}^0|$. Using the Cauchy-Schwarz ineq. $\langle U, W_T\rangle^2 \leq \langle U, U\rangle\,\langle W_T, W_T\rangle$, and putting together gives

$$4\,|\mathcal{M}^0|^2 \leq \langle U, U\rangle\,\left((R^G)^2\,(|\mathcal{M}^0| + |\mathcal{M}^1|) + 2\,\widehat{\theta}\,|\mathcal{M}^0| - 2\,\widehat{\theta}\,|\mathcal{M}^1|\right)$$

So, with $\widehat{\theta} := (R^G)^2$ we have

$$4\,|\mathcal{M}^0|^2 \leq \langle U, U\rangle (R^G)^2(3|\mathcal{M}^0| - |\mathcal{M}^1|) \tag{3.14}$$

Since $4\,|\mathcal{M}^0|^2$ and $\langle U, U\rangle (R^G)^2$ are both non-negative, Equation 3.14 automatically gives us $3|\mathcal{M}^0| - |\mathcal{M}^1| \geq 0$ so we must have $|\mathcal{M}^1| \leq 3|\mathcal{M}^0|$. Also, since $\langle U, U\rangle (R^G)^2|\mathcal{M}^1|$ is non-negative Equation 3.14 automatically gives us $4\,|\mathcal{M}^0|^2 \leq 3\langle U, U\rangle (R^G)^2|\mathcal{M}^0|$ which, dividing by $4|\mathcal{M}^0|$ gives us $|\mathcal{M}^0| \leq \frac{3}{4}\langle U, U\rangle (R^G)^2$. We hence have:

$$\begin{aligned}
|\mathcal{M}| &= |\mathcal{M}^0| + |\mathcal{M}^1| \\
&\leq |\mathcal{M}^0| + 3|\mathcal{M}^0| \\
&= 4|\mathcal{M}^0| \\
&\leq 3\langle U, U\rangle (R^G)^2 \\
&\leq 6(R^G)^2 \sum_{k=1}^{K} |\Phi_k^G|^2
\end{aligned}$$

where the last line comes from Lemma 60.

3.4.3 Winnow Mistake Bound

In this subsection we use the arguments in [57] to aid us in deriving a mistake bound for the matrix-winnow based algorithm

Let X_t and U be as in the previous subsection. In this subsection we define $\tilde{U} := U/(2|\Phi^G|)$. For brevity we define, in this subsection, $W_t := W_t^{\mathrm{w}}$ and $\widehat{\theta} := \widehat{\theta}^{\mathrm{w}}$. We define $\theta := 1/(R^G|\Phi^G|)$

Lemma 61. $\textsc{tr}(\tilde{U}) = 1$

Proof.

$$\begin{aligned}
\textsc{tr}(U) &= \textsc{tr}\left(\Psi\left(\sum_{k=1}^{K} \boldsymbol{u}_k\boldsymbol{u}_k^\top\right)\Psi^\top\right) \\
&= \sum_{k=1}^{K} \textsc{tr}(\Psi\boldsymbol{u}_k\boldsymbol{u}_k^\top\Psi^\top) \\
&= \sum_{k=1}^{K} \textsc{tr}(\boldsymbol{u}_k^\top\Psi^\top\Psi\boldsymbol{u}_k) \\
&= \sum_{k=1}^{K} \boldsymbol{u}_k^\top\Psi^\top\Psi\boldsymbol{u}_k \\
&= \sum_{k=1}^{K} \boldsymbol{u}_k^\top L\boldsymbol{u}_k \\
&= \sum_{k=1}^{K} |\Phi_k^G| \\
&= 2|\Phi^G|
\end{aligned}$$

which implies the result ☐

Lemma 62. $\textsc{tr}(X_t^w) = 1$

Proof.

$$\begin{aligned}\mathrm{TR}(X_t) &= \mathrm{TR}((\Psi^+)^\top(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top\Psi^+)\\ &= \mathrm{TR}((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top\Psi^+(\Psi^+)^\top(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\ &= (\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top\Psi^+(\Psi^+)^\top(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})\\ &= (\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})\end{aligned}$$

which, since $X_t^{\mathrm{W}} = \frac{X_t}{(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})}$, implies the result. □

Lemma 63. *If $z_t = 0$ then $\mathrm{TR}(\tilde{U}X_t^w) = 0$. If $z_t = 1$ then $\mathrm{TR}(\tilde{U}X_t^w) \geq \theta$*

Proof. By Lemma 59 we have:

1. If $z_t = 0$ then:

$$\begin{aligned}\mathrm{TR}(\tilde{U}X_t^{\mathrm{W}}) &= \mathrm{TR}(UX_t)/(2|\Phi^G|((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\ &= \langle U, X_t\rangle/(2|\Phi^G|((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\ &= 0.\end{aligned}$$

2. If $z_t = 1$ then:

$$\begin{aligned}\mathrm{TR}(\tilde{U}X_t^{\mathrm{W}}) &= \mathrm{TR}(UX_t)/(2|\Phi^G|((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\ &= \langle U, X_t\rangle/(2|\Phi^G|((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\ &= 2/(2|\Phi^G|((\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t}-\boldsymbol{e}_{j_t}))\\ &= 1/(|\Phi^G|R^G_{i_t,j_t})\\ &\geq 1/(|\Phi^G|R^G)\\ &= \theta\end{aligned}$$

□

Let $d(\tilde{U}, W)$ be the matrix (or "quantum") relative entropy

$$d(\tilde{U}, W) = \mathrm{TR}(\tilde{U}\ \log\tilde{U} - \tilde{U}, \log W + W - \tilde{U})\ .$$

Then, when t is an updating round, we have

$$\begin{aligned}d(\tilde{U}, W_{t-1}) - d(\tilde{U}, W_t) &= \mathrm{TR}(\tilde{U}\ \log W_t - \tilde{U}\ \log W_{t-1}) + \mathrm{TR}(W_{t-1}) - \mathrm{TR}(W_t)\\ &= \eta\,(z_t - \hat{z}_t)\,\mathrm{TR}(\tilde{U}X_t^{\mathrm{W}}) + \mathrm{TR}(W_{t-1}) - \mathrm{TR}(\exp(\log W_{t-1} + \eta\,(z_t-\hat{z}_t)\,X_t^{\mathrm{W}}))\end{aligned}$$

From Golden-Thompson ineq. we have that for any two symmetric (need not be positive definite) matrices A and B

$$\mathrm{TR}(\exp(A+B)) \leq \mathrm{TR}(\exp(A)\ \exp(B))$$

that we apply to the above with $A = \log W_{t-1}$ and $B = \eta\,(z_t - \hat{z}_t)\,X_t^{\mathrm{W}}$. This gives

$$\begin{aligned}d(\tilde{U}, W_{t-1}) - d(\tilde{U}, W_t) &\geq \eta\,(z_t - \hat{z}_t)\,\mathrm{TR}(\tilde{U}X_t^{\mathrm{W}}) + \mathrm{TR}(W_{t-1} - \exp(\log W_{t-1})\ \exp(\eta\,(z_t-\hat{z}_t)\,X_t^{\mathrm{W}}))\\ &= \eta\,(z_t - \hat{z}_t)\,\mathrm{TR}(\tilde{U}X_t^{\mathrm{W}}) + \mathrm{TR}(W_{t-1}(I - \exp(\eta\,(z_t-\hat{z}_t)\,X_t^{\mathrm{W}}))\ .\end{aligned} \tag{3.15}$$

We then use the following two facts:

1. For any positive semidefinite matrix X with eigenvalues in $[0, 1]$, and any $a \in \mathbb{R}$ we have

$$(1 - e^a)\,X \leq I - \exp(aX),$$

where "$\leq$" is the partial order over positive semidefinite matrices.

2. If A is a positive semidefinite matrix and B and C are two symmetric matrices with $B \leq C$, then $\text{TR}(AB) \leq \text{TR}(AC)$.

Notice that X_t^{W} in (3.15) has eigenvalues in $[0,1]$ since it is positive semidefinite with $\text{TR}(X_t^{\text{W}}) = 1$ (by lemma 62). We use Item 1 with $a = \eta\,(z_t - \hat{z}_t)$, and $X = X_t^{\text{W}}$, and then Item 2 with $A = W_{t-1}$, resulting in

$$d(\tilde{U}, W_{t-1}) - d(\tilde{U}, W_t) \geq \eta\,(z_t - \hat{z}_t)\,\text{TR}(\tilde{U}X_t^{\text{W}}) + (1 - e^{\eta\,(z_t - \hat{z}_t)})\,\text{TR}(W_{t-1}X_t^{\text{W}})\,. \quad (3.16)$$

At this point we distinguish the two cases: 1) $z_t = 1$, $\hat{z}_t = 0$ and 2) $z_t = 0$, $\hat{z}_t = 1$.

In case 1) we have, by lemma 63, that $\text{TR}(\tilde{U}X_t^{\text{W}}) \geq \theta$ and $\text{TR}(W_{t-1}X_t^{\text{W}}) < \widehat{\theta}$. Hence the RHS of (3.16) is at least (notice that $1 - e^{\eta}$ is negative)

$$\eta\,\theta + (1 - e^{\eta})\,\widehat{\theta}.$$

In case 2) we have, by lemma 63, that $\text{TR}(\tilde{U}X_t^{\text{W}}) = 0$ and $\text{TR}(W_{t-1}X_t^{\text{W}}) \geq \widehat{\theta}$. Hence the RHS of (3.16) is at least (notice that $1 - e^{-\eta}$ is positive)

$$(1 - e^{-\eta})\,\widehat{\theta}.$$

Setting

$$\widehat{\theta} = \frac{\eta\theta}{e^{\eta} - e^{-\eta}}$$

makes the two progress lower bound to be equal to $\eta\theta\,\frac{1-e^{-\eta}}{e^{\eta}-e^{-\eta}} = \frac{\eta\theta}{1+e^{\eta}}$, which is at least $\theta/4$ when $\eta \simeq 1.28$.

With the above setting for η and $\widehat{\theta}$ in hand, we sum (3.16) over mistaken trials, and conclude that the resulting algorithm has a number of mistakes m satisfying

$$m\,\theta/4 \leq d(\tilde{U}, W_0) - d(\tilde{U}, W_T) \leq d(\tilde{U}, W_0).$$

Now, since, by lemma 61, $\text{TR}(\tilde{U}) = 1$, and $\text{TR}(W_0) = 1$, it is easy to see that

$$d(\tilde{U}, W_0) = \text{TR}(\tilde{U}\log\tilde{U}) - \text{TR}(\tilde{U}\log W_0) + \text{TR}(W_0) - \text{TR}(\tilde{U}) = 0 - \log\frac{1}{n} + 1 - 1 = \log n$$

i.e.,

$$m \leq 4|\Phi^G|\,R^G\,\log n\,.$$

3.5 Support Tree

In this section we describe the construction of a *binary support tree* (BST) of the graph G (First introduced in [32] and [17]). We will then run the algorithms of the preceding section on the support tree rather than G. The benefits of doing this are threefold:

- We replace the (possibly large) factors of R^G in the mistake bounds by $\log(n)$.
- We replace the cutsize factors in the mistake bound by the (often much smaller) resistance-weighted cutsize in expectation.
- In the next section we will give an implementation of the perceptron algorithm on the support tree that requires only $\mathcal{O}(n)$ initialisation time and only $\mathcal{O}(\log(n)^2)$ time per trial.

The support tree is constructed as follows:

1. Sample an uniform random spanning tree, T, of G. Recall that a uniformly random spanning tree of an unweighted graph can be sampled in expected time $\mathcal{O}(n \ln n)$ for "most" graphs [10]. Using the nice algorithm of [58], the expected time reduces to $\mathcal{O}(n)$ —see also the work of [3]. However, all known techniques take expected time $\Theta(n^3)$ in certain pathological cases.

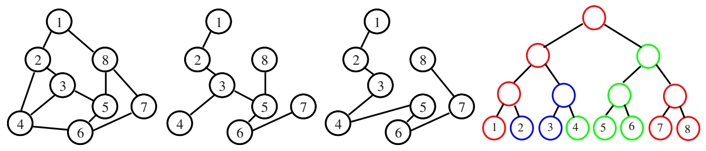


FIGURE 3.1: **From left to right:** The graph G; a random spanning tree T of G (note that the vertices are numbered by the order of a depth-first visit of T, starting from root vertex 1; the path graph P which follows the order of the depth-first visit of T; the BST built on top of P. Notice how the class labels of the 8 vertices in V (corresponding to the three colors) are propagated upwards.

2. Let P be a path graph with vertex set $\mathcal{V}(G)$ and who's vertices follow the ordering given by a depth first search of T.
3. Define the support tree, B, to be a full balanced binary tree with leaves $\mathcal{V}(G)$ which are ordered as in P.

The labelling y, on $\mathcal{V}(G)$ is extended to $\mathcal{V}(B)$ recursively from leaves to root as follows: Given an internal vertex, i, of B, define $y_i := y_j$ where j is the left child of i.

We the run the algorithms of the preceding section on B instead of G.

3.5.1 Mistake Bounds

We now prove theorem 56. We start with the following lemma:

Lemma 64. *Let $(G, \boldsymbol{y})$ be a labeled graph, and T be a spanning tree of G drawn uniformly at random. Then, for all $k = 1, \ldots, K$, we have:*

1. $\mathbb{E}[|\Phi_k^T|] = \sum_{(i,j)\in\Phi_k^G} R_{i,j}^G = \phi_k^G$*, and*
2. $\mathbb{E}[|\Phi_k^T|^2] \le 2(\sum_{(i,j)\in\Phi_k^G} R_{i,j}^G)^2 = (\phi_k^G)^2$.

Proof. Set $s = |\Phi_k^G|$ and $\Phi_k^G = \{(i_1, j_1), (i_2, j_2), \ldots, (i_s, j_s)\}$. Also, for $\ell = 1, \ldots, s$, let Y_ℓ be the random variable which is 1 if (i_ℓ, j_ℓ) is an edge of T, and 0 otherwise. From $\mathbb{E}[Y_\ell] = R_{i_\ell,j_\ell}^G$ (a standard result) we immediately have 1). In order to prove 2), we rely on the negative correlation of variables Y_ℓ, i.e., that $\mathbb{E}[Y_\ell\, Y_{\ell'}] \le \mathbb{E}[Y_\ell]\,\mathbb{E}[Y_{\ell'}]$ for $\ell \ne \ell'$ (a standard result: see, e.g., [40]). Then we can write

$$\begin{aligned}
\mathbb{E}(|\Phi_k^T|^2) &= \mathbb{E}\left[\left(\sum_{\ell=1}^{s} Y_\ell\right)^2\right] \\
&= \mathbb{E}\left[\sum_{\ell=1}^{s}\sum_{\ell'=1}^{s} Y_\ell Y_{\ell'}\right] \\
&= \sum_{\ell=1}^{s}\mathbb{E}[Y_\ell] + \sum_{\ell=1}^{s}\sum_{\ell'\ne\ell}\mathbb{E}[Y_\ell Y_{\ell'}] \\
&\le \sum_{\ell=1}^{s}\mathbb{E}[Y_\ell] + \sum_{\ell=1}^{s}\sum_{\ell'\ne\ell}\mathbb{E}[Y_\ell]\,\mathbb{E}[Y_{\ell'}]\,.
\end{aligned}$$

Now, for any spanning tree T of G, if $s \ge 1$ then it must be the case that $|\Phi_k^T| \ge 1$, and hence $\sum_{\ell=1}^{s}\mathbb{E}[Y_\ell] = \mathbb{E}[|\Phi_k^T|] \ge 1$. Combined with the above we obtain:

$$\mathbb{E}[|\Phi_k^T|^2] \le \left(\sum_{\ell=1}^{s}\mathbb{E}[Y_\ell]\right)^2 + \sum_{\ell=1}^{s}\sum_{\ell'\ne\ell}\mathbb{E}[Y_\ell]\,\mathbb{E}[Y_{\ell'}] \le 2\left(\sum_{\ell=1}^{s}\mathbb{E}[Y_\ell]\right)^2 = 2\left(\sum_{\ell=1}^{s} R_{i_\ell,j_\ell}^G\right)^2 ,$$

as claimed. □

From theorem 55 we have that if we execute Matrix Winnow on B with then the number M^W of mistakes satisfies

$$M^{\mathrm{W}} = \mathcal{O}\left((R^B)\log(n)\sum_{k=1}^{k}|\Phi_k^B|\right) .$$

Since B is a tree, its resistance diameter, R^B, is equal to its diameter, which is $\mathcal{O}(\log n)$. Moreover, $|\Phi_k^B| = \mathcal{O}(|\Phi_k^T|\log n)$ (see [30] for proof), for $k = 1, \dots, K$, hence $|\Phi_k^B| = \mathcal{O}(|\Phi_k^T|\log n)$. Plugging back, taking expectation over T, and using Lemma 64, 1) proves the Matrix Winnow bound. Similarly, if we run the Matrix Perceptron algorithm on B then

$$M^{\mathrm{P}} = \mathcal{O}\left((R^B)^2\sum_{k=1}^{k}|\Phi_k^B|^2\right) .$$

Proceeding as before, in combination with Lemma 64, 2), proves the Matrix Perceptron bound.

3.6 Fast Perceptron

In this section we give an implementation of the matrix perceptron algorithm on the BST B that requires a per-trial time of $\mathcal{O}(\log^2(n))$, an initialisation time of $\mathcal{O}(n)$, a space requirement of $\mathcal{O}(n^2)$ and a consecutive space requirement of $\mathcal{O}(1)$.

3.6.1 The Algorithm

The algorithm operates on the BST B by maintaining a $(2n-1)\times(2n-1)$ symmetric matrix F with integer entries initially set to zero. At time t, when receiving the pair of leaves (i_t, j_t), the algorithm constructs $\mathcal{P}_t$, the (unique) path in B connecting i_t to j_t. Then the prediction $\hat{z}_t \in \{0,1\}$ is computed as

$$\hat{z}_t = \begin{cases} 1 & \text{if } \sum_{\ell,\ell'\in\mathcal{P}_t} F_{\ell,\ell'} \geq 4\log^2 n, \\ 0 & \text{otherwise .} \end{cases} \tag{3.17}$$

Upon receiving label z_t, the algorithm updates F as follows. First of all, the algorithm is mistake driven, so an update takes place only if $z_t \neq \hat{z}_t$. Let $\mathcal{N}_t$ be the set of neighbors of the vertices in $\mathcal{P}_t$, and define $\mathcal{S}_t := \mathcal{N}_t \setminus (\mathcal{P}_t \setminus \{i_t, j_t\})$. We assign integer tags $f_t(\ell)$ to vertices $\ell \in \mathcal{N}_t$ as follows:

1. For all $\ell \in \mathcal{P}_t$, if ℓ is the s-th vertex in $\mathcal{P}_t$ then we set $f_t(\ell) = s$;
2. For all $\ell \in \mathcal{N}_t \setminus \mathcal{P}_t$, let n_ℓ be the (unique) neighbor of ℓ that is contained in $\mathcal{P}_t$. Then we set $f_t(\ell) = f_t(n_\ell)$.

We then update F on each pair $(\ell, \ell') \in \mathcal{S}_t^2$ as

$$F_{\ell,\ell'} \leftarrow F_{\ell,\ell'} + (2z_t - 1)\left(f_t(\ell) - f_t(\ell')\right)^2 . \tag{3.18}$$

Figure 3.2 illustrates the process.

The fact that the algorithm is $\mathcal{O}(\log^2 n)$ per round easily follows from the fact that, since B is a balanced binary tree, the sizes of sets $\mathcal{P}_t$ (prediction step in (3.17)) and $\mathcal{S}_t$ (update step in (3.18)) are both $\mathcal{O}(\log n)$.

3.6.2 Adaptive Representation of F

A naive implementation of F as an array would require a consecutive space of $\Theta(n^2)$ and an initialisation time of $\Theta(n^2)$. Hence, we now outline a method of growing a data structure that stores a representation of F online for which the

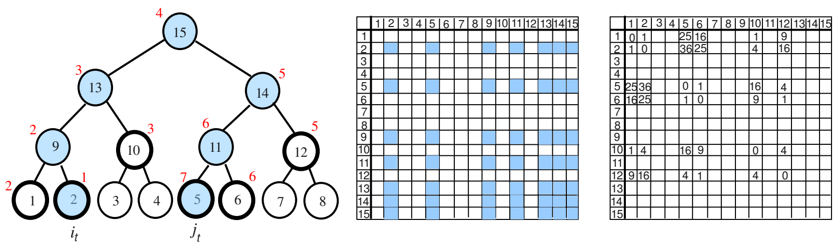


FIGURE 3.2: Matrix Perceptron algorithm at time t with $i_t = 2$ and $j_t = 5$. **Left:** The BST. Light blue vertices are those in $\mathcal{P}_t$. Thick-bordered vertices are those in $\mathcal{S}_t$. The f_t tags are the red numbers near the involved vertices (i.e., those in $\mathcal{P}_t \cup \mathcal{S}_t$). **Middle:** The matrix F. In light blue are the entries of F that are summed over in (3.17). **Right:** The matrix F, where numbers are the values $(f_t(\ell) - f_t(\ell'))^2$ that are added to ($z_t = 1, \hat{z}_t = 0$) or subtracted from ($z_t = 0, \hat{z}_t = 1$) the respective components of F during the update step (3.18).

initialisation time is only $\mathcal{O}(n)$ and has an $\mathcal{O}(1)$ consecutive space requirement while keeping the per round time to $\mathcal{O}(\log^2 n)$.

For every vertex ℓ in B the algorithm maintains a subtree B_ℓ of B, initially set to $\{\rho\}$, being ρ the root of B. At every vertex $\ell' \in B_\ell$ is stored the value $F_{\ell,\ell'}$. At the start of time t, the algorithm climbs B from i_t to ρ, in doing so storing the ordered list $\mathcal{L}_{i_t}$ of vertices in the path from ρ to i_t. The same is done with j_t. The set $\mathcal{S}_t$ is then computed. For all $\ell \in \mathcal{S}_t$, the tree B_ℓ is then extended to include the vertices in $\mathcal{N}_t$ and the path from i_t (note that for each $\ell \in \mathcal{S}_t$ this takes only $\mathcal{O}(\log n)$ time, since we have the list $\mathcal{L}_{i_t}$). Whenever a new vertex ℓ' is added to B_ℓ, the value $F_{\ell,\ell'}$ is set to zero. Hence, we initialize F "on demand", the only initialization step being the allocation of B, i.e., $\mathcal{O}(n)$ time.

3.6.3 Proof of Equivalence

We now show the equivalence of the sequence of predictions issued by (3.17) to those of the Matrix Perceptron algorithm run on B. In this section we let L be the Laplacian of B (instead of G)

Definition 65. *For every $\ell \in \mathcal{S}_t$ define $\Lambda_t(\ell)$ as the maximal subtree of B that contains ℓ and does not contain any nodes in $\mathcal{P}_t \setminus \{i_t, j_t\}$.*

Lemma 66. *$\Lambda_t(\cdot)$ defined above enjoys the following properties (see Figure 3.3, left, for reference).*

1. *For all ℓ, $\Lambda_t(\ell)$ is uniquely defined;*
2. *Any subtree T of B that has no vertices from $\mathcal{P}_t \setminus \{i_t, j_t\}$ (and hence any of the trees Λ_t) contains at most one vertex from $\mathcal{S}_t$;*
3. *The subtrees $\{\Lambda_t(\ell) : \ell \in \mathcal{S}_t\}$ are pairwise disjoint;*
4. *The set $\{\Lambda_t(\ell) : \ell \in \mathcal{S}_t\} \cup (\mathcal{P}_t \setminus \{i_t, j_t\})$ covers B (so in particular $\{\Lambda_t(\ell) : \ell \in \mathcal{S}_t\}$ covers the set of leaves of B).*

Proof. 1. Suppose we have subtrees T and T' with $T \neq T'$ that both satisfy the conditions of $\Lambda_t(\ell)$. Then w.l.o.g assume there exists a vertex ℓ' in T that is not in T'. Since T and T' are both connected and both contain ℓ, the subgraph $T \cup T'$ of B is connected and is hence a subtree. Since neither T nor T' contains vertices in $\mathcal{P}_t \setminus \{i_t, j_t\}$, $T \cup T'$ does not contain any such either. Hence, because T' is a strict subtree of $T \cup T'$, we have contradicted the maximality of T'.

2. Suppose T has distinct vertices $\ell, \ell' \in \mathcal{S}_t$. Since T is connected, it must contain the path in B from ℓ to ℓ'. This path goes from ℓ to the neighbor of ℓ that is in $\mathcal{P}_t \setminus \{i_t, j_t\}$, then follows the path $\mathcal{P}_t \setminus \{i_t, j_t\}$ (in the right direction) until a neighbor of ℓ' is reached. The path then terminates at ℓ'. Such a path contains at least one vertex in $\mathcal{P}_t \setminus \{i_t, j_t\}$, contradicting the initial assumption about T.

3. Assume the converse – that there exist distinct ℓ, ℓ' in $\mathcal{S}_t$ such that $\Lambda_t(\ell)$ and $\Lambda_t(\ell')$ share vertices. Then, since $\Lambda_t(\ell)$ and $\Lambda_t(\ell')$ are connected, $\Lambda_t(\ell) \cup \Lambda_t(\ell')$ must also be connected (and hence must be a subtree of B). Since $\Lambda_t(\ell) \cup \Lambda_t(\ell')$ shares no vertices with $\mathcal{P}_t \setminus \{i_t, j_t\}$, and contains both ℓ and ℓ' (which are both in $\mathcal{S}_t$), the statement in Item 2 above is contradicted.

4. Assume that we have a $\ell \in B \setminus (\mathcal{P}_t \setminus \{i_t, j_t\})$. Then let P' be the path from ℓ to the (first vertex encountered in) the path $\mathcal{P}_t \setminus \{i_t, j_t\}$. Let ℓ' be the second from last vertex in P'. Then ℓ' is a neighbor of a vertex in $\mathcal{P}_t$, but is not in $\mathcal{P}_t \setminus \{i_t, j_t\}$, so it must be in $\mathcal{S}_t$. This implies that the path P'' that goes from ℓ to ℓ' contains no vertices in $\mathcal{P}_t \setminus \{i_t, j_t\}$ and is therefore (Item 1) a subtree of $\Lambda_t(\ell')$. Hence, $\ell \in \Lambda_t(\ell')$.

□

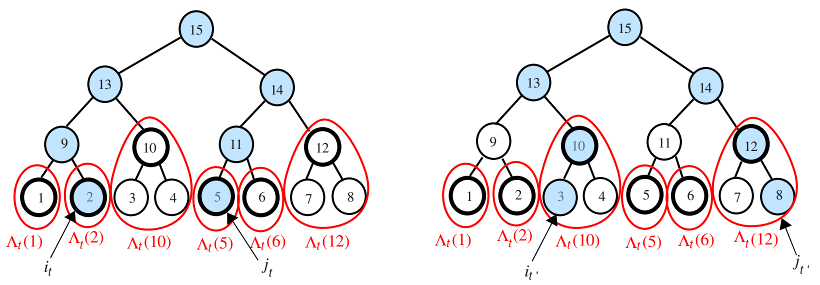


FIGURE 3.3: **Left:** The same BST as in Figure 3.2 with $i_t = 2$ and $j_t = 5$. Light blue vertices are those in $\mathcal{P}_t$. Thick-bordered vertices are those in $\mathcal{S}_t$. Since vertices 3 and 4 are in $\Lambda_t(10)$, we have $\tilde{f}_t(3) = \tilde{f}_t(4) = f_t(10)$. Since vertices 7 and 8 are in $\Lambda_t(12)$, we have $\tilde{f}_t(7) = \tilde{f}_t(8) = f_t(12)$. For all other vertices ℓ, we have $\tilde{f}_t(\ell) = f_t(\ell)$. **Right:** The same BST as in Figure 3.2 with path $\mathcal{P}_{t'}$ (light blue vertices) having endpoints $i_{t'} = 3$ and $j_{t'} = 8$. Thick-bordered vertices are still those in $\mathcal{S}_t$. Path $\mathcal{P}_{t'}$ intersects $\mathcal{S}_t$ at two vertices, 10 and 12, which means that $3 \in \Lambda_t(10)$ and $8 \in \Lambda_t(12)$. We have $\tilde{f}_t(3) = f_t(10)$, $\tilde{f}_t(8) = f_t(12)$, and $((\boldsymbol{e}_3 - \boldsymbol{e}_8)L^+(\boldsymbol{e}_2 - \boldsymbol{e}_5))^2 = (\tilde{f}_t(3) - \tilde{f}_t(8))^2 = (f_t(10) - f_t(12))^2$.

Definition 67. *We extend the tagging function f_t to all vertices of B via the vector*[1] *$\tilde{f}_t$ as follows (note that, by Lemma 66, $\tilde{f}_t$ is well defined):*

1. *For all $\ell \in \mathcal{P}_t \setminus \{i_t, j_t\}$, set $\tilde{f}_t(\ell) = f_t(\ell)$;*

2. *For all $\ell' \in \mathcal{S}_t$ and $\ell \in \Lambda_t(\ell')$, set $\tilde{f}_t(\ell) = f_t(\ell')$.*

Lemma 68. $L\tilde{f}_t = \boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}$.

Proof. For any vertex κ of $B \setminus \{i_t, j_t\}$ one of the following holds:

[1] In our notation, we interchangeably view $\tilde{f}$ both as a tagging function from the $2n - 1$ vertices of B to the natural numbers and as a $(2n - 1)$-dimensional vector.

1. If $\kappa \in \mathcal{P}_t$, then κ has a neighbor κ_1 with $\tilde{f}_t(\kappa_1) = \tilde{f}_t(\kappa) - 1$, one neighbour κ_2 with $\tilde{f}_t(\kappa_2) = \tilde{f}_t(\kappa) + 1$, and (unless κ is the root of B) one neighbour κ_3 with $\tilde{f}_t(\kappa_3) = \tilde{f}_t(\kappa)$. We therefore have that $[L\tilde{f}_t]_\kappa = 3\tilde{f}_t(\kappa) - \tilde{f}_t(\kappa_1) - \tilde{f}_t(\kappa_2) - \tilde{f}_t(\kappa_3) = 0$.
2. If $\kappa \in \mathcal{N}_t \setminus \mathcal{P}_t$, then κ has one neighbor κ_1 in $\mathcal{P}_t$ and we have $\tilde{f}_t(\kappa_1) = \tilde{f}_t(\kappa)$. Let T_κ be the subtree of B containing exactly vertex κ and all neighbors of κ bar κ_1. Since $\mathcal{P}_t$ is connected, it contains κ_1 and does not contain κ, none of the other neighbors of κ being in $\mathcal{P}_t$. Hence T_κ is a subtree of B that contains κ and no vertices from $\mathcal{P}_t \setminus \{i_t, j_t\}$, and so by Lemma 66, item 1 it must be a subtree of $\Lambda_t(\kappa)$. Hence, by definition of $\tilde{f}_t$, all vertices κ_2 in T_κ satisfy $\tilde{f}_t(\kappa_2) = \tilde{f}_t(\kappa)$. This implies that for all neighbors κ_3 of κ we have $\tilde{f}_t(\kappa_3) = \tilde{f}_t(\kappa)$, which in turn gives $[L\tilde{f}_t]_k = 0$.
3. If $\kappa \notin \mathcal{N}_t$ then, by Lemma 66 item 4, let κ be contained in $\Lambda_t(\ell)$ for some $\ell \in \mathcal{S}_t$. Let T_κ be the subtree of B containing exactly vertex κ and all neighbors of κ. Note that T_κ is a subtree of B that contains κ and no vertices from $\mathcal{P}_t \setminus \{i_t, j_t\}$. Since $\Lambda_t(\ell)$ also contains κ (hence $\Lambda_t(\ell) \cup T_\kappa$ is connected), we have that $\Lambda_t(\ell) \cup T_\kappa$ is a subtree of B that contains ℓ and no vertices from $\mathcal{P} \setminus \{i_t, j_t\}$. By Lemma 66 item 1, this implies that $\Lambda_t(\ell) \cup T_\kappa$ is a subtree of (and hence equal to) $\Lambda_t(\ell)$. Hence, by definition of $\tilde{f}_t$, we have that $\tilde{f}_t$ is identical on T_κ. Thus all neighbors κ_1 of κ satisfy $\tilde{f}_t(\kappa_1) = \tilde{f}_t(\kappa)$, implying again $[L\tilde{f}_t]_\kappa = 0$.

So in either case $[L\tilde{f}_t]_\kappa = 0$.

Finally, let i'_t be the neighbor of i_t in B. We have $[L\tilde{f}_t]_{i_t} = \tilde{f}_t(i_t) - \tilde{f}_t(i'_t) = 1 - 2 = -1$. Similarly, we have $[L\tilde{f}_t]_{j_t} = 1$. Putting together, we have shown that $L\tilde{f}_t = \boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}$, thereby concluding the proof. □

Lemma 69. *Suppose we have vertices $\ell, \ell' \in \mathcal{S}_t$. Then for any pair of vertices κ and κ' of B with $\kappa \in \Lambda_t(\ell)$ and $\kappa' \in \Lambda_t(\ell')$ we have*

$$(\boldsymbol{e}_\kappa - \boldsymbol{e}_{\kappa'})^T L^+ (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}) = f_t(\ell') - f_t(\ell) ,$$

where $\boldsymbol{e}_i$ is the i-th element in the canonical basis of $\mathbb{R}^{2n-1}$.

Proof. By lemma 68 and the definition of pseudoinverse,

$$L\tilde{f}_t = LL^+L\tilde{f}_t = LL^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}) .$$

This mplies that $L(\tilde{f}_t - L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})) = 0$. Hence, there exists $c \in \mathbb{R}$ such that $\tilde{f}_t - L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}) = c\mathbf{1}$ (noting that multiples of $\mathbf{1}$ are the only solutions to the equation $Lx = 0$) so there exists a constant c such that $\tilde{f}_t = L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}) + c\mathbf{1}$. From the definition of $\tilde{f}$ we can hence write

$$\begin{aligned} f_t(\ell') - f_t(\ell) &= \tilde{f}_t(\kappa') - \tilde{f}_t(\kappa) \\ &= ([L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})]_{\kappa'} - c) - ([L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})]_\kappa - c) \\ &= [L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})]_{\kappa'} - [L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})]_\kappa \\ &= (\boldsymbol{e}_\kappa - \boldsymbol{e}_{\kappa'})^T L^+(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}), \end{aligned}$$

as claimed. □

Lemma 70. *Let κ, κ' be two vertices of B. Let $\mathcal{P}$ be the path from κ to κ' in B. Then for any t either $|\mathcal{P} \cap \mathcal{S}_t| \le 1$ or $\mathcal{P} \cap \mathcal{S}_t = \{\ell, \ell'\}$, for two distinct vertices ℓ and ℓ'. No other cases are possible. Moreover,*

$$((\boldsymbol{e}_\kappa - \boldsymbol{e}_{\kappa'})^T L^+(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}))^2 = \begin{cases} 0 & \text{if } |\mathcal{P} \cap \mathcal{S}_t| \le 1 \\ (f_t(\ell) - f_t(\ell'))^2 & \text{if } \mathcal{P} \cap \mathcal{S}_t = \{\ell, \ell'\} . \end{cases}$$

Proof. By Lemma 66 item 4, we have two possible cases only:

1. There exists $\ell \in \mathcal{S}_t$ such that both κ and κ' are in $\Lambda_t(\ell)$: In this case (since $\Lambda_t(\ell)$ is connected) the path $\mathcal{P}$ lies in $\Lambda_t(\ell)$. Since, by Lemma 66 item 2, no $\ell' \in \mathcal{S}_t$ with $\ell' \neq \ell$ can be in $\Lambda_t(\ell)$, it is only ever possible that $\mathcal{P}$ contains at most one vertex ℓ (if any) of $\mathcal{S}_t$.
2. There exist two distinct nodes $\ell, \ell' \in \mathcal{S}_t$ such that $\kappa \in \Lambda(\ell)$ and $\kappa' \in \Lambda(\ell')$. In this case, $\mathcal{P}$ corresponds to the following path: First go from κ to ℓ (by Lemma 66 item 2, since this path lies in $\Lambda(\ell)$ the only vertex in $\mathcal{S}_t$ that lies in the section of the path is ℓ); then go to the neighbor of ℓ that is in $\mathcal{P}_t \setminus \{i_t, j_t\}$; then follow the path $\mathcal{P}_t \setminus \{i_t, j_t\}$ until you reach the neighbor of ℓ' (this section of $\mathcal{P}$ contains no vertices in $\mathcal{S}_t$); then go from ℓ' to κ (by Lemma 66 item 2, since this path lies in $\Lambda(\ell')$ the only vertex in $\mathcal{S}_t$ that lies in this section of the path is ℓ'). Thus, $\mathcal{P} \cap \mathcal{S}_t = \{\ell, \ell'\}$.

The result then follows by applying Lemma 69 to the two cases above. $\square$

Figure 3.3 illustrates the above lemmas by means of an example.

To conclude the proof, let $\langle A, B\rangle$ be a shorthand for $\text{TR}(A^\top B)$. We see that from Algorithm 1, Lemma 70, and the definition of F in (3.18) we can write

$$\begin{aligned}
\langle W_t^{\text{P}}, X_t^{\text{P}}\rangle &= \langle \left(\sum_{s=1, s\in\mathcal{M}}^{t-1} (2z_s - 1) X_s^{\text{P}} \right), X_t^{\text{P}}\rangle \\
&= \sum_{s=1, s\in\mathcal{M}}^{t-1} (2z_s - 1)\langle X_s^{\text{P}}, X_t^{\text{P}}\rangle \\
&= \sum_{s=1, s\in\mathcal{M}}^{t-1} (2z_s - 1)((\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T L^+ (\boldsymbol{e}_{i_s} - \boldsymbol{e}_{j_s}))^2 \\
&= \sum_{(\ell,\ell')\in\mathcal{P}_t^2} F_{\ell,\ell'} \ ,
\end{aligned}$$

where $\mathcal{M}$ is the set of mistaken rounds, and the second-last equality follows from a similar argument as the one contained in the proof of lemma 58. Note also that since B is a tree, its resistance diameter is equal to its diameter and since B is balanced this is equal to $2\log(n)$. Hence we have that $(R^B)^2 = 4\log^2(n)$.

Plugging these identities into the definition of the matrix perceptron algorithm (Algorithm 1) gives us the equivalence of the algorithms.

Chapter 4

A Time and Space Efficient Junction Tree Architecture

4.1 Abstract

The junction tree algorithm is a way of computing all marginals of boolean multivariate probability distributions that factorise over sets of random variables. The junction tree algorithm first constructs a tree called a junction tree who's vertices are sets of random variables. The algorithm then performs a generalised version of belief propagation on the junction tree. The Shafer-Shenoy and Hugin architectures are two ways to perform this belief propagation that tradeoff time and space complexities in different ways: Hugin propagation is at least as fast as Shafer-Shenoy propagation and in the cases that we have large vertices of high degree is significantly faster. However, this speed increase comes at the cost of an increased space complexity. This chapter first introduces a simple novel architecture, ARCH-1, which has the best of both worlds: the speed of Hugin propagation and the low space requirements of Shafer-Shenoy propagation. A more complicated novel architecture, ARCH-2, is then introduced which has, up to a factor only linear in the maximum cardinality of any vertex, time and space complexities at least as good as ARCH-1 and in the cases that we have large vertices of high degree is significantly faster than ARCH-1.
This chapter is all my own work.

4.2 Introduction

The junction tree algorithm is a popular tool for the simultaneous computation of all marginals of a multivariate probability distribution stored in a factored form. In this paper we consider the case in which the random variables are boolean. The junction tree algorithm is a generalisation of belief propagation [44] performed on a tree (called a junction tree) who's vertices are sets of random variables. The Shafer-Shenoy [50] [43] and Hugin [23] [43] architectures are two variations of the junction tree algorithm that trade off time and space complexities in different ways: Hugin propagation is faster than Shafer-Shenoy propagation but at the cost of a greater space complexity. Large vertices of high degree cause much inefficiency in both these architectures (especially in that of Shafer-Shenoy) and it is the purpose of this paper to introduce novel architectures that perform better in these cases. In order to tackle the problem of high-degree vertices, an algorithm was given in [52] for constructing a binary junction tree on which Shafer-Shenoy propagation can then be performed. This method was shown empirically to be faster than Hugin propagation (on a generic junction tree constructed in a certain way) in [36]. The drawback of this method, however, is that it can require dramatically more space than Shafer-Sheony propagation on a generic junction tree due to the maximum cardinality of the intersection of two neighbouring vertices being large. It should also be noted that in [62] an architecture was given that eliminated the redundant computations (caused by high-degree vertices) of Shafer-Shenoy propagation. Again though, this architecture can have a dramatically increased space complexity over that of Shafer-Shenoy propagation. In comparison, the architectures introduced in

this paper tackle the problem of high degree vertices whilst retaining the low space complexity of Shafer-Shenoy propagation (on a generic junction tree). Two novel architectures are introduced in this paper: the first, ARCH-1, achieves the speed (up to a constant factor) of Hugin propagation and has the low space requirements of Shafer-Shenoy propagation. ARCH-1 is very simple and serves as a warm up to a more complicated architecture, ARCH-2, which (almost) has space and time complexities at least as good as ARCH-1 and in the cases in which we have large vertices of high degree is significantly faster than ARCH-1/Hugin. In the cases in which we have a large enough (relative to the rest of the junction tree) vertex who's degree is exponential (to some base greater than one) in its cardinality then ARCH-2 has a polynomial saving in the time complexity over that of ARCH-1/Hugin: i.e. there exists $s < 1$ such that a time of $\Theta(t)$ (for ARCH-1/Hugin) becomes a time of $\mathcal{O}(t^s)$ (for ARCH-2). The saving in time complexity in going from Shafer-Shenoy to Hugin/ARCH-1 is similar.
A more detailed description of the results of this paper is given in Section 4.4 after the preliminary definitions have been introduced and the junction tree algorithm has been described.

In this paper we assume that all basic operations such as arithmetic operations and memory reads/writes take constant time. We also assume that storing any number or pointer takes constant space. To ease the reader's understanding, the algorithms given in sections 4.5 though 4.7 are sketches: to achieve the stated time complexities we must be able to find and store variables in constant amortised time and space. The exact implementations that give the stated time and space complexities are given in Section 4.9.

This paper is structured as follows: In Section 4.3 we give the preliminary definitions required by the paper. In Section 4.4 we give an overview of the junction tree algorithm, a detailed overview of the results of the paper and some technicalities relating to time/space complexities. In Section 4.5 we describe the Shafer-Shenoy and Hugin architectures and analyse their complexities. In Section 4.6 we describe the architecture ARCH-1 and analyse its complexity. In Section 4.7 we describe the architecture ARCH-2 and analyse its complexity. In Section 4.8 we describe how to modify ARCH-2 such that it can deal with zeros. In Section 4.9 we give the details of how to implement the algorithms introduced in this paper.

4.3 Preliminaries

In this section we define the notation and concepts used in this paper except for that required exclusively for the implementation details of the algorithms which are defined in Section 4.9. Also, the notation $\mathcal{J}$, S, $\mathcal{F}(C)$ and $M_{H\to C}$, as well as the notion of "sending" and "receiving" messages, is defined in Algorithm 71.

4.3.1 Basic Notation

The symbol := is used for definition: e.g. $x := y$ means "x is defined to be equal to y". Given $a \in \mathbb{N}$ we define $\mathbb{N}_a$ to be equal to the set of the first a natural numbers: i.e. the set $\{1, 2, 3, ..., (a-1), a\}$. Given a set X we define $\mathcal{P}(X)$ to be the *power-set* of X: that is, the set of all subsets of X. In this paper, whenever we use the word "set" we mean a finite set. If we are talking about an infinite set we shall say "(infinite) set".

We now define the pseudo-code used in this paper: The left arrow, $\leftarrow$, denotes assignment: e.g. $a \leftarrow b$ indicates that the value b is computed and then assigned to the variable a. Function names are written in bold with the input coming in brackets after the name. When the assignment symbol, $\leftarrow$, has a function on its right hand side it indicates that the function is run and the output of the function is assigned to the variable on the left hand side: e.g. $a \leftarrow$ **function**(b) indicates that the function **function** is run with input b and its output is assigned to variable a. When the word "return" appears in the pseudo-code for a function it indicates that the function terminates and outputs the object coming after the word "return".

4.3.2 Potentials

A *binary labelling* of a set X is a map from X into $\{0,1\}$. A *potential* on a set X is a map from $\mathcal{P}(X)$ into $\mathbb{R}^+$. Given a potential Ψ on a set X we define $\sigma(\Psi) := X$. Given a set X we define $\mathcal{T}(X)$ to be the (infinite) set of all possible potentials on X. Given a set X we define $\mathbf{1}_X$ to be the potential in $\mathcal{T}(X)$ that satisfies $\mathbf{1}_X(Z) := 1$ for all $Z \in \mathcal{P}(X)$. Note that a potential on a set X is equivalent to a map from all possible binary labellings of X into the positive reals (which is the usual definition of a potential). The equivalence is seen by noting that there is a bijective mapping from $\mathcal{P}(X)$ into the set of all possible binary labellings of X where a subset Y of X maps to the labelling μ_Y of X given by $\mu_Y(x) := 1$ for all $x \in Y$ and $\mu_Y(x) := 0$ for all $x \in X \setminus Y$. The operations in this paper are easier to describe when the domain of a potential is a power-set, which is why we define potentials in this way.

Given a potential Ψ on a set X and a subset $Y \subseteq X$ we define the *Y-marginal*, $\Psi^{\nabla Y}$, of Ψ as the potential in $\mathcal{T}(Y)$ that satisfies, for all $Z \in \mathcal{P}(Y)$:

$$\Psi^{\nabla Y}(Z) := \sum_{U \in \mathcal{P}(X): U \cap Y = Z} \Psi(U) \tag{4.1}$$

Note that, by above, Ψ may be equivalent to a probability distribution on binary labellings of X. If this is the case then $\Psi^{\nabla Y}$ is equivalent to the marginalisation of that probability distribution onto Y.

Given sets X and Y and potentials $\Psi \in \mathcal{T}(X)$ and $\Phi \in \mathcal{T}(Y)$ we define the *product*, $\Psi\Phi$, of Ψ and Φ as the potential in $\mathcal{T}(X \cup Y)$ that satisfies, for all $Z \in \mathcal{P}(X \cup Y)$:

$$[\Psi\Phi](Z) := \Psi(Z \cap X)\Phi(Z \cap Y) \tag{4.2}$$

We represent the product of multiple potentials by the $\prod$ symbol, as in the multiplication of numbers.

Given a set X and potentials $\Psi, \Phi \in \mathcal{T}(X)$ we define the *quotient*, Ψ/Φ, of Ψ and Φ as the potential in $\mathcal{T}(X)$ that satisfies, for all $Z \in \mathcal{P}(X)$:

$$[\Psi/\Phi](Z) := \Psi(Z)/\Phi(Z) \tag{4.3}$$

The reason for the low space complexity of ARCH-2 is that, given a set X and a potential $\Phi \in \mathcal{T}(X)$, we may not need to store the value of $\Phi(Y)$ for every $Y \in \mathcal{P}(X)$. This encourages the following definitions: A set ζ of sets is a *straddle-set* if and only if, for every $Z \in \zeta$ and every $Y \in \mathcal{P}(Z)$ we have $Y \in \zeta$. Given a potential Φ and a straddle-set $\zeta \subseteq \mathcal{P}(\sigma(\Phi))$, the *restriction*, $\Phi^{\oplus\zeta}$, is the data-structure that stores the value $\Phi(Y)$ if and only if $Y \in \zeta$. Note that storing a restriction $\Phi^{\oplus\zeta}$ requires a space of only $\Theta(|\zeta|)$.

ARCH-2 works with the notions of the *p-dual*, $\#\Psi$, and the *m-dual*, $\%\Psi$, of a potential Ψ. These are defined in sections 4.7.1 and 4.7.2 respectively.

4.3.3 Factorisations

Suppose we have a probability distribution $\mathbb{P}$ on the set of binary labellings of a set S. Then a set, $\mathcal{F}$, of potentials is a *factorisation* of $\mathbb{P}$ if and only if $\bigcup_{\Lambda \in \mathcal{F}} \sigma(\Lambda) = S$ and for every binary labelling, μ of S we have:

$$\mathbb{P}(\mu) \propto \left[\prod_{\Lambda \in \mathcal{F}} \Lambda\right](\{x \in S : \mu(x) = 1\}) \tag{4.4}$$

If $\mathcal{F}$ is a factorisation of a probability distribution $\mathbb{P}$ then we refer to the potentials in $\mathcal{F}$ as *factors*.

4.3.4 Junction Trees

Given a tree $\mathcal{J}$ we define $\mathcal{V}(\mathcal{J})$ and $\mathcal{E}(\mathcal{J})$ to be the vertex and edge set of $\mathcal{J}$ respectively. Also, given a tree $\mathcal{J}$ and a vertex $C \in \mathcal{V}(\mathcal{J})$ we define $\deg(C)$ and

$\mathcal{N}(C)$ to be the degree (i.e. number of neighbours) and neighbourhood (i.e. set of neighbours) of C in $\mathcal{J}$ respectively. When a tree $\mathcal{J}$ is rooted we define, for a vertex $C \in \mathcal{V}(\mathcal{J})$, $\uparrow(C)$ and $\downarrow(C)$ to be the parent of C and the set of children of C respectively.

A *junction tree*, $\mathcal{J}$, on a set S is a tree satisfying the following axioms:

- Every vertex of $\mathcal{J}$ is a subset of S.
- $\bigcup \mathcal{V}(\mathcal{J}) = S$
- Given $C, H \in \mathcal{V}(\mathcal{J})$ and some $x \in S$ such that $x \in C \cap H$ then x is a member of every vertex in the path (in $\mathcal{J}$) from C to H.

The *width* of a junction tree is defined as the cardinality of its largest vertex.

4.4 The Junction Tree Algorithm

The goal of this paper is as follows: We have a probability distribution $\mathbb{P}$ on binary labellings, μ, of a set S and a factorisation, $\mathcal{F}$, of $\mathbb{P}$. We wish to compute the marginal probability $\mathbb{P}(\mu(x) = 1)$ for every $x \in S$.

Note that the marginal $\mathbb{P}(\mu(x) = 1)$ is equivalent to a potential $\rho_x \in \mathcal{T}(\{x\})$ defined as $\rho_x(\emptyset) := \mathbb{P}(\mu(x) = 0)$ and $\rho_x(\{x\}) := \mathbb{P}(\mu(x) = 1)$. The *junction tree algorithm* is a way of simultaneously computing the potentials ρ_x for every $x \in S$. The algorithm has three stages: The *junction tree construction stage*, the *message passing stage* and the *computation of marginals stage*:

Algorithm 71. The Junction Tree Algorithm:

1. *Junction tree construction stage:*
   A junction tree $\mathcal{J}$ on S is constructed such that for all $\Lambda \in \mathcal{F}$ we have a vertex $\Lambda^+ \in \mathcal{V}(\mathcal{J})$ for which $\sigma(\Lambda) \subseteq \Lambda^+$. For every $C \in \mathcal{V}(\mathcal{J})$ we define $\mathcal{F}(C) := \{\Lambda \in \mathcal{F} : \Lambda^+ = C\}$.
2. *Message passing stage:*
   For every ordered pair (C, E) of neighbouring vertices of $\mathcal{J}$, we create and store a message $M_{C\to E}$ which is a potential in $\mathcal{T}(C \cap E)$. When such a message is created we say that C "sends" the message and E "receives" the message. The messages are defined recursively by the following equation:

$$M_{C\to E} := \left[\left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\mathbf{1}_{C\cap E} \prod_{H\in\mathcal{N}(C)\setminus\{E\}} M_{H\to C}\right)\right]^{\nabla C\cap E} \tag{4.5}$$

3. *Computation of marginals stage:*
   For every $x \in S$ we compute the potential ρ_x from the messages. Specifically, for any vertex $C \in \mathcal{V}(\mathcal{J})$ with $x \in C$ we have:

$$\rho_x = \left[\left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\prod_{H\in\mathcal{N}(C)} M_{H\to C}\right)\right]^{\nabla\{x\}} \tag{4.6}$$

In the rest of the paper the symbols $\mathcal{J}$, S, $\mathcal{F}(C)$ and $M_{H\to C}$, as well as the notion of "sending" and "receiving" messages, are always defined as above. In this paper we consider, in detail, the message passing stage of the junction tree algorithm We first review the Shafer-Shenoy and Hugin architectures that differ in how the messages are computed. With Shafer-Shenoy propagation each vertex C contributes a time of $\Theta\left(\deg(C)(\deg(C) + |\mathcal{F}(C)|)2^{|C|}\right)$ to the message passing stage whilst with Hugin propagation each vertex C contributes a time of $\Theta\left((\deg(C) + |\mathcal{F}(C)|)2^{|C|}\right)$ to the message passing stage. When we have large vertices of high degree Hugin propagation is hence significantly faster than Shafer-Shenoy propagation. However, this speed increase comes at a cost of a higher space complexity: whilst the space complexity of Shafer-Shenoy propagation is only that required to store the factors and messages, the Hugin architecture must store, for every vertex C, a potential $\Gamma_C \in \mathcal{T}(C)$; leading to a space requirement of $\Omega\left(\sum_{C\in\mathcal{V}(\mathcal{J})} 2^{|C|}\right)$.

We then describe, from a merger of the ideas behind Shafer-Shenoy and Hugin propagation, a simple, novel architecture ARCH-1 which has (up to a constant factor) the best of both worlds: the speed of Hugin propagation and the low space complexity of Shafer-Shenoy propagation.

The main idea behind ARCH-1, that of simultaneously computing many marginals of a factored potential, then leads us into the novel architecture ARCH-2 which has (up to a factor linear in the width of the junction tree) at least the time and space efficiency of ARCH-1 and is considerably faster when we have large vertices of high degree. Specifically, each vertex C now contributes a time of only $\mathcal{O}\left(|C|2^{|C|}\right)$ to the message passing stage and, in addition to storing the factors and messages, ARCH-2 requires a space of only
$\mathcal{O}\left(\max_{C\in\mathcal{V}(\mathcal{J})}|C|\left(\left(\sum_{H\in\mathcal{N}(C)}2^{|H\cap C|}\right)+\left(\sum_{\Lambda\in\mathcal{F}(C)}2^{|\sigma(\Lambda)|}\right)\right)\right)$.

We note that although we don't explicitly describe the computation of marginals stage, the ideas behind ARCH-2 can be used to do this stage with time and space no greater than the message passing stage of ARCH-2 . The details are left to the reader.

As stated in the introduction, to ease the reader's understanding, the algorithms given in sections 4.5 though 4.7 are sketches: to achieve the stated time complexities we must be able to find and store variables in constant amortised time and space. The exact implementations that give the stated time and space complexities are given in Section 4.9.

We also note, that the auxiliary space required by the algorithms in this paper is an additive factor of $\mathcal{O}(|S|)$ more than is stated since we must maintain an array of size $|S|$ (see Section 4.9). However, since $\mathcal{O}(|S|)$ is no greater than the space required to store the factors it is fine to neglect this.

4.5 Shafer-Shenoy and Hugin Propagation

4.5.1 Shafer-Shenoy Propagation

In this subsection we describe and analyse the complexity of Shafer-Shenoy propagation. Shafer-Shenoy propagation follows the following algorithm:

Algorithm 72. Outline of Shafer-Shenoy Propagation:
Given an ordered pair (C,E) of neighbouring vertices, once C has received messages from all vertices in $\mathcal{N}(C)\setminus\{E\}$ the message $M_{C\to E}$ is computed as:

$$M_{C\to E} \leftarrow \left[\left(\prod_{\Lambda\in\mathcal{F}(C)}\Lambda\right)\left(\mathbf{1}_{C\cap E}\prod_{H\in\mathcal{N}(C)\setminus\{E\}}M_{H\to C}\right)\right]^{\nabla C\cap E} \tag{4.7}$$

and is sent from C to E.

Note that the creation of a message in the above algorithm is an instance of the following operation (where $\{D_1,D_2,...,D_k\}:=\{\sigma(\Lambda):\Lambda\in\mathcal{F}(C)\}\cup\{C\cap H:H\in\mathcal{N}(C)\}$):

Operation 73. *We have a set C, subsets $\{D_1,D_2,...,D_k\}\subseteq\mathcal{P}(C)$ and a subset $W\subseteq C$. For every $i\in\mathbb{N}_k$ we have a potential $\Upsilon_i\in\mathcal{T}(D_i)$. We must compute $\left(\prod_{i=1}^{k}\Upsilon_i\right)^{\nabla W}$.*

If Operation 73 is performed by firstly computing $\prod_{i=1}^{k}\Upsilon_i$ and then marginalising it onto W it requires an auxiliary space on $\Theta\left(2^{|C|}\right)$ leading to a space requirement of at least $\Omega\left(\max_{H\in\mathcal{V}(\mathcal{J})}2^{|H|}\right)$ for the whole algorithm. Hence, we now give an algorithm that can be implemented to perform Operation 73 in a time of $\Theta\left(k2^{|C|}\right)$ and which uses only constant auxiliary space:

Algorithm 74. *For every $Y \in \mathcal{P}(W)$ we maintain a variable $h(Y) \in \mathbb{R}$, initially set to zero.*

For every $Z \in \mathcal{P}(C)$, in turn, we do the following:

$$h(Z \cap W) \leftarrow h(Z \cap W) + \prod_{i=1}^{k} \Upsilon_i(Z \cap D_i) \tag{4.8}$$

Note that after we have performed the above for every $Z \in \mathcal{P}(C)$, the function h is equal to the potential $\left(\prod_{i=1}^{k} \Upsilon_i\right)^{\nabla W}$. We then output the potential h.

If Algorithm 74 is used for performing Operation 73 then the computation of each message $M_{C \to E}$ takes a time of $\Theta((\deg(C) + |\mathcal{F}(C)|)2^{|C|})$ and requires only constant auxiliary space. Hence, the space complexity of the entire message passing algorithm is the space required to store the factors and messages. Since each vertex C sends $\deg(C)$ messages, each vertex C contributes a time of $\Theta(\deg(C)(\deg(C) + |\mathcal{F}(C)|)2^{|C|})$ to the entire message passing algorithm.

4.5.2 Hugin Propagation

In this subsection we describe and analyse the complexity of Hugin propagation. Hugin propagation stores the following potentials: For every vertex $C \in \mathcal{V}(\mathcal{J})$ we have a potential $\Gamma_C \in \mathcal{T}(C)$ initialised to be equal to $\mathbf{1}_C \prod_{\Lambda \in \mathcal{F}(C)} \Lambda$. For every edge $\{C, E\} \in \mathcal{E}(\mathcal{J})$ we have a potential $\Psi_{\{C,E\}} \in \mathcal{T}(C \cap E)$ initialised equal to $\mathbf{1}_{C \cap E}$. Hugin propagation follows the following algorithm:

Algorithm 75. Hugin Propagtion:
Given an ordered pair (C, E) of neighbouring vertices, once C has received messages from all vertices in $\mathcal{N}(C) \setminus \{E\}$, it sends a message to E via the following algorithm:

1. *Set* $\Psi^{\text{old}}_{\{C,E\}} \leftarrow \Psi_{\{C,E\}}$
2. *Set* $\Psi_{\{C,E\}} \leftarrow {\Gamma_C}^{\nabla C \cap E}$
3. *Set* $M_{C \to E} \leftarrow \Psi_{\{C,E\}} / \Psi^{\text{old}}_{\{C,E\}}$
4. *Set* $\Gamma_E \leftarrow M_{C \to E} \Gamma_E$

Note that the time required by a vertex C to pass a message to a neighbour E is $\Theta(2^{|C|} + 2^{|E|})$. Since each vertex C sends and receives a message to/from each of its neighbours, and since the potential Γ_C takes a time of $\Theta(|\mathcal{F}(C)|2^{|C|})$ to initialise, we have that C contributes a time of $\Theta((\deg(C) + |\mathcal{F}(C)|)2^{|C|})$ to the entire message passing algorithm. Note then that Hugin propagation is faster than Shafer-Shenoy propagation. The drawback, however, is that storing, for each vertex C, the potential Γ_C has a space requirement of $\Theta(2^{|C|})$. This leads to a total space requirement of $\Theta(\sum_{C \in \mathcal{V}(\mathcal{J})} 2^{|C|})$ in addition to that required to store the factors, which can be significantly more (and never less) than that of Shafer-Shenoy propagation.

Given a vertex $C \in \mathcal{V}(\mathcal{J})$, if the potential Γ_C is initialised by combining (via multiplication) the factors in $\mathcal{F}(C)$ on a binary basis (as is described in [49]) then the initialisation time of Γ_C can be less than $\Theta\left(|\mathcal{F}(C)|2^{|C|}\right)$ so the time complexity of Hugin propagation can be reduced. However, each vertex still contributes a time of at least $\Omega\left(\deg(C)2^{|C|}\right)$ so if the degree of a vertex is greater than the number of associated factors then combining factors on a binary basis does not speed up this time by more than a constant factor. In addition, this faster version of Hugin propagation is still never faster than ARCH-2 by more than a logarithmic factor and when we have large vertices of high degree is still significantly slower than ARCH-2.

4.6 ARCH-1

In this section we describe the architecture ARCH-1 which has (up to a constant factor) the speed of Hugin propagation and the low space complexity of Shafer-Shenoy propagation. The reason for the low time/space complexity is that many marginals are computed simultaneously from a factored potential using a merger of the ideas behind Shafer-Shenoy and Hugin propagation: an algorithm similar to Algorithm 74 and the division idea of the Hugin architecture. Like Shafer-Shenoy propagation we store only the messages and factors.

ARCH-1 selects a vertex R as the root of $\mathcal{J}$ and then (as is often in the description of Hugin and Shafer-Shenoy propagation) has two phases: the *inward phase*, in which messages are passed up the tree to the root and the *outward phase*, in which messages are passed down the tree from the root to the leaves. We first sketch an outline of ARCH-1 (which is also an outline of ARCH-2) before going into the details:

Algorithm 76. Outline of ARCH-1/ARCH-2:
The algorithm has two phases: First the inward phase and then the outward phase.

1. *Inward phase: For every vertex $C \in \mathcal{V}(\mathcal{J}) \setminus \{R\}$, once C has received messages from all its children, it sends a message to its parent as follows:*

   (a) *The message $M_{C\to\uparrow(C)}$ is computed as:*

$$M_{C\to\uparrow(C)} \leftarrow \left[\left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\mathbf{1}_{C\cap\uparrow(C)} \prod_{H\in\downarrow(C)} M_{H\to C}\right)\right]^{\nabla C\cap\uparrow(C)} \tag{4.9}$$

   and is sent from C to $\uparrow(C)$.

2. *Outward phase: For every vertex $C \in \mathcal{V}(\mathcal{J})$, once C has received messages from all its neighbours, it sends messages to all its children as follows:*

   (a) *For every $E \in \downarrow(C)$, simultaneously, the potential M'_E (in $\mathcal{T}(C \cap E)$) is computed as:*

$$M'_E \leftarrow \left[\left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\prod_{H\in\mathcal{N}(C)} M_{H\to C}\right)\right]^{\nabla C\cap E} \tag{4.10}$$

   (b) *For every $E \in \downarrow(C)$ the message $M_{C\to E}$ is computed as:*

$$M_{C\to E} \leftarrow M'_E / M_{E\to C} \tag{4.11}$$

   and is sent to E.

We now prove the correctness of Algorithm 76: i.e. that the messages are equal to those defined in Stage 2 of Algorithm 71.

Consider first the inward phase: Since Equation 4.9 is the same as Equation 4.5 we have, by induction up $\mathcal{J}$ from the leaves to the root, that $M_{C\to\uparrow(C)}$ is correctly computed for every $C \in \mathcal{V}(\mathcal{J}) \setminus \{R\}$.

Consider next the outward phase: We prove, by induction on C down $\mathcal{J}$ from the root to the leaves, that $M_{C\to E}$ is correctly computed for all $E \in \downarrow(C)$. By the inductive hypothesis and the result above that $M_{H\to C}$ is correctly computed for every $H \in \downarrow(C)$ we have that $M_{H\to C}$ is correctly computed for every $H \in \mathcal{N}(C)$.

Hence, for all $E \in \downarrow(C)$ and all $Y \in \mathcal{P}(C \cap E)$ we have:

$$\begin{aligned}
M'_E(Y) &= \left[\left(\prod_{\Lambda \in \mathcal{F}(C)} \Lambda\right)\left(\prod_{H \in \mathcal{N}(C)} M_{H \to C}\right)\right]^{\nabla C \cap E}(Y) && (4.12)\\
&= \left[\left(\prod_{\Lambda \in \mathcal{F}(C)} \Lambda\right) \mathbf{1}_{C \cap E}\left(\prod_{H \in \mathcal{N}(C)} M_{H \to C}\right)\right]^{\nabla C \cap E}(Y) && (4.13)\\
&= \sum_{Z \in \mathcal{P}(C): Z \cap C \cap E = Y} \left[\left(\prod_{\Lambda \in \mathcal{F}(C)} \Lambda\right) \mathbf{1}_{C \cap E}\left(\prod_{H \in \mathcal{N}(C)} M_{H \to C}\right)\right](Z) && (4.14)\\
&= \sum_{Z \in \mathcal{P}(C): Z \cap C \cap E = Y} M_{E \to C}(Z \cap C \cap E)\left[\left(\prod_{\Lambda \in \mathcal{F}(C)} \Lambda\right) \mathbf{1}_{C \cap E}\left(\prod_{H \in \mathcal{N}(C) \setminus \{E\}} M_{H \to C}\right)\right](Z) && (4.15)\\
&= \sum_{Z \in \mathcal{P}(C): Z \cap C \cap E = Y} M_{E \to C}(Y)\left[\left(\prod_{\Lambda \in \mathcal{F}(C)} \Lambda\right) \mathbf{1}_{C \cap E}\left(\prod_{H \in \mathcal{N}(C) \setminus \{E\}} M_{H \to C}\right)\right](Z) && (4.16)\\
&= M_{E \to C}(Y) \sum_{Z \in \mathcal{P}(C): Z \cap C \cap E = Y} \left[\left(\prod_{\Lambda \in \mathcal{F}(C)} \Lambda\right) \mathbf{1}_{C \cap E}\left(\prod_{H \in \mathcal{N}(C) \setminus \{E\}} M_{H \to C}\right)\right](Z) && (4.17)\\
&= M_{E \to C}(Y)\left[\left(\prod_{\Lambda \in \mathcal{F}(C)} \Lambda\right)\left(\prod_{H \in \mathcal{N}(C) \setminus \{E\}} M_{H \to C}\right)\right]^{\nabla C \cap E}(Y) && (4.18)\\
&= M_{E \to C}(Y) M_{C \to E}(Y) && (4.19)
\end{aligned}$$

and hence $[M'_E / M_{E \to C}](Y) = M_{C \to E}(Y)$ so $M'_E / M_{E \to C} = M_{C \to E}$ which proves that the inductive hypothesis holds for C.

Note that Step 1a and Step 2a of Algorithm 76 can be solved by instances of the following operation (where $\{D_1, D_2, ..., D_k\} := \{\sigma(\Lambda) : \Lambda \in \mathcal{F}(C)\} \cup \{C \cap H : H \in \mathcal{N}(C)\}$):

Operation 77. *We have, as input, a set C, and subsets $D_1, D_2, ..., D_k \in \mathcal{P}(C)$ with $\bigcup_{i=1}^{k} D_i = C$. For every $i \in \mathbb{N}_k$ we have, as input, a potential $\Upsilon_i \in \mathcal{T}(D_i)$.*
Define $\Gamma := \prod_{i=1}^{k} \Upsilon_i$ and for every $i \in \mathbb{N}_k$ define $\Psi_i := \Gamma^{\nabla D_i}$.
We must compute Ψ_i for every $i \in \mathbb{N}_k$.

ARCH-1 computes Operation 77 via the following algorithm, which can be implemented in a time of $\Theta(k2^{|C|})$ using only constant auxiliary space:

Algorithm 78. *For every $i \in \mathbb{N}_k$ and every $Y \in \mathcal{P}(D_i)$ we maintain a variable $h_i(Y) \in \mathbb{R}$ initially set equal to zero.*
For every $Z \in \mathcal{P}(C)$, in turn, we do the following:

1. *Set $\alpha \leftarrow \prod_{i=1}^{k} \Upsilon_i(Z \cap D_i)$*
2. *For all $i \in \mathbb{N}_k$ set $h_i(Z \cap D_i) \leftarrow h_i(Z \cap D_i) + \alpha$.*

Note that after we have performed the above for every $Z \in \mathcal{P}(C)$, the function h_i is a potential in $\mathcal{T}(D_i)$. We then output, for every $i \in \mathbb{N}_k$, $\Psi_i \leftarrow h_i$.

The correctness of Algorithm 78 is seen immediately by noting that at the end of the algorithm we have, for all $i \in \mathbb{N}_k$ and $Y \in \mathcal{P}(D_i)$:

$$
\begin{aligned}
h_i(Y) &= \sum_{[Z \in \mathcal{P}(C): Z \cap D_i = Y]} \prod_{j=1}^{k} \Upsilon_j(Z \cap D_j) && (4.20)\\
&= \sum_{Z \in \mathcal{P}(C): Z \cap D_i = Y} \left[\prod_{j=1}^{k} \Upsilon_j\right](Z) && (4.21)\\
&= \sum_{Z \in \mathcal{P}(C): Z \cap D_i = Y} \Gamma(Z) && (4.22)\\
&= \Gamma^{\nabla D_i}(Y) && (4.23)\\
&= \Psi_i(Y) && (4.24)
\end{aligned}
$$

where Γ is as in the statement of Operation 77.

Note that, for every vertex C, Operation 77 is called twice (once during the inward phase and once during the outward phase), each time taking a time, under Algorithm 78, of $\Theta((\deg(C) + |\mathcal{F}(C)|)2^{|C|})$. Each vertex C hence contributes a time of $\Theta((\deg(C) + |\mathcal{F}(C)|)2^{|C|})$ to the time complexity of the whole message passing algorithm. ARCH-1 hence has the same time complexity as Hugin propagation. Like Shafer-Shenoy propagation, the space required by ARCH-1 is only that required to store the factors and messages.

In Section 4.9.3 we show how, by caching various quantities, we can, whilst keeping the same space requirements, speed up Algorithm 78 to take a time of only $\Theta\left(\sum_{i=1}^{|C|} |\{j : y_i \in D_j\}| 2^i\right)$ where $\{y_j : j \in \mathbb{N}_{|C|}\} := C$. In order be free to choose the ordering $(y_1, y_2, ..., y_{|C|})$ of C that minimises this time we require an additional time of $\mathcal{O}\left(\sum_{i=1}^{k} |D_i| 2^{|D_i|}\right)$. However, even this faster implementation of ARCH-1 may still be significantly slower than ARCH-2 and will never be faster by more than a logarithmic factor.

4.7 ARCH-2

We now describe the architecture ARCH-2. The time and space complexities of ARCH-2 are always (up to a factor that is linear in the width of $\mathcal{J}$) at least as good as those of ARCH-1. In cases in which we have large vertices of high degree ARCH-2 is significantly faster than ARCH-1/Hugin.

Specifically, each vertex C contributes a time of only $\Theta(|C|2^{|C|})$ to ARCH-2 and, in addition to storing the factors and messages, ARCH-2 requires a space of only $\mathcal{O}\left(\max_{C \in \mathcal{V}(\mathcal{J})} |C| \left(\left(\sum_{H \in \mathcal{N}(C)} 2^{|H \cap C|}\right) + \left(\sum_{\Lambda \in \mathcal{F}(C)} 2^{|\sigma(C)|}\right)\right)\right)$.

ARCH-2 proceeds similarly to ARCH-1, using Operation 77 to do steps 1a and 2a of Algorithm 76. The only difference between ARCH-2 and ARCH-1 is how Operation 77 is computed. The algorithm for performing Operation 77 is based upon the concepts of the *p-dual* and the *m-dual* of a potential. The p-dual is a novel concept whilst the m-dual was defined in [53] under the name of "inclusion-exclusion format" but was used in a very different way from how we use it in this paper. We first give a definition of the duals and the required theory surrounding them.

4.7.1 The p-Dual

In this subsection we introduce the p-dual and the required theory surrounding it. We first define the p-dual of a potential:

Definition 79. The p-dual:

Given a set X and a potential $\Phi \in \mathcal{T}(X)$, the p-dual, $\#\Phi$, of Φ is the potential in $\mathcal{T}(X)$ that satisfies, for every $Y \in \mathcal{P}(X)$:

$$\#\Phi(Y) := \prod_{Z \in \mathcal{P}(Y)} \Phi(Z)^{(-1)^{|Z|}} \tag{4.25}$$

The next theorem will assist us in the the recovery of a potential from its p-dual

Theorem 80. *Suppose we have a set X, an element $x \in X$ and a potential $\Phi \in \mathcal{T}(X)$. Define Φ_- and Φ_+ to be the potentials in $\mathcal{T}(X \setminus \{x\})$ that satisfy, for every $Z \in \mathcal{P}(X \setminus \{x\})$, $\Phi_-(Z) := \Phi(Z)$ and $\Phi_+(Z) := \Phi(Z \cup \{x\})$. For every $Y \in \mathcal{P}(X \setminus \{x\})$ we have the following:*

1. $\#\Phi_-(Y) = \#\Phi(Y)$
2. $\#\Phi_+(Y) = \#\Phi(Y)/\#\Phi(Y \cup \{x\})$

Proof. 1.

$$\#\Phi_-(Y) = \prod_{Z \in \mathcal{P}(Y)} \Phi_-(Z)^{(-1)^{|Z|}} \tag{4.26}$$

$$= \prod_{Z \in \mathcal{P}(Y)} \Phi(Z)^{(-1)^{|Z|}} \tag{4.27}$$

$$= \#\Phi(Y) \tag{4.28}$$

2.

$$\#\Phi_+(Y) = \prod_{Z \in \mathcal{P}(Y)} \Phi_+(Z)^{(-1)^{|Z|}} \tag{4.29}$$

$$= \prod_{Z \in \mathcal{P}(Y)} \Phi(Z \cup \{x\})^{(-1)^{|Z|}} \tag{4.30}$$

$$= \prod_{U \in \mathcal{P}(Y \cup \{x\}) : x \in U} \Phi(U)^{(-1)^{|U|-1}} \tag{4.31}$$

$$= \left(\prod_{U \in \mathcal{P}(Y \cup \{x\}) : x \in U} \Phi(U)^{(-1)^{|U|}} \right)^{-1} \tag{4.32}$$

$$= \left(\prod_{U \in \mathcal{P}(Y \cup \{x\}) \setminus \mathcal{P}(Y)} \Phi(U)^{(-1)^{|U|}} \right)^{-1} \tag{4.33}$$

$$= \left(\prod_{U \in \mathcal{P}(Y)} \Phi(U)^{(-1)^{|U|}} \right) \left(\prod_{U \in \mathcal{P}(Y \cup \{x\})} \Phi(U)^{(-1)^{|U|}} \right)^{-1} \tag{4.34}$$

$$= \frac{\#\Phi(Y)}{\#\Phi(Y \cup \{x\})} \tag{4.35}$$

□

From Theorem 80 we get the following theorem, which will aid us in the construction of a p-dual.

Theorem 81. *Suppose we have a set X, an element $x \in X$ and a potential $\Phi \in \mathcal{T}(X)$. Define Φ_- and Φ_+ to be the potentials in $\mathcal{T}(X \setminus \{x\})$ that satisfy, for every $Z \in \mathcal{P}(X \setminus \{x\})$, $\Phi_-(Z) := \Phi(Z)$ and $\Phi_+(Z) := \Phi(Z \cup \{x\})$. For every $Y \in \mathcal{P}(X \setminus \{x\})$ we have the following:*

1. $\#\Phi(Y) = \#\Phi_-(Y)$
2. $\#\Phi(Y \cup \{x\}) = \#\Phi_-(Y)/\#\Phi_+(Y)$

Proof. The result comes from solving the equations of Theorem 80 for $\#\Phi(Y)$ and $\#\Phi(Y \cup \{x\})$ □

We next show that the p-dual of a product of potentials with the same domain is the product of the p-duals of the potentials:

Lemma 82. *Given a set X and potentials $\Phi', \Phi \in \mathcal{T}(X)$, we have $\#(\Phi'\Phi) = (\#\Phi')(\#\Phi)$*

Proof. For any $Y \in \mathcal{P}(X)$ we have:

$$\begin{aligned}
\#[\Phi'\Phi](Y) &= \prod_{Z \in \mathcal{P}(Y)} [\Phi'\Phi](Z)^{(-1)^{|Z|}} && (4.36)\\
&= \prod_{Z \in \mathcal{P}(Y)} [\Phi'(Z)\Phi(Z)]^{(-1)^{|Z|}} && (4.37)\\
&= \prod_{Z \in \mathcal{P}(Y)} \Phi'(Z)^{(-1)^{|Z|}} \Phi(Z)^{(-1)^{|Z|}} && (4.38)\\
&= \left(\prod_{Z \in \mathcal{P}(Y)} \Phi'(Z)^{(-1)^{|Z|}}\right)\left(\prod_{Z \in \mathcal{P}(Y)} \Phi(Z)^{(-1)^{|Z|}}\right) && (4.39)\\
&= [\#\Phi'(Y)][\#\Phi(Y)] && (4.40)
\end{aligned}$$

□

With the aid of the following lemma we will show how to compute the p-dual of a product of potentials:

Lemma 83. *Given a set X, a set $Y \subseteq X$, and a potential $\Phi \in \mathcal{T}(Y)$, let Φ' be the potential in $\mathcal{T}(X)$ that satisfies, for all $Z \in \mathcal{P}(X)$, $\Phi'(Z) := \Phi(Z \cap Y)$. Then given $U \in \mathcal{P}(X)$ we have:*

1. *If $U \subseteq Y$ then $\#\Phi'(U) = \#\Phi(U)$*
2. *If $U \nsubseteq Y$ then $\#\Phi'(U) = 1$*

Proof. 1.

$$\begin{aligned}
\#\Phi'(U) &= \prod_{Z \in \mathcal{P}(U)} \Phi'(Z)^{(-1)^{|Z|}} && (4.41)\\
&= \prod_{Z \in \mathcal{P}(U)} \Phi(Z \cap Y)^{(-1)^{|Z|}} && (4.42)\\
&= \prod_{Z \in \mathcal{P}(U)} \Phi(Z)^{(-1)^{|Z|}} && (4.43)\\
&= \#\Phi(U) && (4.44)
\end{aligned}$$

2. We have $U \setminus Y \neq \emptyset$ so choose some $v \in U \setminus Y$. We then have:

$$\begin{aligned}
\#\Phi'(U) &= \prod_{Z \in \mathcal{P}(U)} \Phi'(Z)^{(-1)^{|Z|}} && (4.45)\\
&= \prod_{W \in \mathcal{P}(U \setminus \{v\})} \Phi'(W)^{(-1)^{|W|}} \Phi'(W \cup \{v\})^{(-1)^{|W \cup \{v\}|}} && (4.46)\\
&= \prod_{W \in \mathcal{P}(U \setminus \{v\})} \Phi(W \cap Y)^{(-1)^{|W|}} \Phi((W \cup \{v\}) \cap Y)^{(-1)^{|W \cup \{v\}|}} && (4.47)\\
&= \prod_{W \in \mathcal{P}(U \setminus \{v\})} \Phi(W \cap Y)^{(-1)^{|W|}} \Phi(W \cap Y)^{(-1)^{|W \cup \{v\}|}} && (4.48)\\
&= \prod_{W \in \mathcal{P}(U \setminus \{v\})} \Phi(W \cap Y)^{(-1)^{|W|}} \Phi(W \cap Y)^{(-1)^{|W|+1}} && (4.49)\\
&= \prod_{W \in \mathcal{P}(U \setminus \{v\})} \Phi(W \cap Y)^{(-1)^{|W|}} \Phi(W \cap Y)^{-(-1)^{|W|}} && (4.50)\\
&= \prod_{W \in \mathcal{P}(U \setminus \{v\})} \Phi(W \cap Y)^{(-1)^{|W|}-(-1)^{|W|}} && (4.51)\\
&= \prod_{W \in \mathcal{P}(U \setminus \{v\})} \Phi(W \cap Y)^{0} && (4.52)\\
&= 1 && (4.53)
\end{aligned}$$

□

Theorem 84. *Suppose we have a set X, subsets $\{Y_i : i \in \mathbb{N}_k\} \subseteq \mathcal{P}(X)$ such that $\bigcup\{Y_i : i \in \mathbb{N}_k\} = X$ and potentials $\{\Phi_i : i \in \mathbb{N}_k\}$ such that $\Phi_i \in \mathcal{T}(Y_i)$. Then for every $U \in \mathcal{P}(X)$ we have:*

$$\#\left[\prod_{i=1}^{k} \Phi_i\right](U) = \prod_{i : U \subseteq Y_i} \#\Phi_i(U) \tag{4.54}$$

Proof. For $i \in \mathbb{N}_k$ let Φ'_i be the potential in $\mathcal{T}(X)$ that satisfies, for all $Z \in \mathcal{P}(X)$, $\Phi'(Z) := \Phi(Z \cap Y)$. Then we have:

$$\begin{aligned}
\#\left[\prod_{i=1}^{k} \Phi_i\right](U) &= \#\left[\prod_{i=1}^{k} \Phi'_i\right](U) && (4.55)\\
&= \prod_{i=1}^{k} \#\Phi'_i(U) && (4.56)\\
&= \left(\prod_{i : U \subseteq Y_i} \#\Phi'_i(U)\right)\left(\prod_{i : U \nsubseteq Y_i} \#\Phi'_i(U)\right) && (4.57)\\
&= \left(\prod_{i : U \subseteq Y_i} \#\Phi_i(U)\right)\left(\prod_{i : U \nsubseteq Y_i} 1\right) && (4.58)\\
&= \prod_{i : U \subseteq Y_i} \#\Phi_i(U) && (4.59)
\end{aligned}$$

Where equation 4.56 comes from Lemma 82 and equation 4.58 comes from Lemma 83. □

4.7.2 The m-Dual

In this subsection we introduce the m-dual and the required theory surrounding it. The m-dual was defined in [53] under the name of "inclusion-exclusion format" but was used in a very different way from how we use it in this paper. We first define the m-dual of a potential:

Definition 85. The m-dual:

Given a set X and a potential $\Phi \in \mathcal{T}(X)$, the m-dual, $\%\Phi$, of Φ is the potential in $\mathcal{T}(X)$ that satisfies, for every $Y \in \mathcal{P}(X)$:

$$\%\Phi(Y) := \sum_{Z\in\mathcal{P}(X):Y\subseteq Z} \Phi(Z) \tag{4.60}$$

The following theorem will be useful in the construction of an m-dual:

Theorem 86. *Suppose we have a set X, an element $x \in X$ and a potential $\Phi \in \mathcal{T}(X)$. Define Φ_- and Φ_+ to be the potentials in $\mathcal{T}(X \setminus \{x\})$ that satisfy, for every $Z \in \mathcal{P}(X \setminus \{x\})$, $\Phi_-(Z) := \Phi(Z)$ and $\Phi_+(Z) := \Phi(Z \cup \{x\})$. For every $Y \in \mathcal{P}(X \setminus \{x\})$ we have the following:*

1. $\%\Phi(Y) = \%\Phi_-(Y) + \%\Phi_+(Y)$
2. $\%\Phi(Y \cup \{x\}) = \%\Phi_+(Y)$

Proof. 1.

$$\begin{aligned}
\%\Phi(Y) &= \sum_{Z\in\mathcal{P}(X):Y\subseteq Z} \Phi(Z) && (4.61)\\
&= \left(\sum_{Z\in\mathcal{P}(X):Y\subseteq Z,x\notin Z} \Phi(Z)\right) + \left(\sum_{Z\in\mathcal{P}(X):Y\subseteq Z,x\in Z} \Phi(Z)\right) && (4.62)\\
&= \left(\sum_{Z\in\mathcal{P}(X\setminus\{x\}):Y\subseteq Z} \Phi(Z)\right) + \left(\sum_{Z\in\mathcal{P}(X):Y\subseteq Z,x\in Z} \Phi(Z)\right) && (4.63)\\
&= \left(\sum_{Z\in\mathcal{P}(X\setminus\{x\}):Y\subseteq Z} \Phi(Z)\right) + \left(\sum_{U\in\mathcal{P}(X\setminus\{x\}):Y\subseteq U} \Phi(U\cup\{x\})\right) && (4.64)\\
&= \left(\sum_{Z\in\mathcal{P}(X\setminus\{x\}):Y\subseteq Z} \Phi_-(Z)\right) + \left(\sum_{U\in\mathcal{P}(X\setminus\{x\}):Y\subseteq U} \Phi_+(U)\right) && (4.65)\\
&= \%\Phi_-(Y) + \%\Phi_+(Y) && (4.66)
\end{aligned}$$

where Equation 4.64 comes from setting $U := Z \setminus \{x\}$ in the right-hand sum.

2.

$$\begin{aligned}
\%\Phi(Y\cup\{x\}) &= \sum_{Z\in\mathcal{P}(X):Y\cup\{x\}\subseteq Z} \Phi(Z) && (4.67)\\
&= \sum_{U\in\mathcal{P}(X\setminus\{x\}):Y\subseteq U} \Phi(U\cup\{x\}) && (4.68)\\
&= \sum_{U\in\mathcal{P}(X\setminus\{x\}):Y\subseteq U} \Phi_+(U) && (4.69)\\
&= \%\Phi_+(Y) && (4.70)
\end{aligned}$$

where Equation 4.68 comes from setting $U := Z \setminus \{x\}$.

□

From Theorem 86 we get the following theorem, which will be useful in converting an m-dual back to the original potential:

Theorem 87. *Suppose we have a set X, an element $x \in X$ and a potential $\Phi \in \mathcal{T}(X)$. Define Φ_- and Φ_+ to be the potentials in $\mathcal{T}(X \setminus \{x\})$ that satisfy, for every $Z \in \mathcal{P}(X \setminus \{x\})$, $\Phi_-(Z) := \Phi(Z)$ and $\Phi_+(Z) := \Phi(Z \cup \{x\})$. For every $Y \in \mathcal{P}(X \setminus \{x\})$ we have the following:*

1. $\%\Phi_-(Y) = \%\Phi(Y) - \%\Phi(Y \cup \{x\})$
2. $\%\Phi_+(Y) = \%\Phi(Y \cup \{x\})$

Proof. The result comes from solving the equations of Theorem 86 for $\%\Phi_-(Y)$ and $\%\Phi_+(Y)$ □

We next show how marginals are computed when working with m-duals:

Theorem 88. *Suppose we have sets X and Y with $Y \subseteq X$ and a potential $\Phi \in \mathcal{T}(X)$. Then for every $Z \in \mathcal{P}(Y)$ we have:*

$$\% \left[\Phi^{\nabla Y}\right](Z) = \%\Phi(Z) \tag{4.71}$$

Proof.

$$\% \left[\Phi^{\nabla Y}\right](Z) = \sum_{U\in\mathcal{P}(Y):Z\subseteq U} \Phi^{\nabla Y}(U) \tag{4.72}$$

$$= \sum_{[U\in\mathcal{P}(Y):Z\subseteq U]} \sum_{[W\in\mathcal{P}(X):W\cap Y=U]} \Phi(W) \tag{4.73}$$

Note that if we have $U, U' \in \mathcal{P}(Y)$ with $U \neq U'$ and we have $W, W' \in \mathcal{P}(X)$ with $W \cap Y = U$ and $W' \cap Y = U'$ then $W \cap Y \neq W' \cap Y$ so $W \neq W'$. Hence, each W in the (double) sum is counted only once.
Suppose we have $W \in \mathcal{P}(X)$ with $Z \subseteq W$. Then if $U := W \cap Y$ then since $Z \subseteq Y$ and $Z \subseteq W$ we have $Z \subseteq U$ so W is included in the (double) sum.
Now suppose W is included in the (double) sum. Then there exists a $U \in \mathcal{P}(Y)$ with $Z \subseteq U$ such that $W \cap Y = U$. Hence $Z \subseteq W \cap Y$ so $Z \subseteq W$.
Hence, for each $W \in \mathcal{P}(X)$, W is contained in the (double) sum if and only if $Z \subseteq W$ and so since, by above, each such W is counted only once in the (double) sum we have:

$$\% \left[\Phi^{\nabla Y}\right](Z) = \sum_{[U\in\mathcal{P}(Y):Z\subseteq U]} \sum_{[W\in\mathcal{P}(X):W\cap Y=U]} \Phi(W) \tag{4.74}$$

$$= \sum_{W\in\mathcal{P}(X):Z\subseteq W} \Phi(W) \tag{4.75}$$

$$= \%\Phi(Z) \tag{4.76}$$

□

4.7.3 Functions for Manipulating Potentials

We now describe the functions used by ARCH-2. The functions are **transform1** which transforms a potential into its p-dual, **product** which computes the product of potentials when working with p-duals, **transform2** which transforms a restriction of the p-dual of a potential into a restriction of the m-dual of the potential, **marginalise** which computes marginals of a potential while working with m-duals, and **transform3** which transforms the m-dual of a potential back to the original potential.

The functions **transform1**, **transform2** and **transform3** all rest on the observation that, given a potential Φ with $\sigma(\Phi) = \emptyset$, we have $\%\Phi = \#\Phi = \Phi$.

In the description of the functions **transform1**, **transform2** and **transform3**, Φ_- and Φ_+ are defined from Φ and x as in the statements of theorems 80, 81, 86 and 87: i.e. Φ_- and Φ_+ are the potentials in $\mathcal{T}(\sigma(\Phi) \setminus \{x\})$ that satisfy, for every $Z \in \mathcal{P}(\sigma(\Phi) \setminus \{x\})$, $\Phi_-(Z) := \Phi(Z)$ and $\Phi_+(Z) := \Phi(Z \cup \{x\})$

Also, for the function **transform2** (resp. **transform3**) recall that $\sigma(\Phi) = \sigma(\#\Phi)$ (resp. $\sigma(\Phi) = \sigma(\%\Phi)$)

For a detailed description of how to implement these functions so they have the stated time and space complexities see Section 4.9.4 (which is based on notation and algorithms given earlier in Section 4.9)

We first describe the recursive function **transform1**:

- The function takes, as input, a potential Φ
- The function outputs the p-dual, $\#\Phi$, of Φ.
- The algorithm can be implemented to take a time of $\Theta\left(|\sigma(\Phi)|2^{|\sigma(\Phi)|}\right)$ and to require $\Theta\left(2^{|\sigma(\Phi)|}\right)$ auxiliary space. This is proved immediately by induction over $|\sigma(\Phi)|$.
- The correctness of the algorithm comes directly from Theorem 81, using induction over $|\sigma(\Phi)|$.

Algorithm 89. transform1(Φ)*:*
If $\sigma(\Phi)=\emptyset$ then return Φ. Else, perform the following:

```
1. Choose $x \in \sigma(\Phi)$.
2. For each $Z \in \mathcal{P}(\sigma(\Phi)\setminus\{x\})$ set $\Phi_-(Z) \leftarrow \Phi(Z)$ and $\Phi_+(Z) \leftarrow \Phi(Z\cup\{x\})$.
3. Set $\#\Phi_- \leftarrow$transform1$(\Phi_-)$ and $\#\Phi_+ \leftarrow$transform1$(\Phi_+)$.
4. For each $Y \in \mathcal{P}(\sigma(\Phi)\setminus\{x\})$ set $\#\Phi(Y) \leftarrow \#\Phi_-(Y)$ and $\#\Phi(Y\cup\{x\}) \leftarrow \#\Phi_-(Y)/\#\Phi_+(Y)$
5. Return $\#\Phi$.
```

We now describe the function **product**:

- The function takes, as input, a set $\{\#\Upsilon_i : i \in \mathbb{N}_k\}$ of p-duals of potentials Υ_i.
- The function outputs the restriction $[\#\Gamma]^{\oplus\zeta}$ where $\zeta = \bigcup_{i=1}^k \mathcal{P}(\sigma(\Upsilon_i))$ and $\Gamma = \prod_{i=1}^k \Upsilon_i$. It is the case that for all $Z \in \mathcal{P}(\sigma(\Gamma))$ with $Z \notin \zeta$ we have $\#\Gamma(Z) = 1$.
- The algorithm can be implemented to take a time of $\mathcal{O}\left(2^{|\bigcup_{i=1}^k \sigma(\Upsilon_i)|} + \sum_{i=1}^k 2^{|\sigma(\Upsilon_i)|}\right)$ and to require $\Theta(|\zeta|)$ auxiliary space.
- The correctness of the algorithm comes directly from Theorem 84.

Algorithm 90. product$(\{\#\Upsilon_i : i \in \mathbb{N}_k\})$*:*

```
1. Let $\zeta \leftarrow \bigcup_{i=1}^k \mathcal{P}(\sigma(\#\Upsilon_i))$
2. For each $Z \in \zeta$ set $\#\Gamma(Z) \leftarrow \prod_{i\in\mathbb{N}_k : Z\subseteq\sigma(\Upsilon_i)} \#\Upsilon_i(Z)$.
3. Return $[\#\Gamma]^{\oplus\zeta}$
```

We now describe the recursive function **transform2**:

- The function takes, as input, a restriction, $[\#\Phi]^{\oplus\zeta}$, of the p-dual of a potential Φ where $\zeta \subseteq \mathcal{P}(\sigma(\Phi))$ is a straddle-set such that, for all $Z \in \mathcal{P}(\sigma(\Phi))$ with $Z \notin \zeta$, we have $\#\Phi(Z) = 1$.
- The function outputs the restriction, $[\%\Phi]^{\oplus\zeta}$, of the m-dual of Φ.
- The algorithm can be implemented to take a time of $\mathcal{O}(|\sigma(\Phi)|2^{|\sigma(\Phi|)})$ and to require a space of only $\mathcal{O}(|\sigma(\Phi)||\zeta|)$. This is proved immediately by induction over $|\sigma(\Phi)|$, noting that for $x \in \sigma(\Phi)$ we have that $|\{U \in \zeta : x \notin U\}| \leq |\zeta|$.
- The correctness of the algorithm comes directly from theorems 80 and 86, using induction over $|\sigma(\Phi)|$

Algorithm 91. transform2$([\#\Phi]^{\oplus\zeta})$*:*
If $\sigma(\Phi)=\emptyset$ then return $\#\Phi$. Else, perform the following:

```
1. Choose $x \in \sigma(\Phi)$
2. Set $\vartheta \leftarrow \{U \in \zeta : x \notin U\}$
3. For each $Y \in \vartheta$ set $\#\Phi_-(Y) \leftarrow \#\Phi(Y)$
```

```
4. For each $Y \in \vartheta$ do the following:
   If $Y \cup \{x\} \in \zeta$ then set $\#\Phi_+(Y) \leftarrow \#\Phi(Y)/\#\Phi(Y \cup \{x\})$. Else set $\#\Phi_+(Y) \leftarrow \#\Phi(Y)$
5. Set $[\%\Phi_-]^{\oplus\vartheta} \leftarrow$ **transform2**$([\#\Phi_-]^{\oplus\vartheta})$ and $[\%\Phi_+]^{\oplus\vartheta} \leftarrow$ **transform2**$([\#\Phi_+]^{\oplus\vartheta})$
6. For each $Y \in \vartheta$ set $\%\Phi(Y) \leftarrow \%\Phi_-(Y)+\%\Phi_+(Y)$ and $\%\Phi(Y\cup\{x\}) \leftarrow \%\Phi_+(Y)$
7. Return $[\%\Phi]^{\oplus\zeta}$
```

We now describe the function **marginalise**:

- The function takes, as input, a restriction $[\%\Gamma]^{\oplus\zeta}$ of the m-dual of a potential Γ as well as a set of sets $\{D_i : i \in \mathbb{N}_k\}$ where $\zeta = \bigcup_{i=1}^{k} \mathcal{P}(D_i)$.
- The function outputs the set of potentials $\{\%\Psi_i : i \in \mathbb{N}_k\}$ where, for every $i \in \mathbb{N}_k$, $\Psi_i := \Gamma^{\nabla D_i}$.
- The algorithm can be implemented to take a time of $\Theta\left(2^{|\bigcup_{i=1}^{k} D_i|} + \sum_{i=1}^{k} 2^{|D_i|}\right)$ and to require $\Theta(|\zeta|)$ auxiliary space.
- The correctness of the algorithm comes directly from Theorem 88

Algorithm 92. marginalise$([\%\Gamma]^{\oplus\zeta}, \{D_i : i \in \mathbb{N}_k\})$:

```
1. For every $Z \in \zeta$ perform the following:
   For every $i \in \mathbb{N}_k$ with $Z \subseteq D_i$ set $\%\Psi_i(Z) \leftarrow \%\Gamma(Z)$
2. Return $\{\%\Psi_i : i \in \mathbb{N}_k\}$
```

We now describe the recursive function **transform3**:

- The function takes as input the m-dual, $\%\Phi$, of a potential Φ.
- The function outputs the potential Φ.
- The algorithm can be implemented to take a time of $\Theta\left(|\sigma(\Phi)|2^{|\sigma(\Phi)|}\right)$ and to require $\Theta\left(2^{|\sigma(\Phi)|}\right)$ auxiliary space. This is proved immediately by induction over $|\sigma(\Phi)|$
- The correctness of the algorithm comes directly from Theorem 87, using induction over $|\sigma(\Phi)|$.

Algorithm 93. transform3$(\%\Phi)$:
If $\sigma(\Phi) = \emptyset$ then return $\%\Phi$. Else, perform the following:

```
1. Choose $x \in \sigma(\Phi)$.
2. For each $Y \in \mathcal{P}(\sigma(\Phi) \setminus \{x\})$ set $\%\Phi_-(Y) \leftarrow \%\Phi(Y) - \%\Phi(Y \cup \{x\})$ and $\%\Phi_+(Y) \leftarrow \%\Phi(Y \cup \{x\})$.
3. Set $\Phi_- \leftarrow$ **transform3**$(\%\Phi_-)$ and $\Phi_+ \leftarrow$ **transform3**$(\%\Phi_+)$.
4. For each $Y \in \mathcal{P}(\sigma(\Phi) \setminus \{x\})$ set $\Phi(Y) \leftarrow \Phi_-(Y)$ and $\Phi(Y \cup \{x\}) \leftarrow \Phi_+(Y)$
5. Return $\Phi$.
```

4.7.4 Performing Operation 77

As stated at the start of the section, the only difference between ARCH-1 and ARCH-2 is the way that Operation 77, used to do steps 1a and 2a of Algorithm 76, is performed.
In this subsection let C, k, D_i, Υ_i, Γ and Ψ_i be as in the description of Operation 77. That is: C is a set. $D_1, ..., D_k$ are subsets of C with $\bigcup_{i=1}^{k} D_i = C$. Υ_i is a potential in $\mathcal{T}(D_i)$. $\Gamma := \prod_{i=1}^{k} \Upsilon_i$ and $\Psi_i := \Gamma^{\nabla D_i}$. The goal of Operation 77 is to compute Ψ_i for every $i \in \mathbb{N}_k$.

ARCH-2 performs Operation 77 via the following algorithm:

Algorithm 94. *Performing Operation 77:*

```
1. For every $i \in \mathbb{N}_k$ set $\#\Upsilon_i \leftarrow \mathbf{transform1}(\Upsilon_i)$
2. Set $[\#\Gamma]^{\oplus\zeta} \leftarrow \mathbf{product}(\{\#\Upsilon_i : i \in \mathbb{N}_k\})$
3. Set $[\%\Gamma]^{\oplus\zeta} \leftarrow \mathbf{transform2}([\#\Gamma]^{\oplus\zeta})$
4. Set $\{\%\Psi_i : i \in \mathbb{N}_k\} \leftarrow \mathbf{marginalise}([\%\Gamma]^{\oplus\zeta}, \{D_i : i \in \mathbb{N}_k\})$
5. For every $i \in \mathbb{N}_k$ set $\Psi_i \leftarrow \mathbf{transform3}(\%\Psi_i)$ and return $\Psi_i$
```

To summarise, Algorithm 94 does the following: First the potentials Υ_i are converted into their p-duals. From these p-duals, a restriction of the p-dual of the product, Γ, of the potentials Υ_i is computed. From this potential, a restriction of the m-dual of Γ is computed and is then used to compute the m-duals of the D_i-marginals, Ψ_i, of Γ. These m-duals are then converted into the potentials Ψ_i.

The correctness of the Algorithm 94 is proved as follows: lines 1 and 5 are cleary valid by the descriptions of the functions **transform1** and **transform3**. Since $\Gamma = \prod_{i=1}^{k} \Upsilon_i$ line 2 is valid. Note that, since by line 2, $\zeta = \bigcup_{i=1}^{k} \mathcal{P}(\sigma(\#\Upsilon_i))$, ζ is a straddle set. Hence, since by line 2 it is true that for all $Y \in \mathcal{P}(C)$ with $Y \notin \zeta$ we have $\#\Gamma(Y) = 1$, line 3 is valid. Since, by line 2 we have $\zeta = \bigcup_{i=1}^{k} \mathcal{P}(\sigma(\#\Upsilon_i)) = \bigcup_{i=1}^{k} \mathcal{P}(D_i)$, line 4 is valid.

4.7.5 Time and Space Complexity

We now derive the time complexity of Algorithm 94 and use it to calculate the time complexity of ARCH-2:
Lines 1 and 5 of Algorithm 94 take a time of $\Theta\left(\sum_{i=1}^{k} |D_i| 2^{|D_i|}\right)$. Lines 2 and 4 take a time of
$\mathcal{O}\left(2^{|C|} + \sum_{i=1}^{k} |D_i| 2^{|D_i|}\right)$. Line 3 takes a time of $\mathcal{O}\left(|C|2^{|C|}\right)$. The total time complexity of Algorithm 94 is hence $\mathcal{O}\left(|C|2^{|C|} + \sum_{i=1}^{k} |D_i| 2^{|D_i|}\right)$
Hence Step 1a and Step 2a of Algorithm 76 both take a time of:

$$\mathcal{O}\left(|C|2^{|C|} + \left(\sum_{H \in \mathcal{N}(C)} |H \cap C| 2^{|H \cap C|}\right) + \left(\sum_{\Lambda \in \mathcal{F}(C)} |\Lambda| 2^{|\Lambda|}\right)\right) \tag{4.77}$$

$$= \mathcal{O}\left(|C|2^{|C|} + \left(|{\uparrow}(C) \cap C| 2^{|{\uparrow}(C) \cap C|} + \sum_{H \in {\downarrow}(C)} |H \cap C| 2^{|H \cap C|}\right) + \left(\sum_{\Lambda \in \mathcal{F}(C)} |\Lambda| 2^{|\Lambda|}\right)\right) \tag{4.78}$$

$$\subseteq \mathcal{O}\left(|C|2^{|C|} + \left(|C|2^{|C|} + \sum_{H \in {\downarrow}(C)} |H \cap C| 2^{|H \cap C|}\right) + \left(\sum_{\Lambda \in \mathcal{F}(C)} |\Lambda| 2^{|\Lambda|}\right)\right) \tag{4.79}$$

$$= \mathcal{O}\left(|C|2^{|C|} + \left(\sum_{H \in {\downarrow}(C)} |H \cap C| 2^{|H \cap C|}\right) + \left(\sum_{\Lambda \in \mathcal{F}(C)} |\Lambda| 2^{|\Lambda|}\right)\right) \tag{4.80}$$

$$\subseteq \mathcal{O}\left(|C|2^{|C|} + \left(\sum_{H \in {\downarrow}(C)} |H| 2^{|H|}\right) + \left(\sum_{\Lambda \in \mathcal{F}(C)} |\Lambda| 2^{|\Lambda|}\right)\right) \tag{4.81}$$

We call this time complexity the "computation time at C". Note that every vertex H contributes $\mathcal{O}(|H|2^{|H|})$ to the computation time at H, a time of $\mathcal{O}(|H|2^{|H|})$ to the computation time at ${\uparrow}(H)$ (if it exists), and no time to computation time at any other vertex. Each vertex H hence contributes a total time of $\mathcal{O}(|H|2^{|H|})$ to the running time of ARCH-2. In addition, by Equation 4.81 we have that each factor Λ contributes a time of $\mathcal{O}\left(|\sigma(\Lambda)| 2^{|\sigma(\Lambda)|}\right)$ to the running time of ARCH-2.

Note that the total running time of ARCH-2 is, up to a logarithmic factor, no worse than that of ARCH-1 (and Hugin propagation), and in cases where we have large vertices of high degree ARCH-2 is much faster than ARCH-1 (and Hugin propagation).

We now derive the space complexity of ARCH-2: The auxiliary space requirement of Algorithm 94 is the maximum auxiliary space required by any of the functions which is $\mathcal{O}(|C||\zeta|) \subseteq \mathcal{O}\left(|C| \bigcup_{i=1}^{k} 2^{|D_i|}\right)$. Hence Step 1a and Step 2a of Algorithm 76 both require an auxiliary space of $\mathcal{O}\left(|C| \left(\left(\sum_{H\in\mathcal{N}(C)} 2^{|H\cap C|}\right) + \left(\sum_{\Lambda\in\mathcal{F}(C)} 2^{|\sigma(\Lambda)|}\right)\right)\right)$. This implies that, in addition to storing the messages and factors, ARCH-2 has a space requirement of only $\mathcal{O}\left(\max_{C\in\mathcal{V}(\mathcal{J})} |C| \left(\left(\sum_{H\in\mathcal{N}(C)} 2^{|H\cap C|}\right) + \left(\sum_{\Lambda\in\mathcal{F}(C)} 2^{|\sigma(\Lambda)|}\right)\right)\right)$. Hence, the space requirement of ARCH-2 is not greater than that of ARCH-1 (and Shafer-Shenoy propagation) by more than a factor that is linear in width of the junction tree (and since this factor is logarithmic in the time complexity of the algorithm it is negligible).

4.8 Incorporating Zeros

So far we have only considered potentials that map into the positive reals (since ARCH-2 involves division and we can't divide by zero). We now show how to generalise so that the codomain of a potential can be $\mathbb{R}^+ \cup \{0\}$. In [43] the concept of a *zero-concious number* is introduced to do this with Hugin propagation (for a wider range of queries). However, to work with ARCH-2 we need a slightly different object:

Definition 95. MZC (multi-zero conscious) number:

- *An MZC number is a pair* $(a, i) \in \mathbb{R}^+ \times \mathbb{Z}$.
- *The product,* $(a, i) \times (b, j)$*, of two MZC numbers,* (a, i) *and* (b, j)*, is defined to be equal to* (c, k) *where* $c := ab$ *and* $k := i + j$.
- *For two MZC numbers* (a, i) *and* (b, j) *we define* $(a, i)/(b, j) := (a, i) \times (1/b, -j)$.
- *The sum,* $(a, i) + (b, j)$*, of two MZC numbers,* (a, i) *and* (b, j)*, is defined to be equal to* (c, k) *where* c *and* k *are defined are follows: If* $i < j$ *then* $c := a$ *and* $k := i$*, if* $i = j$ *then* $c := a + b$ *and* $k := i$*, and if* $i > j$ *then* $c := b$ *and* $k := j$.

A real number $x \in \mathbb{R} \cup \{0\}$ is converted into an MZC number as follows: If $x = 0$ then it is converted into the MZC number $(1, 1)$. Else it is converted into the MZC number $(x, 0)$. An MZC number (a, i) is converted into a real number as follows: If $i \neq 0$ then it is converted into 0. Else it is converted into a.

Zeros are incorporated into ARCH-2 as follows: Before running Algorithm 94 all numbers (that is: the quantities $\Upsilon_i(Z)$) are converted from reals numbers into MZC numbers. Lines 1 to 3 of Algorithm 94 are then run with MZC numbers instead of reals. After Line 3 is complete then all MZC numbers (that is: the quantities $\%\Gamma(Z)$) are converted from MZC numbers to real numbers. Lines 4 and 5 of Algorithm 94 are then run using real numbers.

Due to the equivalence of ARCH-1/ARCH-2 to Hugin propagation we can, in Line 2b of Algorithm 76, define division of a real number by zero to be equal to zero (or any other number) as is done in Hugin propagation (see [43]).

4.9 Implementation Details

In this section we make use of tree-structured data-structures. Whenever we use the word "vertex" or "leaf" we are referring to a vertex in one of these tree-structured data-structures (not a junction tree).

In this section we assume, without loss of generality, that $S = \mathbb{N}_n$ for some $n \in \mathbb{N}$. Throughout the entire junction tree algorithm we maintain an array $\mathcal{A}$ of size n such that each element of $\mathcal{A}$ is (a pointer to) a set of (pointers to) internal vertices of trees. Note that in our pseudo-code we will regard each element of $\mathcal{A}$ to be a set of vertices rather than a pointer to a set of pointers to vertices. We denote the e-th element of $\mathcal{A}$ by $\mathcal{A}(e)$. We also maintain a set $\mathcal{L}$ of (pointers to) leaves of trees. Like $\mathcal{A}$ we shall, in our pseudo-code, regard $\mathcal{L}$ as a set of leaves rather than a set of pointers to leaves.

$\mathcal{A}$ and $\mathcal{L}$ are used only for synchronised-searches and full-searches (see later) and in between different synchronised-searches/full-searches we have $\mathcal{L} = \emptyset$ and $\mathcal{A}(e) = \emptyset$ for all $e \in \mathbb{N}_n$.

4.9.1 Data-Structures

An *oriented binary tree* is a rooted tree in which every internal vertex v has two children: one child is called the *left-child* of v and is denoted by $\triangleleft(v)$. The other is called the *right-child* of v and is denoted by $\triangleright(v)$. Given a vertex v in an oriented binary tree we define $\Uparrow(v)$ to be the set of proper ancestors of v and we define $\Downarrow(v)$ to be the subtree of v and its descendants.

Given a straddle-set $\zeta \subseteq \mathcal{P}(\mathbb{N}_n)$ we define the *straddle-tree*, $\mathfrak{B}(\zeta)$ as follows: $\mathfrak{B}(\zeta)$ is an oriented binary tree who's internal vertices are labelled with numbers in $\bigcup \zeta$. Given an internal vertex v we let $\phi(v)$ be the label of v. The labels are such that given internal vertices v and w such that w is a child of v we have that $\phi(w) > \phi(v)$. We have a bijection, τ, from the leaves of $\mathfrak{B}(\zeta)$ into the set ζ such that, for any leaf l we have $\tau(l) := \{\phi(v) : v \in \Uparrow(l),\ \triangleright(v) \in \Uparrow(l)\}$.

Given a straddle-tree $\mathfrak{B}(\zeta)$ we will also refer to the tree that $\mathfrak{B}(\zeta)$ is based on by $\mathfrak{B}(\zeta)$. Note that since a straddle-tree $\mathfrak{B}(\zeta)$ is a full binary tree with $|\zeta|$ leaves it has only $2|\zeta| - 1$ vertices in total. Note also that for some set $X \subseteq \mathbb{N}_n$ the straddle-tree $\mathfrak{B}(\mathcal{P}(X))$ is a balanced oriented binary tree of height $|X|$ such that all internal vertices at depth i are labeled with the $(i+1)$th smallest number in X.

Given a potential Φ and a straddle-set $\zeta \subseteq \mathcal{P}(\sigma(\Phi))$ we define the *info-tree*, $\mathfrak{T}(\Phi, \zeta)$ as the straddle-tree $\mathfrak{B}(\zeta)$ with a map ψ from the leaves of $\mathfrak{B}(\zeta)$ into $\mathbb{R}^+$ such that, for any leaf l we have $\psi(l) := \Phi(\tau(l))$. Any restriction, $\Phi^{\oplus\zeta}$ is stored as the info-tree $\mathfrak{T}(\Phi, \zeta)$. Any potential Φ (that is not a restriction) is stored as the info-tree $\mathfrak{T}(\Phi, \mathcal{P}(\sigma(\Phi)))$ which we shall denote by $\mathfrak{T}(\Phi)$

4.9.2 Searches

In this section we describe the ways that the algorithms perform efficient, simultaneous searches over straddle-trees. There are two types of simultaneous search we describe: *full-searches* which are used in ARCH-1 and *synchronised-searches* which are used in ARCH-2. We start by defining a *ghost-search* which is what the simultaneous searches are based on.

Ghost-Search: Given a set $X \subseteq \mathbb{N}_n$, a *ghost-search* of X is the following algorithm, which is split up into a sequence of steps called *time-steps*:

We maintain a stack $\mathfrak{Z}$ such that each element of $\mathfrak{Z}$ is either 0 or of the form (e, f) where $e \in X$ and $f \in \{1, 2, 3\}$. $\mathfrak{Z}$ is initialised to contain $(\min(X), 1)$ as a single element. On each time-step we do the following:

If the top element of $\mathfrak{Z}$ is 0 then remove it from $\mathfrak{Z}$ which completes the time-step. Else, the top element of $\mathfrak{Z}$ is (e, f) for some $e \in X$ and $f \in \{1, 2, 3\}$ so we have the following cases:

1. $f = 1$: In this case we remove (e, f) from $\mathfrak{Z}$ and then place $(e, 2)$ on the top of $\mathfrak{Z}$. If $e = \max(X)$ then next place 0 on the top of $\mathfrak{Z}$. Else place $(\min\{b \in X : b > e\}, 1)$ on the top of $\mathfrak{Z}$. This completes the time-step.
2. $f = 2$: In this case we remove (e, f) from $\mathfrak{Z}$ and place $(e, 3)$ on the top of $\mathfrak{Z}$. If $e = \max(X)$ then next place 0 on the top of $\mathfrak{Z}$. Else place $(\min\{b \in X : b > e\}, 1)$ on the top of $\mathfrak{Z}$. This completes the time-step.
3. $f = 3$: In this case we remove (e, f) from $\mathfrak{Z}$ which completes the time-step.

The algorithm terminates when $\mathfrak{Z}$ becomes empty.

We call a time-step in a ghost-search a *leaf-step* if and only if at the start of the time-step we have that the top element of the stack, $\mathfrak{Z}$, is 0. Note that a ghost search of a set X simulates a depth-first search (in which, given an internal vertex v, $\Downarrow(\triangleleft(v))$ is explored before $\Downarrow(\triangleright(v)))$ of $\mathfrak{B}(\mathcal{P}(X))$ without having to store the whole tree in the memory. The time-steps in which (at the start of the time-step) (e, f) is on the top of the stack corresponds to the times when the depth first search is at some internal vertex v in $\mathfrak{B}(\mathcal{P}(X))$ with $\phi(v) = e$ and it is the f-th time that we have encountered v throughout the depth first search. The leaf-steps correspond to the times when the depth-first search is at the leaves of $\mathfrak{B}(\mathcal{P}(X))$. Hence, by the bijection τ (from the leaves of $\mathfrak{B}(\mathcal{P}(X))$ into $\mathcal{P}(X)$), we have a one to one correspondence between the leaf-steps and the sets in $\mathcal{P}(X)$.

Full-Search: Given a multi-set $\mathfrak{X}$ of straddle-trees (or info-trees, as every info-tree has an underlying straddle-tree), a *full-search* of $\mathfrak{X}$ is the following algorithm:

We first define X to be the set of all labels, $\phi(v)$, of the internal vertices, v, of the trees in $\mathfrak{X}$. Note that X can be found and ordered quickly. Note that before running the synchronised-search the set $\mathcal{L}$ is empty and the array $\mathcal{A}$ has the empty set for every element (see the start of this section). Let $\mathcal{R}$ be the set of roots of the trees in $\mathfrak{X}$. We initialise by, for every $r \in \mathcal{R}$, adding r to the set $\mathcal{A}(\phi(r))$. After this initialisation we perform a ghost search of X. Let $\mathfrak{Z}$ be the stack in the ghost search. At the end of every time-step in the ghost search we do the following:

1. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is 0 (i.e. the time-step is a leaf-step) then we do nothing.
2. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is $(e, 1)$ for some $e \in X$ then for every $v \in \mathcal{A}(e)$ we do the following: If $\triangleleft(v)$ is a leaf then we add $\triangleleft(v)$ to $\mathcal{L}$. Else we add $\triangleleft(v)$ to $\mathcal{A}(\phi(\triangleleft(v)))$.
3. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is $(e, 2)$ for some $e \in X$ then for every $v \in \mathcal{A}(e)$ we do the following: We first remove $\triangleleft(v)$ from $\mathcal{A}(\phi(\triangleleft(v)))$. If $\triangleright(v)$ is a leaf then we add $\triangleright(v)$ to $\mathcal{L}$. Else we add $\triangleright(v)$ to $\mathcal{A}(\phi(\triangleright(v)))$.
4. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is $(e, 3)$ for some $e \in X$ then for every $v \in \mathcal{A}(e)$ we remove $\triangleright(v)$ from $\mathcal{A}(\phi(\triangleright(v)))$.

Once the ghost search terminates, we set $\mathcal{A}(\min(X)) \leftarrow \emptyset$ and then the full-search terminates. Note that upon termination of the full-search we have that $\mathcal{L} = \emptyset$ and for all $e \in \mathbb{N}_n$ we have $\mathcal{A}(e) = \emptyset$ as required.

Given $Y_1, Y_2, ..., Y_a \subseteq \mathbb{N}_n$, a full-search of $\{\mathfrak{B}(\mathcal{P}(Y_i)) : i \in \mathbb{N}_a\}$ essentially does the following: Recall from above that given $X = \bigcup_{i=1}^{a} Y_i$ there is a one to one correspondence between the leaf-steps and sets in $\mathcal{P}(X)$. Suppose we are at a leaf-step. Let Z be the set in $\mathcal{P}(X)$ corresponding to the leaf-step. Then at the start of the leaf-step we have that $\mathcal{L}$ is equal to the set of leaves l in the trees $\{\mathfrak{B}(\mathcal{P}(Y_i)) : i \in \mathbb{N}_a\}$ such that, given l is a leaf of $\mathfrak{B}(\mathcal{P}(Y_j))$, we have $\tau(l) = Z \cap Y_j$.

Note that, given $Y_1, Y_2, ..., Y_a \subseteq \mathbb{N}_n$ and $y_1, y_2, ..., y_c \in \mathbb{N}_n$ with $\{y_i : i \in \mathbb{N}_c\} = \bigcup_{i=1}^{a} Y_i$ and $y_i < y_j$ for all $i, j \in \mathbb{N}_c$ with $i < j$, then a full-search of $\{\mathfrak{B}(\mathcal{P}(Y_i)) : i \in \mathbb{N}_a\}$ takes a time of $\Theta\left(\sum_{i=1}^{c} |\{j \in \mathbb{N}_a : y_i \in Y_j\}| 2^i\right)$ and that this is bounded above by $\mathcal{O}\left(a 2^c\right)$

Syncronised-Search: Given a multi-set $\mathfrak{X}$ of straddle-trees (or info-trees, as every info-tree has an underlying straddle-tree), a *synchronised-search* of $\mathfrak{X}$ is the following algorithm:
We first define X to be the set of all labels, $\phi(v)$, of the internal vertices, v, of the trees in $\mathfrak{X}$. Note that X can be found and ordered quickly. Note that before running the synchronised-search the set $\mathcal{L}$ is empty and the array $\mathcal{A}$ has the empty set for every element (see the start of this section). Let $\mathcal{R}$ be the set of roots of the trees in $\mathfrak{X}$. We initialise by, for every $r \in \mathcal{R}$, adding r to the set $\mathcal{A}(\phi(r))$. After this initialisation we perform a ghost search of X. Let $\mathfrak{Z}$ be the stack in the ghost search. At the end of every time-step in the ghost search we do the following:

1. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is 0 (i.e. the time-step is a leaf-step) then we set $\mathcal{L} \leftarrow \emptyset$.

2. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is $(e, 1)$ for some $e \in X$ then for every $v \in \mathcal{A}(e)$ we do the following: If $\triangleleft(v)$ is a leaf then we add $\triangleleft(v)$ to $\mathcal{L}$. Else we add $\triangleleft(v)$ to $\mathcal{A}(\phi(\triangleleft(v)))$.

3. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is $(e, 2)$ for some $e \in X$ then for every $v \in \mathcal{A}(e)$ we do the following: If $\triangleright(v)$ is a leaf then we add $\triangleright(v)$ to $\mathcal{L}$. Else we add $\triangleright(v)$ to $\mathcal{A}(\phi(\triangleright(v)))$.

4. If, at the start of the time-step, the top element of $\mathfrak{Z}$ is $(e, 3)$ for some $e \in X$ then we set $\mathcal{A}(e) \leftarrow \emptyset$.

Once the ghost search terminates, the synchronised-search also terminates. Note that upon termination of the synchronised-search we have that $\mathcal{L} = \emptyset$ and for all $e \in \mathbb{N}_n$ we have $\mathcal{A}(e) = \emptyset$ as required.

Given straddle-sets $\zeta_1, \zeta_2, ..., \zeta_a \subseteq \mathcal{P}(\mathbb{N}_n)$, a synchronised-search of $\{\mathfrak{B}(\zeta_i) : i \in \mathbb{N}_a\}$ essentially does the following: Recall from above that given $X = \bigcup_{i=1}^{a} (\bigcup \zeta_i)$ there is a one to one correspondence between the leaf-steps and sets in $\mathcal{P}(X)$. Suppose we are at a leaf-step. Let Z be the set in $\mathcal{P}(X)$ corresponding to the leaf-step. Then at the start of the leaf-step we have that $\mathcal{L}$ is equal to the set of leaves l in the trees $\{\mathfrak{B}(\zeta_i) : i \in \mathbb{N}_a\}$ such that $\tau(l) = Z$.

Note that given straddle sets $\zeta_1, \zeta_2, ..., \zeta_a \subseteq \mathcal{P}(\mathbb{N}_n)$, a synchronised-search of $\{\mathfrak{B}(\zeta_i) : i \in \mathbb{N}_a\}$ takes a time of $\mathcal{O}\left(2^{|\bigcup_{i=1}^{a}(\bigcup \zeta_i)|} + \sum_{i=1}^{a} |\zeta_i|\right)$. However, often much of the ghost-search underlying the synchronised-search is unnecessary, meaning that the additive factor of $\mathcal{O}\left(2^{|\bigcup_{i=1}^{a}(\bigcup \zeta_i)|}\right)$ can be reduced.

4.9.3 Implementing Algorithm 78

In this subsection let C, k, D_i, Υ_i, Γ and Ψ_i be as in the description of Operation 77. That is: C is a subset of $\mathbb{N}_n$. $D_1, ..., D_k$ are subsets of C with $\bigcup_{i=1}^{k} D_i = C$. Υ_i is a potential in $\mathcal{T}(D_i)$. $\Gamma := \prod_{i=1}^{k} \Upsilon_i$ and $\Psi_i := \Gamma^{\nabla D_i}$. The goal of Operation 77 is to compute Ψ_i for every $i \in \mathbb{N}_k$.

We first describe a simple implementation of Algorithm 78 that takes a time of $\Theta\left(k2^{|C|}\right)$:

We have, as input, the set, $\{\mathfrak{T}(\Upsilon_i) : i \in \mathbb{N}_k\}$. Initially, for every $i \in \mathbb{N}_k$ we set $\mathfrak{A}_i \leftarrow \mathfrak{B}(\mathcal{P}(D_i))$ (which is copied from $\mathfrak{T}(\Upsilon_i)$) and set $\psi(l) \leftarrow 0$ for every leaf l of $\mathfrak{A}_i$. We then do a full-search of $\{\mathfrak{T}(\Upsilon_i) : i \in \mathbb{N}_k\} \cup \{\mathfrak{A}_i : i \in \mathbb{N}_k\}$. At the start of every leaf-step in the full-search we do the following:

Let U be the set of leaves in $\mathcal{L}$ that are in the trees $\{\mathfrak{T}(\Upsilon_i) : i \in \mathbb{N}_k\}$ and let W be the set of leaves in $\mathcal{L}$ that are in the trees $\{\mathfrak{A}_i : i \in \mathbb{N}_k\}$. Set $\alpha \leftarrow \sum_{l \in U} \psi(l)$ and then set, for every $l \in W$, $\psi(l) \leftarrow \psi(l) + \alpha$.

After the full-search has terminated we have $\mathfrak{A}_i = \mathfrak{T}(\Psi_i)$ for every $i \in \mathbb{N}_k$.

We now describe how, by caching various quantities, Algorithm 78 can, while retaining the low space complexity, be sped up to take a time of only $\Theta\left(\sum_{i \in \mathbb{N}_{|C|}} |\{j \in \mathbb{N}_k : y_i \in D_j\}| 2^i\right)$ where y_i is the i-th least element of C:

We have, as input, the set, $\{\mathfrak{T}(\Upsilon_i) : i \in \mathbb{N}_k\}$. Initially, for every $i \in \mathbb{N}_k$ we set $\mathfrak{A}_i \leftarrow \mathfrak{B}(\mathcal{P}(D_i))$ (which is copied from $\mathfrak{T}(\Upsilon_i)$) and set $\psi(q) \leftarrow 0$ for every leaf q of $\mathfrak{A}_i$. We then do a full-search of $\{\mathfrak{T}(\Upsilon_i) : i \in \mathbb{N}_k\} \cup \{\mathfrak{A}_i : i \in \mathbb{N}_k\}$. For all $i \in \mathbb{N}_k$ let l_i^t (resp. q_i^t) be the leaf of $\mathfrak{T}(\Upsilon_i)$ (resp. $\mathfrak{A}_i$) that is in $\mathcal{L}$ at the start of the t-th leaf-step in the full-search. During the full-search, in addition to maintaining the variable α we also maintain a variable β as well as, for every $i \in \mathbb{N}_k$, a variable δ_i. At the start of the first leaf step we do the following:

1. For all $i \in \mathbb{N}_k$ set $\delta_i \leftarrow 0$

2. Set $\alpha \leftarrow \prod_{i \in \mathbb{N}_{|C|}} \psi(l_i^1)$
3. Set $\beta \leftarrow \alpha$

At the start of the t-th leaf-step, for $t > 1$, we do the following:

1. For all $i \in \mathbb{N}_k$ such that $q_i^t \neq q_i^{t-1}$ set $\psi(q_i^{t-1}) \leftarrow \psi(q_i^{t-1}) + \beta - \delta_i$.
2. For all $i \in \mathbb{N}_k$ such that $q_i^t \neq q_i^{t-1}$ set $\delta_i \leftarrow \beta$
3. Set $Q \leftarrow \{i \in \mathbb{N}_k : l_i^t \neq l_i^{t-1}\}$
4. Set $\alpha \leftarrow \alpha \prod_{i \in Q}(\psi(l_i^t)/\psi(l_i^{t-1}))$
5. Set $\beta \leftarrow \beta + \alpha$

Once the full-search has terminated we set $\psi(q_i^{|C|}) \leftarrow \psi(q_i^{|C|}) + \beta - \delta_i$ for every $i \in \mathbb{N}_k$. We then have $\mathfrak{A}_i = \mathfrak{T}(\Psi_i)$ for every $i \in \mathbb{N}_k$.

Note that, in the above implementation we can first re-order C in order to minimise the time. However, re-ordering C means that we must re-construct the info-trees Υ_i to be consistent with the new order which takes a time of $\Theta\left(|D_i|2^{|D_i|}\right)$ for every $i \in \mathbb{N}_k$. Note also that since the above implementation involves division we should use MZC-numbers (see Section 4.8) instead of real numbers to avoid division by zero.

4.9.4 Implementing the Functions of ARCH-2

In this subsection we describe the implementation of the functions described in Section 4.7.3. The notation of this subsection is as in the description of the algorithms in Section 4.7.3. First note that in Step 1 of Algorithm 89, Step 1 of Algorithm 91 and Step 1 of Algorithm 93 we are asked to choose $x \in \sigma(\Phi)$. In these times we will always choose to set $x \leftarrow \min(\sigma(\Phi))$.

Steps 2, 3 and 4 of Algorithm 91 are performed together as follows:

First define r to be the root of $\mathfrak{T}(\#\Phi, \zeta)$. Note that the straddle-tree underlying $\Downarrow(\triangleleft(r))$ is $\mathfrak{B}(\vartheta)$ so we can copy this tree and set $\mathfrak{A}_- \leftarrow \mathfrak{B}(\vartheta)$ and $\mathfrak{A}_+ \leftarrow \mathfrak{B}(\vartheta)$. We then do a synchronised-search of $\{\mathfrak{A}_-, \Downarrow(\triangleleft(r)), \Downarrow(\triangleright(r))\}$ (resp. $\{\mathfrak{A}_+, \Downarrow(\triangleleft(r)), \Downarrow(\triangleright(r))\}$).
At the start of every leaf-step we do the following:
If $\mathcal{L}$ doesn't contain a leaf of $\mathfrak{A}_-$ (resp. $\mathfrak{A}_+$) we do nothing. Else, we have two cases:

1. $\mathcal{L}$ contains a leaf of $\Downarrow(\triangleright(r))$: In this case we have $\mathcal{L} = \{l_0, l_1, l_2\}$ where l_0 is a leaf of $\mathfrak{A}_-$ (resp. $\mathfrak{A}_+$), l_1 is a leaf of $\Downarrow(\triangleleft(r))$ and l_2 is a leaf of $\Downarrow(\triangleright(r))$. We set $\psi(l_0) \leftarrow \psi(l_1)$ (resp. $\psi(l_0) \leftarrow \psi(l_1)/\psi(l_2)$)
2. $\mathcal{L}$ doesn't contain a leaf of $\Downarrow(\triangleright(r))$: In this case we have $\mathcal{L} = \{l_0, l_1\}$ where l_0 is a leaf of $\mathfrak{A}_-$ (resp. $\mathfrak{A}_+$) and l_1 is a leaf of $\Downarrow(\triangleleft(r))$. We set $\psi(l_0) \leftarrow \psi(l_1)$ (resp. $\psi(l_0) \leftarrow \psi(l_1)$).

After the synchronised searches we have $\mathfrak{A}_- = \mathfrak{T}(\#\Phi_-, \vartheta)$ and $\mathfrak{A}_+ = \mathfrak{T}(\#\Phi_+, \vartheta)$

Step 2 of Algorithm 89 and Step 2 of Algorithm 93 are performed similarly (with $\mathfrak{T}(\Phi)$ or $\mathfrak{T}(\%\Phi)$ instead of $\mathfrak{T}(\#\Phi, \zeta)$ and $\mathcal{P}(\sigma(\Phi) \setminus \{x\})$ instead of ϑ).

Steps 6 and 7 of Algorithm 91 are performed together as follows:
First set $\mathfrak{A} \leftarrow \mathfrak{B}(\zeta)$ (copied from the input). Let r be the root of $\mathfrak{A}$. Do a synchronised search of $\{\Downarrow(\triangleleft(r)), \mathfrak{T}(\%\Phi_-, \vartheta), \mathfrak{T}(\%\Phi_+, \vartheta)\}$
(resp. $\{\Downarrow(\triangleright(r)), \mathfrak{T}(\%\Phi_-, \vartheta), \mathfrak{T}(\%\Phi_+, \vartheta)\}$). At the start of every leaf-step we do the following:

If $\mathcal{L}$ doesn't contain a leaf of $\mathfrak{A}$ then we do nothing. Otherwise we have that $\mathcal{L} = \{l_0, l_1, l_2\}$ where l_0 is a leaf of $\mathfrak{A}$, l_1 is a leaf of $\mathfrak{T}(\%\Phi_-, \vartheta)$ and l_2 is a leaf $\mathfrak{T}(\%\Phi_+, \vartheta)$. In these cases we set $\psi(l_0) \leftarrow \psi(l_1) + \psi(l_2)$ (resp. $\psi(l_0) \leftarrow \psi(l_2)$).
After the synchronised searches we have $\mathfrak{A} = \mathfrak{T}(\Phi, \zeta)$

Step 4 of Algorithm 89 and Step 4 of Algorithm 93 are performed similarly (with $\mathcal{P}(\sigma(\Phi))$ instead of ζ).

Step 1 of Algorithm 90 requires us to construct $\mathfrak{B}(\zeta)$ where $\zeta := \bigcup_{i=1}^{k}\{\mathcal{P}(\sigma(\#\Upsilon_i))\}$. We do this as follows:
We maintain (and grow) a subtree $\mathfrak{A}$ of $\mathfrak{B}(\zeta)$ initialised to contain the root, r, as a single vertex (with $\phi(r) \leftarrow \min\bigcup\zeta$). At every point in the algorithm there is a single vertex of $\mathfrak{A}$ that is designated as the *active vertex*. Also, at every point in the algorithm we say that the active vertex is either *left-oriented* or *right-oriented*. We initialise such that the vertex r is the active vertex and is left-oriented. After this initialisation we do a synchronised-search of $\{\mathfrak{T}(\#\Upsilon_i) : i \in \mathbb{N}_k\}$. Let $\mathfrak{Z}$ be the stack used in the synchronised search. At the start of every time-step after the first we do the following:

1. If the top element of $\mathfrak{Z}$ is 0 (i.e. the time-step is a leaf-step) then do the following: If the active vertex v is currently left-oriented then add a vertex w to $\mathfrak{A}$ such that $w = \triangleleft(v)$. If the active vertex v is currently right-oriented then add a vertex w to $\mathfrak{A}$ such that $w = \triangleright(v)$. It is the case that w is a leaf of $\mathfrak{B}(\zeta)$.
2. If the top element of $\mathfrak{Z}$ is $(e, 1)$ for some $e \in \mathbb{N}_n$ then do the following: If $\mathcal{A}(e) = \emptyset$ then do nothing. Otherwise, given that the active vertex is currently v and is currently left-oriented (resp. right-oriented), we add a vertex w to $\mathfrak{A}$ such that $w = \triangleleft(v)$ (resp. $w = \triangleright(v)$) and set $\phi(w) \leftarrow e$. We then make w the active vertex and designate it as left-oriented.
3. If the top element of $\mathfrak{Z}$ is $(e, 2)$ for some $e \in \mathbb{N}_n$ then do the following: If $\mathcal{A}(e) = \emptyset$ then do nothing. Otherwise, given that the active vertex is currently v, we keep v as the active vertex but now designate it as right-oriented.
4. If the top element of $\mathfrak{Z}$ is $(e, 3)$ for some $e \in \mathbb{N}_n$ then do the following: If $\mathcal{A}(e) = \emptyset$ then do nothing. Otherwise, given v is currently the active vertex, we let the parent of v become the active vertex (it doesn't matter whether it is left-oriented or right-oriented).

After the synchronised-search is complete we have that $\mathfrak{A} = \mathfrak{B}(\zeta)$

After we have constructed $\mathfrak{B}(\zeta)$, Step 2 of Algorithm 90 is performed as follows: Set $\mathfrak{A} \leftarrow \mathfrak{B}(\zeta)$. Do a synchronised-search of $\{\mathfrak{A}\} \cup \{\mathfrak{T}(\#\Upsilon_i) : i \in \mathbb{N}_k\}$. Whenever we are at the start of a leaf-step such that there exists a leaf, l, of $\mathfrak{A}$ in $\mathcal{L}$ we set $\psi(l) \leftarrow \prod_{s\in\mathcal{L}\setminus\{l\}} \psi(s)$. After the synchronised search we have that $\mathfrak{A} = \mathfrak{T}(\#\Gamma, \zeta)$

Algorithm 92 is implemented as follows: For every $i \in \mathbb{N}_k$ set $\mathfrak{A}_i \leftarrow \mathfrak{B}(\mathcal{P}(D_i))$. Do a synchronised-search of $\{\mathfrak{T}(\%\Gamma, \zeta)\} \cup \{\mathfrak{A}_i : i \in \mathbb{N}_k\}$. Whenever we are at the start of a leaf-step such that there exists a leaf, l, of $\mathfrak{T}(\%\Gamma, \zeta)$ in $\mathcal{L}$ we set, for every leaf $s \in \mathcal{L} \setminus \{l\}$, $\psi(s) \leftarrow \psi(l)$. After the synchronised-search we have that $\mathfrak{A}_i = \mathfrak{T}(\%\Psi_i)$

Chapter 5

Online Bayesian Classification in Ising Models

5.1 Abstract

In this chapter we first define the general Ising model, show how it comes from a natural process and show that the Bayes classifier minimises the expected number of mistakes in an online learning game with the Ising model. We show how to convert the junction tree algorithm so that it implements efficient online Bayesian classification in general Ising models. We give the online versions of Shafer-Shenoy propagation/ARCH-1, Hugin propagation and the novel architecture ARCH-2. We show that online Shafer-Shenoy is more space efficient than online Hugin but at the expense of a higher time complexity. We then show that the online version of ARCH-2 essentially achieves the best of both worlds: it has, up to a factor linear in the width of the junction tree, the speed of online Hugin propagation and the space efficiency of online Shafer-Shenoy propagation. Finally we develop an algorithm that, for any tree structured Ising model, constructs, in linear time and space, a junction tree that gives optimal time per prediction for online Hugin/ARCH-2. We show that this prediction time is logarithmic in the maximum cardinality of a binary subtree of the tree (that is, a subtree for which every vertex has degree of at most three). The space requirement for Hugin/ARCH-2 with the junction tree is linear in the cardinality of the tree which is optimal for the online junction tree algorithm.
The final section of this chapter is based on our paper "Online sum-product computations over trees" in NIPS 2012 which was written in collaboration with Mark Herbster and Fabio Vitale. All other sections in this chapter are all my own work.

5.2 Related Work

The idea of performing, for general Ising models, online Bayesian inference by updating messages along paths of a junction tree was essentially studied in [2] where they use an approach similar to online Shafer-Shenoy (see below). In [2] they also consider updating the data structure when edges of the graph are inserted or deleted whilst we consider only the case for which the graph structure is fixed. Previous work relating to Section 5.6 will be given in that section.

5.3 Notation

In this chapter we will use the notation and terminology of the last chapter. However, we will, in this chapter, call the vertices of a junction tree *super-vertices* to avoid confusion with the vertices of a standard graph.

5.4 Ising Models and the Bayes' Classifier

In this section we introduce the online learning problem that we will solve, show that it is solved via Bayesian classification and then reduce this solution to a database update/query. We start by introducing the Ising model.

5.4.1 The General Ising Model

In this subsection we introduce the general (note that this is more general than the standard definition) binary Ising model (from here referred to as simply "Ising model") and show how it occurs from a very natural process.

Definition 96. The Ising Model: *An Ising model on a graph G is a probability distribution on the set of possible binary labellings, μ, of G given by:*

$$\mathbb{P}(\mu) \propto \left(\prod_{(v,w)\in\mathcal{E}(G)} \theta_{(v,w)}(\mu(v),\mu(w)) \right) \left(\prod_{v\in\mathcal{V}(G)} \theta_v(\mu(v)) \right) \tag{5.1}$$

for some $\theta_{(v,w)} : \{0,1\}^2 \to [0,1]$ and some $\theta_v : \{0,1\} \to [0,1]$.

We now show how the Ising model comes directly from a process in which people "meet" independently from there musical tastes and then those that meet may form "friendships" depending on their musical tastes.

We have a set X (e.g. of people). Every $x \in X$ has a stochastic binary label $\kappa(x) \in \{0,1\}$ (e.g. his/her musical taste) such that the labels $\kappa(x)$ are mutually independent. We have stochastic sets $A, B \subset \{\{x,y\} : x,y \in X, x \neq y\}$ (e.g. whether people meet and whether people become friends respectively). We have that A is independent of the labelling κ (e.g. people meet independently of their musical tastes). We have that $B \subseteq A$ (e.g. for people to become friends they first must meet) and for all $x,y \in X$ the event that $\{x,y\} \in B$ is independent of the labels $\{\kappa(z) : z \neq x,y\}$ and the set A given $\{x,y\} \in A$ (e.g. whether or not people become friends after meeting is only dependant on who they are and their musical tastes). For all $x,y \in X$ the events $\{x,y\} \in B$ are also mutually independent given A and κ (e.g. whether or not two people who meet make friends does not influence whether two other people make friends). We are given the sets A and B (e.g. we know who's met and who's become friends) but we don't know κ (e.g. we don't know people's musical tastes).

Define $\mathbb{P}(\mu) := \mathbb{P}(\kappa = \mu | A, B)$ for all binary labellings μ. We now show that $\mathbb{P}(\mu)$ is an Ising model on the graph G defined by $\mathcal{V}(G) := X$ and $\mathcal{E}(G) := A$. To see this let $\hat{\theta}_{(x,y)}(i,j) := \mathbb{P}(\{x,y\} \in B | \{x,y\} \in A, \kappa(x) = i, \kappa(y) = j)$ for all $x,y \in X$ and $i,j \in \{0,1\}$. We have:

$$\mathbb{P}(\kappa = \mu | B = B', A = A') \tag{5.2}$$
$$\propto \mathbb{P}(B = B' | \kappa = \mu, A = A')\mathbb{P}(\kappa = \mu | A = A') \tag{5.3}$$
$$\propto \mathbb{P}(B = B' | \kappa = \mu, A = A')\mathbb{P}(A = A' | \kappa = \mu)\mathbb{P}(\kappa = \mu) \tag{5.4}$$
$$= \mathbb{P}(B = B' | \kappa = \mu, A = A')\mathbb{P}(A = A')\mathbb{P}(\kappa = \mu) \tag{5.5}$$
$$\propto \mathbb{P}(B = B' | \kappa = \mu, A = A')\mathbb{P}(\kappa = \mu) \tag{5.6}$$
$$= \left(\prod_{\{x,y\} \in B'} \mathbb{P}(\{x,y\} \in B | A = A', \kappa = \mu)\right)\left(\prod_{\{x,y\} \in A' \setminus B'} \mathbb{P}(\{x,y\} \notin B | A = A', \kappa = \mu)\right)\mathbb{P}(\kappa = \mu) \tag{5.7}$$
$$= \left(\prod_{\{x,y\} \in B'} \mathbb{P}(\{x,y\} \in B | \{x,y\} \in A, \kappa = \mu)\right)\left(\prod_{\{x,y\} \in A' \setminus B'} \mathbb{P}(\{x,y\} \notin B | \{x,y\} \in A, \kappa = \mu)\right)\mathbb{P}(\kappa = \mu) \tag{5.8}$$
$$= \left(\prod_{\{x,y\} \in B'} \mathbb{P}(\{x,y\} \in B | \{x,y\} \in A, \kappa = \mu)\right)\left(\prod_{\{x,y\} \in A' \setminus B'} (1 - \mathbb{P}(\{x,y\} \in B | \{x,y\} \in A, \kappa = \mu))\right)\mathbb{P}(\kappa = \mu \tag{5.9}$$
$$= \left(\prod_{\{x,y\} \in B'} \hat{\theta}_{(x,y)}(\mu(x), \mu(y))\right)\left(\prod_{\{x,y\} \in A' \setminus B'} (1 - \hat{\theta}_{(x,y)}(\mu(x), \mu(y)))\right)\mathbb{P}(\kappa = \mu) \tag{5.10}$$
$$= \left(\prod_{\{x,y\} \in B'} \hat{\theta}_{(x,y)}(\mu(x), \mu(y))\right)\left(\prod_{\{x,y\} \in A' \setminus B'} (1 - \hat{\theta}_{(x,y)}(\mu(x), \mu(y)))\right)\left(\prod_{x \in X} \mathbb{P}(\kappa(x) = \mu(x))\right) \tag{5.11}$$

which is the Ising model on G (where G is defined by $\mathcal{V}(G) := X$ and $\mathcal{E}(G) := A'$) defined by (see definition 96) $\theta_v(i) = \mathbb{P}(\kappa(v) = i)$ and $\theta_{(v,w)}(i,j) = \hat{\theta}_{(x,y)}(i,j)$ if $\{x,y\} \in B'$ and $\theta_{(v,w)}(i,j) = 1 - \hat{\theta}_{(x,y)}(i,j)$ if $\{x,y\} \notin B'$.

Equations 5.3 and 5.4 are Bayes' rule. Equation 5.5 is since A is independent of the labelling κ. Equation 5.7 is since $B \subseteq A$, and since the events $\{x,y\} \in B$ are mutually independent given A and κ. Equation 5.8 is because for all $x, y \in X$ the event that $\{x,y\} \in B$ is independent of the set A given $\{x,y\} \in A$. Equation 5.10 is since for all $x, y \in X$ the event that $\{x,y\} \in B$ is independent of the labels $\{\kappa(z) : z \neq x, y\}$ given $\{x,y\} \in A$. Equation 5.11 is since the labels $\kappa(x)$ are mutually independent.

5.4.2 The Problem

The online learning problem of this chapter is as follows: We have an Ising model (on a graph G) which is known to learner and nature. Before the online learning game a labelling of the graph is drawn from the Ising model. Neither nature nor learner know the labelling. On each trial t, nature chooses a vertex x_t and queries the learner with it. The learner then gives its prediction $\hat{y}_t$ of the (noisy) label of vertex x_t. A noisy label, y_t, of x_t (note that both nature and learner know the noise distribution from the start) is then revealed to both nature and learner. The learner makes a mistake on trial t if and only if $\hat{y}_t \neq y_t$. The aim for the learner is to minimise the expected number of mistakes (we will give a strategy for the learner that, whatever nature's strategy, the expected number of mistakes will be minimised - the learner need not know natures strategy). Formally, the problem is as follows:

We first have a graph G and a binary Ising model on G: that is, a probability distribution on the set of possible binary labellings, μ, of G given by $\mathbb{P}(\mu) \propto \left(\prod_{(v,w) \in \mathcal{E}(G)} \theta_{(v,w)}(\mu(v), \mu(w))\right)\left(\prod_{v \in \mathcal{V}(G)} \theta_v(\mu(v))\right)$ for some $\theta_{(v,w)} : \{0,1\}^2 \rightarrow$

$[0,1]$ and some $\theta_v : \{0,1\} \to [0,1]$. At every trial t we have a (vertex/label) pair $o_t := (x_t, y_t)$ where $x_t \in \mathcal{V}(G)$ (i.e. x_t is the query vertex at trial t) and $y_t \in \{0,1\}$ (i.e. y_t is the observed (noisy) label of x_t on trial t). For any vertex w we have that the event $x_t = w$ is independent of μ given $o_1, o_2, ..., o_{t-1}$ (since nature doesn't know the labelling μ a-priori but only receives the observed label y_t on each trial t). For any $i \in \{0,1\}$ we have that the event $y_t = i$ is independent of $o_1, o_2, ..., o_{t-1}$ given x_t and μ. We also have a function $\epsilon : \{0,1\}^2 \to [0,1]$ (the noise) such that, for $i \in \{0,1\}$ we have $\mathbb{P}(y_t = i|\mu, x_t) \propto \epsilon(\mu(x_t), i)$. Note that we can generalise ϵ by making it a function also of t and x_t without much change in what follows.

At every trial t we have a value $\hat{y}_t \in \{0,1\}$ which is a function (the learners strategy) of $\{\theta_{(v,w)} : (v,w) \in \mathcal{E}(G)\}$, $\{\theta_v : v \in \mathcal{V}(G)\}$, ϵ, x_t and $\{o_s : s < t\}$ (since this is the only information revealed to the learner by trial t). The objective of the learner is to choose this function such that the expected number of mistakes (i.e. those trials in which $\hat{y}_t \neq y_t$) is minimised, and then to compute it efficiently at every trial.

5.4.3 The Bayes Classifier

Here we introduce the Bayes classifier and show that it minimises the expected number of mistakes:

Definition 97. *The Bayes classifier predicts* $\hat{y}_t := \operatorname{argmax}_{i\in\{0,1\}} \mathbb{P}(y_t = i|o_1, o_2, ..., o_{t-1}, x_t)$.

We now show that the Bayes classifier minimises the expected number of mistakes. To see this let T be the total number of trials. For $t \leq T$ let $\mathcal{M}_t$ be the cumulative mistakes up until (and including) trial t. For $t \leq T$ let $\mathcal{D}_t := (o_1, o_2, ..., o_t)$. We prove by reverse induction (i.e. from $t := T$ to $t := 0$) that the Bayes classifier minimises $\mathbb{E}(\mathcal{M}_T - \mathcal{M}_t|\mathcal{D}_t)$. This clearly holds in the case that $t := T$ since $\mathbb{E}(\mathcal{M}_T - \mathcal{M}_T|\mathcal{D}_T) = 0$ whatever the strategy. So now suppose it holds for some $t \leq T$. We now show it holds for $t-1$. We have:

$$\mathbb{E}(\mathcal{M}_T - \mathcal{M}_{t-1}|\mathcal{D}_{t-1}) = \sum_{o_t} \mathbb{P}(o_t|\mathcal{D}_{t-1})\mathbb{E}(\mathcal{M}_T - \mathcal{M}_{t-1}|o_t, \mathcal{D}_{t-1}) \tag{5.12}$$

$$= \sum_{o_t} \mathbb{P}(o_t|\mathcal{D}_{t-1})\mathbb{E}((\mathcal{M}_T - \mathcal{M}_t) + (\mathcal{M}_t - \mathcal{M}_{t-1})|o_t, \mathcal{D}_{t-1}) \tag{5.13}$$

$$= \sum_{o_t} \mathbb{P}(o_t|\mathcal{D}_{t-1})(\mathbb{E}(\mathcal{M}_T - \mathcal{M}_t|o_t, \mathcal{D}_{t-1}) + \mathbb{E}(\mathcal{M}_t - \mathcal{M}_{t-1}|o_t, \mathcal{D}_{t-1})) \tag{5.14}$$

Note that the term $\mathbb{E}(\mathcal{M}_T - \mathcal{M}_t|o_t, \mathcal{D}_{t-1})$ appearing in Equation 5.14 is equal to $\mathbb{E}(\mathcal{M}_T - \mathcal{M}_t|\mathcal{D}_t)$ which, by the inductive hypothesis is minimised by the Bayes classifier. Hence, we now need only show that the Bayes classifier minimises the quantity $\sum_{o_t} \mathbb{P}(o_t|\mathcal{D}_{t-1})\mathbb{E}(\mathcal{M}_t - \mathcal{M}_{t-1}|o_t, \mathcal{D}_{t-1})$ so, since $\mathbb{E}(\mathcal{M}_t - \mathcal{M}_{t-1}|o_t, \mathcal{D}_{t-1}) = \mathbb{P}(\hat{y}_t \neq y_t|o_t, \mathcal{D}_{t-1}))$ we need only show that the Bayes classifier minimises the quantity $\sum_{o_t} \mathbb{P}(o_t|\mathcal{D}_{t-1})\mathbb{P}(\hat{y}_t \neq y_t|o_t, \mathcal{D}_{t-1}))$. We have:

$$\sum_{o_t} \mathbb{P}(o_t|\mathcal{D}_{t-1})\mathbb{P}(\hat{y}_t \neq y_t|o_t, \mathcal{D}_{t-1})) \tag{5.15}$$

$$= \sum_{x_t\in\mathcal{V}(G)} \sum_{y_t\in\{0,1\}} \mathbb{P}(x_t, y_t|\mathcal{D}_{t-1})\mathbb{P}(\hat{y}_t \neq y_t|o_t, \mathcal{D}_{t-1}) \tag{5.16}$$

$$= \sum_{x_t\in\mathcal{V}(G)} \sum_{y_t\in\{0,1\}} \mathbb{P}(x_t|\mathcal{D}_{t-1})\mathbb{P}(y_t|x_t, \mathcal{D}_{t-1})\mathbb{P}(\hat{y}_t \neq y_t|o_t, \mathcal{D}_{t-1}) \tag{5.17}$$

$$= \sum_{x_t\in\mathcal{V}(G)} \mathbb{P}(x_t|\mathcal{D}_{t-1}) \sum_{y_t\in\{0,1\}} \mathbb{P}(y_t|x_t, \mathcal{D}_{t-1})\mathbb{P}(\hat{y}_t \neq y_t|o_t, \mathcal{D}_{t-1}) \tag{5.18}$$

Since $\hat{y}_t$ is a function of x_t we can minimise each of the terms $\sum_{y_t\in\{0,1\}} \mathbb{P}(y_t|x_t, \mathcal{D}_{t-1})\mathbb{P}(\hat{y}_t \neq y_t|o_t, \mathcal{D}_{t-1}))$ individually. Since $\hat{y}_t$ is a function of only $\mathcal{D}_{t-1}$ and x_t we have that

$\sum_{y_t\in\{0,1\}}\mathbb{P}(y_t|x_t,\mathcal{D}_{t-1})\mathbb{P}(\hat{y}_t \neq y_t|o_t,\mathcal{D}_{t-1})) = \mathbb{P}(y_t \neq \hat{y}_t|x_t,\mathcal{D}_{t-1})$ which is minimised by setting $\hat{y}_t := \operatorname{argmax}_{i\in\{0,1\}}\mathbb{P}(y_t = i|x_t,\mathcal{D}_{t-1})$ which is the Bayes classifier. This completes the proof of the inductive hypothesis.

Since $\mathcal{M}_T = \mathcal{M}_T - \mathcal{M}_0$ and $\mathcal{D}_0 = \emptyset$ we have that $\mathbb{E}(\mathcal{M}_T) = \mathbb{E}(\mathcal{M}_T - \mathcal{M}_0|\mathcal{D}_0)$ which, by above, is minimised by the Bayes classifier. This completes the proof.

5.4.4 An evolving factorisation

To predict using the Bayes classifier we need to compute the probability $\mathbb{P}(y_t = i|o_1,o_2,...,o_{t-1},x_t)$ for $i\in\{0,1\}$. In this section we show how this is done by computing marginals in an evolving factorisation.

We first inductively compute $\mathbb{P}(\mu|o_1,o_2,...,o_t)$. For $t=0$ this quantity is equal to $\mathbb{P}(\mu)$ which is proportional to $\left(\prod_{(v,w)\in\mathcal{E}(G)}\theta_{(v,w)}(\mu(v),\mu(w))\right)\left(\prod_{v\in\mathcal{V}(G)}\theta_v(\mu(v))\right)$. For $t>0$ we have:

$$\begin{aligned}
\mathbb{P}(\mu|o_1,o_2,...,o_t) &= \mathbb{P}(\mu|y_t,x_t,o_1,o_2,...,o_{t-1}) &(5.19)\\
&\propto \mathbb{P}(y_t|\mu,x_t,o_1,o_2,...,o_{t-1})\mathbb{P}(\mu|x_t,o_1,o_2,...,o_{t-1}) &(5.20)\\
&= \mathbb{P}(y_t|\mu,x_t)\mathbb{P}(\mu|x_t,o_1,o_2,...,o_{t-1}) &(5.21)\\
&\propto \epsilon(\mu(x_t),y_t)\mathbb{P}(\mu|x_t,o_1,o_2,...,o_{t-1}) &(5.22)\\
&\propto \epsilon(\mu(x_t),y_t)\mathbb{P}(x_t|\mu,o_1,o_2,...,o_{t-1})\mathbb{P}(\mu|o_1,o_2,...,o_{t-1}) &(5.23)\\
&= \epsilon(\mu(x_t),y_t)\mathbb{P}(x_t|o_1,o_2,...,o_{t-1})\mathbb{P}(\mu|o_1,o_2,...,o_{t-1}) &(5.24)\\
&\propto \epsilon(\mu(x_t),y_t)\mathbb{P}(\mu|o_1,o_2,...,o_{t-1}) &(5.25)
\end{aligned}$$

Equation 5.20 is Bayes' rule. Equation 5.21 comes from the fact that for any $i\in\{0,1\}$ we have that the event $y_t = i$ is independent of $o_1,o_2,...,o_{t-1}$ given x_t and μ. Equation 5.23 is Bayes' rule. Equation 5.24 comes from the fact that for any vertex w we have that the event $x_t = w$ is independent of μ given $o_1,o_2,...,o_{t-1}$.

Hence, by induction on t we can write
$\mathbb{P}(\mu|o_1,o_2,...,o_t) \propto \left(\prod_{(v,w)\in\mathcal{E}(G)}\theta_{(v,w)}(\mu(v),\mu(w))\right)\left(\prod_{v\in\mathcal{V}(G)}\theta_v^t(\mu(v))\right)$ where $\theta_v^0 := \theta_v$ and for all $t>0$ we have $\theta_v^t := \theta_v^{t+1}$ if $v\neq x_t$ and $\theta_v^t(\mu(v)) := \epsilon(\mu(v),y_t)\theta_v^{t+1}$ if $v = x_t$.

We can hence store the probability distribution $\mathbb{P}(\mu|o_1,o_2,...,o_t)$ as a factorisation $\mathcal{F}_t$ which contains, for every edge $(v,w)\in\mathcal{E}(G)$ a potential in $\Lambda_{(v,w)}\in\mathcal{T}(\{v,w\})$ corresponding to $\theta_{(v,w)}$ (i.e. $\Lambda_{(v,w)}(\emptyset) := \theta_{(v,w)}(0,0)$, $\Lambda_{(v,w)}(\{v\}) := \theta_{(v,w)}(1,0)$, $\Lambda_{(v,w)}(\{w\}) := \theta_{(v,w)}(0,1)$ and $\Lambda_{(v,w)}(\{v,w\}) := \theta_{(v,w)}(1,1)$) and, for every vertex $v\in\mathcal{V}(G)$ a potential Λ_v in $\mathcal{T}(\{v\})$ coresponding to θ_v^t (i.e. $\Lambda_v(\emptyset) = \theta_v^t(0)$ and $\Lambda_v(\{v\}) = \theta_v^t(1)$). Note that on going from $\mathcal{F}_{t-1}$ to $\mathcal{F}_t$ we only have to update the potential Λ_{x_t}.

We also have:

$$\begin{aligned}
\mathbb{P}(y_t = i|o_1, o_2, ..., o_{t-1}, x_t) &= \sum_{\mu} \mathbb{P}(y_t = i|\mu, o_1, o_2, ..., o_{t-1}, x_t)\mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}, x_t) && (5.26)\\
&= \sum_{\mu} \mathbb{P}(y_t = i|\mu, x_t)\mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}, x_t) && (5.27)\\
&\propto \sum_{\mu} \epsilon(\mu(x_t), i)\mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}, x_t) && (5.28)\\
&\propto \sum_{\mu} \epsilon(\mu(x_t), i)\mathbb{P}(x_t|\mu, o_1, o_2, ..., o_{t-1})\mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}) && (5.29)\\
&= \sum_{\mu} \epsilon(\mu(x_t), i)\mathbb{P}(x_t|o_1, o_2, ..., o_{t-1})\mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}) && (5.30)\\
&\propto \sum_{\mu} \epsilon(\mu(x_t), i)\mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}) && (5.31)\\
&= \sum_{j\in\{0,1\}} \sum_{\mu:\mu(x_t)=j} \epsilon(j, i)\mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}) && (5.32)\\
&= \sum_{j\in\{0,1\}} \epsilon(j, i) \sum_{\mu:\mu(x_t)=j} \mathbb{P}(\mu|o_1, o_2, ..., o_{t-1}) && (5.33)\\
&= \sum_{j\in\{0,1\}} \epsilon(j, i)\mathbb{P}(\mu(x_t) = j|o_1, o_2, ..., o_{t-1}) && (5.34)\\
&= \epsilon(0, i)\rho_{x_t}(\emptyset) + \epsilon(1, i)\rho_{x_t}(\{x_t\}) && (5.35)
\end{aligned}$$

where ρ_{x_t} is the result of the junction tree algorithm when run with factorisation $\mathcal{F}_{t-1}$ and the sums are over all labellings μ. Equation 5.26 is the law of total probability. Equation 5.27 comes from the fact that for any $i \in \{0, 1\}$ we have that the event $y_t = i$ is independent of $o_1, o_2, ..., o_{t-1}$ given x_t and μ. Equation 5.29 is Bayes' rule. Equation 5.30 comes from the fact that for any vertex w we have that the event $x_t = w$ is independent of μ given $o_1, o_2, ..., o_{t-1}$.

5.5 The Online Junction Tree Algorithm

From the proceeding subsection we see that online Bayesian classification in the Ising model can be solved via the following database update/query:

1. **Initialisation:** We have a set $\mathcal{F}$ of the following potentials:
   (a) For every edge $(v, w) \in \mathcal{E}(G)$ we have a potential $\Lambda_{(v,w)} \in \mathcal{T}(\{v, w\})$.
   (b) For every vertex $v \in \mathcal{V}(G)$ we have a potential $\Lambda_v \in \mathcal{T}(\{v\})$
2. **Update:** Change the potential Λ_v for some vertex $v \in \mathcal{V}(G)$
3. **Query:** Compute the potential ρ_v for some vertex v where ρ_v is the result of the junction tree algorithm when run with factorisation $\mathcal{F}$.

where the time required per trial is the maximum time taken for both update and query at some vertex v.

Note that the computation of a query at vertex v could be done by running the junction tree algorithm on factorisation $\mathcal{F}$. Recall that for every potential $\Lambda \in \mathcal{F}$ the junction tree must contain a super-vertex Λ^+ that has $\sigma(\Lambda)$ as a subset. This motivates the following definition:

Definition 98. Junction tree of a graph *A junction tree $\mathcal{J}$ of a graph G is a tree that satisfies the following axioms:*

1. *Every vertex of $\mathcal{J}$ is a subset of $\mathcal{V}(G)$ (note that in this chapter we call the vertices of $\mathcal{J}$ super-vertices).*

2. $\bigcup \mathcal{V}(\mathcal{J}) = \mathcal{V}(G)$
3. *For every edge $(v, w) \in \mathcal{E}(G)$ we have a super-vertex of $\mathcal{J}$ that contains both v and w.*
4. *(Running intersection property:) For every $C, D \in \mathcal{V}(\mathcal{J})$ and $v \in C \cap D$ we have that v is contained in every super-vertex in the path from C to D.*

i.e. $\mathcal{J}$ is a junction tree on $\mathcal{V}(G)$ such that for every edge $(v, w) \in \mathcal{E}(G)$ we have a super-vertex of $\mathcal{J}$ that contains both v and w.

In this section we assume that we have a rooted junction tree $\mathcal{J}$ of the graph G. Let R be the root of $\mathcal{J}$. As an initialisation step we assign, for every potential $\Lambda \in \mathcal{F}$, a super-vertex $\Lambda^+ \in \mathcal{V}(\mathcal{J})$ that has $\sigma(\Lambda)$ as a subset. As in the previous chapter, for every super-vertex $C \in \mathcal{V}(\mathcal{J})$ we define $\mathcal{F}(C) := \{\Lambda \in \mathcal{F} : \Lambda^+ = C\}$. We denote ${\Lambda_v}^+$ by v^+ for all $v \in \mathcal{V}(G)$
As was stated above, the computation of a query at vertex v could be done by running the junction tree algorithm on factorisation $\mathcal{F}$. However, by splitting up the work between update and query we can be much faster:

1. To initialise we run the junction tree algorithm of factorisation $\mathcal{F}$ to compute all messages. We define an *inward message* (resp. *outward message*) to be a message $M_{C \to D}$ for which C is a child of D (resp. D is a child of C).
2. On update at a vertex v we update the inward messages to be consistent with the new factorisation. Note that since all the inward messages were correct before the change in Λ_v we only have update all the inward messages along the path from v^+ to the root.
3. On query at a vertex v we compute ρ_v by updating, in turn, all the outward messages from the root to v^+ (which can be done since all the inward messages are correct).

We now formalise the above algorithmically. Like in the previous chapter, there are different *architectures* for performing the online junction tree algorithm: Shafer-Shenoy/ARCH-1, Hugin and ARCH-2 (which correspond directly to the architectures of the previous chapter). We shall show that online Shafer-Shenoy/ARCH-1 is space efficient, online Hugin is time efficient, and online ARCH-2 has, up to a factor linear in the width of the junction tree, the best of both worlds: the space efficiency of Shafer-Shenoy and the time efficiency of Hugin.

All the architectures store messages $M_{C \to D}$ (between neighbouring super-vertices C an D) as defined as in the previous chapter (note though, that these messages are now only updated when nesscessary). Hugin propagation and ARCH-2 also store additional potentials. To initialise, the junction tree algorithm (of the previous chapter) is first run (using the appropriate architecture) to compute the messages and any additional potentials. All the architectures have three operations (that differ depending on the architecture): **insert**$(v, \Lambda_v^{\text{new}})$ that changes the potential Λ_v, **pass**(C, E) (for neighbouring super-vertices C and E) that computes the message $M_{C \to E}$ and passes it to E, and **result**(v) that calculates the potential ρ_v. All architectures follow the following algorithms for update and query:
Update at vertex v:
Suppose we need to change Λ_v to Λ_v^{new}. We perform the following algorithm:

1. Run **insert**$(v, \Lambda_v^{\text{new}})$
2. Let $(v^+ = C_1, C_2, C_3, ..., C_j = R)$ be the path from v^+ to R in $\mathcal{J}$
3. For $i = 1, 2, 3, ..., j - 1$ in turn run **pass**(C_i, C_{i+1})

Query at vertex v:
Suppose we need to compute ρ_v. We perform the following algorithm

1. Let $(R = C_1, C_2, C_3, ..., C_j = v^+)$ be the path from R to v^+ in $\mathcal{J}$
2. For $i = 1, 2, 3, ..., j - 1$ in turn run **pass**(C_i, C_{i+1})
3. $\rho_v \leftarrow$**result**(v).

We now turn to the different architectures and how they implement the functions **insert**, **pass** and **result**.

5.5.1 Online Shafer-Shenoy

Online Shafer-Shenoy is the simplest of architectures. We store only the factors and messages. The operation **insert**$(v, \Lambda_v^{\text{new}})$ simply sets $\Lambda_v \leftarrow \Lambda_v^{\text{new}}$.Given neighbouring super-vertices C and E the operation **pass**(C, E) sets:

$$M_{C\to E} \leftarrow \left[\left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\mathbf{1}_{C\cap E} \prod_{H\in\mathcal{N}(C)\setminus\{E\}} M_{H\to C}\right)\right]^{\nabla C\cap E} \tag{5.36}$$

using Algorithm 74 (Operation 73) of the preceding chapter.
Given a vertex $v \in \mathcal{V}(G)$ the operation **result**(v) sets:

$$\rho_v \leftarrow \left[\left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\prod_{H\in\mathcal{N}(C)} M_{H\to C}\right)\right]^{\nabla \{v\}} \tag{5.37}$$

using Algorithm 74 (Operation 73) of the preceding chapter
Computational Complexity: The space complexity of online Shafer-Shenoy is only that required to store the factors and messages. The operation **pass**(C, D) takes a time of $\Theta\left((\deg(C) + |\mathcal{F}(C)|)2^{|C|}\right)$ and the operation **result**(v) takes a time of $\Theta\left((\deg(v^+) + |\mathcal{F}(C)|)2^{|v^+|}\right)$. This means that during update/query each super-vertex C in the path from v^+ to R contributes a time of $\Theta\left((\deg(C) + |\mathcal{F}(C)|)2^{|C|}\right)$ (except that on update the super-vertex R contributes no time).

5.5.2 Online Hugin

In addition to storing the messages, online Hugin stores, for every super-vertex $C \in \mathcal{V}(\mathcal{J})$, a potential $\Gamma_C \in \mathcal{T}(C)$ which is defined as:

$$\Gamma_C := \left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\prod_{H\in\mathcal{N}(C)} M_{H\to C}\right) \tag{5.38}$$

The operation **insert**$(v, \Lambda_v^{\text{new}})$ is the following algorithm:

1. Set $\Lambda_v^{\text{old}} \leftarrow \Lambda_v$
2. Set $\Lambda_v \leftarrow \Lambda_v^{\text{new}}$
3. Set $\Gamma_{v^+} \leftarrow (\Lambda_v/\Lambda_v^{\text{old}})\Gamma_{v^+}$

Note that **insert**$(v, \Lambda_v^{\text{new}})$ only updates Λ_v and Γ_{v^+}.
Given neighbouring super-vertices C and E the operation **pass**(C, E) is the following algorithm:

1. Set $M_{C\to E}^{\text{old}} \leftarrow M_{C\to E}$
2. Set $M' \leftarrow {\Gamma_C}^{\nabla C\cap E}$
3. Set $M_{C\to E} \leftarrow M'/M_{E\to C}$
4. Set $\Gamma_E \leftarrow (M_{C\to E}/M_{C\to E}^{\text{old}})\Gamma_E$

Note that **pass**(C, E) only updates the message $M_{C\to E}$ and the potential Γ_E.
Given a vertex $v \in \mathcal{V}(G)$ the operation **result**(v) simply sets:

$$\rho_v \leftarrow {\Gamma_{v^+}}^{\nabla\{v\}} \tag{5.39}$$

Computational Complexity:

Since online Hugin needs to store the potential Γ_C for every super-vertex C it has a total space complexity of $\Theta\left(\sum_{C\in\mathcal{V}(\mathcal{J})} 2^{|C|}\right)$ which can be significantly more than the space complexity of online Shafer-Shenoy.

The operation **insert**$(v, \Lambda_v^{\mathrm{new}})$ takes a time of $\Theta\left(2^{|v^+|}\right)$. The operation **pass**$(C, E)$ takes a time of $\Theta\left(2^{|C|} + 2^{|E|}\right)$. The operation **result**(v) takes a time of $\Theta\left(2^{|v^+|}\right)$. This means that upon update/query every super-vertex C in the path from v^+ to R contributes a time of only $\Theta\left(2^{|C|}\right)$ to the total time complexity.

5.5.3 Online ARCH-2

In addition to storing the messages, online ARCH-2 stores, for every super-vertex $C \in \mathcal{V}(\mathcal{J})$, the restriction $[\#\Gamma_C]^{\oplus\zeta_C}$ where:

$$\Gamma_C := \left(\prod_{\Lambda\in\mathcal{F}(C)} \Lambda\right)\left(\prod_{H\in\mathcal{N}(C)} M_{H\to C}\right) \tag{5.40}$$

and

$$\zeta_C := \left(\bigcup_{\Lambda\in\mathcal{F}(C)} \mathcal{P}(\sigma(\Lambda))\right) \cup \left(\bigcup_{H\in\mathcal{N}(C)} \mathcal{P}(C\cap H)\right) \tag{5.41}$$

Online ARCH-2 uses the functions **transform1**, **transform2** and **transform3** defined in section 4.7.3 of the previous chapter.

The operation **insert**$(v, \Lambda_v^{\mathrm{new}})$ is the following algorithm:

1. Set $\Lambda_v^{\mathrm{old}} \leftarrow \Lambda_v$
2. Set $\Lambda_v \leftarrow \Lambda_v^{\mathrm{new}}$
3. Set $\#\Lambda_v \leftarrow$ **transfom1**(Λ_v)
4. Set $\#\Lambda_v^{\mathrm{old}} \leftarrow$ **transfom1**$(\Lambda_v^{\mathrm{old}})$
5. For all $Z \in \mathcal{P}(\{v\})$ set $\#\Gamma_{v^+}(Z) \leftarrow [\#\Gamma_{v^+}(Z)][\#\Lambda_v(Z)]/[\#\Lambda_v^{\mathrm{old}}(Z)]$

Note that **insert**$(v, \Lambda_v^{\mathrm{new}})$ only updates Λ_v and the restriction, $[\#\Gamma_{v^+}]^{\oplus\zeta_{v^+}}$ of the potential $\#\Gamma_{v^+}$. The correctness of Line 5 comes directly from Theorem 84.
Given neighbouring super-vertices C and E the operation **pass**(C, E) is the following algorithm:

1. Set $M_{C\to E}^{\mathrm{old}} \leftarrow M_{C\to E}$
2. Set $[\%\Gamma_C]^{\oplus\zeta_C} \leftarrow$ **transform2**$([\#\Gamma_C]^{\oplus\zeta_C})$
3. For all $Z \in \mathcal{P}(C\cap E)$ set $\%M'(Z) \leftarrow \%\Gamma_C(Z)$. Note that $\%M'$ is now a potential in $\mathcal{T}(C\cap E)$.
4. Set $M' \leftarrow$ **transform3**$(\%M')$
5. Set $M_{C\to E} \leftarrow M'/M_{E\to C}$
6. Set $\#M_{C\to E}^{\mathrm{old}} \leftarrow$ **transform1**$(M_{C\to E}^{\mathrm{old}})$
7. Set $\#M_{C\to E} \leftarrow$ **transform1**$(M_{C\to E})$
8. For all $Z \in \mathcal{P}(C\cap E)$ set $\#\Gamma_E(Z) \leftarrow [\#\Gamma_E(Z)][\#M_{C\to E}(Z)]/[\#M_{C\to E}^{\mathrm{old}}(Z)]$

Note that **pass**(C, E) only updates the message $M_{C\to E}$ and the restriction, $[\#\Gamma_E]^{\oplus\zeta_E}$, of the potential $\#\Gamma_E$. Note also that M' is identical to that in online Hugin and the correctness of lines 3 and 8 come directly from theorems 88 and 84 respectively.
Given a vertex $v \in \mathcal{V}(G)$ the operation **result**(v) is the following algorithm:

1. Set $[\%\Gamma_{v^+}]^{\oplus\zeta_{v^+}} \leftarrow$ **transform2**$([\#\Gamma_{v^+}]^{\oplus\zeta_{v^+}})$
2. For all $Z \in \mathcal{P}(\{v\})$ set $\%\rho_v(Z) \leftarrow \%\Gamma_{v^+}(Z)$. Note that $\%\rho_v$ is now a potential in $\mathcal{T}(\{v\})$.
3. Output $\rho_v \leftarrow$ **transform3**$(\%\rho_v)$

Note that the correctness of Line 2 comes directly from Theorem 88.

Computational Complexity: We first derive the basic space requirements of the data-structure: In addition to storing the factors and messages we also store, for every $C \in \mathcal{V}(\mathcal{J})$, the restriction $[\#\Gamma_C]^{\oplus\zeta_C}$ which has a space requirement of:

$$\mathcal{O}(|\zeta_C|) = \mathcal{O}\left(| \left(\bigcup_{\Lambda\in\mathcal{F}(C)} \mathcal{P}(\sigma(\Lambda)) \right) \cup \left(\bigcup_{H\in\mathcal{N}(C)} \mathcal{P}(C\cap H) \right) | \right) \tag{5.42}$$

$$\subseteq \mathcal{O}\left(\left(\sum_{\Lambda\in\mathcal{F}(C)} |\mathcal{P}(\sigma(\Lambda))| \right) + \left(\sum_{H\in\mathcal{N}(C)} |\mathcal{P}(C\cap H)| \right) \right) \tag{5.43}$$

$$= \mathcal{O}\left(\left(\sum_{\Lambda\in\mathcal{F}(C)} 2^{|\sigma(\Lambda)|} \right) + \left(\sum_{H\in\mathcal{N}(C)} 2^{|C\cap H|} \right) \right) \tag{5.44}$$

we call this space complexity the "space required at C". Note that each edge $\{C, E\}$ of the junction tree contributes a space of $\mathcal{O}\left(2^{|C\cap E|}\right)$ to the space required at C and the space required at E and contributes no space to the space required at any other super-vertex. Also the edge $\{C, E\}$ contributes a space of $\mathcal{O}\left(2^{|C\cap E|}\right)$ for the messages $M_{C\to E}$ and $M_{E\to C}$. Hence the edge $\{C, E\}$ contributes a total space of $\mathcal{O}\left(2^{|C\cap E|}\right)$ to the whole data structure. Also, every factor $\Lambda \in \mathcal{F}$ contributes a space of only $2^{|\sigma(\Lambda)|}$ to the data structure. This means that the data-structure has, up to a constant factor, the same space requirements of online Shafer-Shenoy.

We now analyse the complexity of the operation **pass**(C, E): The time/space bottleneck is Line 2 which takes a time of $\mathcal{O}\left(|C|2^{|C|}\right)$ and requires an auxiliary space of:

$$\mathcal{O}\left(|C||\zeta_C|\right) \subseteq \mathcal{O}\left(|C| \left(\left(\sum_{\Lambda\in\mathcal{F}(C)} 2^{|\sigma(\Lambda)|} \right) + \left(\sum_{H\in\mathcal{N}(C)} 2^{|C\cap H|} \right) \right) \right) \tag{5.45}$$

Note also that in Line 8, finding the variable $\#\Gamma_E(Z)$ can take a time of $|E|$ if the potentials are stored in binary trees as in the previous chapter. Hence, the whole of Line 8 will take a time no greater than $\mathcal{O}\left(|E|\mathcal{P}(|E|)\right)$ so it can be ignored as we pay $\mathcal{O}\left(|E|\mathcal{P}(|E|)\right)$ to pass the message (via the function **pass**) from E to the next super-vertex.

The operations **insert**$(v, \Lambda_v^{\text{new}})$ and **result**(v) take a time no greater than $\mathcal{O}\left(|v^+|2^{|v^+|}\right)$ and require only constant auxiliary space.

This implies that during update/query each super-vertex C on the path from v^+ to R contributes a time of only $\mathcal{O}\left(|C|2^{|C|}\right)$ to the entire time complexity meaning that online ARCH-2 is only slower by online Hugin by no more than a logarithmic factor. In addition to the space requirements of the data structure (which, as we showed, is, up to a constant factor, the same as that of Shafer-Shenoy) we require an additional space of:

$$\mathcal{O}\left(\max_{C\in\mathcal{V}(\mathcal{J})} |C| \left(\left(\sum_{\Lambda\in\mathcal{F}(C)} 2^{|\sigma(\Lambda)|} \right) + \left(\sum_{H\in\mathcal{N}(C)} 2^{|C\cap H|} \right) \right) \right) \tag{5.46}$$

which is, up to a factor linear in the width of the junction tree, no greater than the space requirements of online Shafer-Shenoy.

5.6 An Optimal Junction Tree of a Tree

In this section we develop an algorithm that, for any tree T, constructs, in linear time and space, a junction tree of T that has (up to a constant factor) optimal speed for update/query under online Hugin/ARCH-2. We show that this update/query time is logarithmic in the maximum cardinality of a binary subtree (i.e. a subtree in which every vertex has a cardinality of at most three) of T. The space requirements for online Hugin/ARCH-2 with this junction tree are linear in the cardinality of T which is optimal for the online junction tree algorithm.

Related work. In [20] an algorithm was given for this model, in which each prediction required $\min\{\Delta(T), \log n\}$ time where $\Delta(T)$ is the diameter of the tree. We significantly improve on this result we observe that $\log \Delta(T) \leq \chi(T) \leq \min\{\Delta(T), \log n\}$ (where $\chi(T)$ is the time complexity of our resulting junction tree algorithm) and the lower bound is tight. For example, consider the tree T formed by $n/\log n$ path graphs having length $\log n$ that overlap at same central vertex. In this case it is not difficult to show that the total time required by our algorithm is $\chi(T) = \mathcal{O}(\log\log n)$ an exponential improvement over the result in [20]. Our algorithm was inspired by ideas for predicting efficiently on a path graph in [31]. In [1] they use an algorithm similar to ours for the construction of (what is essentially) a junction tree. However, their algorithm is random and it's possible that a very inefficient junction tree is produced. Also, by caching products of potentials our resulting online junction tree algorithm is faster.

Graph-theoretical preliminaries. A graph G is a pair of sets (V, E) such that E is a set of unordered pairs of distinct elements from V. The elements of V are called vertices and those of E are called edges. In order to avoid ambiguities deriving from dealing with different graphs, in some cases we will highlight the membership to graph G denoting these sets as $\mathcal{V}(G)$ and $\mathcal{E}(G)$ respectively. With slight abuse of notation, by writing $v \in G$, we mean $v \in \mathcal{V}(G)$. S is a subgraph G (we write $S \subseteq G$) iff $\mathcal{V}(S) \subseteq \mathcal{V}(G)$ and $\mathcal{E}(S) = \{(i,j) : i,j \in \mathcal{V}(S), (i,j) \in \mathcal{E}(G)\}$. Given any subgraph $S \subseteq G$, we define its **boundary** (or inner border) $\partial_0^G(S)$ and its **neighbourhood** (or outer border) $\partial_e^G(S)$ as: $\partial_0^G(S) := \{i : i \in S, j \notin S, (i,j) \in \mathcal{E}(G)\}$, and $\partial_e^G(S) := \{j : i \in S, j \notin S, (i,j) \in \mathcal{E}(G)\}$. With slight abuse of notation, $\partial_e^G(v) := \partial_e^G(\{v\})$, and thus the degree of a vertex v is $|\partial_e^G(v)|$. Given any graph G, we define the set of its leaves as $\text{leaves}(G) := \{i \in G : |\partial_e^G(i)| = 1\}$, and its **interior** $G^{\bullet} := \{i \in G : |\partial_e^G(i)| \neq 1\}$.

A path P in a graph G is a sequence of vertices $(v_1, v_2, ..., v_n)$ of G, such that for all $i < n$ we have that $(v_i, v_{i+1}) \in \mathcal{E}(G)$.In this case we say that v_1 and v_n are connected by P.

A tree T is a graph in which for all $v, w \in T$ there exists a unique path connecting v with w. In this paper we will only consider trees with a non-empty edge set and thus the vertex set will always have a cardinality of at least 2. Such a path is denoted by $\text{PATH}_T(v,w)$. The distance $d_T(v,w)$ between $v, w \in T$ is the path length $|\mathcal{E}(P)|$. It may be the case that a vertex r in a tree T is selected as the root of T. In this case we call (T, r) a "rooted tree. Given a rooted tree (T, r) and any vertex $i \in \mathcal{V}(T)$, the (*proper*) descendants of i are all vertices that can be connected with r via paths $P \subseteq T$ containing i (excluding i). Analogously, the (*proper*) ancestors of i are all vertices that lie on the path $P \subseteq T$ connecting i with r (excluding i). We denote the set of all descendants (resp. all ancestors) of i by $\Downarrow_T^r(i)$ (resp. $\Uparrow_T^r(i)$). We shall omit the root r when it is clear from the context. Vertex i is the parent (resp. child) of j, which is denoted by $\uparrow_T^r(j)$ (resp. $i \in \downarrow_T^r(j)$) if $(i,j) \in \mathcal{E}(T)$ and $i \in \Uparrow_T^r(j)$ (resp. $i \in \Downarrow_T^r(j)$). Given a tree T we use the notation $S \subseteq T$ *only* if S is a tree and subgraph of T. The height of a rooted tree (T, r) is the maximum length of a path $P \subseteq T$ connecting the root to any vertex: $h_r(T) := \max_{v \in T} d_T(v, r)$. The diameter $\Delta(T)$ of a tree T is defined as the length of the longest path between any two vertices in T. The second diameter $\Delta_2(T)$ of a tree T is defined as the maximum cardinality of a subtree of T in which every vertex has degree at most three.

5.6.1 Hierarchical Covers

In this section we describe a splitting process that recursively decomposes the input tree T. A (decomposition) tree (D, r) identifies this splitting process, generating a tree-structured collection $\mathcal{H}$ of subtrees that hierarchically cover the given input tree T.

This process recursively splits at each step a subtree of T (that we call a "component") resulting from some previous splits. More precisely, a subtree $S \subseteq T$ is split into two or more subcomponents and the decomposition of S depends only on the choice of a vertex $v \in S^{\bullet}$, which we call **splitting vertex**, in the following way. The splitting vertex $v \in S^{\bullet}$ of S induces the **split** set $\Omega(S, v) = \{S_1, \ldots, S_{|\partial_e^S(v)|}\}$ which is the unique set of S's subtrees overlapping at vertex v solely and representing a cover for S, i.e. it satisfies (i) $\cup_{S' \in \Omega(S,v)} S' = S$ and (ii) $\{v\} = S_i \cap S_j$ for all $1 \leq i < j \leq |\partial_e^S(v)|$. Thus the split may be visualized by considering the forest F resulting from removing a vertex from S, but afterwards each component $S_1, \ldots, S_{|\partial_e^S(v)|}$ of F has the "removed vertex" v added back to it. A component having only two vertices is called **atomic**, since it cannot be split further. We indicate with $S^v \subseteq T$ the component subtree whose splitting vertex is v, and we denote atomic components by $S^{(i,j)}$, where $\mathcal{E}(S^{(i,j)}) = \{(i, j)\}$. We finally denote by $\mathcal{H}$ the set of all component subtrees obtained by this splitting process. Since the method is recursive, we can associate a rooted tree (D, r), with T's decomposition into a hierarchical cover, whose internal vertices are the splitting vertices of the splitting process. Its leaves correspond to the single edges (of $\mathcal{E}(T)$) of each atomic component, and a vertex "parent-child" relation $c \in \downarrow_D^r(p)$ corresponds to the "splits-into" relation $S^c \in \Omega(S^p, p)$ (see Figure 5.1).

We will now formalize the splitting process by defining the **hierarchical cover** $\mathcal{H}$ of a tree T, which is a key concept used by our algorithm.

Definition 99. *Given a tree S and a vertex $x \in S^{\bullet}$, the* **split set**, *$\Omega(S, x)$ is the set of all subtrees, Q, of S that satisfy:*

1. $|\mathcal{V}(Q)| \geq 2$
2. *x is a leaf of Q*
3. *Given any $v \in \mathcal{V}(Q) \setminus \{x\}$ all neighbours of v (in S) are in $\mathcal{V}(Q)$*

Definition 100. *An **hierarchical cover**, $\mathcal{H}$, of a tree T is a tree-structured collection of subtrees that satisfy the following three properties:*

1. *$T \in \mathcal{H}$,*
2. *for all $S \in \mathcal{H}$ with $S^{\bullet} \neq \emptyset$ there exists an $x \in S^{\bullet}$ such that $\Omega(S, x) \subset \mathcal{H}$,*
3. *for all $S, R \in \mathcal{H}$ such that $S \not\subseteq R$ and $R \not\subseteq S$, we have $|\mathcal{V}(R) \cap \mathcal{V}(S)| \leq 1$.*

The above definition recursively generates a cover. The splitting process that generates a hierarchical cover $\mathcal{H}$ of T is formalized as rooted tree (D, r) in the following definition.

Definition 101. *If $\mathcal{H}$ is a hierarchical cover of T we define the associated **decomposition tree** (D, r) as a rooted tree, whose vertex set $\mathcal{V}(D) := T^{\bullet} \cup \mathcal{E}(T)$ where $D^{\bullet} = T^{\bullet}$ and* $\text{leaves}(D) \equiv \mathcal{E}(T)$*, such that the following three properties hold:*

1. *$S^r = T$,*
2. *for all $c, p \in D^{\bullet}$, $c \in \downarrow_D^r(p)$ iff $S^c \in \Omega(S^p, p)$,*
3. *for all $(c, p) \in \mathcal{E}(T)$ [1], we have $(c, p) \in \downarrow_D^r(p)$ iff $S^{(c,p)} \in \Omega(S^p, p)$.*

where, for $v \in T^{\bullet}$, S^v is the tree in $\mathcal{H}$ who's splitting vertex is v and for $(c, p) \in \mathcal{E}(T)$, $S^{(c,p)}$ is the component who's vertex set is $\{c, p\}$.

The following lemma shows that with any given hierarchical cover $\mathcal{H}$ it is possible to associate a unique decomposition tree (D, r).

[1]Observe that $(c, p) \in \mathcal{E}(T)$ implies $c, p \in \mathcal{V}(T)$ and $(c, p) \in \text{leaves}(D)$.

Lemma 102. *A hierarchical cover $\mathcal{H}$ of T defines a unique decomposition tree (D, r) such that if $S \in \mathcal{H}$ there exists a $v \in \mathcal{V}(D)$ such that $S = S^v$ and if $v, w \in \mathcal{V}(D)$ and $v \neq w$, then $S^v \neq S^w$.*

Definition 103. *Given an hierarchical cover $\mathcal{H}$ of a tree T, we define the* **height** *of $\mathcal{H}$ to be the height of its associated decomposition tree.*

Definition 104. *Given a tree T, the* **decomposition potential***, $\chi^*(T)$, is the minimum height of a hierarchical cover of T.*

5.6.2 Bounds on the decomposition potential

The update and query times of our algorithm will be linear in the decomposition potential of the tree. Hence, we now derive tight bounds for this quantity.

Definition 105. *Given a tree T, a subtree, B, of T is a* **binary subtree** *if and only if each vertex of B has a degree (in B) of at most three.*

Definition 106. *The* **second-diameter***, $\Delta_2(T)$, of a tree T is the cardinality of the largest binary subtree of T.*

We now prove that $\chi^*(T) \in \Theta(\Delta_2(T))$:

Lemma 107. $\chi^*(T) \in \Omega(\log \Delta_2(T))$

Proof. Let B be a binary subtree of T with $\Delta_2(T)$ vertices and let $\mathcal{H}(T)$ be a hierarchical cover of T with height $\chi^*(T)$. The set $\mathcal{H}(B) := \{S \cap B | S \in \mathcal{H}\}$ is then a hierarchical cover of B with height no greater than $\chi^*(T)$. The decomposition tree associated with $\mathcal{H}(B)$ is a ternary tree, in that every vertex has at most three children, since, for any tree S in the hierarchical cover, and any $v \in S^\bullet$, we have $|\Omega(S, v)| \leq 3$. The height of $\mathcal{H}(B)$, and hence that of $\mathcal{H}(T)$, must then be bounded below by the height of a full, balanced ternary tree with $|\Delta_2(T)|$ vertices, which is logarithmic in $\Delta_2(T)$. $\square$

Lemma 108. $\chi^*(T) \in \mathcal{O}(\log \Delta_2(T))$

Proof. We prove, by induction on $\chi^*(T)$, that given distinct leaves u and v of T, there exists a binary subtree B of T which contains u and v and has cardinality at least $2^{\frac{1}{3}\chi^*(T)}$. Note that this implies that $\chi^*(T) \in \mathcal{O}(\log \Delta_2(T))$

Since $|B| \geq 2$ the result holds for $\chi^*(T) \leq 2$.

If $\chi^*(T) > 2$ then let $\mathcal{H}$ be an hierarchical cover of T with height $\chi^*(T)$ and let x be the split point of T in $\mathcal{H}$. Choose $R \in \mathrm{argmax}_{Q \in \Omega(T,x)}(\chi^*(Q))$ and choose $S \in \mathrm{argmax}_{Q \in \Omega(T,x) \setminus \{R\}}(\chi^*(Q))$. We must have that $\chi^*(R) = \chi^*(T) - 1$. If $\chi^*(S) < \chi^*(R) - 2$ then we do the following:
Let x' be the unique neighbour of x that is contained in R, let S' be the unique tree in $\Omega(T, x')$ that contains S as a subtree, and let $R' = \mathrm{argmax}_{Q \in \Omega(T,x') \setminus \{S'\}}(\chi^*(Q))$. By splitting S' at x we see that $\chi^*(S') < \chi^*(T) - 1$ and hence also that $\chi^*(R') = \chi^*(T) - 1$ (since else, for all $Q \in \Omega(T, x')$, we'd have $\chi^*(Q) < \chi^*(T) - 1$ meaning that there would exist a hierarchical cover of T of height less than $\chi^*(T)$). If we have that $\chi^*(S') < \chi^*(T) - 2$ then we continue recursively on R' and S'. Eventually we will either have that $\chi^*(S') = \chi^*(T) - 2$ or hit reductio ad absurdam: in that $|R'| = 2$ and hence $\chi^*(R') = 0 < \chi^*(T) - 1$ which is is contradiction. Hence, without loss of generality, we may assume that $\chi^*(S) \geq \chi^*(T) - 2$.

Let y be the vertex of least distance from x which is on both the path from x to v and the path from u to v. We have three cases:

1. $y \notin (S \setminus \partial_0^T(S)) \cup (R \setminus \partial_0^T(R))$: By the inductive hypothesis, we have the result by taking binary subtrees $B(S)$ and $B(R)$ (of S and R respectively), containing x and with cardinality at least $2^{\frac{1}{3}(\chi^*(T)-2)}$, and setting B to be the subtree of T containing exactly those vertices in either $B(S)$, $B(R)$, the path from x to y, the path from y to u, or the path from y to v.

2. $y \in R \setminus \partial_0^T(R)$: Choose $U \in \operatorname{argmax}_{Q\in\Omega(R,y)}(\chi^*(U))$. We must have that $\chi^*(U) \geq \chi^*(F) - 1 = \chi^*(T) - 2$. By the inductive hypothesis, we then have the result by taking binary subtrees $B(U)$ and $B(R)$ (of U and R respectively), containing y (and x if $x \in U$) and x respectively, and with cardinality at least $2^{\frac{1}{3}(\chi^*(T)-2)}$, and setting B to be the subtree of T containing exactly those vertices in either $B(U)$, $B(S)$, the path from x to y, the path from y to u, or the path from y to v.
3. $y \in S \setminus \partial_0^T(S)$: Choose $U \in \operatorname{argmax}_{Q\in\Omega(S,y)}(\chi^*(U))$. We must have that $\chi^*(U) \geq \chi^*(S) - 1 \geq \chi^*(T) - 3$. By the inductive hypothesis, we then have the result by taking binary subtrees $B(S)$ and $B(U)$ (of S and U respectively), containing x and y (and x if $x \in U$) respectively, and with cardinality at least $2^{\frac{1}{3}(\chi^*(T)-3)}$, and setting B to be the subtree of T containing exactly those vertices in either $B(R)$, $B(U)$, the path from x to y, the path from y to u, or the path from y to v.

□

5.6.3 $(2, s)$-covers

Our algorithms for update and query will work on an hierarchical cover $\mathcal{H}$ and will be exponential on the **exposure** of $\mathcal{H}$, that is, the maximum cardinality of the boundary of a component in $\mathcal{H}$. We hence introduce $(2, s)$-covers, which are a subset of hierarchical covers with exposure two, and show that we don't lose much by considering only these hierarchical covers.

Definition 109. *Given a rooted tree T, a vertex $w \in T$, and a descendant, v, of w we define*

$$\begin{bmatrix} w \\ v \end{bmatrix} := \operatorname*{argmax}_{S\subseteq T}\{|S| : v, w \in \text{leaves}(S)\} \tag{5.47}$$

Definition 110. *Given a tree T, rooted at s, a vertex $w \in T$ and a child, v, of w, we define by $\left[\begin{smallmatrix} w \\ \vec{v} \end{smallmatrix}\right]$ the subtree of T with vertex set $\Downarrow_T^s(v) \cup \{w\}$. i.e. For any leaf l of T that is a descendant of v we have*

$$\begin{bmatrix} w \\ \vec{v} \end{bmatrix} := \begin{bmatrix} w \\ l \end{bmatrix} \tag{5.48}$$

Definition 111. *Given a tree T, rooted at s, an hierarchical cover $\mathcal{H}$ is a $(2, s)$-cover if and only if, for every $S \in \mathcal{H}$ there exist vertices $v, w \in T$ such that $v \in \Downarrow_T^s(w)$ and $S = \left[\begin{smallmatrix} w \\ v \end{smallmatrix}\right]$. Given a subtree S of T whose root is s' when T is rooted at s we define a $(2, s)$-cover of S to be the same as a $(2, s')$-cover of S.*

Definition 112. *Given a tree T, rooted at s, we define $\chi(T)$ to be the minimum height of a $(2, s)$-cover of T.*

The update and query times of our algorithm will be linear in the height of the hierarchical cover that we use. We now show that $\chi(T) \leq 2\chi^*(T)$ and hence that we don't lose out by considering only $(2, s)$-covers.

Lemma 113. *For any subtree $\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]$ of T, a $(2, s)$-cover of $\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]$ exists and we have that $\chi(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) \leq 2\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right])$. In particular we have that $\chi(T) \leq 2\chi^*(T)$.*

Proof. We prove by induction on $\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right])$.

If $\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) = 0$ then $w = \uparrow_T(u)$ and hence $\{\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]\}$ is a $(2, s)$cover of $\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]$ so $\chi(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) = 0$.

If $\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) \neq 0$ then choose a hierarchical cover $\mathcal{H}$ of height $\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right])$. Let v be the split point of $\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]$ in $\mathcal{H}$. Let v' be the vertex of maximum depth in the intersection of the paths from u to w and from v to w. For every $S \in \Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v)$, let $\mathcal{H}(S)$ be the hierarchical cover induced on S by $\mathcal{H}$. Note that the height of each of these is no greater that $\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1$ and hence for all such S, $\chi^*(S) \leq \chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1$.

Two cases to consider.

First, if $v = v'$ we then have for every $S \in \Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v)$ both that $|\partial_0^T(S)| \leq 2$ and by the inductive hypothesis, a $(2,s)$-cover $\mathcal{G}(S)$ of S of height no greater than $2(\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right] - 1)$. Hence $\{\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]\} \cup \bigcup\{\mathcal{G}(S) : S \in \Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v)\}$ is a $(2,s)$-cover of $\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]$ with height no greater than $2\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1$.

Second, if $v \neq v'$ then let Q be the tree in $\Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v')$ that contains v and let R be the tree in $\Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v)$ that contains u and w. We have that, for all trees $S_R \in \Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v') \backslash \{Q\}$ that $S_R \subseteq R$ and hence, since $\{S_R \cap U : U \in \mathcal{H}(R)\}$ is an hierarchical cover of S_R with height at most that of $\mathcal{H}(R)$, we have that $\chi^*(S_R) \leq \chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1$. Since the tree $\left[\begin{smallmatrix} v' \\ v \end{smallmatrix}\right]$ is also a subset of R we likewise obtain $\chi^*(\left[\begin{smallmatrix} v' \\ v \end{smallmatrix}\right]) \leq \chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1$. Likewise, for all trees $S_Q \in \Omega(Q, v) \setminus \{\left[\begin{smallmatrix} v' \\ v \end{smallmatrix}\right]\}$ is in $\Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v)$ so similarly we see that $\chi^*(S_Q) \leq \chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1$ and $|\partial_0^T(S_Q)| \leq 2$. Hence, by the inductive hypothesis, we may find $(2,s)$-covers $\mathcal{G}(S_Q)$ of every tree $S_Q \in \Omega(Q, v)$ which have height at most $2(\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1)$ and hence create a $(2,s)$-cover $\mathcal{G}(Q) := \{Q\} \cup \bigcup\{\mathcal{G}(S_Q) : S' \in \Omega(Q, v)\}$ of Q with height at most $2\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]) - 1$.

Thus for every $S \in \Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v')$, there exists $(2,s)$-cover $\mathcal{G}(S)$ of S with height at most $2(\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right])) - 1$ therefore we may $(2,s)$-cover $\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]$ with $\{\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right]\} \cup \bigcup\{\mathcal{G}(S) : S \in \Omega(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right], v')\}$ in height no more that $2\chi^*(\left[\begin{smallmatrix} w \\ u \end{smallmatrix}\right])$. □

5.6.4 Greedy covers

Our algorithm works by constructing a $(2,s)$-hierarchical cover in a greedy fashion. We shall now formalise what we mean by "greedy". A greedy cover consists only of trees which are "maximal," as defined below.

Definition 114. *A tree* $\left[\begin{smallmatrix} w \\ v \end{smallmatrix}\right] \subseteq T$ *is called* η-**maximal** *with respect to rooted tree* (T, s) *iff:*

1. $\chi(\left[\begin{smallmatrix} w \\ v \end{smallmatrix}\right]) \leq \eta$,
2. *for every proper ancestor, z, of w we have* $\chi(\left[\begin{smallmatrix} z \\ v \end{smallmatrix}\right]) > \eta$.

Intuitively, an η-maximal tree (with respected to a rooted tree) is a tree whose optimal cover height is no more than η and if we should grow the tree toward the root the optimal cover height must exceed η. We give an example, consider a path graph with edge set $\{(1,2),(2,3),(3,4),(4,5)\}$ and root vertex "1", the subtrees identified with the four edges are 0-maximal, the subtrees $\{\{1,2,3\},\{2,3,4\},\{3,4,5\},\{1,2\}\}$ are all 1-maximal while the subtrees $\{\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,3,4,5\}\}$ are 2-maximal and the subtree $\{2,3,4,5\}$ is not η-maximal for any η. Thus we also observe that the second condition of definition is vacuously satisfied for $\left[\begin{smallmatrix} s \\ v \end{smallmatrix}\right] \subseteq T$ for any v, since s is the root of T.

Definition 115. *A $(2,s)$-cover $\mathcal{R}_S$ of a tree $S \subseteq T$ is* **greedy** *iff for every tree $Q \in \mathcal{R}_S$ with $Q^\bullet \neq \emptyset$, there exists an $x \in Q^\bullet$ such that $\Omega(Q,x) \subset \mathcal{R}_S$ and there exists an $\eta \in \mathbb{N}$ such that every $P \in \Omega(Q,x)$ is η-maximal with respect to (T,s).*

Observe that it is possible to find a greedy $(2,s)$-cover $\mathcal{R}_Z \subseteq \mathcal{R}_S$ for every element Z of a greedy cover $\mathcal{R}_S$ of any tree $S \subseteq T$. In the following lemma we show that the height of a greedy $(2,s)$-cover is optimal.

Lemma 116. *If $\mathcal{G}$ is a greedy $(2,s)$-cover of $\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right]$ then it has optimal height $\chi(\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right])$.*

Proof. We prove by induction on the height of $\mathcal{G}$.

It holds if the height of $\mathcal{G}$ is equal to zero since in this case $\{\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right]\}$ is the unique cover.

If the height of $\mathcal{G}$ is greater than zero then let w be the splitting point of $\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right]$ in $\mathcal{G}$. By definition, there exists some η such that all trees in $\Omega(\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right], w)$ are η-maximal. We also have, by definition, that the $(2,s)$-covers induced by $\mathcal{G}$ on all trees in $\Omega(\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right], w)$ are greedy. So, since all trees $S \in \Omega(\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right], w)$ satisfy $\chi(S) \leq \eta$ we have, by the inductive hypothesis, that the height of the $(2,s)$-cover induced by $\mathcal{G}$ on each such S is at most η and hence the height of $\mathcal{G}$ is at most $\eta + 1$. Since $\left[\begin{smallmatrix} w \\ v \end{smallmatrix}\right]$ is η-maximal and x is a proper ancestor of w we have that $\chi(\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right]) \geq \eta + 1$ so since $\chi(\left[\begin{smallmatrix} x \\ v \end{smallmatrix}\right])$ is no more than the height of $\mathcal{G}$, both are equal to $\eta + 1$. □

We shall now derive sufficient conditions for determining that a subtree is η-maximal, hence giving sufficient conditions for the recursive construction of greedy $(2,s)$-covers. We shall first introduce the notion of a w-cousin.

Definition 117. *The vertex u is a w-cousin of v if both u and v are proper descendants of w and the path from u to v contains w.*

Lemma 118. *Given a rooted tree (T,s) and $\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right] \subseteq T$ with some vertex x in the interior of the path from w to y, if every tree in $\Omega(\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right],x)$ is $(\eta-1)$-maximal then $\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right]$ is η-maximal.*

Proof. By construction we have that $\chi(\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right]) \leq \eta$ since we can take a minimum height $(2,s)$-cover $\mathcal{R}_S$ (of height at most $\eta-1$) for every $S \in \Omega(\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right],x)$ and present $\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right] \cup \bigcup\{\mathcal{R}_S : S \in \Omega(\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right],x)\}$ as a $(2,s)$-cover of $\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right]$ with height at most η. Thus we have ve shown the first condition of Definition 114 holds.

We now consider the trivialized case when $y = s$. We now observe that the second condition of Definition 114 holds vacuously.

For the case $y \neq s$, we proceed by contradiction, by supposing that $\left[\begin{smallmatrix}y\\w\end{smallmatrix}\right]$ is not η-maximal.

Then by the second property of Definition 114 there exists a proper ancestor y' of y for which $\chi(\left[\begin{smallmatrix}y'\\w\end{smallmatrix}\right]) \leq \eta$. So take a $(2,s)$-cover $\mathcal{Q}$ of $\left[\begin{smallmatrix}y'\\w\end{smallmatrix}\right]$ of minimum height $(\leq \eta)$ and suppose $\left[\begin{smallmatrix}y'\\w\end{smallmatrix}\right]$ splits at x' in this $(2,s)$-cover. Suppose that, for some proper ancestor z of x, x' is either equal to z or some z-cousin of w. Then since for every tree in $\mathcal{Q}$ the cardinality of its boundary in T bounded by 2 we must have that $\left[\begin{smallmatrix}z\\w\end{smallmatrix}\right] \in \mathcal{Q}$. This would mean (since $z \neq y'$) that $\chi(\left[\begin{smallmatrix}z\\w\end{smallmatrix}\right]) \leq \eta - 1$ which would contradict the fact that $\left[\begin{smallmatrix}x\\w\end{smallmatrix}\right]$ was $(\eta-1)$-maximal. The alternative is that x' is a descendant of x. This implies that $\left[\begin{smallmatrix}y'\\x\end{smallmatrix}\right] \subseteq \left[\begin{smallmatrix}y'\\x'\end{smallmatrix}\right]$ and hence $\{S \cap \left[\begin{smallmatrix}y'\\x\end{smallmatrix}\right] : S \in \mathcal{Q}\}$ is a $(2,s)$-cover of $\left[\begin{smallmatrix}y'\\x\end{smallmatrix}\right]$ of height at most $\chi(\left[\begin{smallmatrix}y'\\x'\end{smallmatrix}\right]) \leq \chi(\left[\begin{smallmatrix}y'\\w\end{smallmatrix}\right]) - 1 \leq \eta - 1$ which contradicts the fact that $\left[\begin{smallmatrix}y\\x\end{smallmatrix}\right]$ is $(\eta-1)$-maximal. Thus our original supposition is false. $\square$

Lemma 119. *Given a rooted tree (T,s), the following conditions imply that $\left[\begin{smallmatrix}y\\x\end{smallmatrix}\right] \subseteq T$ is η-maximal:*

1. *$\left[\begin{smallmatrix}y\\x\end{smallmatrix}\right]$ is $(\eta-1)$-maximal,*
2. *There exists some leaf ℓ which is a y-cousin of x, some v which is a proper ancestor of ℓ, and some w which is both a proper ancestor of v and descendant of y, such that all trees in $\Omega(\left[\begin{smallmatrix}w\\\ell\end{smallmatrix}\right],v)$ are $(\eta-1)$-maximal.*

Proof. In the trivial case that $y = s$ condition 1, above immediately implies the result.

When $y \neq s$ suppose $\left[\begin{smallmatrix}y\\x\end{smallmatrix}\right]$ is not η-maximal. Then for some proper ancestor z of y we have that $\chi(\left[\begin{smallmatrix}z\\x\end{smallmatrix}\right]) \leq \eta$. By Lemma 118 we have that $\left[\begin{smallmatrix}w\\\ell\end{smallmatrix}\right]$ is η-maximal so x cannot be a leaf else $\left[\begin{smallmatrix}z\\x\end{smallmatrix}\right] = \left[\begin{smallmatrix}z\\\ell\end{smallmatrix}\right]$ which contradicts the proceeding statement. Let $\mathcal{H}$ be a $(2,s)$-cover of $\left[\begin{smallmatrix}z\\x\end{smallmatrix}\right]$ with minimum height $(\leq \eta)$. Since every tree in $\mathcal{H}$ has the cardinality of its boundary in T bounded by 2 we must have (since x is not a leaf) that $\left[\begin{smallmatrix}y\\\ell\end{smallmatrix}\right] \in \mathcal{H}$ which would imply that $\chi(\left[\begin{smallmatrix}y\\\ell\end{smallmatrix}\right]) \leq \eta - 1$, contradicting the $(\eta-1)$-maximality of $\left[\begin{smallmatrix}v\\\ell\end{smallmatrix}\right]$. $\square$

5.6.5 Constructing a greedy cover

From a "big picture" perspective, a greedy $(2,s)$-cover $\mathcal{G}$ is recursively constructed in a bottom-up fashion: in the initialization phase $\mathcal{G}$ contains only the atomic components covering T, i.e. the ones formed only by a pair of adjacent vertices of $\mathcal{V}(T)$. We have then at this stage $|\mathcal{G}| = |\mathcal{E}(T)|$. Then $\mathcal{G}$ grows step by step through the addition of new covering subtrees of T. At each time step t, at least one subtree of T is added to $\mathcal{G}$. All the subtrees S^v, that are added at step t, must be such that all trees in $\Omega(S,v)$ are added before step t.

We now introduce the formal description of our method for constructing a greedy $(2,s)$-hierarchical cover $\mathcal{G}$. As we said, the construction of $\mathcal{G}$ proceeds in incremental steps. At each step t the method operates on a tree T_t, whose vertices are part of $\mathcal{V}(T)$. The construction of T_t is accomplished starting by T_{t-1} (if $t > 0$)

in such a way that $\mathcal{V}(T_t) \subset \mathcal{V}(T_{t-1})$, where T_0 is set to be the subtree of (T, s) containing the root and all the internal vertices.

During each step t all the while-loop instructions of Figure 5.1 are executed: (1) some vertices (the black ones in Figure 5.1) are selected through a depth-first visit (during the backtracking steps) of T_t starting from s [2], (2) for each selected vertex v, subtree S^v is obtained from merging subtrees added to $\mathcal{G}$ in previous steps and overlapping at vertex v, (3) in order to create tree T_{t+1} from T_t the previously selected vertices of T_t are removed, (4) the edge set $\mathcal{E}(T_{t+1})$ is created from $\mathcal{E}(T_t)$ in such a way to preserve the T_t's structure, but all the edges incident to the vertices removed from $\mathcal{V}(T_t)$ (the black vertices Figure 5.1) in the while-loop step 3 need to be deleted. The possible disconnection that would arise by the removal of these parts of T_t is avoided by completing the construction of $\mathcal{E}_{t+1}$ through the addition of some new edges. These additional edges are not part of $\mathcal{E}(T)$ and link each vertex v with its grand-parent in T_t if vertex v's parent was deleted (see the dashed line edges in Figure 5.1) during the construction of T_{t+1} from T_t. In the final while-loop step the variable t gets incremented by 1.

The idea of this recursive construction is, at each time step t, to find all the t-maximal trees in the greedy decomposition of T. This is either done by merging together various $(t-1)$-maximal trees (lemma 118) or by simply recognising that a certain $(t-1)$-maximal tree is also t-maximal (lemma 119).

Before we provide the detailed description (figure 5.1) of the algorithm for constructing an optimal $(2, s)$-hierarchical cover we need some ancillary definitions. We call a vertex $v \in \mathcal{V}(T_t) \setminus \{s\}$ **mergeable** (at time t) if and only if either (i) $v \in \operatorname{leaves}(T_t)$ or (ii) v has a single child in T_t and that child is not mergeable. If $v \in \mathcal{V}(T_t) \setminus \{s\}$ is mergeable we write $v \in \mathcal{Z}_t$. We also use the following shorthands for making more intuitive our notation: We set $c_v^t := \downarrow^s_{T_t}(v)$ when $|\downarrow^s_{T_t}(v)| = 1$, $p_v^t := \uparrow^s_{T_t}(v)$ when $v \neq s$ and $g_v^t := \uparrow^s_{T_t}(p_v^t)$ when $v, p_v^t \neq s$. Finally, given $u, u' \in \mathcal{V}(T)$ such that $u' \in \Downarrow^s_T(u)$, we indicate with with $\downarrow^s_T(u \mapsto u')$ the child of u which is ancestor of u' in T.

We now show that our algorithm outputs the greedy decomposition of T in linear time.

Lemma 120. *The following two properties hold for the hierarchical cover algorithm in Figure 5.1,*

1. *for every $v \in T_t$ and every leaf ℓ of T which is a proper descendant of v with $\left[{v \atop \ell}\right] \cap T_t = \{v\}$, we have that $\left[{v \atop \ell}\right]$ is t-maximal,*
2. *for every $v \in \mathcal{V}(T_t) \setminus \{s\}$, we have that $\left[{\uparrow_{T_t}(v) \atop v}\right]$ is t-maximal.*

Proof. We prove by induction on t.

At $t = 0$, for every $v \in T_t$ and every leaf ℓ of T which is a proper descendant of v, we have that $v = \uparrow_T(\ell)$ and hence $\left[{v \atop \ell}\right]$ is 0-maximal. Similarly, for every $v \in \mathcal{V}(T_0) \setminus \{s\}$ we have that $\uparrow_{T_0}(v) = \uparrow_T(v)$ so $\left[{\uparrow_{T_0}(v) \atop v}\right]$ is 0-maximal.

[Property 1]: given $t \neq 0$ consider $v \in T_t$ and a leaf ℓ of T which is a proper descendant of v with $\left[{v \atop \ell}\right] \cap T_t = \{v\}$. First we consider the trivial argument when $v = s$ is the root of T. Inductively (property 1) we have that $\left[{v \atop \ell}\right]$ is $(t-1)$-maximal and hence as v is the root, $\left[{v \atop \ell}\right]$ is then also t-maximal trivially. We then have two cases when $v \neq s$:

First, consider the case that $\left[{v \atop \ell}\right] \cap T_{t-1} = \{v\}$. Since $\left[{v \atop \ell}\right] \cap T_{t-1} = \{v\}$ and $v \in T_t$ the vertex v cannot be a leaf in T_{t-1} and thus there exists a child c of v in T_{t-1} that is a v-cousin of ℓ. Now choose a leaf ℓ' of T that is proper descendent of c and let d denote a descendent of c (it might be the case that $d = c$) that is a leaf of T_{t-1} on the path from c to ℓ'. Now by the inductive hypothesis we have that $\left[{d \atop \ell'}\right]$ is $(t-1)$-maximal (property 1), $\left[{\uparrow_{T_{t-1}}(d) \atop d}\right]$ is $(t-1)$-maximal (property 2), and

[2]Observe that s is the unique vertex belonging to $\mathcal{V}(T_t)$ for all time step $t \geq 0$.

$\genfrac{[}{]}{0pt}{}{v}{\ell}$ is $(t-1)$-maximal (property 1). Furthermore, if there exists any leaf $\ell'' \neq \ell'$ of T that is a proper descendent of d then also inductively $\genfrac{[}{]}{0pt}{}{d}{\ell''}$ is $(t-1)$-maximal (property 1). Thus the conditions of Lemma 119 are satisfied and we have that $\genfrac{[}{]}{0pt}{}{v}{\ell}$ is t-maximal.

Second, consider the case that $\genfrac{[}{]}{0pt}{}{v}{\ell} \cap T_{t-1} \neq \{v\}$.

Note that $|\genfrac{[}{]}{0pt}{}{v}{\ell} \cap T_{t-1}| \leq 2$ as given the vertices in *any* path in tree T_t at most one vertex in every edge in the path on T_{t-1} is removed in the construction of T_t via line 3 and 4 of the pseudocode of the algorithm. Thus if $|\genfrac{[}{]}{0pt}{}{v}{\ell} \cap T_{t-1}| \geq 3$ this implies $|\genfrac{[}{]}{0pt}{}{v}{\ell} \cap T_t| \geq 2$ which is contradiction. Thus we now have $\genfrac{[}{]}{0pt}{}{v}{\ell} \cap T_{t-1} = \{u, v\}$ for u a child of v in T_{t-1}. By the inductive hypothesis $\genfrac{[}{]}{0pt}{}{u}{\ell}$ is $(t-1)$-maximal (property 1) and $\genfrac{[}{]}{0pt}{}{v}{u}$ is $(t-1)$-maximal (property 2). If the leaf ℓ has any u-cousin leaf ℓ' the intersection $\genfrac{[}{]}{0pt}{}{u}{\ell'} \cap T_{t-1} = \{u\}$ (otherwise $|\genfrac{[}{]}{0pt}{}{v}{\ell} \cap T_t| \geq 2$) and hence by the inductive hypothesis (property 1) $\genfrac{[}{]}{0pt}{}{u}{\ell'}$ is also $(t-1)$-maximal. Thus the conditions of Lemma 118 are satisfied and we have that $\genfrac{[}{]}{0pt}{}{v}{\ell}$ is t-maximal.

[Property 2]: given $t \neq 0$ consider $v \in T_t \setminus s$. We again have two cases:

First, consider the case that $p := \uparrow_{T_t}(v) = \uparrow_{T_{t-1}}(v)$.

We have two sub-cases, first the trivial case is the one in which $p = s$, which implies by the inductive hypothesis (property 2) $\genfrac{[}{]}{0pt}{}{p}{v}$ is $(t-1)$-maximal and hence as p is the root, $\genfrac{[}{]}{0pt}{}{p}{v}$ is then also t-maximal trivially. If $p \neq s$ then by construction of T_t from T_{t-1} it must be the case that p has at least one child in T_{t-1} distinct from v; denote it as c. Now choose a leaf ℓ of T that is a proper descendent of c. Let d denote a descendent of c (it might be the case that $d = c$) that is a leaf of T_{t-1} on the path from c to ℓ. Now by the inductive hypothesis we have that $\genfrac{[}{]}{0pt}{}{d}{\ell}$ is $(t-1)$-maximal (property 1), $\genfrac{[}{]}{0pt}{}{\uparrow_{T_{t-1}}(d)}{d}$ is $(t-1)$-maximal (property 2), and $\genfrac{[}{]}{0pt}{}{p}{v}$ is $(t-1)$-maximal (property 2). Furthermore, if there exists any leaf $\ell' \neq \ell$ of T that is a proper descendent of d then also inductively $\genfrac{[}{]}{0pt}{}{d}{\ell'}$ is $(t-1)$-maximal (property 1). Thus the conditions of Lemma 119 are satisfied and we have that $\genfrac{[}{]}{0pt}{}{p}{v}$ is t-maximal.

Second, consider the case that $\uparrow_{T_t}(v) \neq \uparrow_{T_{t-1}}(v)$.

Then by the construction of T_t, lines 3 and 4 in the algorithm, we shall denote the parent and grandparent of v (in T_{t-1}) as $p := \uparrow_{T_{t-1}}(v)$ and $g := \uparrow_{T_t}(v) = \uparrow_{T_{t-1}}(p)$. Let ℓ denote any leaf of T that is a p-cousin of v; if such a leaf exists. Note that no p-cousin of v is contained in T_{t-1} because that would imply $\uparrow_{T_t}(v) = \uparrow_{T_{t-1}}(v)$, thus $\genfrac{[}{]}{0pt}{}{p}{\ell} \cap T_{t-1} = \{p\}$. Hence if any such ℓ exists by the inductive hypothesis (property 1) then $\genfrac{[}{]}{0pt}{}{p}{\ell}$ is $(t-1)$-maximal. Furthermore inductively (property 2) we have that $\genfrac{[}{]}{0pt}{}{p}{v}$ and $\genfrac{[}{]}{0pt}{}{g}{p}$ are $(t-1)$-maximal. Thus the conditions of Lemma 118 are satisfied and we have that $\genfrac{[}{]}{0pt}{}{g}{v}$ is t-maximal. □

Theorem 121. *Given a rooted tree (T, s), the algorithm in Figure 5.1 outputs $\mathcal{G}$, a $(2, s)$-hierarchical cover of height $\chi(T)$, in time linear in $|\mathcal{V}(T)|$.*

Proof. We first prove that when $\mathcal{V}(T_t) = \{s\}$, $\mathcal{G}$ is an optimal $(2, s)$-cover of T.

We observe at time t for every edge $(p, v) \in T_t$ that $\genfrac{[}{]}{0pt}{}{p}{v} \in \mathcal{G}$ as either $(p, v) \in T$ and thus added in the initialisation of the algorithm or the edge was created by the merge of an internal vertex in line 2 of the algorithm pseudocode Likewise if $v \in \text{leaves}(T_t)$, $c \in \downarrow_T(v)$ and $\ell_c \in \text{leaves}(T) \cap \Downarrow_T(c)$ then $\genfrac{[}{]}{0pt}{}{v}{\ell_c} \in \mathcal{G}$ as $\genfrac{[}{]}{0pt}{}{v}{\ell_c}$ was added in the initialisation step if $c \in \text{leaves}(T)$ otherwise there exists a $t' < t$ when $|T_{t'} \cap \Downarrow_T(c)| = 1$ when $\genfrac{[}{]}{0pt}{}{v}{\ell_c}$ was added to $\mathcal{G}$ via line 2 of the pseudocode

Suppose S is a tree added to $\mathcal{G}$ at time t by the merge operation (line 2) on a vertex v. Then, by Lemma 120, all trees in $\Omega(S, v)$ are t-maximal (and thus S is $(t+1)$-maximal by Lemma 118). By the discussion in the preceding paragraph all trees in $\Omega(S, v)$ have been added to $\mathcal{G}$ by time t so v is the split point of S in $\mathcal{G}$. Recalling Definition 100 we see that T (property 1) is in $\mathcal{G}$ from the initialisation step; by the preceding discussion property 2 is satisfied; and the fact that trees added to cover are formed by "merging" trees already within the cover (line 2) ensures that all added trees respect property 3 Thus $\mathcal{G}$ is a is a $(2, s)$-cover of T and furthermore it is greedy (Definition 115) therefore Lemma 116 applies so $\mathcal{G}$ is optimal height $\chi(T)$.

Now we argue that the algorithm requires no more than linear time. Note that each tree added to $\mathcal{G}$ in line 2 of the pseudocode is uniquely indexed by two vertices (trees with an boundary of two (in T) are indexed by p_v and c_v and trees with a boundary one (in T) are indexed by p_v and z) so it takes constant time to add each tree. Computing the set M_t takes linear time via a depth first search. Hence, at each time t, we have a time complexity proportional to $|T_{t-1}|$ (where $T_{-1} := T$). Let X_t be the set of vertices of degree at most two in T_t when $\mathcal{V}(T_t) \neq s$. By the handshaking lemma the sum of the degrees of all vertices in the subtree containing all vertices of T_t except s is equal to $2|T_t| - 4$ so the number of vertices of degree 3 or more is less than $\frac{2}{3}|T_t|$. Hence, $|X_t| > \frac{1}{3}|T_t|$. We have that $|M_{t+1}| \geq \frac{1}{2}|X_t| - 1$, since if some $v \in X_t/s$ is not in M_{t+1}, then it must be an internal vertex of T_t and its child must be in M_{t+1}. Thus since $|T_{t+2}| = |T_{t+1}| - |M_{t+1}| < |T_t| - |M_{t+1}|$ we have that $|T_{t+2}| - 1 < \frac{5}{6}|T_t| + 1$. Since the time complexity of time step t is linear in $|T_t| - 1$ the total time complexity is upper bounded by a quantity proportional to $2[(|T| + (\frac{5}{6}|T| + 1) + (\frac{5}{6}(\frac{5}{6}|T| + 1) + 1) + ...)]$. Since at least one vertex is removed in each time step there are no more that $|T|$ terms in this sum. Hence, this quantity is upper bounded by $2[\sum_{t\geq 0} \left(\frac{5}{6}\right)^t |T| + |T| \sum_{t\geq 0} \left(\frac{5}{6}\right)^t]$ which is equal to $24|T|$.

□

5.6.6 Converting the Greedy Cover into a Junction Tree

The greedy cover is converted into a junction tree $\mathcal{U}$ of T as follows:
Let (D, r) be the decomposition tree associated with the greedy cover. We have an isomorphism in tree structure, $\lambda : \mathcal{V}(D) \to \mathcal{V}(\mathcal{U})$ such that for all $v \in T^\bullet$ we have $\lambda(v) := \{v\} \cup \partial_0^T(S^v)$ and for all $(v, w) \in \mathcal{E}(T)$ we have $\lambda((v, w)) := \{v, w\}$

Note that $\mathcal{U}$ has a height of $\mathcal{O}(\chi^*(T))$ and each super-vertex is of constant cardinality. Hence, online Hugin/ARCH-2 takes a time of $\mathcal{O}(\chi^*(T))$ per update/query. Hence, update/query time is logarithmic in the second-diameter of T. Note also that since $\mathcal{V}(\mathcal{U})$ has a cardinality no greater than $2|\mathcal{V}(T)|$ and since each super-vertex has constant cardinality, the space requirement for online Hugin/ARCH-2 is only linear in $|\mathcal{V}(T)|$ which is (up to a constant factor) optimal for the online junction tree algorithm.

5.6.7 The Optimality of the Junction Tree

In this subsection we prove that, given a junction tree of a tree T, online Hugin/ARCH-2 has an update/query time of at least $\Omega(\chi^*(T))$ thereby proving the optimality of the junction tree constructed in this section.

In this subsection we shall assume that all junction trees are rooted: the root of a junction tree $\mathcal{J}$ is denoted by $r(\mathcal{J})$. In this section we denote super-vertices of junction trees by upper-case greek letters rather than the upper-case latin letters. Subtrees of T will be represented by upper-case latin letters.

In this subsection we will need the following definitions:

Definition 122. *Given a junction tree $\mathcal{J}$, the* **complexity**, *$\mathfrak{P}(\mathcal{J})$, of $\mathcal{J}$ is defined as:*

$$\mathfrak{P}(\mathcal{J}) := \max_{\Gamma \in \mathcal{J}} \sum_{\Lambda \in \Uparrow_{\mathcal{J}}(\Gamma)} 2^{|\Lambda|} \tag{5.49}$$

Note that the complexity of a junction tree is the time complexity of online Hugin propagation when run on that junction tree.

Definition 123. *Given a subtree $S \subseteq T$, we define S° to be the subtree of S who's vertices are those in S that aren't leaves of T.*

Definition 124. *Given a subtree $S \subseteq T$, a junction tree $\mathcal{J}$ is an S-junction tree if and only if it is a junction tree of S° with $\partial_0^T(S) \subseteq r(\mathcal{J})$. Given an S-junction tree $\mathcal{J}$ we define $\mathfrak{C}(\mathcal{J}) := r(\mathcal{J}) \setminus \partial_0^T(S)$.*

Definition 125. *Generalising the definition of the* **split set** *we define, for a subtree $S \subseteq T$ and a set $X \subseteq S^\bullet$, $\Omega(S, X)$ to be the set of subtrees, Q, of S that satisfy:*

1. $|\mathcal{V}(Q)| \geq 2$
2. *Given any* $v \in \mathcal{V}(Q) \setminus X$ *all neighbours of* v *(in* S*) are in* $\mathcal{V}(Q)$
3. *No element of* X *is in* $Q^{\bullet}$*.*

Intuitively, when S is a tree and $X \subset S^{\bullet}$, $\Omega(S, X)$ is the set of trees formed by splitting S at all vertices in X simultaneously.

Lemma 126. *Given an* S*-junction tree* $\mathcal{J}$*, then for every* $Q \in \Omega(S, \mathfrak{C}(\mathcal{J}))$ *with* $|Q| > 2$ *there exists a child,* Γ*, of* $r(\mathcal{J})$ *such that* $\mathcal{V}(Q^{\circ}) \in \bigcup \Downarrow_{\mathcal{J}}(\Gamma)$ *and, for every edge* $(x, y) \in \mathcal{E}(Q^{\circ})$ *there exists some* $\Lambda' \in \Downarrow_{\mathcal{J}}(\Gamma)$ *with* $x, y \in \Lambda'$*. In addition we have that* $\partial_0^T(Q) \subseteq \Gamma$*.*

Proof. Since $|Q| > 2$ there exists a vertex v in $Q^{\bullet}$. Note that $v \in S^{\circ}$ so there exists $\Lambda \in \mathcal{V}(\mathcal{J})$ with $v \in \Lambda$. Since $v \in Q^{\bullet}$, $v \notin \partial_0^T(S) \cup \mathfrak{C}(\mathcal{J}) = r(\mathcal{J})$. Hence we have $\Lambda \neq r(\mathcal{J})$ so let Γ be the child of $r(\mathcal{J})$ that has Λ as a descendant.
Let w be an arbitrary neighbour of v in Q°. Then since (v, w) is an edge in S° we have (by definition of a junction tree of S°) that $v, w \in \Lambda'$ for some $\Lambda' \in \mathcal{V}(\mathcal{J})$. If Λ' is not a descendant of Γ then $r(\mathcal{J})$ is in the path from Λ' to Λ. Since Λ and Λ' both contain v this implies, by the running intersection property, that $r(\mathcal{J})$ contains v which contradicts the above. We hence have that $\Lambda' \in \Downarrow_{\mathcal{J}}(\Gamma)$ so we have $w \in \bigcup \Downarrow_{\mathcal{J}}(\Gamma)$ and $v, w \in \Lambda' \in \Downarrow_{\mathcal{J}}(\Gamma)$
We have hence shown that every neighbour of v in Q° is in $\bigcup \Downarrow_{\mathcal{J}}(\Gamma)$ and also for every neighbour, w of v there is a super-vertex $\Lambda' \in \Downarrow_{\mathcal{J}}(\Gamma)$ such that $v, w \in \Lambda'$. Doing the same argument recursively on the neighbours of v in Q° gives us $\mathcal{V}(Q^{\circ}) \in \bigcup \Downarrow_{\mathcal{J}}(\Gamma)$ and, for every edge $(x, y) \in \mathcal{E}(Q^{\circ})$ there exists some $\Lambda' \in \Downarrow_{\mathcal{J}}(\Gamma)$ with $x, y \in \Lambda'$. This proves the first part of the lemma.
We now prove the second part of the lemma: Note first that, by definition of $\Omega(S, \mathfrak{C}(\mathcal{J}))$ we have that the leaves of Q are either leaves of S or in the set $\mathfrak{C}(\mathcal{J})$ which implies that $\partial_0^T(Q) \subseteq \partial_0^T(S) \cup \mathfrak{C}(\mathcal{J}) = r(\mathcal{J})$. Let v be an arbitrary vertex in $\partial_0^T(Q)$ and let w be the neighbour of v in Q. Since $|Q| > 2$, $w \in Q^{\bullet}$ so, by definition of $\Omega(S, \mathfrak{C}(\mathcal{J}))$ we have that $w \notin \mathfrak{C}(\mathcal{J})$. Since $w \in Q^{\bullet}$ we have $w \in S^{\bullet}$ so $w \notin \partial_0^T(S)$. We have hence shown that $w \notin r(\mathcal{J})$. Since (v, w) is an edge in S° we have, by definition of a junction tree, that there exists $\Lambda' \in \mathcal{V}(\mathcal{J})$ with $v, w \in \Lambda'$. From above we have that there exists $\Lambda \in \Downarrow_{\mathcal{J}}(\Gamma)$ with $w \in \Lambda$. If Λ' is not a descendant of Γ then $r(\mathcal{J})$ is in the path from Λ to Λ' so by the running intersection property (since $w \in \Lambda$ and $w \in \Lambda'$) we have $w \in r(\mathcal{J})$ which contradicts the above. Hence we have that $\Lambda' \in \Downarrow_{\mathcal{J}}(\Gamma)$ so Γ is in the path from $r(\mathcal{J})$ to Λ'. Since $v \in \Lambda'$ and (from above) $v \in r(\mathcal{J})$ we have that $v \in \Gamma$. Since v was an arbitrary member of $\partial_0^T(Q)$ we have hence proved the second part of the lemma. □

Lemma 127. *Given an* S*-junction tree* $\mathcal{J}$*, then for every* $Q \in \Omega(S, \mathfrak{C}(\mathcal{J}))$ *with* $|Q| > 2$ *there exists a* Q*-junction tree* $\mathcal{K}$ *which satisfies:*

$$\mathfrak{P}(\mathcal{K}) \leq \mathfrak{P}(\mathcal{J}) - 2^{|r(\mathcal{J})|} \tag{5.50}$$

Proof. Choose a child, Γ, as in lemma 126. Let $\mathcal{K}'$ be the tree $\Downarrow_{\mathcal{J}}(\Gamma)$. Form $\mathcal{K}$ from $\mathcal{K}'$ by removing, from all super-vertices in $\mathcal{K}'$, all vertices that aren't in $\mathcal{V}(Q)$. Since $\mathcal{V}(Q^{\circ}) \in \bigcup \Downarrow_{\mathcal{J}}(\Gamma)$ we automatically have that $\bigcup \mathcal{V}(\mathcal{K}) = \mathcal{V}(Q^{\circ})$. The running intersection property on $\mathcal{K}$ is also inherited from $\mathcal{J}$. Since for every edge $(x, y) \in \mathcal{E}(Q^{\circ})$ there exists some $\Lambda' \in \Downarrow_{\mathcal{J}}(\Gamma)$ with $x, y \in \Lambda'$ we have that there exists $\Lambda \in \mathcal{K}$ with $x, y \in \Lambda$. This proves that $\mathcal{K}$ is a junction tree of Q°. By lemma 126 we have that $\partial_0^T(Q) \subseteq \Gamma$ so $\partial_0^T(Q) \subseteq r(\mathcal{K})$. We have hence shown that $\mathcal{K}$ is a Q-junction tree.
We now bound the complexity of $\mathcal{K}$: Since, in going from $\mathcal{K}'$ to $\mathcal{K}$, no super-vertex

increases in size we have:

$$
\begin{aligned}
\mathfrak{P}(\mathcal{K}) &= \max_{\Lambda\in\mathcal{K}} \sum_{\Lambda'\in\Uparrow_{\mathcal{K}}(\Lambda)} 2^{|\Lambda'|} && (5.51)\\
&\leq \max_{\Lambda\in\mathcal{K}'} \sum_{\Lambda'\in\Uparrow_{\mathcal{K}'}(\Lambda)} 2^{|\Lambda'|} && (5.52)\\
&= \max_{\Lambda\in\mathcal{K}'} \sum_{\Lambda'\in\Uparrow_{\mathcal{J}}(\Lambda)\setminus r(\mathcal{J})} 2^{|\Lambda'|} && (5.53)\\
&= \max_{\Lambda\in\mathcal{K}'} \left(\sum_{\Lambda'\in\Uparrow_{\mathcal{J}}(\Lambda)} 2^{|\Lambda'|}\right) - 2^{|r(\mathcal{J})|} && (5.54)\\
&\leq \max_{\Lambda\in\mathcal{J}} \left(\sum_{\Lambda'\in\Uparrow_{\mathcal{J}}(\Lambda)} 2^{|\Lambda'|}\right) - 2^{|r(\mathcal{J})|} && (5.55)\\
&= \mathfrak{P}(\mathcal{J}) - 2^{|r(\mathcal{J})|} && (5.56)
\end{aligned}
$$

which completes the proof of the lemma. □

Lemma 128. *Given an S-junction tree, $\mathcal{J}$, there exists an hierarchical cover of S with height no greater than $\mathfrak{P}(\mathcal{J})$.*

Proof. We prove by induction on $\mathfrak{P}(\mathcal{J})$. If $\mathfrak{P}(\mathcal{J}) = 2$ then (since $\mathfrak{P}(\mathcal{J}) \geq 2^{|r(\mathcal{J})|}$, with the inequality being strict if $\mathcal{J}$ has more than one vertex) we must have that $\mathcal{J}$ has a single super-vertex $r(\mathcal{J})$ which contains a single vertex v. By definition of an S-junction tree we must then have that $\mathcal{V}(S^\circ) = v$. Hence, all vertices in $\mathcal{V}(S) \setminus \{v\}$ must be leaves of T, hence leaves of S and hence are neighbours of v. We hence have that all trees in $\Omega(S, \{v\})$ contain only two vertices so the set $\{S\} \cup \Omega(S, \{v\})$ is an hierarchical cover of S of height 1. The inductive hypothesis hence holds for $\mathfrak{P}(\mathcal{J}) = 2$.
Suppose that the inductive hypothesis holds for all $\mathfrak{P}(\mathcal{J}) \leq i$ (for some $i \geq 2$). Now suppose that $\mathfrak{P}(\mathcal{J}) = i + 1$. We construct an hierarchical cover, $\mathcal{H}$ of S via the following algorithm:

1. Set $\mathcal{H} = \{S\}$
2. For every $v \in \mathfrak{C}(\mathcal{J})$ in turn do the following:
   (a) Let Q be the tree in $\mathcal{H}$ of which no subtrees of Q are in $\mathcal{H}$ and we have $v \in \mathcal{V}(Q)$. Set $\mathcal{H} \leftarrow \mathcal{H} \cup \Omega(Q, \{v\})$.
3. For every $Q \in \Omega(S, \mathfrak{C}(\mathcal{J}))$ in turn do the following:
   (a) Choose an minimum height hierarchical cover, $\mathcal{H}'(Q)$, of Q and set $\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{H}'(Q)$.

We now bound the height of $\mathcal{H}$. By lemma 127 we have that, for every $Q \in \Omega(S, \mathfrak{C}(\mathcal{J}))$ there exists a Q-junction tree $\mathcal{K}$ which satisfies $\mathfrak{P}(\mathcal{K}) \leq \mathfrak{P}(\mathcal{J}) - 2^{|r(\mathcal{J})|}$. Since $\mathfrak{P}(\mathcal{K}) < \mathfrak{P}(\mathcal{J})$ we have, by the inductive hypothesis, that there exists an hierarchical cover of Q with height no greater than $\mathfrak{P}(\mathcal{K})$. We must hence have that the height of $\mathcal{H}'(Q)$ (in the above algorithm) is no greater than $\mathfrak{P}(\mathcal{K})$ and hence no greater than $\mathfrak{P}(\mathcal{J}) - 2^{|r(\mathcal{J})|}$. We hence have that the height of $\mathcal{H}$ is no greater than $|\mathfrak{C}(\mathcal{J})| + \mathfrak{P}(\mathcal{J}) - 2^{|r(\mathcal{J})|}$ which (since $|\mathfrak{C}(\mathcal{J})| \leq 2^{|r(\mathcal{J})|}$) is no greater than $\mathfrak{P}(\mathcal{J})$. The inductive hypothesis hence holds in this case. □

Theorem 129. *Given that $\mathcal{J}$ is a junction tree of T we have that $\chi^*(T) \leq \mathfrak{P}(\mathcal{J})$.*

Proof. Form the junction tree $\mathcal{K}$ by removing, from all super-vertices $\Gamma \in \mathcal{V}(\mathcal{J})$, all leaves of T. It is easy to check that $\mathcal{K}$ satisfies all the rules of a junction tree of T°. Since $\partial_0^T(T) = \emptyset$ we hence have that $\mathcal{K}$ is a T-junction tree. By Lemma 128 there hence exists an hierarchical cover of T with height no greater than $\mathfrak{P}(\mathcal{K})$. We hence have that $\chi^*(T) \leq \mathfrak{P}(\mathcal{K})$. Since, in forming $\mathcal{K}$, no super-vertex of $\mathcal{J}$ increases in cardinality, we have that $\mathfrak{P}(\mathcal{K}) \leq \mathfrak{P}(\mathcal{J})$. Combining with the above we get the result. $\square$

```
Input: Rooted tree $(T, s)$.

Initialisation: $T_0 \leftarrow T^\bullet \cup \{s\}$;        $t \leftarrow 0$;
                $\mathcal{G} \leftarrow \left\{ \left[\begin{smallmatrix}\uparrow^s_T(v)\\ v\end{smallmatrix}\right] \;:\; v \in \mathcal{V}(T) \setminus \{s\} \right\}$.

While $\big(\mathcal{V}(T_t) \neq \{s\}\big)$
    1. Construct $\mathcal{Z}_t$ via depth-first search
       of $T_t$ from $s$.
    2. For all $v \in \mathcal{Z}_t$, merge as follows:
       If $v \in \text{leaves}(T_t)$ then
              $z \leftarrow \downarrow^s_T(p^t_v \mapsto v)$.
              $\mathcal{G} \leftarrow \mathcal{G} \cup \left[\begin{smallmatrix}p^t_v\\ \vec{z}\end{smallmatrix}\right]$.
          Else $\mathcal{G} \leftarrow \mathcal{G} \cup \left[\begin{smallmatrix}p^t_v\\ c^t_v\end{smallmatrix}\right]$.
    3. $\mathcal{V}(T_{t+1}) \leftarrow \mathcal{V}(T_t) \setminus \mathcal{Z}_t$.
    4. $\mathcal{E}(T_{t+1}) \leftarrow \{(v, p^t_v) : v, p^t_v \in \mathcal{V}(T_{t+1})\} \cup$
                         $\{(v, g^t_v) : v, g^t_v \in \mathcal{V}(T_{t+1}),$
                           $p^t_v \notin \mathcal{V}(T_{t+1})\}$.
    5. $t \leftarrow t + 1$.

Output: Optimal $(2, s)$-hierarchical cover $\mathcal{G}$ of $T$.
```

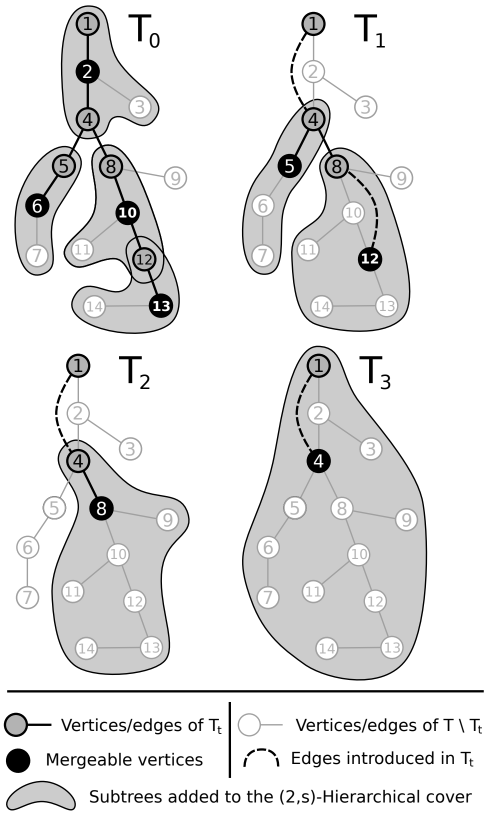


FIGURE 5.1: **Left**: Pseudocode for the linear time construction algorithm for an optimal $(2, s)$-hierarchical cover. **Right**: Pictorial explanation of the pseudocode and the details of the hierarchical cover.

In order to clarify the method, we describe some of the details of the cover and some merge operations that are performed in the diagram. Vertex 1 is the root vertex s. In each component, depicted as enclosed in a line, the black node is the splitting vertex, i.e. the mergeable vertex for the tree T_t in which is depicted. The boundary definition can be clarified by highlighting, for instance, that $\partial^T_0(S^2) = \{4\}$ and $\partial^T_0(S^{10}) = \{8, 12\}$. Subtree S^2 contains vertices 1, 2, 3 and 4. Vertex 2 is the splitting vertex of S^2. $\Omega(S^2, 2) = \{S^{(1,2)}, S^{(2,3)}, S^{(2,4)}\}$, i.e. at time $t = 0$, S^2 is formed by merging the three atomic subtrees $S^{(1,2)}$, $S^{(2,3)}$ and $S^{(2,4)}$, which were added in the initialization step. These three subtrees overlap at only vertex 2, which is depicted in black because it is mergeable in T_0. For what concerns the decomposition tree (D, r), we have $\downarrow^r_D(5) = \{(4, 5), 6\}$, which implies that S^5 is therefore formed by the atomic component $S^{(4,5)}$ and the non-atomic component S^6. At time $t = 1$, S^{12} is obtained by merging S^{10} together with S^{13}, which have been both created at time $t = 0$. Observe that in T_1 vertex 12 is a leaf and the z variable in the while-loop step 2 is assigned to vertex 10 (v and and p^t_v is respectively vertex 12 and 8). Regarding the subtree representation with the square bracket notation we can write, for instance, $S^2 = \left[\begin{smallmatrix}1\\ 4\end{smallmatrix}\right]$ and $S^{12} = \left[\begin{smallmatrix}8\\ \vec{10}\end{smallmatrix}\right] (\equiv \left[\begin{smallmatrix}8\\ 11\end{smallmatrix}\right] \equiv \left[\begin{smallmatrix}8\\ 14\end{smallmatrix}\right])$. Observe that, according to the definition of $(2, s)$-hierarchical cover, we have $4 \in \Downarrow^1_T(1)$ and $10 \in \downarrow^1_T(8)$. Finally, notice that the height of the $(2, s)$-cover of S^v is equal to $t + 1$ iff v is depicted in black in T_t.

Bibliography

[1] U. A. Acar et al. “Adaptive Bayesian Inference”. In: *NIPS*. 2007.

[2] U. A. Acar et al. “Adaptive Inference on General Graphical Models”. In: *UAI*. 2008.

[3] N. Alon et al. “Many random walks are faster than one”. In: *Comb. Probab. Comput.* 20.4 (2011), pp. 481–502.

[4] Michael O. Ball and J. Scott Provan. “Calculating bounds on reachability and connectedness in stochastic networks”. In: *Networks* 13.2 (1983), pp. 253–278.

[5] J. M. Barzdin and R. V. Frievald. “On the prediction of general recursive functions”. In: *Soviet Math. Doklady* 13 (1972), pp. 1224–1228.

[6] J. Basilico and T. Hofmann. “Unifying collaborative and content-based filtering”. In: *Proc of the 21st ICML*. ICML. 2004.

[7] Mikhail Belkin and Partha Niyogi. “Semi-Supervised Learning on Riemannian Manifolds”. In: *Mach. Learn.* 56.1-3 (2004), pp. 209–239. ISSN: 0885-6125. DOI: `http://dx.doi.org/10.1023/B:MACH.0000033120.25363.1e`.

[8] A. Ben-Dor, R. Shamir, and Z. Yakhini. “Clustering gene expression patterns”. In: *Journal of Computational Biology* 6(3/4) (1999).

[9] Avrim Blum and Shuchi Chawla. “Learning from labeled and unlabeled data using graph mincuts.” In: *Proceedings of the Eighteenth International Conference on Machine Learning, ICML 1*. 2001, pp. 19–26.

[10] A. Broder. “Generating random spanning trees”. In: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*. SFCS ’89. IEEE Computer Society, 1989, pp. 442–447.

[11] C. Brunner et al. “Pairwise support vector machines and their application to large scale problems”. In: *Journal of Machine Learning Research* 13 (2012), pp. 2279–2292.

[12] E. Candes and B. Recht. “Exact matrix completion via convex optimization”. In: *Foundations of Computational Mathematics* 9(6) (2009), pp. 717–772.

[13] E. Candes and T. Tao. “The power of convex relaxation: near-optimal matrix completion”. In: *IEEE Transactions on Information Theory* 56 (2010), pp. 2053–2080.

[14] Q. Cao, Z. Guo, and Y. Ying. “Generalization Bounds for Metric and Similarity Learning”. In: *CoRR* abs/1207.5437 (2012).

[15] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. “Linear Algorithms for Online Multitask Classification”. In: *Journal of Machine Learning Research* 11 (2010), pp. 2901–2934.

[16] Nicolò Cesa-Bianchi, Claudio Gentile, and Fabio Vitale. “Fast and Optimal Prediction on a Labeled Tree.” In: *Proceedings of the 22nd Annual Conference on Learning*. Omnipress, 2009.

[17] Nicolò Cesa-Bianchi et al. "Random Spanning Trees and the Prediction of Weighted Graphs". In: *Proceedings of the 27th International Conference on Machine Learning (27th ICML)*. 2010, pp. 175–182.

[18] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. "Cluster Kernels for Semi-Supervised Learning". In: *Advances in Neural Information Processing Systems 15*. Ed. by S. Becker, S. Thrun, and K. Obermayer. MIT Press, 2003, pp. 601–608. URL: http://papers.nips.cc/paper/2257-cluster-kernels-for-semi-supervised-learning.pdf.

[19] J. Davis et al. "Information-theoretic metric learning". In: *Proceedings of the 24th international conference on Machine learning*. ICML '07. 2007, pp. 209–216.

[20] Arthur L. Delcher et al. "Logarithmic-time updates and queries in probabilistic networks". In: *J. Artif. Int. Res.* 4 (1 1996), pp. 37–59. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622737.1622740.

[21] A. Demiriz, K. Bennett, and M.J. Embrechts. "Semi-supervised clustering using genetic algorithms". In: *In Artificial Neural Networks in Engineering (ANNIE-99)*. 1999, pp. 809–814.

[22] L. R. Ford and D. R. Fulkerson. "Maximal Flow through a Network." In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. URL: http://www.rand.org/pubs/papers/P605/.

[23] S. Lauritzen F.V. Jensen and K. Olesen. "Baysesian updating in recursive graphical models by local computation". In: *Computational Statistics Quarterly* 4 (1990), pp. 269–282.

[24] Thomas Gärtner and Gemma C. Garriga. "The Cost of Learning Directed Cuts." In: *Proceedings of the 18th European Conference on Machine Learning*. 2007.

[25] Leslie Ann Goldberg and Mark Jerrum. "The Complexity of Ferromagnetic Ising with Local Fields." In: *Combinatorics, Probability & Computing* 16.1 (Nov. 20, 2008), pp. 43–61.

[26] D. Gross. "Recovering low-rank matrices from few coefficients in any basis". In: *IEEE Transactions on Information Theory* 57/3 (2011), pp. 1548–1566.

[27] E. Hazan, S. Kale, and S. Shalev-Shwartz. "Near-optimal algorithms for online matrix prediction". In: *Proceedings of the 25th Annual Conference on Learning Theory (COLT'12)*. 2012.

[28] Mark Herbster. "Exploiting cluster-structure to predict the labeling of a graph." In: *Proceedings of the 19th International Conference on Algorithmic Learning Theory*. 2008, pp. 54–69.

[29] Mark Herbster and Guy Lever. "Predicting the labelling of a graph via minimum p-seminorm interpolation". In: *Proceedings of the 22nd Annual Conference on Learning Theory (COLT'09)*. 2009.

[30] Mark Herbster, Guy Lever, and Massimiliano Pontil. "Online Prediction on Large Diameter Graphs". In: *Advances in Neural Information Processing Systems (NIPS 22)*. MIT Press, 2009, pp. 649–656.

[31] Mark Herbster, Guy Lever, and Massimiliano Pontil. "Online Prediction on Large Diameter Graphs." In: *NIPS*. Note, referenced prediction algorithm is in an extended version in preparation, 2011. MIT Press, Apr. 15, 2009, pp. 649–656.

[32] Mark Herbster, Massimiliano Pontil, and Lisa Wainer. "Online learning over graphs". In: *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM, 2005, pp. 305–312.

[33] S. M. Kakade, S. Shalev-Shwartz, and A. Tewari. "Regularization Techniques for Learning with Matrices". In: *The Journal of Machine Learning Research* (2012), pp. 1865–1890.

[34] V. Koltchinskii, K. Lounici, and A. Tsybakov. "Nuclear norm penalization and optimal rates for noisy matrix completion". In: *Annals of Statistics* 39(5) (2011), pp. 2302–2329.

[35] V. Koltchinskii and P. Rangel. "Low rank estimation of similarities on graphs". In: *CoRR* (2012). eprint: arXiv/1205.1868.

[36] V. Lepar and P.P. Shenoy. "A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions". In: *UAI'98 Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (1998), pp. 328–337.

[37] Z. Li, J. Liu, and X. Tang. "Pairwise constraint propagation by semidefinite programming for semi-supervised classification". In: *Proc of the 25th ICML*. ICML. 2008.

[38] Nick Littlestone. "Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm". In: *Machine Learning* 2 (1988), pp. 285–318.

[39] Nick Littlestone and Manfred K. Warmuth. "The Weighted Majority Algorithm". In: *Inf. Comput.* 108.2 (1994), pp. 212–261.

[40] R. Lyons and Y. Peres. *Probability on Trees and Networks*. In preparation. Current version available at http://mypage.iu.edu/~rdlyons/. Cambridge University Press, 2012.

[41] A. Maurer. "Learning similarity with operator-valued large-margin classifiers". In: *Journal of Machine Learning Research* 9 (2008), pp. 1049–1082.

[42] S. Negahban and M. Wainwright. "Restricted strong convexity and weighted matrix completion with noise". In: *Preprint* (2010).

[43] J.D. Park and A. Darwiche. "A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions". In: *UAI'98 Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (1998), pp. 328–337.

[44] J. Pearl. "Reverend Bayes on inference engines: A distributed hierarchical approach". In: *Proceedings of the American Association of Artificial Intelligence National Conference on AI* (1982), pp. 133–136.

[45] Jean-Claude Picard and Maurice Queyranne. "On the structure of all minimum cuts in a network and applications". In: *Combinatorial Optimization II*. Ed. by V.J. Rayward-Smith. Vol. 13. Mathematical Programming Studies. Springer Berlin Heidelberg, 1980, pp. 8–16. ISBN: 978-3-642-00803-0. DOI: 10.1007/BFb0120902. URL: http://dx.doi.org/10.1007/BFb0120902.

[46] J. Scott Provan and Michael O. Ball. “The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected”. In: *SIAM Journal on Computing* 12.4 (1983), pp. 777–788. DOI: 10.1137/0212053. eprint: http://epubs.siam.org/doi/pdf/10.1137/0212053. URL: http://epubs.siam.org/doi/abs/10.1137/0212053.

[47] S. S. Rangapuram and M. Hein. “Constrained 1-Spectral Clustering”. In: *Proc. 15th International Conference on Artificial Intelligence and Statistics*. AISTATS. 2012.

[48] A. Rohde and A. Tsybakov. “Estimation of high-dimensional low rank matrices”. In: *Annals of Statistics* 39(2) (2011), pp. 887–930.

[49] T. Schmidt and P.P. Shenoy. “Some improvements to the Shenoy-Shafer and Hugin architectures for computing marginals”. In: *Artificial Intelligence* 102 (1998), pp. 323–333.

[50] G.R. Shafer and P.P. Shenoy. “Probability Propagation”. In: *Annals of Mathematics and Artificial Intelligence* 2 (1990), pp. 327–351.

[51] S. Shalev-Shwartz, Y. Singer, and A. Ng. “Online and batch learning of pseudo-metrics”. In: *Proceedings of the twenty-first international conference on Machine learning*. ICML ’04. ACM, 2004.

[52] P.P. Shenoy. “Binary Join Trees for Computing Marginals in the Shenoy-Shafer Architecture”. In: *International Journal of Approximate Reasoning* 17 (1997), pp. 239–263.

[53] P.P. Shenoy. “The inclusion-exclusion rule and its application to the junction tree algorithm”. In: *IJCAI ’13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence* (2013), pp. 2568–2575.

[54] Martin Szummer and Tommi Jaakkola. “Partially labeled classification with Markov random walks”. In: *NIPS*. 2001, pp. 945–952.

[55] K. Tsuda, G. Rätsch, and M. K. Warmuth. “Matrix exponentiated gradient updates for on-line learning and Bregman projections”. In: *Journal of Machine Learning Research* 6 (2005), pp. 995–1018.

[56] Fabio Vitale et al. “See the Tree Through the Lines: The Shazoo Algorithm”. In: *NIPS*. Ed. by John Shawe-Taylor et al. 2011, pp. 1584–1592.

[57] M. K. Warmuth. “Winnowing Subspaces”. In: *Proceedings of the 24th International Conference on Machine Learning*. ACM, 2007, pp. 999–1006.

[58] D. B. Wilson. “Generating random spanning trees more quickly than the cover time”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 296–303.

[59] J. Zhang and R. Yan. “On the value of pairwise constraints in classification and consistency”. In: *Proc of the 24th ICML*. ICML. 2007.

[60] Dengyong Zhou et al. “Learning with Local and Global Consistency”. In: *NIPS*. 2003.

[61] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. “Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions”. In: *ICML*. 2003, pp. 912–919.

[62] H. Zu. “An efficient implementation of belief function propagation”. In: *UAI’91 Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence* (1991), pp. 425–432.