Nearest Neighbour with Bandit Feedback

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper we adapt the nearest neighbour rule to the contextual bandit problem. Our algorithm handles the fully adversarial setting in which no assumptions at all are made about the data-generation process. When combined with a sufficiently fast data-structure for (perhaps approximate) adaptive nearest neighbour search, such as a navigating net, our algorithm is extremely efficient - having a per trial running time polylogarithmic in both the number of trials and actions, and taking only quasi-linear space. We give generic regret bounds for our algorithm and further analyse them in a semi-stochastic setting. A side result of this paper is that, when applied to the online classification problem with stochastic labels, our algorithm can have sublinear regret whilst only finding a single nearest neighbour per trial - in stark contrast to the k-nearest neighbours algorithm.

1 Introduction

In this paper we adapt the classic *nearest neighbour* rule to the contextual bandit problem and develop an extremely efficient algorithm. The problem proceeds in trials, where on trial t: (1) a *context* x_t is revealed to us, (2) we must select an *action* a_t, and (3) the *loss* $\ell_{t,a_t} \in [0,1]$ of action a_t on trial t is revealed to us. We assume that the contexts are points in a metric space and the distance between two contexts represents their similarity. A *policy* is a mapping from contexts to actions and the inductive bias of our algorithm is towards learning policies that typically map similar contexts to similar actions. Our main result has absolutely no assumptions whatsoever about the generation of the context/loss sequence and has no restriction on what policies we can compare our algorithm to.

Our algorithm requires, as a subroutine, a data-structure that performs c-nearest neighbour search. This data-structure must be *adaptive* - in that new contexts can be inserted into it over time. An example of such a data-structure is the *Navigating net* [15] which, given mild conditions on our metric and dataset, performs both search and insertion in polylogarithmic time. When utilising a data-structure of this speed our algorithm is extremely efficient - with a per-trial time complexity polylogarithmic in both the number of trials and actions, and requiring only quasi-linear space.

As an example we will further analyse the special case of the contextual bandit problem in which the context sequence is drawn i.i.d. from a probability distribution over the d-dimensional hypercube, whilst the loss vectors can still be arbitrary. In this case, for any policy y with a finite-volume decision boundary, our algorithm achieves sub-linear regret w.r.t. y without the need to know any parameters.

A side result of this paper is that, when applied to the online classification task with stochastic labels, our algorithm can achieve sublinear regret whilst only finding *one* nearest neighbour per trial: in stark contrast to the k-nearest neighbour algorithm. Our algorithm can hence be viewed also as a potentially faster alternative to k-nearest neighbours when faced with the online classification problem.

In the course of this paper we develop some novel algorithmic techniques, including a new algorithmic framework CanProp and efficient algorithms for searching in trees, which may find further application.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

We now describe related works. The bandit problem [17] was first introduced in [22] but was originally studied in the stochastic setting in which all losses are drawn i.i.d. at random [16], [1], [2]. However, our world is very often not i.i.d. stochastic. The work of [3] introduced the seminal EXP3 algorithm which handled the case in which the losses were selected arbitrarily. This work also introduced the EXP4 algorithm for contextual bandits. In general this algorithm is exponential time but in some situations can be implemented in polynomial time - such as their EXP3.S algorithm, which was a bandit version of the classic FIXEDSHARE algorithm [13]. In [11] the EXP3.S setting was greatly generalised to the situation in which the contexts where vertices of a graph. They utilised the methodology of [7], [14] and [12] in order to develop extremely efficient algorithms. Although inspiring this work, these algorithms cannot be utilised in our situation as they inherently require the set of queried contexts to be known a-priori. In the stochastic case another class of contextual bandit problems are *linear bandits* [18], [4] in which the contexts are mappings from the actions into $\mathbb{R}^d$. Here the queried contexts need not be known in advance but the losses must be drawn i.i.d. from a distribution that has mean linear in the respective context. The k nearest neighbour algorithm was first analysed in [5]. The work [21] utilised the k nearest neighbour methodology and the works [8] and [16] to handle a stochastic contextual bandit problem. However, their setting is extremely more restricted than ours. In particular, the context/loss pairs must be drawn i.i.d. at random and the probability distribution they are sampled from must obey certain strict conditions. In addition, on each trial the contexts seen so far must be ordered in increasing distance from the current context and operations must be performed on this sequence, making their algorithm exponentially slower than ours. Our algorithm utilises the works of [19] and [6] as subroutines. It should be noted that the later work, which was based on [20], was improved on in [10] - we leave it as an open problem as to whether we can utilise their work in our algorithm.

2 Notation

Let $\mathbb{N}$ be the set of natural numbers not including 0. Given a natural number $m \in \mathbb{N}$ we define $[m] := \{j \in \mathbb{N} \mid j \leq m\}$. Given a predicate p we define $[\![p]\!] := 1$ if p is true and $[\![p]\!] := 0$ otherwise. We define $\log(\cdot)$ and $\ln(\cdot)$ to be the logarithms with base 2 and e respectively. Given sets $\mathcal{A}$ and $\mathcal{B}$ we denote by $\mathcal{B}^{\mathcal{A}}$ the set of all functions $f : \mathcal{A} \to \mathcal{B}$ and by $2^{\mathcal{A}}$ the set of all subsets of $\mathcal{A}$.

All trees in this paper are considered rooted. Given a tree $\mathcal{J}$ we denote its root by $r(\mathcal{J})$, its vertex set by $\mathcal{J}$, its leaves by $\mathcal{J}^{\star}$, and its internal vertices by $\mathcal{J}^{\dagger}$. Given a vertex v in a tree $\mathcal{J}$ we denote its parent by $\uparrow_{\mathcal{J}}(v)$ and the subtree of all its descendants by $\Downarrow_{\mathcal{J}}(v)$. Given an internal node v in a (full) binary tree $\mathcal{J}$ we denote its left and right children by $\triangleleft_{\mathcal{J}}(v)$ and $\triangleright_{\mathcal{J}}(v)$ respectively. Internal nodes v in a (full) ternary tree $\mathcal{J}$ have an additional child $\triangledown_{\mathcal{J}}(v)$ called the *centre* child. Given vertices v and v' in a tree $\mathcal{J}$ we denote by $\Gamma_{\mathcal{J}}(v, v')$ the *least common ancestor* of v and v': i.e. the vertex of maximum depth which is an ancestor of both v and v'. We will drop the subscript $\mathcal{J}$ in all these functions when unambiguous. Given a tree $\mathcal{J}$, a *subtree* of $\mathcal{J}$ is a tree whose edge set is a subset of that of $\mathcal{J}$.

Given a probability distribution μ we write $x \sim \mu$ to signify that x is a random element drawn from μ. Given, in addition, some $m \in \mathbb{N}$, we define μ^m to be the probability distribution over sets formed by drawing m elements i.i.d. with replacment from distribution μ. With a slight overloading of notation we denote the uniform distribution over $[0, 1]$ also by $[0, 1]$.

3 Problem and Result

3.1 The Contextual Bandit Problem

We consider the following game between *Learner* (us) and *Nature* (our adversary). We have K *actions* and a metric space $(\mathcal{C}, \Delta)$ where $\mathcal{C}$ is a (possibly infinite) set of *contexts* and for all $x, x' \in \mathcal{C}$ we have that $\Delta(x, x')$ is the *distance* from x to x'. Learning proceeds in T trials. A-priori Nature chooses a sequence of contexts $\mathcal{X} = \{x_t \mid t \in [T]\} \subseteq \mathcal{C}$ and a sequence of loss vectors $\{\boldsymbol{\ell}_t \mid t \in [T]\} \subseteq [0, 1]^K$, but does not reveal them to Learner. On the t-th trial the following happens:

1. Nature reveals x_t to Learner.
2. Learner chooses some action $a_t \in [K]$.
3. Nature reveals ℓ_{t,a_t} to Learner.

A *policy* is a function from $\mathcal{C}$ into $[K]$. i.e. a policy associates each possible context with an action. Given a policy $y : \mathcal{C} \rightarrow [K]$ we define the *y-regret* of Learner as:

$$R(y) := \sum_{t \in [T]} \ell_{t,a_t} - \sum_{t \in [T]} \ell_{t,y(x_t)}$$

which is the difference between the total cumulative loss suffered by Learner and that which Learner would have suffered if it had instead chosen a_t equal to $y(x_t)$ for all trials t.

3.2 The (k) Nearest Neighbour Classifier

We now digress from the contextual bandit problem in order to study the nearest neighbour methodology. The *nearest neighbour classifier* is a method to solve the following supervised learning problem. We assume that there exists an unknown function $y : \mathcal{C} \rightarrow [K]$. We are given a finite set $\mathcal{S} \subseteq \mathcal{C}$ along with the restriction of y onto $\mathcal{S}$. We are then asked to predict the value of $y(x)$ for some given $x \in \mathcal{C}$. The nearest neighbour classifier works by first finding the nearest neighbour $\hat{x}$ of x, defined as:

$$\hat{x} := \operatorname{argmin}_{x' \in \mathcal{S}} \Delta(x, x') \,,$$

and then choosing $y(\hat{x})$ as its prediction of $y(x)$. In many important metric spaces the time taken to find such a nearest neighbour is in $\Theta(|\mathcal{S}|)$. This fact has lead to the idea of instead using $y(\tilde{x})$ as our prediction, where $\tilde{x} \in \mathcal{S}$ is an arbitrary *c-nearest neighbour* which is defined as satisfying:

$$\Delta(x, \tilde{x}) \leq c \min_{x' \in \mathcal{S}} \Delta(x, x') \,.$$

By utilising special data-structures the time complexity of finding, for any fixed $c > 1$, such a c-nearest neighbour is, for many metric spaces, only polylogarithmic in $|\mathcal{S}|$.

Given a probability distribution μ over $\mathcal{C}$, some $c \geq 1$ and some $m \in \mathbb{N}$ we define the *generalisation error* as:

$$g_m(\mu, y, c) := \mathbb{P}_{\mathcal{S} \sim \mu^m, x \sim \mu} \left[\exists\, z \in \mathcal{S} \,:\, \left(\Delta(x, z) \leq c \min_{x' \in \mathcal{S}} \Delta(x, x') \right) \wedge\, y(z) \neq y(x) \right]$$

which is the probability that it is possible for the nearest neighbour classifier to make a mistake on a context drawn from μ when $\mathcal{S}$ is formed by drawing m contexts i.i.d. from μ.

We will now bound this quantity when in euclidean space. We first make the following definitions. For any $\delta > 0$ define the *δ-margin* of y by:

$$\mathcal{M}(y, \delta) := \{x \in \mathcal{C} \mid \exists\, x' \in \mathcal{C} \,:\, \Delta(x, x') \leq \delta \,\wedge\, y(x) \neq y(x')\} \tag{1}$$

which is the set of contexts that are at distance no more than δ from the *decision boundary* of y. The *volume* (w.r.t. μ) of the decision boundary is then given by:

$$\alpha(y, \mu) := \lim_{\delta \rightarrow 0} \frac{\mu(\mathcal{M}(y, \delta))}{\delta} \,. \tag{2}$$

When in euclidean space the following theorem bounds the generalisation error:

Theorem 3.1. *Given $\mathcal{C} := [0,1]^d$ and Δ is the euclidean metric then for any $y : \mathcal{C} \rightarrow [K]$, $c \geq 1$, $\epsilon > 0$, and probability distribution μ such that the probability density of μ is always at least ϵ, we have:*

$$g_m(\mu, y, c) \in \tilde{\mathcal{O}} \left(c\, \alpha(y, \mu)\, (\epsilon m)^{-1/d} \right) \,.$$

So far we have only considered deterministic functions $y : \mathcal{C} \rightarrow [K]$ with decision boundaries of finite volume. But what happens if instead we have that $y(x)$ is drawn from some probability distribution dependent on x (which is itself drawn from μ). Here, the *Bayes optimal classifier* is defined as that which always predicts $y^*(x) := \operatorname{argmax}_{a \in [K]} \mathbb{P}[y(x) = a | x]$ as the label of x. In general, even if $g_m(\mu, y^*, c) \in o(1)$, the probability of making a mistake with the nearest neighbour classifier does not approach that of the Bayes optimal classifier as $m \rightarrow \infty$. In order to converge optimally, the k-nearest neighbour classifier was introduced. In this algorithm, when given a context $x \in \mathcal{C}$, the k nearest neighbours to x are found and the predicted value of $y(x)$ is decided by majority vote. In order to converge optimally we need that $k \rightarrow \infty$ as $m \rightarrow \infty$.

A remarkable side-result of this paper is that, given $g_m(\mu, y^*, c) \in \mathcal{O}(m^{-p})$ for some $p > 0$, our algorithm can be applied to learning this situation online whilst only finding *one* nearest neighbour per trial. Since the additional overhead of our algorithm is so small it can be significantly faster than k-nearest neighbours. We strongly suspect that we don't need the condition on $g_m(\mu, y^*, c)$ if we are working in a bounded subset of euclidean space and $\mathbb{P}[y(x) = a | x]$ is Lipschitz.

3.3 Adaptive Nearest Neighbour Search

Our algorithm will require a data-structure for performing *adaptive nearest neighbour search*. This problem is as follows. We maintain a finite set $\mathcal{S} \subseteq \mathcal{C}$. At any point in time we must either (1) insert a new context into the set $\mathcal{S}$ and update the data-structure, or (2) given a context, utilise the data-structure to find a c-nearest neighbour in the set $\mathcal{S}$.

We are especially interested in data-structures that can do both operations in a time polylogarithmic in $|\mathcal{S}|$. An example of such a data-structure is the *navigating net* [15] which has time complexity $\tilde{\mathcal{O}}(\ln(|\mathcal{S}|))$ given that $c > 1$, the set $|\mathcal{S}|$ is of bounded doubling dimension (w.r.t. Δ) and has aspect ratio (ratio between the largest and smallest distances between contexts in $\mathcal{S}$) polynomial in $|\mathcal{S}|$, as is the case in many applications and can be enforced by quantisation when working in a bounded subset of euclidean space. We note that the $\tilde{\mathcal{O}}$ hides a constant factor that is exponential in the doubling dimension of $\mathcal{S}$. In high-dimensional applications our dataset will often lie on a low-dimensional manifold so this factor should be small.

3.4 Our Results

We now turn back to the contextual bandit problem and give our main results.

Theorem 3.2. *Consider the contextual bandit problem defined in Section 3.1. Suppose that for all trials* $t > 1$ *we are given, in addition to* x_t*, a context* $n(x_t)$ *which satisfies:*

$$n(x_t) \in \{x_s \mid s \in [t-1]\} .$$

Our algorithm CBNN *takes a single parameter* $\rho > 0$ *and, for all policies* $y : \mathcal{C} \to [K]$ *simultaneously, obtains an expected* y*-regret bounded by:*

$$\mathbb{E}[R(y)] \in \tilde{\mathcal{O}}\left(\left(\rho + \frac{\Phi(y)}{\rho}\right)\sqrt{KT}\right)$$

where:

$$\Phi(y) := 1 + \sum_{t \in [T] \setminus \{1\}} [\![y(x_t) \neq y(n(x_t))]\!]$$

and the expectation is taken over the randomisation of the algorithm. CBNN *needs no initialisation time and has a per-trial time complexity of:*

$$\mathcal{O}(\ln(T)^2 \ln(K)) .$$

We note that, although n can be any valid function, we are particularly interested in the case that $n(x_t)$ is a c-nearest neighbour of x_t. i.e. that we have:

$$\Delta(x_t, n(x_t)) \leq c \min_{s \in [t-1]} \Delta(x_t, x_s) . \tag{3}$$

In this case finding $n(x_t)$ typically requires only $\tilde{\mathcal{O}}(\ln(T))$ time per trial when using a navigating net or other fast data-structure for adaptive nearest neighbour search, as explained in Section 3.3. Furthermore, the quantity $\Phi(y)$ can now be bounded in a way that is dependent only on the set of queried contexts $\mathcal{X}$ and not their order. This bound is given in the following theorem.

Theorem 3.3. *Suppose we have a policy* $y : \mathcal{C} \to [K]$*. For any context* $x \in \mathcal{C}$ *we define* $\gamma(x, y) := \max\{\delta \geq 0 \mid x \notin \mathcal{M}(y, \delta)\}$ *which is the distance of* x *from the decision boundary of* y*. Then when Equation* (3) *is satisfied we have that* $\Phi(y)$ *is no greater than the minimum cardinality of any set* $\mathcal{S} \subseteq \mathcal{C}$ *in which for all* $t \in [T]$ *there exists* $x \in \mathcal{S}$ *with* $\Delta(x, x_t) < \gamma(x, y)/3c$*.*

A direct corollary of this theorem is that for all $\delta > 0$ we have that:

$$\Phi(y) \leq N_\delta(\mathcal{X}) + |\mathcal{X} \cap \mathcal{M}(4c\delta, y)|$$

where $N_\delta(\mathcal{X})$ is the (external) covering number of $\mathcal{X}$ with radius δ, and $|\mathcal{X} \cap \mathcal{M}(4c\delta, y)|$ is the number of contexts in $\mathcal{X}$ lying within distance $4c\delta$ of the decision boundary.

It will be common in applications that the set $\mathcal{X}$ of observed contexts will come from a finite set of seperated clusters and there will be a good policy y which, on any such cluster, is constant on that cluster. Theorem 3.3 then implies that, as long as the inter-cluster distances are positive and the clusters have finite covering numbers (which is guaranteed in many metric spaces), then $\Phi(y)$ will be bounded independent of T and hence, by Theorem 3.2, the y-regret of CBNN will scale like $\tilde{\mathcal{O}}(\sqrt{T})$, whatever the value of ρ.

However, it will not always be the case that the dataset splits into such clusters. We shall investigate what happens when this is not the case by restricting ourselves to the situation in which the contexts $\{x_t \mid t \in [T]\}$ are drawn i.i.d. from a probability distribution μ. Here we have, by linearity of expectation, that:

$$\mathbb{E}[\Phi(y)] \leq 1 + \sum_{t \in [T]} g_t(\mu, y, c) .$$

When in euclidean space, theorems 3.1 and 3.2 then lead to the following theorem:

Theorem 3.4. *Consider the contextual bandit problem defined in Section 3.1. Suppose that $\mathcal{C} = [0,1]^d$, Δ is the euclidean metric, and the contexts are drawn i.i.d. at random from a probability distribution μ with density always at least $\epsilon > 0$. Note that the loss vectors can be arbitrary. Set ρ equal to $T^{-(d-1)/d}c^{-1/2}$. Then when Equation* (3) *is satisfied we have, for all policies $y : \mathcal{C} \to [K]$ simultaneously, that the y-regret of* CBNN *is bounded by:*

$$\mathbb{E}[R(y)] \in \tilde{\mathcal{O}}\left((\epsilon^{-1/d}\alpha(y,\mu) + 1)c^{1/2}K^{1/2}T^{(2d-1)/(2d)}\right)$$

where $\alpha(y, \mu)$ is the volume (w.r.t. μ) of the decision boundary of y as defined in equations (1) *and* (2)*. The existence of such an ϵ can be relaxed (with an effect on the bound) but we assume it for simplicity.*

Note that, given the decision boundary of y is of finite volume, the expected regret is guaranteed to be sub-linear in T. This implies that the per-trial performance of CBNN approaches that of always selecting $a_t = y(x_t)$. We note that if T is unknown or infinite (i.e. learning never stops) then a simple doubling trick can be used to make the algorithm parameter-free (with no knowledge of μ). The fact that, in this non-separated case, the regret scales like $\tilde{\mathcal{O}}(T^{(2d-1)/(2d)})$ is a facet of the well known *curse of dimensionality* and is the price we pay for being able to learn from such a vast class of policies. We note that in many high-dimensional applications the set $\mathcal{X}$ will lie on a low-dimensional manifold so that the curse of dimensionality will be significantly reduced.

4 The Algorithm

In this section we describe our algorithm CBNN and give the pseudocode for the novel subroutines. In appendices C to E we give a more detailed description of how CBNN works, and prove that it obtains its bounds.

To give the reader intuition, in Appendix B we describe our initial idea - an algorithm, based on EXP4 [3] and *Belief propagation* [20], which attains our regret bound but is exponentially slower - taking a per-trial time of $\tilde{\Theta}(KT)$.

4.1 Cancellation Propagation

In this subsection we describe a novel algorithmic framework CANPROP for designing contextual bandit algorithms with a running time logarithmic in K. It is inspired by EXP3 [3], specialist algorithms [7] and online decision-tree pruning algorithms [9] but is certainly not a simple combination of these works. CBNN will be an efficient implementation of an instance of CANPROP. Although in general CANPROP requires a-priori knowledge of the set $\mathcal{X} := \{x_t \mid t \in [T]\}$, CBNN is designed in a way that, crucially, does not need this set to be known.

We assume, without loss of generality, that K and T are integer powers of two. CANPROP, which takes a parameter $\eta > 0$, works on a full, balanced binary tree $\mathcal{B}$ with leaves $\mathcal{B}^\star = [K]$. On every trial t each pair $(v, \mathcal{S}) \in \mathcal{B} \times 2^{\mathcal{X}}$ has a weight $w_t(v, \mathcal{S}) \in [0, 1]$. These weights induce a function $\theta_t : \mathcal{B} \to [0, 1]$ defined by:

$$\theta_t(v) := \sum_{\mathcal{S} \in 2^{\mathcal{X}}} [\![x_t \in \mathcal{S}]\!] w_t(v, \mathcal{S}) .$$

On each trial t a root-to-leaf path $\{z_{t,j} \mid j \in [\log(K)] \cup \{0\}\}$ is sampled such that, given $z_{t,j}$, we have that $z_{t,(j+1)}$ is sampled from $\{\triangleleft(z_{t,j}), \triangleright(z_{t,j})\}$ with probability proportional to the value of θ_t

when applied to each of these vertices. The action a_t is then chosen equal to $z_{t,\log(K)}$. Once the loss has been observed we climb back up the root-to-leaf path, updating the function w_t to w_{t+1}.

CANPROP (at trial t) is given in Algorithm 1. We note that if $w_{t+1}(v, \mathcal{S})$ is not set in the pseudocode then it is defined to be equal to $w_t(v, \mathcal{S})$.

Algorithm 1 CANPROP at trial t

$v_{t,0} \leftarrow r(\mathcal{B})$
for $j = 0, 1, \cdots, (\log(K) - 1)$ **do**
  for $v \in \{\triangleleft(v_{t,j}), \triangleright(v_{t,j})\}$ **do**
    $\theta_t(v) \leftarrow \sum_{\mathcal{S} \in 2^{\mathcal{X}}} [\![x_t \in \mathcal{S}]\!] w_t(v, \mathcal{S})$
  end for
  $z_{t,j} \leftarrow \theta_t(\triangleleft(v_{t,j})) + \theta_t(\triangleright(v_{t,j}))$
  for $v \in \{\triangleleft(v_{t,j}), \triangleright(v_{t,j})\}$ **do**
    $\pi_t(v) \leftarrow \theta_t(v)/z_{t,j}$
  end for
  $\zeta_{t,j} \sim [0, 1]$
  if $\zeta_{t,j} \leq \pi_t(\triangleleft(v_{t,j}))$ **then**
    $v_{t,j+1} \leftarrow \triangleleft(v_{t,j})$
  else
    $v_{t,j+1} \leftarrow \triangleright(v_{t,j})$
  end if
end for
$a_t \leftarrow v_{t,\log(K)}$
$\tilde{\pi}_t \leftarrow \prod_{j \in [\log(K)]} \pi_t(v_{t,j})$
$\psi_{t,\log(K)} \leftarrow \exp(-\eta \ell_{t,a_t}/\tilde{\pi}_t)$
for $j = \log(K), (\log(K) - 1), \cdots, 1$ **do**
  $\psi_{t,(j-1)} \leftarrow 1 - (1 - \psi_{t,j})\pi_t(v_{t,j})$
  $\psi'_{t,j} \leftarrow \psi_{t,j}/\psi_{t,j-1}$
  if $v_{t,j} = \triangleleft(v_{t,j-1})$ **then**
    $\tilde{v}_{t,j} \leftarrow \triangleright(v_{t,j-1})$
  else
    $\tilde{v}_{t,j} \leftarrow \triangleleft(v_{t,j-1})$
  end if
  for $\mathcal{S} \in 2^{\mathcal{X}} : x_t \in \mathcal{S}$ **do**
    $w_{t+1}(v_{t,j}, \mathcal{S}) \leftarrow w_t(v_{t,j}, \mathcal{S})\psi'_{t,j}$
    $w_{t+1}(\tilde{v}_{t,j}, \mathcal{S}) \leftarrow w_t(\tilde{v}_{t,j}, \mathcal{S})/\psi_{t,j-1}$
  end for
end for

In Appendix C we give a general regret bound for CANPROP. For CBNN we set $\eta := \rho\sqrt{\ln(K)\ln(T)/KT}$ and for all $(v, \mathcal{S}) \in \mathcal{B} \times 2^{\mathcal{X}}$ set:

$$w_1(v, \mathcal{S}) := \frac{1}{4\log(T)} \sum_{i \in [\log(T)]} \prod_{x \in \mathcal{X} \setminus \{x_1\}} \left(\sigma(x, \mathcal{S}) \frac{2^i}{T} + (1 - \sigma(x, \mathcal{S})) \left(1 - \frac{2^i}{T}\right) \right) \tag{4}$$

where $\sigma(x, \mathcal{S}) := [\![[\![x \in \mathcal{S}]\!] \neq [\![n(x) \in \mathcal{S}]\!]]\!]$. This choice gives us the regret bound in Theorem 3.2. We note that CBNN will be implemented in such a way that $\mathcal{X}$ and n need not be known a-priori.

4.2 Ternary Search Trees

As we shall see, CBNN works by storing a binary tree $\mathcal{A}(v)$ at each vertex $v \in \mathcal{B}$. In order to perform efficient operations on these trees we will utilise the rebalancing data-structure defined in [19] which here we shall call a *ternary search tree* (TST) due to the fact that it is a generalisation of the classic *binary search tree* and, as we shall show, has searching applications. However, as for binary search trees, the applications of TSTs are more than just searching: we shall also utilise them for online belief propagation.

We now define what is meant by a TST. Suppose we have a full binary tree $\mathcal{J}$. A TST of $\mathcal{J}$ is a full ternary tree $\mathcal{D}$ which satisfies the following. The vertex set of $\mathcal{D}$ is partitioned into two sets $\mathcal{D}^\circ$ and $\mathcal{D}^\bullet$ where each vertex $s \in \mathcal{D}$ is associated with a vertex $\mu(s) \in \mathcal{J}$ and every $s \in \mathcal{D}^\bullet$ is also associated with a vertex $\mu'(s) \in \Downarrow(\mu(s))^\dagger$. In addition, each internal vertex $s \in \mathcal{D}^\dagger$ is associated with a vertex $\xi(s) \in \mathcal{J}$. For all $u \in \mathcal{J}$ there exists an unique leaf $\Upsilon_{\mathcal{D}}(u) \in \mathcal{D}^\star$ in which $\mu(\Upsilon_{\mathcal{D}}(u)) = u$.

For completeness we now describe the rules that a TST $\mathcal{D}$ of $\mathcal{J}$ must satisfy. We have that $r(\mathcal{D}) \in \mathcal{D}^\circ$ and $\mu(r(\mathcal{D})) := r(\mathcal{J})$. Each vertex $s \in \mathcal{D}$ represents a subtree $\hat{\mathcal{J}}(s)$ of $\mathcal{J}$. If $s \in \mathcal{D}^\circ$ then $\hat{\mathcal{J}}(s) := \Downarrow(\mu(s))$ and otherwise $\hat{\mathcal{J}}(s)$ is the set of descendants of $\mu(s)$ which are not proper descendants of $\mu'(s)$. Given that $s \in \mathcal{D}^\dagger$ this subtree is *split* at the vertex $\xi(s)$ where if $s \in \mathcal{D}^\bullet$ we have that $\xi(s)$ lies on the path from $\mu(s)$ to $\mu'(s)$. The children of s are then defined so that $\hat{\mathcal{J}}(\triangleleft(s)) = \hat{\mathcal{J}}(s) \cap \Downarrow(\triangleleft(\xi(s)))$ and $\hat{\mathcal{J}}(\triangleright(s)) = \hat{\mathcal{J}}(s) \cap \Downarrow(\triangleright(\xi(s)))$ and $\hat{\mathcal{J}}(\triangledown(s)) = \hat{\mathcal{J}}(s) \setminus (\hat{\mathcal{J}}(\triangleleft(s)) \cup \hat{\mathcal{J}}(\triangleright(s)))$. i.e. $\xi(s)$ partitions the subtree $\hat{\mathcal{J}}(s)$ into the subtrees $\hat{\mathcal{J}}(\triangleleft(s))$, $\hat{\mathcal{J}}(\triangleright(s))$, and $\hat{\mathcal{J}}(\triangledown(s))$. This process continues recursively until $|\hat{\mathcal{J}}(s)| = 1$ in which case s is a leaf of $\mathcal{D}$.

For all binary trees $\mathcal{J}$ in our algorithm we shall maintain a TST $\mathcal{H}(\mathcal{J})$ of $\mathcal{J}$ with height $\mathcal{O}(\ln(|\mathcal{J}|))$. Such trees $\mathcal{J}$ are *dynamic* in that on any trial it is possible that two vertices, u and u', are added to the tree $\mathcal{J}$ such that u' is inserted between a non-root vertex of $\mathcal{J}$ and its parent, and u is designated as a child of u'. We define the subroutine $\textsc{Rebalance}(\mathcal{H}(\mathcal{J}), u)$ as one which rebalances the TST $\mathcal{H}(\mathcal{J})$ after this insertion, so that the height of $\mathcal{H}(\mathcal{J})$ always remains in $\mathcal{O}(\ln(|\mathcal{J}|))$. The work of [19] describes how this subroutine can be implemented in a time of $\mathcal{O}(\ln(|\mathcal{J}|))$ and we refer the reader to this work for details (noting that they use different notation).

4.3 Contractions

At any trial t the contexts in $\{x_s \mid s \in [t]\}$ naturally form a tree by designating $n(x_s)$ as the parent of x_s. However, to utilise the TST data-structure we must only have binary trees. Hence, we will work with a (dynamic) full binary tree $\mathcal{Z}$ which, on trial t, is a *binarisation* of the above tree. The relationship between these two trees is given by a map $\gamma : \mathcal{Z}_t \to \{x_s \mid s \in [t]\}$ where $\mathcal{Z}_t$ is the tree $\mathcal{Z}$ on trial t. For all $x \in \{x_s \mid s \in [t]\}$ we will always have an unique leaf $\tilde{\gamma}(x) \in \mathcal{Z}_t^\star$ in which $\gamma(\tilde{\gamma}(x)) = x$. We also maintain a balanced TST $\mathcal{H}(\mathcal{Z})$ of $\mathcal{Z}$.

Algorithm 2 gives the subroutine $\textsc{Grow}_t$ which updates $\mathcal{Z}$ at the start of trial t. Note that $\textsc{Grow}_t$ also defines a function $d : \mathcal{Z} \to \mathbb{N}$ such that $d(u)$ is the number of times the function n must be applied to $\gamma(u)$ to reach x_1.

Algorithm 2 $\textsc{Grow}_t$ which works on $\mathcal{Z}$

```
u ← γ̃(n(x_t))
u* ← ↑(u)
u' ← NewVertex
u'' ← NewVertex
γ(u') ← n(x_t)
γ(u'') ← x_t
γ̃(x_t) ← u''
if u = ◁(u*) then
    ◁(u*) ← u'
else
    ▷(u*) ← u'
end if
◁(u') ← u''
▷(u') ← u
d(u') ← d(u)
d(u'') ← d(u) + 1
Rebalance(H(Z), u'')
```

A *contraction* (of $\mathcal{Z}$) is defined as a full binary tree $\mathcal{J}$ in which the following holds. (1) The vertices of $\mathcal{J}$ are a subset of those of $\mathcal{Z}$. (2) $r(\mathcal{J}) = r(\mathcal{Z})$. (3) Given a vertex $u \in \mathcal{J}$ we have $\triangleleft_{\mathcal{J}}(u) \in \Downarrow_{\mathcal{Z}}(\triangleleft_{\mathcal{Z}}(u))$ and $\triangleright_{\mathcal{J}}(u) \in \Downarrow_{\mathcal{Z}}(\triangleright_{\mathcal{Z}}(u))$. (4) Any leaf of $\mathcal{J}$ is a leaf of $\mathcal{Z}$.

CBNN will maintain, on every vertex $v \in \mathcal{B}$, a contraction $\mathcal{A}(v)$ as well as a TST $\mathcal{H}(\mathcal{A}(v))$ of $\mathcal{A}(v)$. Given $\mathcal{J}$ is one of these contractions, we also maintain, for all $i, i' \in \{0,1\}$, all $u \in \mathcal{J}$ and all $j \in [\log(T)]$, a value $\tau_{i,i'}(\mathcal{J}, u, j) \in \mathbb{R}_+$. Technically these quantities, which depend on the above function d, define a sequence of *bayesian networks* on $\mathcal{J}$ which is explained in Appendix D.3. For all $i \in \{0,1\}$ and all $u \in \mathcal{J}$ we also maintain a value $\kappa_i(\mathcal{J}, u)$ initialised equal to 1.

On each of our contractions $\mathcal{J}$ we will define, on trial t, a subroutine $\textsc{Insert}_t(\mathcal{J})$ that simply modifies $\mathcal{J}$ so that $\tilde{\gamma}(x_t)$ is added to its leaves. This subroutine is only called on certain trials t. Specifically, it is called on the contraction $\mathcal{A}(v)$ only when v is involved in $\textsc{CanProp}$ on trial t. Although the effect of this subroutine is simple to describe, its polylogarithmic-time implementation is quite complex. A function that is used many times during this subroutine is $\nu : \mathcal{Z} \times \mathcal{Z} \to \{\blacktriangleleft, \blacktriangleright, \blacktriangle\}$ in which $\nu(u, u')$ is equal to $\blacktriangleleft, \blacktriangleright, \blacktriangle$ if u' is contained in $\Downarrow_{\mathcal{Z}}(\triangleleft_{\mathcal{Z}}(u))$, in $\Downarrow_{\mathcal{Z}}(\triangleright_{\mathcal{Z}}(u))$ or in neither, respectively. Algorithm 3 shows how to compute this function. Now that we have a subroutine for computing ν we can turn to the pseudocode for the subroutine $\textsc{Insert}_t(\mathcal{J})$ in Algorithm 4. In the appendix we give a full description of how and why this subroutine works.

4.4 Online Belief Propagation

In this subsection we utilise the work of [6] in order to be able to efficiently compute the function θ_t that appears in $\textsc{CanProp}$.

Algorithm 3 Computing $\nu(u, u')$ for $u, u' \in \mathcal{Z}$

$\mathcal{E} \leftarrow \mathcal{H}(\mathcal{Z})$
if $u = u'$ **then**
  return ▲
end if
$\tilde{s} \leftarrow \Upsilon_{\mathcal{E}}(u)$
$\tilde{s}' \leftarrow \Upsilon_{\mathcal{E}}(u')$
$s^* \leftarrow \Gamma_{\mathcal{E}}(\tilde{s}, \tilde{s}')$
for $s \in \{\triangleleft(s^*), \nabla(s^*), \triangleright(s^*)\}$ **do**
  if $\tilde{s} \in \Downarrow(s)$ **then**
    $\hat{s} \leftarrow s$
  end if
  if $\tilde{s}' \in \Downarrow(s)$ **then**
    $\hat{s}' \leftarrow s$
  end if
end for
if $\hat{s} \neq \nabla(s^*)$ **then**
  return ▲
end if
if $\xi(s^*) = u \wedge \hat{s}' = \triangleleft(s^*)$ **then**
  return ◄
else if $\xi(s^*) = u \wedge \hat{s}' = \triangleright(s^*)$ **then**
  return ►
end if
$s \leftarrow \hat{s}$
while TRUE **do**
  if $s \in \mathcal{E}^\circ$ **then**
    return ▲
  else if $u = \xi(s) \wedge \triangleleft(s) \in \mathcal{E}^\bullet$ **then**
    return ◄
  else if $u = \xi(s) \wedge \triangleright(s) \in \mathcal{E}^\bullet$ **then**
    return ►
  end if
  for $s' \in \{\triangleleft(s), \nabla(s), \triangleright(s)\}$ **do**
    if $\tilde{s} \in \Downarrow(s')$ **then**
      $s \leftarrow s'$
    end if
  end for
end while

Algorithm 4 The operation $\text{INSERT}_t(\mathcal{J})$ on a contraction $\mathcal{J}$ of $\mathcal{Z}$ at trial t

$\mathcal{E} \leftarrow \mathcal{H}(\mathcal{Z})$
$\mathcal{D} \leftarrow \mathcal{H}(\mathcal{J})$
$s \leftarrow r(\mathcal{D})$
$u_t \leftarrow \tilde{\gamma}(x_t)$
while $s \in \mathcal{D}^\dagger$ **do**
  if $\nu(\xi(s), u_t) =$ ◄ **then**
    $s \leftarrow \triangleleft(s)$
  else if $\nu(\xi(s), u_t) =$ ► **then**
    $s \leftarrow \triangleright(s)$
  else if $\nu(\xi(s), u_t) =$ ▲ **then**
    $s \leftarrow \nabla(s)$
  end if
end while
$\hat{u} \leftarrow \mu(s)$
$s \leftarrow r(\mathcal{E})$
while $s \in \mathcal{E}^\dagger$ **do**
  if $\nu(\xi(s), u_t) = \nu(\xi(s), \hat{u})$ **then**
    if $\nu(\xi(s), u_t) =$ ◄ **then**
      $s \leftarrow \triangleleft(s)$
    else if $\nu(\xi(s), u_t) =$ ► **then**
      $s \leftarrow \triangleright(s)$
    else if $\nu(\xi(s), u_t) =$ ▲ **then**
      $s \leftarrow \nabla(s)$
    end if
  else
    $s \leftarrow \nabla(s)$
  end if
end while
$u^* \leftarrow \mu(s)$
$u' \leftarrow \uparrow_{\mathcal{J}}(\hat{u})$
if $\hat{u} = \triangleleft_{\mathcal{J}}(u')$ **then**
  $\triangleleft_{\mathcal{J}}(u') \leftarrow u^*$
else
  $\triangleright_{\mathcal{J}}(u') \leftarrow u^*$
end if
if $\nu(u^*, \hat{u}) =$ ◄ **then**
  $\triangleleft_{\mathcal{J}}(u^*) \leftarrow \hat{u}$
  $\triangleright_{\mathcal{J}}(u^*) \leftarrow u_t$
else
  $\triangleright_{\mathcal{J}}(u^*) \leftarrow \hat{u}$
  $\triangleleft_{\mathcal{J}}(u^*) \leftarrow u_t$
end if
for $i \in \{0, 1\}$ **do**
  $\kappa_i(\mathcal{J}, u^*) \leftarrow 1$
  $\kappa_i(\mathcal{J}, u_t) \leftarrow 1$
end for
for $(j, i, i') \in [\log(T)] \times \{0, 1\} \times \{0, 1\}$ **do**
  for $u \in \{u^*, \hat{u}, u_t\}$ **do**
    $\delta(u) \leftarrow d(u) - d(\uparrow_{\mathcal{J}}(u))$
    if $i = i'$ **then**
      $\tau_{i,i'}(\mathcal{J}, u, j) \leftarrow 1 - \phi_{\delta(u)}(2^j/T)$
    else
      $\tau_{i,i'}(\mathcal{J}, u, j) \leftarrow \phi_{\delta(u)}(2^j/T)$
    end if
  end for
end for
$\text{REBALANCE}(\mathcal{H}(\mathcal{J}), u_t)$

Given a vertex u in one of our contractions $\mathcal{J}$ we define $\mathcal{F}(\mathcal{J}, u) := \{f \in \{0,1\}^{\mathcal{J}} \mid f(u) = 1\}$ and then for all $j \in [\log(T)]$ define:

$$\Lambda(\mathcal{J}, u, j) := \prod_{f \in \mathcal{F}(\mathcal{J}, u)} \prod_{u' \in \mathcal{J} \setminus \{r(\mathcal{J})\}} \tau_{f(\uparrow_{\mathcal{J}}(u')), f(u')}(\mathcal{J}, u', j) \kappa_{f(u')}(\mathcal{J}, u') \,.$$

As stated in the previous subsection, when a vertex $v \in \mathcal{B}$ becomes involved in CanProp on trial t, CBNN will add $\tilde{\gamma}(x_t)$ to the leaves of $\mathcal{A}(v)$ via the operation $\textsc{Insert}_t(\mathcal{A}(v))$. In the appendix we shall show that for each such v we then have:

$$\theta_t(v) = \frac{1}{4\log(T)} \sum_{j \in [\log(T)]} \Lambda(\mathcal{A}(v), \tilde{\gamma}(x_t), j) \,.$$

We now outline how to compute this efficiently, deferring a full description for Appendix E.3. First note that for all contractions $\mathcal{J}$ and all $(j, u) \in [\log(T)] \times \mathcal{J}$ we have that $\Lambda(\mathcal{J}, u, j)$ is of the exact form to be solved by the classic *Belief propagation* algorithm [20]. The work of [6] shows how to compute this term in logarithmic time by maintaining a data-structure based on a balanced TST of $\mathcal{J}$ - in our case the TST $\mathcal{H}(\mathcal{J})$. Whenever, for some $i \in \{0,1\}$ and $u' \in \mathcal{J}$, the value $\kappa_i(\mathcal{J}, u')$ changes, the data-structure is updated in logarithmic time.

We shall maintain, for each contraction $\mathcal{J}$, a set of $\log(T)$ such data-structures - one for each value of j. We define the subroutine $\textsc{Evidence}(\mathcal{J}, u')$ as that which updates all these data-structures after $\kappa_i(\mathcal{J}, u')$ changes. We also make sure that the data-structures are updated whenever $\textsc{Rebalance}(\mathcal{H}(\mathcal{J}), \cdot)$ is called. We then define the subroutine $\textsc{Marginal}(\mathcal{J}, u)$ as that which computes $\Lambda(\mathcal{J}, u, j)$ for each $j \in [\log(T)]$, and then sums the results and divides by $4\log(T)$. Hence, the output of $\textsc{Marginal}(\mathcal{A}(v), \tilde{\gamma}(x_t))$ is equal to $\theta_t(v)$.

4.5 CBNN

Now that we have defined all our subroutines we give, in Algorithm 5, the algorithm CBNN which is an efficient implementation of CanProp with initial weighting given in Equation (4).

Algorithm 5 CBNN at trial t

$\textsc{Grow}_t$
$u_t \leftarrow \tilde{\gamma}(x_t)$
$v_{t,0} \leftarrow r(\mathcal{B})$
for $j = 0, 1, \cdots, (\log(K) - 1)$ **do**
  for $v \in \{\triangleleft(v_{t,j}), \triangleright(v_{t,j})\}$ **do**
    $\textsc{Insert}_t(\mathcal{A}(v))$
    $\theta_t(v) \leftarrow \textsc{Marginal}(\mathcal{A}(v), u_t)$
  end for
  $z_{t,j} \leftarrow \theta_t(\triangleleft(v_{t,j})) + \theta_t(\triangleright(v_{t,j}))$
  for $v \in \{\triangleleft(v_{t,j}), \triangleright(v_{t,j})\}$ **do**
    $\pi_t(v) \leftarrow \theta_t(v)/z_{t,j}$
  end for
  $\zeta_{t,j} \sim [0,1]$
  if $\zeta_{t,j} \leq \pi_t(\triangleleft(v_{t,j}))$ **then**
    $v_{t,j+1} \leftarrow \triangleleft(v_{t,j})$
  else
    $v_{t,j+1} \leftarrow \triangleright(v_{t,j})$
  end if
end for
$a_t \leftarrow v_{t,\log(K)}$
$\tilde{\pi}_t \leftarrow \prod_{j \in [\log(K)]} \pi_t(v_{t,j})$
$\psi_{t,\log(K)} \leftarrow \exp(-\eta \ell_{t,a_t}/\tilde{\pi}_t)$
for $j = \log(K), (\log(K) - 1), \cdots, 1$ **do**
  $\psi_{t,(j-1)} \leftarrow 1 - (1 - \psi_{t,j})\pi_t(v_{t,j})$
  if $v_{t,j} = \triangleleft(v_{t,j-1})$ **then**
    $\tilde{v}_{t,j} \leftarrow \triangleright(v_{t,j-1})$
  else
    $\tilde{v}_{t,j} \leftarrow \triangleleft(v_{t,j-1})$
  end if
  $\kappa_1(\mathcal{A}(v_{t,j}), u_t) \leftarrow \psi_{t,j}/\psi_{t,j-1}$
  $\kappa_1(\mathcal{A}(\tilde{v}_{t,j}), u_t) \leftarrow 1/\psi_{t,j-1}$
  $\textsc{Evidence}(\mathcal{A}(v_{t,j}), u_t)$
  $\textsc{Evidence}(\mathcal{A}(\tilde{v}_{t,j}), u_t)$
end for

5 Conclusion

In this paper we introduced the use of the nearest neighbour methodology for the fully adversarial contextual bandit problem when the contexts are selected from a metric space. We developed an extremely efficient algorithm CBNN. We gave a regret bound for CBNN and, as an example, further analysed it in the case in which the contexts (but not necessarily the losses) are drawn i.i.d. from a distribution on a multi-dimensional hypercube: where CBNN requires no knowledge of parameters.

References

[1] R. Agrawal. Sample mean based index policies by o(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27:1054 – 1078, 1995.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32:48–77, 2002.

[4] W. Chu, L. Li, L. Reyzin, and R. E. Schapire. Contextual bandits with linear payoff functions. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[5] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 13:21–27, 1967.

[6] A. L. Delcher, A. J. Grove, S. Kasif, and J. Pearl. Logarithmic-time updates and queries in probabilistic networks. *J. Artif. Intell. Res.*, 4:37–59, 1995.

[7] Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth. Using and combining predictors that specialize. In *Symposium on the Theory of Computing*, 1997.

[8] A. Garivier and O. Cappé. The kl-ucb algorithm for bounded stochastic bandits and beyond. In *Annual Conference Computational Learning Theory*, 2011.

[9] D. P. Helmbold and R. E. Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27:51–68, 1995.

[10] M. Herbster, S. Pasteris, and F. Vitale. Online sum-product computation over trees. In *NIPS*, 2012.

[11] M. Herbster, S. Pasteris, F. Vitale, and M. Pontil. A gang of adversarial bandits. In *Neural Information Processing Systems*, 2021.

[12] M. Herbster and J. Robinson. Online prediction of switching graph labelings with cluster specialists. In *Neural Information Processing Systems*, 2018.

[13] M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1995.

[14] W. M. Koolen, D. Adamskiy, and M. K. Warmuth. Putting bayes to sleep. In *NIPS*, 2012.

[15] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[16] T. L. Lai and H. E. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.

[17] T. Lattimore and C. Szepesvári. Bandit algorithms. 2020.

[18] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *The Web Conference*, 2010.

[19] K. Matsuzaki and A. Morihata. Mathematical engineering technical reports balanced ternary-tree representation of binary trees and balancing algorithms. 2008.

[20] J. Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. *Probabilistic and Causal Inference*, 1982.

[21] H. W. J. Reeve, J. C. Mellor, and G. Brown. The k-nearest neighbour ucb algorithm for multi-armed bandits with covariates. *ArXiv*, abs/1803.00316, 2018.

[22] H. E. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952.

A Guide to the Appendices

To give the reader some intuition behind CBNN we present, in Appendix B, our initial idea: an algorithm which obtains the regret bound of CBNN but is exponentially slower. We then give a detailed description of CBNN in appendices C to E. Specifically, in Appendix C we describe our novel algorithmic framework CANPROP. In Appendix D we describe contractions and bayesian networks on them, showing how CANPROP can be implemented with them. Finally, in Appendix E we describe TSTs and how they are used to perform our required operations efficiently. In Appendix F we prove, in order, all of the theorems stated in this paper.

B The Initial Idea

Here we describe our initial idea - an algorithm, based on EXP4 [3] and *Belief propagation* [20], which attains the regret bound of CBNN but is exponentially slower - taking a per-trial time of $\tilde{\Theta}(KT)$. Since this section is only to give intuition, and the results are surpassed by CBNN, we do not prove the statements made in this section.

To begin with we assume a-priori knowledge of the set $\mathcal{X} := \{x_t \mid t \in [T]\}$ and function n but the final algorithm will not need this knowledge. Without loss of generality assume that T is an integer power of two.

The algorithm is based on EXP4 [3] which we now describe. On every trial t we maintain a weighting $\hat{w}_t : [K]^{\mathcal{X}} \to [0, 1]$. We are free to choose any $\hat{w}_1$ satisfying:

$$\sum_{y \in [K]^{\mathcal{X}}} \hat{w}_1(y) = 1 \,.$$

On each trial t the following happens:

1. x_t is revealed
2. For all $a \in [K]$ set $p_{t,a} \leftarrow \sum_{y \in [K]^{\mathcal{X}}} [\![y(x_t) = a]\!] \hat{w}_t(y)$
3. Set $a_t \leftarrow a$ with probability proportional to $p_{t,a}$
4. Receive ℓ_{t,a_t}
5. For all $a \in [K]$ set $\hat{\ell}_{t,a} \leftarrow [\![a = a_t]\!] \ell_{t,a_t} \|\boldsymbol{p}_t\|_1 / p_{t,a_t}$
6. For all $y \in [K]^{\mathcal{X}}$ set $\hat{w}_{t+1}(y) \leftarrow \hat{w}_t(y) \exp(-\eta \hat{\ell}_{t,y(x_t)})$

It is a classic result [3] that, for any policy $y : \mathcal{X} \to [K]$, the expected y-regret of EXP4 is bounded by:

$$\mathbb{E}[R(y)] \leq \frac{\eta K T}{2} - \frac{\ln(\hat{w}_1(y))}{\eta} \,.$$

If $i \in [\log(T)]$ is such that $2^i \leq \Phi(y) \leq 2^{i+1}$ then:

$$\ln\left(\left(\frac{2^i}{T(K-1)} \right)^{\Phi(y)} \left(1 - \frac{2^i}{T} \right)^{(T-1-\Phi(y))} \right) \in \mathcal{O}\left(\ln\left(\frac{KT}{|\Phi(y)|} \right) \Phi(y) \right)$$

so setting:

$$\eta := \rho \sqrt{\frac{1}{KT}}$$

and:

$$\hat{w}_1(y) := \frac{1}{K \log(T)} \sum_{i \in [\log(T)]} \left(\frac{2^i}{T(K-1)} \right)^{\Phi(y)} \left(1 - \frac{2^i}{T} \right)^{(T-1-\Phi(y))}$$

gives us our desired regret bound.

However, we have two issues - the algorithm takes exponential time and the set $\mathcal{X}$ and function n need to be known a-priori. We will hence discuss how to bring the time complexity down to

$\tilde{\Theta}(KT)$ and with no a-priori knowledge. To do this first define, for all $i \in [\log(T)]$, the function $\hat{\tau}_i : [K] \times [K] \to [0,1]$ by:

$$\hat{\tau}_i(a, a') := [\![a \neq a']\!] \frac{2^i}{T(K-1)} + [\![a = a']\!] \left(1 - \frac{2^i}{T}\right)$$

Note then that for all $y \in [K]^{\mathcal{X}}$ we have:

$$\hat{w}_1(y) \propto \sum_{i \in [\log(T)]} \left(\prod_{s \in [T] \setminus \{1\}} \hat{\tau}_i(y(x_s), y(n(x_s))) \right)$$

so that for all trials t:

$$\hat{w}_t(y) \propto \sum_{i \in [\log(T)]} \left(\prod_{s \in [T] \setminus \{1\}} \hat{\tau}_i(y(x_s), y(n(x_s))) \right) \prod_{s \in [T]} \exp(-[\![s < t]\!] \eta \hat{\ell}_{s, y(x_s)})$$

and hence for all $a \in [K]$ we have:

$$p_{t,a} \propto \sum_{i \in [\log(T)]} \sum_{y \in [K]^{\mathcal{X}}} [\![y(x_t) = a]\!] \left(\prod_{s \in [T] \setminus \{1\}} \hat{\tau}_i(y(x_s), y(n(x_s))) \right) \prod_{s \in [T]} \exp(-[\![s < t]\!] \eta \hat{\ell}_{s, y(x_s)})$$

For all $s \in [t]$ and $a \in [K]$ define:

$$\phi'_t(x_s, a) := [\![s < t]\!] \exp(-\eta \hat{\ell}_{s,a}) + [\![s = t]\!]$$

A crucial insight is that:

$$\sum_{y \in [K]^{\mathcal{X}}} [\![y(x_t) = a]\!] \left(\prod_{s \in [T] \setminus \{1\}} \hat{\tau}_i(y(x_s), y(n(x_s))) \right) \prod_{s \in [T]} \exp(-[\![s < t]\!] \eta \hat{\ell}_{s, y(x_s)})$$

is equal to:

$$\sum_{y \in [K]^{\mathcal{X}_t}} [\![y(x_t) = a]\!] \phi'_t(x_1, y(x_1)) \prod_{s \in [t] \setminus \{1\}} \hat{\tau}_s(y(x_s), y(n(x_s))) \phi'_t(x_s, y(x_s)) \tag{5}$$

which is why the algorithm needs only know $\{x_s \mid s \in [t]\}$ and $\{n(x_s) \mid s \in [t] \setminus \{1\}\}$. On trial t we construct a tree with vertex set $\{x_s \mid s \in [t]\}$ which is rooted at x_1 and is such that for all $s \in [t] \setminus \{1\}$ we have that $n(x_s)$ is the parent of x_s. We note that computing the quantity in Equation (5) for all $a \in [K]$ can be done in a time of $\Theta(Kt)$ by Belief propagation [20] on this tree. Hence we have that $\boldsymbol{p}_t$ can be computed in a time of $\Theta(Kt \log(T))$ without a-priori knowledge of $\mathcal{X}$ and n.

C Cancellation Propagation

We now turn to the description and analysis of our algorithm CBNN, starting with our novel algorithmic framework CanProp.

C.1 The General CanProp Algorithm

Let $\mathcal{X} := \{x_t \mid t \in [T]\}$. Note that we do not know $\mathcal{X}$ a-priori but for now let's assume we do. We now introduce a general algorithmic framework CanProp for handling contextual bandit problems with a per-trial time logarithmic in K. Without loss of generality assume that K is an integer power of two. Let $\mathcal{B}$ be a full, balanced, oriented binary tree whose leaves are the set of actions $[K]$. Let $\mathcal{B}' := \mathcal{B} \setminus \{r(\mathcal{B})\}$. CanProp takes a parameter $\eta \in \mathbb{R}_+$ called the *learning rate*. On each trial t CanProp maintains a function:

$$w_t : \mathcal{B}' \times 2^{\mathcal{X}} \to [0,1]$$

The function w_1 is free to be defined how one likes, as long as it satisfies the constraint that for all internal vertices $v \in \mathcal{B}^\dagger$ we have:

$$\sum_{\mathcal{S} \in 2^{\mathcal{X}}} (w_1(\triangleleft(v), \mathcal{S}) + w_1(\triangleright(v), \mathcal{S})) = 1$$

We now describe how CANPROP acts on trial t. For all $v \in \mathcal{B}'$ we define:

$$\theta_t(v) := \sum_{\mathcal{S} \in 2^{\mathcal{X}}} [\![x_t \in \mathcal{S}]\!] w_t(v, \mathcal{S})$$

and for all $v \in \mathcal{B}^{\dagger}$ we define:

$$\pi_t(\triangleleft(v)) := \frac{\theta_t(\triangleleft(v))}{\theta_t(\triangleleft(v)) + \theta_t(\triangleright(v))} \quad ; \quad \pi_t(\triangleright(v)) := \frac{\theta_t(\triangleright(v))}{\theta_t(\triangleleft(v)) + \theta_t(\triangleright(v))}$$

As we shall see CANPROP needs only compute these values for $\mathcal{O}(\ln(K))$ vertices v. CANPROP samples a root-to-leaf path $\{v_{t,j} \mid j \in [\log(K)] \cup \{0\}\}$ as follows. $v_{t,0}$ is defined equal to $r(\mathcal{B})$. For all $j \in [\log(K) - 1] \cup \{0\}$, once $v_{t,j}$ has been sampled we sample $v_{t,(j+1)}$ from the probability distribution defined by:

$$\mathbb{P}[v_{t,(j+1)} = v] := [\![\uparrow(v) = v_{t,j}]\!] \pi_t(v) \quad \forall v \in \mathcal{B}'$$

noting that $v_{t,(j+1)}$ is a child of $v_{t,j}$. We define:

$$\mathcal{P}_t := \{v_{t,j} \mid j \in [\log(K)] \cup \{0\}\}$$

CANPROP then selects:

$$a_t := v_{t,\log(K)}$$

and then receives the loss ℓ_{t,a_t}. The function w_t is then updated to w_{t+1} as follows. Firstly we define,

$$w_{t+1}(v, \mathcal{S}) := w_t(v, \mathcal{S}) \qquad \forall (v, \mathcal{S}) \in \{v' \in \mathcal{B}' \mid \uparrow(v') \notin \mathcal{P}_t\} \times 2^{\mathcal{X}}$$

We then define:

$$\psi_{t,\log(K)} := \exp\left(\frac{-\eta \ell_{t,a_t}}{\prod_{j \in [\log(K)]} \pi_t(v_{t,j})} \right)$$

Once we have defined $\psi_{t,j}$ for some $j \in [\log(K)]$ we then define:

$$\psi_{t,(j-1)} := 1 - (1 - \psi_{t,j}) \pi_t(v_{t,j})$$

$$\beta_t(v) := \frac{[\![v \in \mathcal{P}_t]\!] \psi_{t,j} + [\![v \notin \mathcal{P}_t]\!]}{\psi_{t,(j-1)}} \quad \forall v \in \{\triangleleft(v_{t,(j-1)}), \triangleright(v_{t,(j-1)})\}$$

$$w_{t+1}(v, \mathcal{S}) := ([\![x_t \in \mathcal{S}]\!] \beta_t(v) + [\![x_t \notin \mathcal{S}]\!]) w_t(v, \mathcal{S}) \quad \forall (v, \mathcal{S}) \in \{\triangleleft(v_{t,(j-1)}), \triangleright(v_{t,(j-1)})\} \times 2^{\mathcal{X}}$$

The regret bound of CANPROP is given by the following theorem.

Theorem C.1. *Suppose we have a function $y : \mathcal{X} \to [K]$. For all $v \in \mathcal{B}$ define:*

$$\mathcal{Q}(v) := \{x \in \mathcal{X} \mid y(x) \in \Downarrow(v)\}$$

Then the expected y-regret of CANPROP *is bounded by:*

$$\mathbb{E}[R(y)] \leq \frac{\eta K T}{2} - \frac{1}{\eta} \sum_{v \in \mathcal{B}'} [\![\mathcal{Q}(v) \neq \emptyset]\!] \ln(w_1(v, \mathcal{Q}(v)))$$

C.2 Our Parameter Tuning

We now describe and analyse the initial weighting w_1 that we will use. Without loss of generality assume T is an integer power of two. Define $\mathcal{X}' := \mathcal{X} \setminus \{x_1\}$. For all $(x, \mathcal{S}) \in \mathcal{X}' \times 2^{\mathcal{X}}$ define:

$$\sigma(x, \mathcal{S}) := [\![[\![x \in \mathcal{S}]\!] \neq [\![n(x) \in \mathcal{S}]\!]]\!]$$

For all $(v, \mathcal{S}) \in \mathcal{B}' \times 2^{\mathcal{X}}$ we define:

$$w_1(v, \mathcal{S}) := \frac{1}{4 \log(T)} \sum_{i \in [\log(T)]} \prod_{x \in \mathcal{X}'} \left(\sigma(x, \mathcal{S}) \frac{2^i}{T} + (1 - \sigma(x, \mathcal{S})) \left(1 - \frac{2^i}{T} \right) \right)$$

Given our parameter ρ we choose our learning rate as:

$$\eta := \rho \sqrt{\frac{\ln(K) \ln(T)}{KT}}$$

Given this initial weighting and learning rate, Theorem C.1 implies the following regret bound.

Theorem C.2. *Given w_1 and η are defined as above, then for any policy $y : \mathcal{C} \to [K]$ the expected y-regret of* CANPROP *is bounded by:*

$$\mathbb{E}[R(y)] \in \mathcal{O}\left(\left(\rho + \frac{\Phi(y)}{\rho} \right) \sqrt{\ln(K) \ln(T) K T} \right)$$

D Binarisation and Implementation with Contractions

D.1 A Sequence of Binary Trees

For any trial t we have a natural tree-structure on the set $\{x_{t'} \mid t' \in [t]\}$ formed by making $n(x_{t'})$ the parent of $x_{t'}$ for all $t' \in [t] \setminus \{1\}$. However, in order to utilise the methodology of [19] we need to work with binary trees. Hence, we now inductively define a sequence of binary trees $\{\mathcal{Z}_t \mid t \in [T] \setminus \{1\}\}$ where the vertices of $\mathcal{Z}_t$ are a subset of those of $\mathcal{Z}_{t+1}$. We also define a function $\gamma : \mathcal{Z}_T \to \mathcal{X}$. This function γ has the property that for any $t \in [T]$ and for any distinct leaves $u, u' \in \mathcal{Z}_t^\star$ we have that $\gamma(u) \neq \gamma(u')$, and that:

$$\{\gamma(u) \mid u \in \mathcal{Z}_t^\star\} = \{x_{t'} \mid t' \in [t]\}$$

We define $\mathcal{Z}_2$ to contain three vertices $\{r(\mathcal{Z}_2), \triangleleft(r(\mathcal{Z}_2)), \triangleright(r(\mathcal{Z}_2))\}$ where:

$$\gamma(r(\mathcal{Z}_2)) := \gamma(\triangleleft(r(\mathcal{Z}_2)) := x_1 \quad \text{and} \quad \gamma(\triangleright(r(\mathcal{Z}_2)) := x_2$$

Now consider a trial $t \in [T]$. We have that $\mathcal{Z}_{t+1}$ is constructed from $\mathcal{Z}_t$ via the following algorithm $\textsc{Grow}_{t+1}$:

1. Let u be the unique leaf in $\mathcal{Z}_t^\star$ in which $\gamma(u) = n(x_{t+1})$ and let $u^* := \uparrow(u)$.
2. Create two new vertices u' and u''.
3. Set $\gamma(u') \leftarrow n(x_{t+1})$ and $\gamma(u'') \leftarrow x_{t+1}$.
4. If $u = \triangleleft(u^*)$ then set $\triangleleft(u^*) \leftarrow u'$. Else set $\triangleright(u^*) \leftarrow u'$.
5. Set $\triangleleft(u') \leftarrow u''$ and $\triangleright(u') \leftarrow u$

We also define a function $d : \mathcal{Z}_T \to \mathbb{N} \cup \{0\}$ as follows. Define $d'(x_1) := 0$ and for all $t \in [T] \setminus \{1\}$ inductively define $d'(x_t) := d'(n(x_t)) + 1$. Finally define $d(u) := d'(\gamma(u))$ for all $u \in \mathcal{Z}_T$. Since for all $t \in [T]$ we have that the vertices of $\mathcal{Z}_t$ are a subset of those of $\mathcal{Z}_T$ we have that d also defines a function over $\mathcal{Z}_t$ for all $t \in [T]$.

For all $t \in [T]$ we define u_t to be the unique leaf of $\mathcal{Z}_t$ for which $\gamma(u_t) = x_t$.

D.2 Contractions

Our efficient implementation of $\textsc{CanProp}$ will have a data-structure at every vertex $v \in \mathcal{B}'$. However, to achieve polylogarithmic time per trial we can only update a polylogarithmic number of these data-structures per trial. This necessitates the use of *contractions* of our trees $\{\mathcal{Z}_t \mid t \in [T] \setminus \{1\}\}$ which are defined as follows. A *contraction* of a full binary tree $\mathcal{Q}$ is another full binary tree $\mathcal{J}$ which satisfies the following:

- The vertices of $\mathcal{J}$ are a subset of those of $\mathcal{Q}$.
- $r(\mathcal{J}) = r(\mathcal{Q})$
- Given an internal vertex $u \in \mathcal{J}^\dagger$ we have $\triangleleft_{\mathcal{J}}(u) \in \Downarrow_{\mathcal{Q}}(\triangleleft_{\mathcal{Q}}(u))$ and $\triangleright_{\mathcal{J}}(u) \in \Downarrow_{\mathcal{Q}}(\triangleright_{\mathcal{Q}}(u))$
- Any leaf of $\mathcal{J}$ is a leaf of $\mathcal{Q}$.

Note that any contraction of $\mathcal{Z}_t$ is also a contraction of $\mathcal{Z}_{t+1}$ and hence, by induction, a contraction of $\mathcal{Z}_{t'}$ for all $t' \geq t$. Given a trial t and a contraction $\mathcal{J}$ of $\mathcal{Z}_{t-1}$ we now define the operation $\textsc{Insert}_t(\mathcal{J})$ which acts on $\mathcal{J}$ by the following algorithm:

1. Let $\hat{u}$ be the unique vertex in $\mathcal{J} \setminus r(\mathcal{J})$ such that u_t lies in the maximal spanning tree of $\mathcal{Z}_t$ with $\uparrow_{\mathcal{J}}(\hat{u})$ and $\hat{u}$ as leaves.
2. Let $u^* := \Gamma_{\mathcal{Z}_t}(u_t, \hat{u})$.
3. Add the vertices u^* and u_t to the tree $\mathcal{J}$.
4. Let $u' := \uparrow_{\mathcal{J}}(\hat{u})$.
5. If $\hat{u} = \triangleleft_{\mathcal{J}}(u')$ then set $\triangleleft_{\mathcal{J}}(u') \leftarrow u^*$. Else set $\triangleright_{\mathcal{J}}(u') \leftarrow u^*$.
6. If $\hat{u} \in \Downarrow_{\mathcal{Z}_t}(\triangleleft_{\mathcal{Z}_t}(u^*))$ then set $\triangleleft_{\mathcal{J}}(u^*) \leftarrow \hat{u}$ and $\triangleright_{\mathcal{J}}(u^*) \leftarrow u_t$. Else set $\triangleright_{\mathcal{J}}(u^*) \leftarrow \hat{u}$ and $\triangleleft_{\mathcal{J}}(u^*) \leftarrow u_t$

Later in this paper we will show how this operation can be done in polylogarithmic time. Note that after the operation we have that $\mathcal{J}$ is a contraction of $\mathcal{Z}_t$ and u_t has been added to it's leaves. From now on when we use the term *contraction* we mean any contraction of $\mathcal{Z}_T$.

D.3 Contraction-Based Bayesian Networks

Here we shall define a bayesian network over any contraction $\mathcal{J}$ and show how it can be utilised to compute certain quantities required by CanProp. This bayesian network takes a parameter $\epsilon \in [0,1]$. First define the quantity $\phi_0(\epsilon) := 0$ and for all $j \in \mathbb{N} \cup \{0\}$ inductively define:

$$\phi_{j+1}(\epsilon) := (1-\epsilon)\phi_j(\epsilon) + \epsilon(1-\phi_j(\epsilon))$$

The algorithm must compute these quantities for various values of ϵ. However, for all $t \in [T]$ we have that $\phi_t(\epsilon)$ doesn't have to be computed until trial t so computing these quantities is constant time per trial (for each value of ϵ). Given a contraction $\mathcal{J}$, a value $\epsilon \in [0,1]$, a vertex $u \in \mathcal{J} \setminus r(\mathcal{J})$ and indices $i, i' \in \{0,1\}$ define:

$$\tilde{\tau}_{i,i'}(\mathcal{J}, u, \epsilon) := [\![i \neq i']\!] \phi_{(d(u)-d(\uparrow_{\mathcal{J}}(u)))}(\epsilon) + [\![i = i']\!](1 - \phi_{(d(u)-d(\uparrow_{\mathcal{J}}(u)))}(\epsilon))$$

which defines the transition matrix from $\uparrow_{\mathcal{J}}(u)$ to u in a bayesian network. We shall now show how belief propagation over such bayesian networks can be used to compute the quantities we need in CanProp. Suppose we have a contraction $\mathcal{J}$, a value $\epsilon \in [0,1]$ and a function $\lambda : \mathcal{J}^\star \to \mathbb{R}_+$. This function λ induces a function $\lambda' : \mathcal{X} \to \mathbb{R}_+$ defined as follows. Given $x \in \mathcal{X}$, if there exists a leaf $u \in \mathcal{J}^\star$ with $\gamma(u) = x$ then $\lambda'(x) = \lambda(u)$. Otherwise $\lambda'(x) = 1$. We then define a weighting $\tilde{w}(\lambda, \epsilon, \cdot) : 2^{\mathcal{X}} \to \mathbb{R}_+$ such that for all $\mathcal{S} \in 2^{\mathcal{X}}$ we have:

$$\tilde{w}(\lambda, \epsilon, \mathcal{S}) := \left(\prod_{x \in \mathcal{S}} \lambda'(x) \right) \left(\prod_{x \in \mathcal{X}'} (\sigma(x, \mathcal{S})\epsilon + (1 - \sigma(x, \mathcal{S}))(1-\epsilon)) \right)$$

For the CanProp algorithm we will need to compute

$$\sum_{\mathcal{S} \in 2^{\mathcal{X}}} [\![\gamma(\hat{u}) \in \mathcal{S}]\!] \tilde{w}(\lambda, \epsilon, \mathcal{S}) \tag{6}$$

for some leaf $\hat{u} \in \mathcal{J}^\star$. We shall now show how we can compute this quantity via belief propagation on the bayesian network. In particular we shall construct a quantity $\tilde{\Lambda}(\mathcal{J}, \lambda, \epsilon, u)$ equal to the quantity in Equation (6). To do this first define the function $\lambda^* : \mathcal{J} \to \mathbb{R}_+$ so that for all $u \in \mathcal{J}^\star$ we have $\lambda^*(u) = \lambda(u)$ and for all $u \in \mathcal{J}^\dagger$ we have $\lambda^*(u) = 1$. For all vertices $u \in \mathcal{J}$ and all indices $i \in \{0,1\}$ define:

$$\tilde{\kappa}_i(\lambda, u) := [\![i = 0]\!] + [\![i = 1]\!] \lambda^*(u)$$

For all $\hat{u} \in \mathcal{J}$ define:

$$\mathcal{F}(\mathcal{J}, \hat{u}) := \{ f \in \{0,1\}^{\mathcal{J}} \mid f(\hat{u}) = 1 \}$$

and then define:

$$\tilde{\Lambda}(\mathcal{J}, \lambda, \epsilon, \hat{u}) := \sum_{f \in \mathcal{F}(\mathcal{J}, \hat{u})} \prod_{u \in \mathcal{J} \setminus r(\mathcal{J})} \tilde{\tau}_{f(\uparrow_{\mathcal{J}}(u)), f(u)}(\mathcal{J}, u, \epsilon) \tilde{\kappa}_{f(u)}(\lambda, u)$$

The equality of this quantity and that given in Equation (6) is given by the following theorem.

Theorem D.1. *Given a contraction $\mathcal{J}$, a function $\lambda : \mathcal{J}^\star \to \mathbb{R}_+$, some $\epsilon \in [0,1]$ and some leaf $\hat{u} \in \mathcal{J}^\star$ we have:*

$$\tilde{\Lambda}(\mathcal{J}, \lambda, \epsilon, \hat{u}) = \sum_{\mathcal{S} \in 2^{\mathcal{X}}} [\![\gamma(\hat{u}) \in \mathcal{S}]\!] \tilde{w}(\lambda, \epsilon, \mathcal{S})$$

Note that $\tilde{\Lambda}(\mathcal{J}, \lambda, \epsilon, \hat{u})$ is of the exact form to be solved via belief propagation over $\mathcal{J}$. However, belief propagation is still too slow (taking $\Theta(|\mathcal{J}|)$ time) - we will remedy this later.

D.4 Cancelation Propogation with Contractions

We now describe how to implement CanProp with contractions. For each $v \in \mathcal{B}'$ we maintain a contraction $\mathcal{A}(v)$ and a function $\zeta(v, \cdot) : \mathcal{A}(v)^\star \to \mathbb{R}_+$. We initialise with $\mathcal{A}(v)$ identical to $\mathcal{Z}_2$ and $\zeta(v, u) = 1$ for both leaves $u \in \mathcal{Z}_2^\star$. Via induction over t we will have that at the start of each trial t we have, for all sets $\mathcal{S} \in 2^{\mathcal{X}}$, that:

$$w_t(v, \mathcal{S}) = \frac{1}{4\log(T)} \sum_{i \in [\log(T)]} \tilde{w}(\zeta(v, \cdot), 2^i/T, \mathcal{S}) \tag{7}$$

On trial t we do as follows. First we update $\mathcal{Z}_{t-1}$ to $\mathcal{Z}_t$ using the algorithm $\textsc{Grow}_t$. We will perform the necessary modifications to our contractions as we sample the path $\mathcal{P}_t$. In particular we first set $v_{t,0} \leftarrow r(\mathcal{B})$ and then for each $j \in [\log(K)-1] \cup \{0\}$ in turn we do as follows. For each $v \in \{\triangleleft(v_{t,j}), \triangleright(v_{t,j})\}$ run $\textsc{Insert}_t(\mathcal{A}(v))$ and set $\zeta(v, u_t) \leftarrow 1$. Since $\zeta(v, u_t) = 1$ Equation (7) still holds and hence, by Theorem D.1, we have:

$$\theta_t(v) = \frac{1}{4\log(T)} \sum_{i \in [\log(T)]} \tilde{\Lambda}(\mathcal{A}(v), \zeta(v, \cdot), 2^i/T, u_t)$$

where $\zeta(v, \cdot)$ is the function that maps each $u \in \mathcal{A}(v)$ to $\zeta(v, u)$. After $\theta_t(v)$ has been computed for both $v \in \{\triangleleft(v_{t,j}), \triangleright(v_{t,j})\}$ we can now sample $v_{t,j+1}$.

Once we have selected the action a_t we then update the functions $\{\zeta(v, \cdot) \mid \uparrow_{\mathcal{B}}(v) \in \mathcal{P}_t\}$ by setting $\zeta(v, u_t) \leftarrow \beta_t(v)$ for all $v \in \mathcal{B}'$ with $\uparrow_{\mathcal{B}}(v) \in \mathcal{P}_t$. It is clear now that Equation (7) holds inductively.

D.5 Notational Relationship to the Main Body

We now point out how the notation in this section relates to that of the main body. In particular we have, for all $v \in \mathcal{B}'$, all $u \in \mathcal{A}(v)$, all $j \in [\log(T)]$ and all $i, i' \in \{0, 1\}$, that:

- $\tau_{i,i'}(\mathcal{A}(v), u, j) = \tilde{\tau}_{i,i'}(\mathcal{A}(v), u, 2^j/T)$
- $\kappa_i(\mathcal{A}(v), u) = \tilde{\kappa}_i(\zeta(v, \cdot), u)$
- $\Lambda(\mathcal{A}(v), u, j) = \tilde{\Lambda}(\mathcal{A}(v), \zeta(v, \cdot), 2^j/T, u)$

E Utilising Ternary Search Trees

There are now only two things left to do in order to achieve polylogarithmic time per trial - to make an efficient online implementation of the $\textsc{Insert}_t(\cdot)$ operation and an efficient online algorithm to perform belief propagation over our contractions. In order to do this we will utilise the methodology of [19] which we now describe. However, we do not give the full details of the rebalancing technique and refer the reader to [19] for these details.

E.1 Ternary Search Trees

In this section we will consider a full binary tree $\mathcal{J}$. A *(full) ternary tree* $\mathcal{D}$ is a rooted tree in which each internal vertex $s \in \mathcal{D}^\dagger$ has three children denoted by $\triangleleft(s)$, $\triangledown(s)$, $\triangleright(s)$ and called the left, centre, and right children respectively. We now define what it means for a ternary tree $\mathcal{D}$ to be a ternary search tree (TST) of $\mathcal{J}$. Firstly, the vertex set of $\mathcal{D}$ is partitioned into two sets $\mathcal{D}^\circ$ and $\mathcal{D}^\bullet$. Every vertex $s \in \mathcal{D}$ is associated with a vertex $\mu(s) \in \mathcal{J}$ and every $s \in \mathcal{D}^\bullet$ is also associated with a vertex $\mu'(s) \in \Downarrow_{\mathcal{J}}(\mu(s))^\dagger$. The root $r(\mathcal{D})$ of $\mathcal{D}$ is contained in $\mathcal{D}^\circ$ and $\mu(r(\mathcal{D})) := r(\mathcal{J})$. Each internal vertex $s \in \mathcal{D}^\dagger$ is associated with a vertex $\xi(s) \in \mathcal{J}$. If $s \in \mathcal{D}^\circ$ then $\xi(s) \in \Downarrow(\mu(s))^\dagger$ and if $s \in \mathcal{D}^\bullet$ then $\xi(s)$ lies on the path (in $\mathcal{J}$) from $\mu(s)$ to $\uparrow(\mu'(s))$. For all $s \in \mathcal{D}^\dagger$ we have:

- $\triangledown(s) \in \mathcal{D}^\bullet$, $\mu(\triangledown(s)) := \mu(s)$ and $\mu'(\triangledown(s)) := \xi(s)$.
- $\triangleleft(s)$ satisfies:
  - If $s \in \mathcal{D}^\circ$ then $\triangleleft(s) \in \mathcal{D}^\circ$ and $\mu(\triangleleft(s)) := \triangleleft(\xi(s))$.
  - If $s \in \mathcal{D}^\bullet$ and $\mu'(s) \in \Downarrow(\triangleright(\xi(s)))$ then $\triangleleft(s) \in \mathcal{D}^\circ$ and $\mu(\triangleleft(s)) := \triangleleft(\xi(s))$
  - Else $\triangleleft(s) \in \mathcal{D}^\bullet$, $\mu(\triangleleft(s)) := \triangleleft(\xi(s))$ and $\mu'(\triangleleft(s)) := \mu'(s)$
- $\triangleright(s)$ satisfies:
  - If $s \in \mathcal{D}^\circ$ then $\triangleright(s) \in \mathcal{D}^\circ$ and $\mu(\triangleright(s)) := \triangleright(\xi(s))$.
  - If $s \in \mathcal{D}^\bullet$ and $\mu'(s) \in \Downarrow(\triangleleft(\xi(s)))$ then $\triangleright(s) \in \mathcal{D}^\circ$ and $\mu(\triangleright(s)) := \triangleright(\xi(s))$
  - Else $\triangleright(s) \in \mathcal{D}^\bullet$, $\mu(\triangleright(s)) := \triangleright(\xi(s))$ and $\mu'(\triangleright(s)) := \mu'(s)$

Finally, for each leaf $s \in \mathcal{D}^\star$ we have:

- If $s \in \mathcal{D}^\circ$ then $\mu(s)$ is a leaf of $\mathcal{J}$.

- If $s \in \mathcal{D}^\bullet$ then there exists $u \in \mathcal{J}^\dagger$ such that $\mu(s) = \mu'(s) = u$.

Intuitively each vertex $s \in \mathcal{D}$ is associated with a subtree $\hat{\mathcal{J}}(s)$ of $\mathcal{J}$. If $s \in \mathcal{D}^\circ$ then $\hat{\mathcal{J}}(s) := \Downarrow(\mu(s))$ and if $s \in \mathcal{D}^\bullet$ then $\hat{\mathcal{J}}(s)$ is the subtree of descendants of $\mu(s)$ which are not proper descendants of $\mu'(s)$. For every $s \in \mathcal{D}$ such that $\hat{\mathcal{J}}(s)$ contains only a single vertex, we have that s is a leaf of $\mathcal{D}$. Otherwise s is an internal vertex of $\mathcal{D}$ and its children are as follows. We say that $\hat{\mathcal{J}}(s)$ is *split* at the vertex $\xi(s) \in \hat{\mathcal{J}}(s)^\dagger$. If $s \in \mathcal{D}^\bullet$ we require that $\xi(s)$ is on the path in $\mathcal{J}$ from $\mu(s)$ to $\mu'(s)$. The action of splitting $\hat{\mathcal{J}}(s)$ at $\xi(s)$ partitions $\hat{\mathcal{J}}(s)$ into the subtrees $\hat{\mathcal{J}}(\triangleleft(s))$, $\hat{\mathcal{J}}(\triangledown(s))$ and $\hat{\mathcal{J}}(\triangleright(s))$ defined as follows:

- $\hat{\mathcal{J}}(\triangleleft(s)) := \Downarrow(\triangleleft(\xi(s))) \cap \hat{\mathcal{J}}(s)$
- $\hat{\mathcal{J}}(\triangleright(s)) := \Downarrow(\triangleright(\xi(s))) \cap \hat{\mathcal{J}}(s)$
- $\hat{\mathcal{J}}(\triangledown(s)) := \hat{\mathcal{J}}(s) \setminus (\hat{\mathcal{J}}(\triangleleft(s)) \cup \hat{\mathcal{J}}(\triangleright(s)))$

Utilising the methodology of [19] we will maintain TSTs of $\mathcal{Z}_t$ (at each trial t) and the trees in $\{\mathcal{A}(v) \mid v \in \mathcal{B}'\}$, each with height $\mathcal{O}(\ln(T))$. Note that these trees are dynamic, in that vertices are inserted into them over time. [19] shows how, after such an insertion, the corresponding TST can be *rebalanced* so that its height is still in $\mathcal{O}(\ln(T))$. This rebalancing is performed via a sequence of $\mathcal{O}(\ln(T))$ *tree rotations*, which generalise the concept of tree rotations in binary search trees.

E.2 Searching

In this section we show how we can use our TSTs to implement the operation $\text{I}\textsc{nsert}_t(\mathcal{J})$ on any trial t and contraction $\mathcal{J}$ of $\mathcal{Z}_{t-1}$. To do this we need to perform the following two search operations:

1. Find the unique vertex $\hat{u} \in \mathcal{J} \setminus r(\mathcal{J})$ such that u_t lies in the maximal spanning tree of $\mathcal{Z}_t$ with $\uparrow_{\mathcal{J}}(\hat{u})$ and $\hat{u}$ as leaves.
2. Find $u^* := \Gamma_{\mathcal{Z}_t}(u_t, \hat{u})$

To perform these tasks in polylogarithmic time we will utilise TSTs $\mathcal{E}$ and $\mathcal{D}$ of $\mathcal{Z}_t$ and $\mathcal{J}$ respectively. Both the searching tasks utilise a function $\nu : \mathcal{Z}_t^2 \to \{\blacktriangle, \blacktriangleleft, \blacktriangleright\}$ defined, for all $u, u' \in \mathcal{Z}_t$ as follows. If $u' \in \Downarrow_{\mathcal{Z}_t}(\triangleleft(u))$ or $u' \in \Downarrow_{\mathcal{Z}_t}(\triangleright(u))$ then $\nu(u, u') := \blacktriangleleft$ or $\nu(u, u') := \blacktriangleright$ respectively. Otherwise $\nu(u, u') := \blacktriangle$. This can be computed as follows. If $u = u'$ then $\nu(u, u') := \blacktriangle$. Otherwise let $\tilde{s}$ and $\tilde{s}'$ be the unique leaves of $\mathcal{E}$ such that $\mu(\tilde{s}) = u$ and $\mu(\tilde{s}') = u'$. Let $s^* := \Gamma_{\mathcal{E}}(\tilde{s}, \tilde{s}')$ and let $\hat{s}$ and $\hat{s}'$ be the children of s^* which are ancestors of $\tilde{s}$ and $\tilde{s}'$ respectively. If $\hat{s} \neq \triangledown(s^*)$ then we have $\nu(u, u') = \blacktriangle$. If $\xi(s^*) = u$ then we have $\nu(u, u') = \blacktriangleleft$ or $\nu(u, u') = \blacktriangleright$ if $\hat{s}' = \triangleleft(s^*)$ or $\hat{s}' = \triangleright(s^*)$ respectively. If $\hat{s} = \triangledown(s^*)$ and $\xi(s^*) \neq u$ then we perform the following process. Start with s equal to $\hat{s}$. At any point in the process we do as follows. If $s \in \mathcal{E}^\circ$ then the process terminates with $\nu(u, u') := \blacktriangle$. If $s \in \mathcal{E}^\bullet$ and $u = \xi(s)$ then the process terminates with $\nu(u, u') = \blacktriangleleft$ or $\nu(u, u') = \blacktriangleright$ if $\triangleleft(s) \in \mathcal{E}^\bullet$ or $\triangleright(s) \in \mathcal{E}^\bullet$ respectively. If $s \in \mathcal{E}^\bullet$ and $u \neq \xi(s)$ then we reset s as equal to the child of s which is an ancestor of $\tilde{s}$ and continue the process.

The vertex $\hat{u}$ can be found as follows. We construct a root-to-leaf path in $\mathcal{D}$ such that, given a vertex s in the path, the next vertex in the path is $\triangleleft(s)$, $\triangleright(s)$ or $\triangledown(s)$ if $\nu(\xi(s), u_t)$ is equal to $\blacktriangleleft$, $\blacktriangleright$ or $\blacktriangle$ respectively. Given that s' is the leaf of $\mathcal{D}$ that is in this path we have $\hat{u} = \mu(s')$.

The vertex u^* can then be found as follows. We construct a root-to-leaf path in $\mathcal{E}$ such that, given a vertex s in the path, the next vertex in the path is found as follows. If $\nu(\xi(s), u_t) = \nu(\xi(s), \hat{u})$ then given $\nu(\xi(s), u_t)$ is equal to $\blacktriangleleft$, $\blacktriangleright$ or $\blacktriangle$, the next vertex is equal to $\triangleleft(s)$, $\triangleright(s)$ or $\triangledown(s)$ respectively. Otherwise, the next vertex is $\triangledown(s)$. Given that s' is the leaf of $\mathcal{E}$ that is in this path we have $u^* = \mu(s')$.

The fact that these algorithms find the correct vertices is given in the following theorem:

Theorem E.1. *The above algorithms are correct.*

E.3 Belief Propagation

Here we utilise the methodology of [6] in order to efficiently compute the function $\tilde{\Lambda}$ that appears in the $\text{C}\textsc{an}\text{P}\textsc{rop}$ implementation. i.e. given a contraction $\mathcal{J}$, a function $\lambda : \mathcal{J}^\star \to \mathbb{R}_+$, some

$\epsilon \in [0,1]$ and some leaf $\hat{u} \in \mathcal{J}^\star$ we need to compute $\tilde{\Lambda}(\mathcal{J}, \lambda, \epsilon, \hat{u})$. For brevity let us define, for all $i, i' \in \{0,1\}$, and all vertices $u \in \mathcal{J} \setminus \{r(\mathcal{J})\}$, the quantities:

$$\hat{\tau}_{i,i'}(u) := \tilde{\tau}_{i,i'}(\mathcal{J}, u, \epsilon) \quad ; \quad \hat{\kappa}_i(u) := \tilde{\kappa}_i(\lambda, u)$$

For simplicity of presentation we will utilise a tree $\mathcal{J}'$ which is defined as identical to $\mathcal{J}$ except with a single vertex added as the parent of $r(\mathcal{J})$. For all $i, i' \in \{0,1\}$ we define $\hat{\kappa}_i(r(\mathcal{J}')) := 1$ and $\hat{\tau}_{i,i'}(r(\mathcal{J})) = [\![i = i']\!]$. For all $u \in \mathcal{J}$ we will define $\uparrow(u) := \uparrow_{\mathcal{J}'}(u)$

We will utilise a TST $\mathcal{D}$ of $\mathcal{J}$ by maintaining *potentials* on the vertices of $\mathcal{D}$ defined as follows. First, for any vertex $s \in \mathcal{D}$ define the subtree $\hat{\mathcal{J}}(s)$ of $\mathcal{J}$ to be equal to $\Downarrow_{\mathcal{J}}(\mu(s))$ if $s \in \mathcal{D}^\circ$ and equal to the maximal subtree with $\mu(s)$ and $\mu'(s)$ as leaves if $s \in \mathcal{D}^\bullet$. For all $s \in \mathcal{D}^\circ$ and $i \in \{0,1\}$ we define:

$$\Psi_i(s) := \sum_{f \in \{0,1\}^{\hat{\mathcal{J}}(s) \cup \{\uparrow(\mu(s))\}}} [\![f(\uparrow(\mu(s))) = i]\!] \prod_{u \in \hat{\mathcal{J}}(s)} \hat{\tau}_{f(\uparrow(u)), f(u)}(u) \hat{\kappa}_{f(u)}(u)$$

and for all $s \in \mathcal{D}^\bullet$ and $i, i' \in \{0,1\}$ we define:

$$\Omega_{i,i'}(s) := \sum_{f \in \{0,1\}^{\hat{\mathcal{J}}(s) \cup \{\uparrow(\mu(s))\}}} [\![f(\uparrow(\mu(s))) = i]\!][\![f(\mu'(s)) = i']\!] \prod_{u \in \hat{\mathcal{J}}(s)} \hat{\tau}_{f(\uparrow(u)), f(u)}(u) \hat{\kappa}_{f(u)}(u)$$

We have the following recurrence relations for these potentials. Suppose we have an internal vertex $s \in \mathcal{D}^\dagger$ and $i, i' \in \{0,1\}$. If $s \in \mathcal{D}^\circ$ we have:

$$\Psi_i(s) = \sum_{i'' \in \{0,1\}} \Omega_{i,i''}(\nabla(s)) \Psi_{i''}(\triangleleft(s)) \Psi_{i''}(\triangleright(s))$$

If, instead, $s \in \mathcal{D}^\bullet$ then, by letting $s' := \triangleleft(s)$, $s'' := \triangleright(s)$ if $\triangleleft(s) \in \mathcal{D}^\bullet$ and $s' := \triangleright(s)$, $s'' := \triangleleft(s)$ otherwise, we have:

$$\Omega_{i,i'}(s) = \sum_{i'' \in \{0,1\}} \Omega_{i,i''}(\nabla(s)) \Omega_{i'',i'}(s') \Psi_{i''}(s'')$$

If, on a trial t, we perform the operation $\textsc{Insert}_t(\mathcal{J})$ or change the value of $\lambda(u_t)$ these recurrence relations can be used to update the potentials (in conjunction with the tree rotations) in logarithmic time.

Now that we have defined our potentials we will show how to use them to compute $\tilde{\Lambda}(\mathcal{J}, \lambda, \epsilon, \hat{u})$ in logarithmic time. To do this we recursively define the following quantities for $i \in \{0,1\}$. Let $\omega_i(r(\mathcal{D})) := 1$. Given an internal vertex $s \in \mathcal{D}^\circ$ we define:

$$\omega_i(\nabla(s)) := \omega_i(s) \quad ; \quad \omega'_i(\nabla(s)) := \Psi_i(\triangleleft(s)) \Psi_i(\triangleright(s))$$

$$\omega_i(\triangleleft(s)) := \Psi_i(\triangleright(s)) \sum_{i' \in \{0,1\}} \omega_{i'}(s) \Omega_{i',i}(s) \quad ; \quad \omega_i(\triangleright(s)) := \Psi_i(\triangleleft(s)) \sum_{i' \in \{0,1\}} \omega_{i'}(s) \Omega_{i',i}(s)$$

Given an internal vertex $s \in \mathcal{D}^\bullet$ define $s' := \triangleleft(s)$, $s'' := \triangleright(s)$ if $\triangleleft(s) \in \mathcal{D}^\bullet$ and $s' := \triangleright(s)$, $s'' := \triangleleft(s)$ otherwise. Then:

$$\omega_i(\nabla(s)) := \omega_i(s) \quad ; \quad \omega'_i(\nabla(s)) := \Psi_i(s'') \sum_{i' \in \{0,1\}} \Omega_{i,i'}(s') \omega'_{i'}(s)$$

$$\omega_i(s') := \sum_{i' \in \{0,1\}} \omega_{i'}(s) \Omega_{i',i}(s) \Psi_i(s'') \quad ; \quad \omega'_i(s') := \omega'_i(s)$$

$$\omega_i(s'') := \sum_{i', i'' \in \{0,1\}} \omega_{i'}(s) \Omega_{i',i}(\nabla(s)) \omega'_{i''}(s) \Omega_{i,i''}(s')$$

For $s \in \mathcal{D}^\circ$, $\omega'_i(s)$ is not required and hence is arbitrary. We inductively compute the values $\{\omega_i(s), \omega'_i(s) \mid i \in \{0,1\}\}$ for all s in the path from $r(\mathcal{D})$ to the unique leaf $\hat{s} \in \mathcal{D}^\star$ in which $\mu(\hat{s}) = \hat{u}$. We then have $\tilde{\Lambda}(\mathcal{J}, \lambda, \epsilon, \hat{u}) = \omega_1(\hat{s})$.

Since this is known methodology we do not include a proof in this paper and direct the reader to [6].

F Proofs

F.1 Theorem 3.1

For brevity we write α instead of $\alpha(y,\mu)$. Choose some $\delta > 0$. Let $\mathcal{E} := \mathcal{C} \setminus \mathcal{M}(y,\delta)$. Let $\mathcal{X}$ be a set of m contexts drawn i.i.d. at random from μ. Now consider some x drawn from μ and let $\hat{x}$ be a c-nearest neighbour of x in $\mathcal{X}$.

Suppose that $x \in \mathcal{E}$. Let $\mathcal{A}$ be the ball of radius δ/c centred at x. We have that:

$$\mu(\mathcal{A}) \geq \epsilon\lambda\left(\frac{\delta}{c}\right)^d$$

where λ is a constant dependent on d. This means that for any x' drawn from μ we have that:

$$\mathbb{P}[x' \notin \mathcal{A}] \leq 1 - \lambda\epsilon\left(\frac{\delta}{c}\right)^d \leq \exp\left(-\lambda\epsilon\left(\frac{\delta}{c}\right)^d\right)$$

Suppose that:

$$m \geq \frac{-\ln(\alpha\delta)}{\lambda\epsilon}\left(\frac{c}{\delta}\right)^d$$

Note that if there exists $x'' \in \mathcal{X}$ with $x'' \in \mathcal{A}$ then $\Delta(x,\hat{x}) \leq \delta$ so that $y(x) = y(\hat{x})$. The above equations then give us:

$$\mathbb{P}[y(x) \neq y(\hat{x}) \,|\, x \in \mathcal{E}] \leq \exp\left(-m\lambda\epsilon\left(\frac{\delta}{c}\right)^d\right) \leq \alpha\delta$$

We then have that:

$$\mathbb{P}[y(x) \neq y(\hat{x})] \leq \alpha\delta + \mu(\mathcal{M}(\delta)) \in \mathcal{O}(\alpha\delta)$$

Since:

$$\delta \in \tilde{\mathcal{O}}(c(\epsilon m)^{-1/d})$$

we now have:

$$\mathbb{P}[y(x) \neq y(\hat{x})] \in \tilde{\mathcal{O}}\left(c\alpha(\epsilon m)^{-1/d}\right)$$

F.2 Theorem 3.2

This theorem is proved in appendices C to E and the theorems therein.

F.3 Theorem 3.3

Choose a set $\mathcal{S} \subseteq \mathcal{C}$ in which for all $t \in [T]$ there exists $x \in \mathcal{S}$ with $\Delta(x, x_t) < \gamma(x,y)/3c$. For all trials t let $\mathcal{S}_t$ be the set of all contexts $x \in \mathcal{S}$ in which there exists $s \in [t]$ with $\Delta(x, x_s) < \gamma(x,y)/3c$.

Now consider a trial t in which $y(x_t) \neq y(n(x_t))$ and choose $x \in \mathcal{S}$ with $\Delta(x, x_t) < \gamma(x,y)/3c$.

Assume, for contradiction, that $x \in \mathcal{S}_{t-1}$. Then there exists $s \in [t-1]$ with $\Delta(x, x_s) < \gamma(x,y)/3c$ so that by the triangle inequality we have:

$$\Delta(x_t, x_s) \leq \Delta(x, x_s) + \Delta(x, x_t) < 2\gamma(x,y)/3c$$

which implies that $\Delta(x_t, n(x_t)) < 2\gamma(x,y)/3$. By the triangle inequality we then have that:

$$\Delta(x, n(x_t)) \leq \Delta(x_t, n(x_t)) + \Delta(x, x_t) < 2\gamma(x,y)/3 + \gamma(x,y)/3c \leq 3\gamma(x,y)/3 = \gamma(x,y)$$

Since $\Delta(x, x_t) < \gamma(x,y)$ we have $y(x) = y(x_t)$ and hence that $y(x) \neq y(n(x_t))$. But this contradicts the fact that $\Delta(x, n(x_t)) < \gamma(x,y)$.

We have hence shown that $x \notin \mathcal{S}_{t-1}$. Since $x \in \mathcal{S}_t$ we then have that $|\mathcal{S}_t| \geq |\mathcal{S}_{t-1}|$. This implies that:

$$\Phi(y) = \sum_{t\in[T]} [\![y(x_t) \neq y(n(x_t))]\!] \leq |\mathcal{S}_T| \leq |\mathcal{S}|$$

as required.

F.4 Theorem 3.4

By linearity of expectation we have:

$$\mathbb{E}[\Phi(y)] \leq 1 + \sum_{t\in[T]} g_t(\mu, y, c)$$

and from Theorem 3.1 we have:

$$g_t(\mu, y, c) \in \mathcal{O}\left(c\alpha(y, \mu)(\epsilon t)^{-1/d}\right)$$

so that:

$$\mathbb{E}[\Phi(y)] \in \mathcal{O}\left(c\alpha(y, \mu)\epsilon^{-1/d}T^{(d-1)/d}\ln(T)\right)$$

By setting:

$$\rho := T^{(d-1)/(2d)}c^{1/2}$$

we then have:

$$\rho + \frac{\mathbb{E}[\Phi(y)]}{\rho} \in \mathcal{O}\left(c^{1/2}(1 + \alpha(y, \mu)\epsilon^{-1/d})T^{(d-1)/(2d)}\ln(T)\right)$$

so that by Theorem 3.2 we have:

$$\mathbb{E}[R(y)] \in \tilde{\mathcal{O}}\left(c^{1/2}(1 + \alpha(y, \mu)\epsilon^{-1/d})K^{1/2}T^{(2d-1)/(2d)}\right)$$

F.5 Theorem C.1

For every trial $t \in [T]$ define:

$$\Delta_t := -\sum_{v\in\mathcal{B}'} [\![\mathcal{Q}(v) \neq \emptyset]\!] \ln(w_t(v, \mathcal{Q}(v)))$$

Choose some arbitrary trial $t \in [T]$. From here until we say otherwise all probabilities and expectations (i.e. whenever we use $\mathbb{P}[\cdot]$ or $\mathbb{E}[\cdot]$) are implicitly conditional on the state of the algorithm at the start of trial t. Note first that we have:

$$\Delta_t - \Delta_{t+1} = \sum_{v\in\mathcal{B}'} [\![\mathcal{Q}(v) \neq \emptyset]\!] \ln\left(\frac{w_{t+1}(v, \mathcal{Q}(v))}{w_t(v, \mathcal{Q}(v))}\right) \tag{8}$$

For all $j \in [\log(K)] \cup \{0\}$ let $\gamma_{t,j}$ be the ancestor (in $\mathcal{B}$) of $y(x_t)$ at depth j. Note that for all $v \in \mathcal{X} \setminus \{\gamma_{t,j} \mid j \in [\log(K)] \cup \{0\}\}$ we have $y(x_t) \notin \Downarrow(v)$ so that $x_t \notin \mathcal{Q}(v)$ and hence, directly from the CANPROP algorithm, we have $w_{t+1}(v, \mathcal{Q}(v)) = w_t(v, \mathcal{Q}(v))$. By Equation (8) and the fact that $\mathcal{Q}(v) \neq \emptyset$ for all ancestors v of $y(x_t)$ this implies that:

$$\Delta_t - \Delta_{t+1} = \sum_{j\in[\log(K)]} \ln\left(\frac{w_{t+1}(\gamma_{t,j}, \mathcal{Q}(\gamma_{t,j}))}{w_t(\gamma_{t,j}, \mathcal{Q}(\gamma_{t,j}))}\right) \tag{9}$$

For all $j \in [\log(K)]$ define:

$$\lambda_{t,j} := \ln\left(\frac{w_{t+1}(\gamma_{t,j}, \mathcal{Q}(\gamma_{t,j}))}{w_t(\gamma_{t,j}, \mathcal{Q}(\gamma_{t,j}))}\right)$$

and:

$$\epsilon_{t,j} := \mathbb{E}[\ln(\psi_{t,j}) \mid \gamma_{t,j} \in \mathcal{P}_t]$$

Now choose some arbitrary $j \in [\log(K)]$. If $\gamma_{t,(j-1)} \in \mathcal{P}_t$ then $\gamma_{t,(j-1)} = v_{t,(j-1)}$ so $\uparrow(\gamma_{t,j}) = v_{t,(j-1)}$ and hence, since $x_t \in \mathcal{Q}(\gamma_{t,j})$, we have $\lambda_{t,j} = \ln(\beta_t(\gamma_{t,j}))$. By definition of $\beta_t(\gamma_{t,j})$ this means that:

$$\mathbb{E}[\lambda_{t,j} \mid \gamma_{t,j} \in \mathcal{P}_t\,,\, \gamma_{t,(j-1)} \in \mathcal{P}_t] = \epsilon_{t,j} - \mathbb{E}[\ln(\psi_{t,(j-1)}) \mid \gamma_{t,j} \in \mathcal{P}_t\,,\, \gamma_{t,(j-1)} \in \mathcal{P}_t]$$

and that:

$$\mathbb{E}[\lambda_{t,j} \mid \gamma_{t,j} \notin \mathcal{P}_t\,,\, \gamma_{t,(j-1)} \in \mathcal{P}_t] = -\mathbb{E}[\ln(\psi_{t,(j-1)}) \mid \gamma_{t,j} \notin \mathcal{P}_t\,,\, \gamma_{t,(j-1)} \in \mathcal{P}_t]$$

Multiplying these two equations by $\mathbb{P}[\gamma_{t,j} \in \mathcal{P}_t \mid \gamma_{t,(j-1)} \in \mathcal{P}_t]$ and $\mathbb{P}[\gamma_{t,j} \notin \mathcal{P}_t \mid \gamma_{t,(j-1)} \in \mathcal{P}_t]$ respectively, and summing them together, then gives us:

$$\mathbb{E}[\lambda_{t,j} \mid \gamma_{t,(j-1)} \in \mathcal{P}_t] = \mathbb{P}[\gamma_{t,j} \in \mathcal{P}_t \mid \gamma_{t,(j-1)} \in \mathcal{P}_t]\epsilon_{t,j} - \mathbb{E}[\ln(\psi_{t,(j-1)} \mid \gamma_{t,(j-1)} \in \mathcal{P}_t]$$

Since $\mathbb{P}[\gamma_{t,j} \in \mathcal{P}_t \mid \gamma_{t,(j-1)} \in \mathcal{P}_t] = \pi_t(\gamma_{t,j})$ we then have:

$$\mathbb{E}[\lambda_{t,j} \mid \gamma_{t,(j-1)} \in \mathcal{P}_t] = \pi_t(\gamma_{t,j})\epsilon_{t,j} - \epsilon_{t,(j-1)} \tag{10}$$

If, on the other hand, $\gamma_{t,(j-1)} \notin \mathcal{P}_t$ then $\uparrow(\gamma_{t,j}) \notin \mathcal{P}_t$ so $\lambda_{t,j} = 0$. This means that:

$$\mathbb{E}[\lambda_{t,j}] = \mathbb{P}[\gamma_{t,(j-1)} \in \mathcal{P}_t]\mathbb{E}[\lambda_{t,j} \mid \gamma_{t,(j-1)} \in \mathcal{P}_t] \tag{11}$$

Since the probability that $\gamma_{t,(j-1)} \in \mathcal{P}_t$ is equal to $\prod_{j' \in [j-1]} \pi_t(\gamma_{t,j'})$ we then have, by combining equations (10) and (11), that:

$$\mathbb{E}[\lambda_{t,j}] = \epsilon_{t,j} \prod_{j' \in [j]} \pi_t(\gamma_{t,j'}) - \epsilon_{t,(j-1)} \prod_{j' \in [j-1]} \pi_t(\gamma_{t,j'})$$

By substituting into Equation (9) (after taking expectations) we then have that:

$$\begin{aligned}\mathbb{E}[\Delta_t - \Delta_{t+1}] &= -\epsilon_{t,0} + \epsilon_{t,\log(K)} \prod_{j \in [\log(K)]} \pi_t(\gamma_{t,j}) \\ &= -\mathbb{E}[\ln(\psi_{t,0})] + \mathbb{E}[\ln(\psi_{t,\log(K)}) \mid a_t = \gamma_{t,\log(K)}] \prod_{j \in [\log(K)]} \pi_t(\gamma_{t,j}) \end{aligned} \tag{12}$$

Note that if $a_t = \gamma_{t,\log(K)}$ then $\gamma_{t,j} = v_{t,j}$ for all $j \in [\log(K)]$. By definition of $\psi_{t,\log(K)}$ and the fact that $\gamma_{t,\log(K)} = y(x_t)$, Equation (12) then gives us:

$$\mathbb{E}[\Delta_t - \Delta_{t+1}] = -\mathbb{E}[\ln(\psi_{t,0})] - \eta\ell_{t,y(x_t)} \tag{13}$$

For all $(v, a) \in \mathcal{B} \times [K]$ define:

$$p_{t,a}(v) = \mathbb{P}[a_t = a \mid v \in \mathcal{P}_t]$$

noting that this is non-zero only when $a \in \Downarrow(v)$. Suppose we have some $v \in \mathcal{B} \setminus \{r(\mathcal{B})\}$ and some $a \in \Downarrow(v) \cap [K]$. Then, since $\mathbb{P}[a_t = a \mid v \notin \mathcal{P}_t] = 0$, we have:

$$p_{t,a}(\uparrow(v)) = \mathbb{P}[a_t = a \mid \uparrow(v) \in \mathcal{P}_t] = \mathbb{P}[a_t = a \mid v \in \mathcal{P}_t]\mathbb{P}[v \in \mathcal{P}_t \mid \uparrow(v) \in \mathcal{P}_t] = \pi_t(v)p_{t,a}(v)$$

Since $p_{t,a}(v) = 0$ whenever $a \notin \Downarrow(v)$, this implies that for all $(v, a) \in \mathcal{B}^\dagger \times [K]$ we have:

$$p_{t,a}(v) = \pi_t(\triangleleft(v))p_{t,a}(\triangleleft(v)) + \pi_t(\triangleright(v))p_{t,a}(\triangleright(v)) \tag{14}$$

For all $a \in [K]$ define:

$$\hat{\ell}_{t,a} = \frac{[\![a_t = a]\!]\ell_{t,a}}{\mathbb{P}[a_t = a]}$$

We now take the inductive hypothesis that for all $j \in [\log(K)] \cup \{0\}$ we have:

$$\psi_{t,j} = \sum_{a \in [K]} p_{t,a}(v_{t,j}) \exp(-\eta\hat{\ell}_{t,a})$$

and prove this via reverse induction (i.e. from $j = \log(K)$ to $j = 0$). Note that given $a' := a_t$ we have $\mathbb{P}[a_t = a'] = \prod_{j \in [\log(K)]} \pi_t(v_{t,j})$ and hence:

$$\psi_{t,\log(K)} = \exp(-\eta\hat{\ell}_{t,a_t})$$

so the inductive hypothesis holds for $j = \log(K)$. Now suppose that we have some $j' \in [\log(K)]$ and that the inductive hypothesis holds for $j = j'$. We shall now show that it holds also for $j = j' - 1$. Let v' be the child of $v_{t,(j'-1)}$ that is not equal to $v_{t,j'}$. Note that $a_t \notin \Downarrow(v')$ and hence $\exp(-\eta\hat{\ell}_{t,a}) = 1$ for all $a \in \Downarrow(v')$ (i.e. whenever $p_{t,a}(v') \neq 0$) which implies:

$$\sum_{a \in [K]} p_{t,a}(v') \exp(-\eta\hat{\ell}_{t,a}) = 1 \tag{15}$$

For all $a \in [K]$, Equation (14) gives us:

$$p_{t,a}(v_{t,(j'-1)}) \exp(-\eta\hat{\ell}_{t,a}) = \pi_t(v')p_{t,a}(v') \exp(-\eta\hat{\ell}_{t,a}) + \pi_t(v_{t,j'})p_{t,a}(v_{t,j'}) \exp(-\eta\hat{\ell}_{t,a})$$

Substituting Equation (15) and the inductive hypothesis into this equation (when summed over all $a \in [K]$) then gives us:

$$\sum_{a\in[K]} p_{t,a}(v_{t,(j'-1)}) \exp(-\eta\hat{\ell}_{t,a}) = \pi_t(v') + \pi_t(v_{t,j'})\psi_{t,j'}$$

Since $\pi_t(v') + \pi_t(v_{t,j'}) = 1$ we have, direct from the algorithm, that $\pi_t(v') + \pi_t(v_{t,j'})\psi_{t,j'} = \psi_{t,(j'-1)}$ so the inductive hypothesis holds for $j = j' - 1$. We have hence shown that the inductive hypothesis holds for all $j \in [\log(K)] \cup \{0\}$ and in particular for $j = 0$. Since $p_{t,a}(v_{t,0}) = \mathbb{P}[a_t = a]$ we then have:

$$\psi_{t,0} = \sum_{a\in[K]} \mathbb{P}[a_t = a] \exp(-\eta\hat{\ell}_{t,a}) \tag{16}$$

Since $\exp(-z) \leq 1 - z + z^2/2$ for all $z \in \mathbb{R}_+$ we have, from Equation (16), that:

$$\psi_{t,0} \leq \sum_{a\in[K]} \mathbb{P}[a_t = a] \left(1 - \eta\hat{\ell}_{t,a} + \frac{\eta^2\hat{\ell}_{t,a}}{2}\right) = 1 - \eta \sum_{a\in[K]} \mathbb{P}[a_t = a]\hat{\ell}_{t,a} + \frac{\eta^2}{2} \sum_{a\in[K]} \mathbb{P}[a_t = a]\hat{\ell}^2_{t,a}$$

so since $\ln(1 + z) \leq z$ for all $z \in \mathbb{R}$ we have:

$$\ln(\psi_{t,0}) \leq -\eta \sum_{a\in[K]} \mathbb{P}[a_t = a]\hat{\ell}_{t,a} + \frac{\eta^2}{2} \sum_{a\in[K]} \mathbb{P}[a_t = a]\hat{\ell}^2_{t,a} \tag{17}$$

Noting that $\mathbb{P}[a_t = a]\hat{\ell}_{t,a} = [\![a_t = a]\!]\ell_{t,a}$ for all $a \in [K]$, we have:

$$\mathbb{E}\left[\sum_{a\in[K]} \mathbb{P}[a_t = a]\hat{\ell}_{t,a}\right] = \mathbb{E}[\ell_{t,a_t}]$$

and:

$$\mathbb{E}\left[\sum_{a\in[K]} \mathbb{P}[a_t = a]\hat{\ell}^2_{t,a}\right] = \mathbb{E}\left[\sum_{a\in[K]} \frac{[\![a_t = a]\!]\hat{\ell}^2_{t,a}}{\mathbb{P}[a_t = a]}\right] = \sum_{a\in[K]} \hat{\ell}^2_{t,a} \leq K$$

Substituting these equations into Equation (17) (after taking expectations) gives us:

$$\mathbb{E}[\ln(\psi_{t,0})] \leq -\eta\mathbb{E}[\ell_{t,a_t}] + \eta^2 K/2$$

which, upon substitution into Equation (13) gives us:

$$\mathbb{E}[\Delta_t - \Delta_{t+1}] \geq \eta(\mathbb{E}[\ell_{t,a_t}] - \ell_{t,y(x_t)}) - \eta^2 K/2 \tag{18}$$

Note that this equation implies that the same equation also holds when the expectation is not implicitly conditional on the state of the algorithm at the start of trial t. Hence, we now drop the assumption that the expectation is conditional on the state of the algorithm at the start of trial t. Summing Equation (18) over all trials $t \in [T]$ and then rearranging gives us:

$$\mathbb{E}[R(y)] \leq \frac{1}{\eta}(\mathbb{E}[\Delta_1] - \mathbb{E}[\Delta_{T+1}]) + \frac{\eta KT}{2} \tag{19}$$

Now consider a trial t. For all $v \in \mathcal{B}^\dagger$ let:

$$V_t(v) := \sum_{\mathcal{S}\in 2^{\mathcal{X}}} [\![x_t \in \mathcal{S}]\!] w_{t+1}(\triangleleft(v), \mathcal{S}) + \sum_{\mathcal{S}\in 2^{\mathcal{X}}} [\![x_t \in \mathcal{S}]\!] w_{t+1}(\triangleright(v), \mathcal{S})$$

Now take any $j \in [\log(K) - 1] \cup \{0\}$ and let $v := v_{t,j}$. Note that:

$$V_t(v) = \beta_t(\triangleleft(v))\theta_t(\triangleleft(v)) + \beta_t(\triangleright(v))\theta_t(\triangleright(v))$$

so that by definition of $\pi_t(\triangleleft(v))$ and $\pi_t(\triangleright(v))$ we have:

$$V_t(v) = (\theta_t(\triangleleft(v)) + \theta_t(\triangleright(v)))(\pi_t(\triangleleft(v))\beta_t(\triangleleft(v)) + \pi_t(\triangleright(v))\beta_t(\triangleright(v)))$$

Without loss of generality assume that $\triangleleft(v) \in \mathcal{P}_t$. Then the above equation implies that:

$$V_t(v) = (\theta_t(\triangleleft(v)) + \theta_t(\triangleright(v)))\frac{\pi_t(\triangleleft(v))\psi_{t,j+1} + \pi_t(\triangleright(v))}{\psi_{t,j}}$$

so by definition of $\psi_{t,j}$ we have:

$$V_t(v) = (\theta_t(\triangleleft(v)) + \theta_t(\triangleright(v))) = \sum_{\mathcal{S}\in 2^{\mathcal{X}}} [\![x_t \in \mathcal{S}]\!] w_t(\triangleleft(v), \mathcal{S}) + \sum_{\mathcal{S}\in 2^{\mathcal{X}}} [\![x_t \in \mathcal{S}]\!] w_t(\triangleright(v), \mathcal{S})$$

Note that this equation trivially holds for all $v \in \mathcal{B}^\dagger \setminus \mathcal{P}_t$ and hence holds for all $v \in \mathcal{B}^\dagger$. Since for all such v and all $\mathcal{S}$ with $x_t \notin \mathcal{S}$ we have $w_{t+1}(\triangleleft(v), \mathcal{S}) = w_t(\triangleleft(v), \mathcal{S})$ and $w_{t+1}(\triangleright(v), \mathcal{S}) = w_t(\triangleright(v), \mathcal{S})$ we then have:

$$\sum_{\mathcal{S}\in 2^{\mathcal{X}}} w_{t+1}(\triangleleft(v), \mathcal{S}) + \sum_{\mathcal{S}\in 2^{\mathcal{X}}} w_{t+1}(\triangleright(v), \mathcal{S}) = \sum_{\mathcal{S}\in 2^{\mathcal{X}}} w_t(\triangleleft(v), \mathcal{S}) + \sum_{\mathcal{S}\in 2^{\mathcal{X}}} w_t(\triangleright(v), \mathcal{S})$$

so, by induction on t we have, for all $t \in [T+1]$, that:

$$\sum_{\mathcal{S}\in 2^{\mathcal{X}}} w_t(\triangleleft(v), \mathcal{S}) + \sum_{\mathcal{S}\in 2^{\mathcal{X}}} w_t(\triangleright(v), \mathcal{S}) = 1$$

Hence, for all $v \in \mathcal{B} \setminus r(\mathcal{B})$ and $\mathcal{S} \in 2^{\mathcal{X}}$, we have $w_t(v, \mathcal{S}) \in [0,1]$. We have now shown that $\Delta_{T+1} \geq 0$ so that Equation 19 gives us:

$$\mathbb{E}[R(y)] \leq \frac{1}{\eta}\mathbb{E}[\Delta_1] + \frac{\eta K T}{2}$$

which, by definition of Δ_1, gives us the desired result.

F.6 Theorem C.2

The fact that the weighting w_t is valid is given by the following lemma:

Lemma F.1. *For all $v \in \mathcal{B}^\dagger$ we have:*

$$\sum_{\mathcal{S}\in 2^{\mathcal{X}}} (w_1(\triangleleft(v), \mathcal{S}) + w_t(\triangleright(v), \mathcal{S})) = 1$$

Proof. We will show that for all $v \in \mathcal{B}'$ we have:

$$\sum_{\mathcal{S}\in 2^{\mathcal{X}}} w_1(v, \mathcal{S}) = \frac{1}{2}$$

which directly implies the result. So take some arbitrary $v \in \mathcal{B}'$. Define, for all $t \in [T]$, the sets:

$$\mathcal{X}'_t := \{x_s \mid s \in [t]\} \setminus \{x_1\} \quad \text{and} \quad \mathcal{F}_t := \{0,1\}^{\mathcal{X}'_t \cup \{x_1\}}$$

and for all $x \in \mathcal{X}'_t$, $f \in \mathcal{F}_t$ and $i \in [\log(T)]$, define the quantity:

$$\beta_i(x, f) := [\![f(x) \neq f(n(x))]\!] 2^i/T + [\![f(x) = f(n(x))]\!](1 - 2^i/T)$$

which is defined since $n(x) \in \mathcal{X}'_t \cup \{x_1\}$. Now fix some $i \in [\log(T)]$. For all $t \in [T-1]$ we have:

$$\sum_{f\in\mathcal{F}_{t+1}} \prod_{x\in\mathcal{X}'_{t+1}} \beta_i(x,f) = \sum_{f\in\mathcal{F}_t} \left(\prod_{x\in\mathcal{X}'_t} \beta_i(x,f)\right) \sum_{f(x_{t+1})\in\{0,1\}} \beta_i(x_{t+1}, f)$$

Given any $f \in \mathcal{F}_t$ we have:

$$\sum_{f(x_{t+1})\in\{0,1\}} \beta_i(x_{t+1}, f) = \left(1 - \frac{2^i}{T}\right) + \frac{2^i}{T} = 1$$

and hence:

$$\sum_{f\in\mathcal{F}_{t+1}} \prod_{x\in\mathcal{X}'_{t+1}} \beta_i(x,f) = \sum_{f\in\mathcal{F}_t} \prod_{x\in\mathcal{X}'_t} \beta_i(x,f)$$

Since $\mathcal{X}'_T = \mathcal{X}'$ this implies, by induction, that:

$$\sum_{f\in\mathcal{F}_T}\prod_{x\in\mathcal{X}'}\beta_i(x,f)=\sum_{f\in\mathcal{F}_1}\prod_{x\in\mathcal{X}'_1}\beta_i(x,f)=\sum_{f\in\mathcal{F}_1}\prod_{x\in\emptyset}\beta_i(x,f)=\sum_{f\in\mathcal{F}_1}1=|\mathcal{F}_1|=2 \tag{20}$$

Note that we have a bijection $\mathcal{G}:\mathcal{F}_T\rightarrow 2^{\mathcal{X}}$ defined by:

$$\mathcal{G}(f):=\{x\in\mathcal{X}\mid f(x)=1\}\quad\forall f\in\mathcal{F}_T$$

and that for all $(i,f,x)\in[\log(T)]\times\mathcal{F}_T\times\mathcal{X}'$ we have:

$$\beta_i(x,f)=\sigma(x,\mathcal{G}(f))2^i/T+(1-\sigma(x,\mathcal{G}(f)))(1-2^i/T)$$

Hence, Equation (20) shows us that for all $i\in[\log(T)]$ we have:

$$\sum_{\mathcal{S}\in 2^{\mathcal{X}}}\prod_{x\in\mathcal{X}'}\left(\sigma(x,\mathcal{S})\frac{2^i}{T}+(1-\sigma(x,\mathcal{S}))\left(1-\frac{2^i}{T}\right)\right)=2$$

This implies that:

$$\sum_{\mathcal{S}\in 2^{\mathcal{X}}}w_1(v,\mathcal{S})=\frac{1}{2}$$

which implies the result. □

Now that we have shown that the weighting w_1 is valid we can utilise Theorem C.1 to prove our regret bound. For any set $\mathcal{S}\in 2^{\mathcal{X}}$ define:

$$\phi(\mathcal{S}):=\sum_{x\in\mathcal{X}'}\sigma(x,\mathcal{S})$$

For any $i\in[\log(T)]$ define the function $f_i:[T-1]\rightarrow\mathbb{R}$ by

$$f_i(c):=\left(1-\frac{2^i}{T}\right)^{T-1-c}\left(\frac{2^i}{T}\right)^c$$

for all $c\in[T-1]$. Choose any set $\mathcal{S}\in 2^{\mathcal{X}}$ and define:

$$j:=\min\{\lceil\log(\phi(\mathcal{S})+1)\rceil,\log(T)-1\}$$

If $\phi(\mathcal{S})\geq T/2$ then $j=\log(T)-1$ so $2^j/T=1/2$ and hence:

$$-\ln(f_j(\phi(\mathcal{S})))=(T-1)\ln(2)\leq 2\phi(\mathcal{S})\ln(2)\in\mathcal{O}(\phi(\mathcal{S})) \tag{21}$$

Now consider the case in which $\phi(\mathcal{S})<T/2$. Let $h:=2^j/T$. In this case $2^j/T\leq 1/2$ and hence f_j is monotonic decreasing so since $2^j\geq\phi(\mathcal{S})$ we have:

$$\ln(f_j(\phi(\mathcal{S}))\geq\ln(f_j(2^j))=\ln(f_j(Th))\geq T((1-h)\ln(1-h)+h\ln(h))\geq -Th\ln(e/h)$$

so since $\phi(\mathcal{S})+1\geq 2^j/2=Th/2$ and $h\geq 1/T$ we have:

$$-\ln(f_j(\phi(\mathcal{S}))\leq 2(\phi(\mathcal{S})+1)\ln(eT)\in\mathcal{O}((\phi(\mathcal{S})+1)\ln(T)) \tag{22}$$

Equations (21) and (22) show us that for all possible values of $\phi(\mathcal{S})$ we have:

$$-\ln(f_j(\phi(\mathcal{S}))\in\mathcal{O}(\ln(T)(\phi(\mathcal{S})+1))$$

Noting that for all $v\in\mathcal{B}'$ we have $w_1(v,\mathcal{S})\geq(1/4\log(T))f_j(\phi(\mathcal{S}))$ we have now shown that:

$$-\ln(w_1(v,\mathcal{S}))\in\mathcal{O}(\ln(T)(\phi(\mathcal{S})+1)) \tag{23}$$

for all $v\in\mathcal{B}'$. As in the statement of Theorem C.1 define, for all $v\in\mathcal{B}$, the set:

$$\mathcal{Q}(v):=\{x\in\mathcal{X}\mid y(x)\in\Downarrow(v)\}$$

First note that the graph (with vertex set $\mathcal{X}$) formed by linking x to $n(x)$ for every $x\in\mathcal{X}'$ is a tree so that $\Phi(y)\geq|\{y(x)\mid x\in\mathcal{X}\}|-1$. So since for all $v\in\mathcal{B}'$ we have $\mathcal{Q}(v)\neq\emptyset$ if and only if v has a descendent in $\{y(x) \mid x \in \mathcal{X}\}$ and each element of $\{y(x) \mid x \in \mathcal{X}\}$ has $\log(K)$ ancestors in $\mathcal{B}'$ we have:

$$\sum_{v \in \mathcal{B}'} [\![\mathcal{Q}(v) \neq \emptyset]\!] \leq \log(K)|\{y(x) \mid x \in \mathcal{X}\}| \leq \log(K)(\Phi(y)+1) \tag{24}$$

Now suppose we have some $x \in \mathcal{X}'$. If $y(x) = y(n(x))$ then for all $v \in \mathcal{B}'$ we have $x, n(x) \in \mathcal{Q}(v)$ or $x, n(x) \notin \mathcal{Q}(v)$ and hence $\sigma(x, \mathcal{Q}(v)) = 0$. On the other hand, if $y(x) \neq y(n(x))$ then for any $v \in \mathcal{B}'$ with $\sigma(x, \mathcal{Q}(v)) = 1$ we must have that either $x \in \mathcal{Q}(v)$ or $n(x) \in \mathcal{Q}(v)$ so v is an ancestor of either x or $n(x)$ and hence there can be at most $2\log(K)$ such v. So in any case we have:

$$\sum_{v \in \mathcal{B}'} \sigma(x, \mathcal{Q}(v)) \leq [\![y(x) \neq y(n(x))]\!] 2\log(K)$$

Hence we have:

$$\sum_{v \in \mathcal{B}'} \phi(\mathcal{Q}(v)) = \sum_{x \in \mathcal{X}'} \sum_{v \in \mathcal{B}'} \sigma(x, \mathcal{Q}(v)) \leq 2\log(K)\Phi(y) \tag{25}$$

Equation (23) gives us:

$$-\sum_{v \in \mathcal{B}'} [\![\mathcal{Q}(v) \neq \emptyset]\!] \ln(w_1(v, \mathcal{Q}(v))) \in \mathcal{O}\left(\ln(T) \sum_{v \in \mathcal{B}'} \phi(\mathcal{Q}(v)) + \ln(T) \sum_{v \in \mathcal{B}'} [\![\mathcal{Q}(v) \neq \emptyset]\!]\right)$$

Substituting in equations (24) and (25) then gives us:

$$-\sum_{v \in \mathcal{B}'} [\![\mathcal{Q}(v) \neq \emptyset]\!] \ln(w_1(v, \mathcal{Q}(v))) \in \mathcal{O}(\ln(K)\ln(T)\Phi(y))$$

so by Theorem C.1 we have:

$$\mathbb{E}[R(y)] \in \mathcal{O}\left(\frac{\eta K T}{2} + \frac{\ln(K)\ln(T)\Phi(y)}{\eta}\right)$$

Since $\eta = \rho\sqrt{\ln(K)\ln(T)/KT}$ we obtain the result.

F.7 Theorem D.1

Define $\lambda' : \mathcal{X} \to \mathbb{R}_+$ as follows. Given $x \in \mathcal{X}$, if there exists a leaf $u \in \mathcal{J}^\star$ with $\gamma(u) = x$ then $\lambda'(x) = \lambda(u)$. Otherwise $\lambda'(x) = 1$. Given $t \in [T]$ define $\hat{\lambda}_t : \mathcal{Z}_t \to \mathbb{R}_+$ such that for all $u \in \mathcal{Z}_t$ we have that $\hat{\lambda}_t(u) := \lambda'(\gamma(u))$ if u is a leaf of $\mathcal{Z}_t$ and $\hat{\lambda}_t(u) := 1$ otherwise. For all $t \in [T]$ and $f : \{x_{t'} \mid t' \in [t]\} \to \{0, 1\}$ define:

$$\mathcal{N}(f) := \{f' \in \{0,1\}^{\mathcal{Z}_t} \mid \forall u \in \mathcal{Z}_t^\star, f'(u) = f(\gamma(u))\}$$

and:

$$\hat{w}(f) := \left(\prod_{t' \in [t] : f(x_{t'})=1} \lambda'(x_t)\right) \prod_{t' \in [t] \setminus \{1\}} ([\![f(x_t) \neq f(n(x_t))]\!]\epsilon + [\![f(x_t) = f(n(x_t))]\!](1-\epsilon))$$

and:

$$\hat{\nu}(f) := \sum_{f' \in \mathcal{N}(f)} \prod_{u \in \mathcal{Z}_t \setminus r(\mathcal{Z}_t)} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_t}(u)), f'(u)}(\mathcal{Z}_t, u, \epsilon) \tilde{\kappa}_{f'(u)}(\hat{\lambda}_t, u)$$

We now have the following lemma:

Lemma F.2. *For all $t \in [T]$ and $f : \{x_{t'} \mid t' \in [t]\} \to \{0, 1\}$ we have:*

$$\hat{w}(f) = \hat{\nu}(f)$$

Proof. We prove by induction on t. Suppose the result holds for $t = s$ (for some $s \geq 2$). We now show that it holds for $t = s+1$ as well. Let f^* be the restriction of f onto the set $\{x_{t'} \mid t' \in [s]\}$. Let u^* and u' be the unique leaves in $\mathcal{Z}_{s+1}^\star$ of which $\gamma(u') = n(x_{s+1})$ and $\gamma(u^*) = x_{s+1}$. By the construction of $\mathcal{Z}_{s+1}$ these vertices are siblings. Let u'' be the parent (in $\mathcal{Z}_{s+1}$) of both u^* and u'. First note that:

$$[\![f(x_{s+1}) = 0]\!] + [\![f(x_{s+1}) = 1]\!]\lambda'(x_{s+1}) = \tilde{\kappa}_{f(x_{s+1})}(\hat{\lambda}_{s+1}, u^*) \tag{26}$$

Since, by the construction of $\mathcal{Z}_{s+1}$, we have $\gamma(\uparrow_{\mathcal{Z}_{s+1}}(u^*)) = \gamma(u'') = n(x_{s+1})$ we also have that $d(\uparrow_{\mathcal{Z}_{s+1}}(u^*)) = d(u^*) - 1$ so that, since $\phi_1(\epsilon) = \epsilon$, we have:

$$[\![f(x_{s+1}) \neq f(n(x_{s+1}))]\!]\epsilon + [\![f(x_{s+1}) = f(n(x_{s+1}))]\!](1-\epsilon) = \tilde{\tau}_{f(n(x_{s+1})),f(x_{s+1})}(\mathcal{Z}_{s+1}, u^*, \epsilon) \tag{27}$$

Equations (26) and (27) give us:

$$\hat{w}(f) = \hat{w}(f^*)\tilde{\tau}_{f(n(x_{s+1})),f(x_{s+1})}(\mathcal{Z}_{s+1}, u^*, \epsilon)\tilde{\kappa}_{f(x_{s+1})}(\hat{\lambda}_{s+1}, u^*) \tag{28}$$

Now suppose we have some $f' \in \mathcal{N}(f)$. We have $\gamma(u'') = \gamma(u')$ and hence $d(\uparrow_{\mathcal{Z}_{s+1}}(u')) = d(u'') = d(u')$ so since $f'(u') = f(n(x_{s+1}))$ and $\phi_0(\epsilon) = 0$ we have:

$$\tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_{s+1}}(u')),f'(u')}(\mathcal{Z}_{s+1}, u', \epsilon) = \tilde{\tau}_{f'(u''),f'(u')}(\mathcal{Z}_{s+1}, u', \epsilon) = [\![f'(u'') = f(n(x_{s+1}))]\!] \tag{29}$$

Since, by the construction of $\mathcal{Z}_{s+1}$, we have $\uparrow_{\mathcal{Z}_{s+1}}(u'') = \uparrow_{\mathcal{Z}_s}(u')$ and (as above) we have $d(u'') = d(u')$, we also have:

$$\tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_{s+1}}(u'')),f'(u'')}(\mathcal{Z}_{s+1}, u'', \epsilon) = \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_s}(u')),f'(u'')}(\mathcal{Z}_s, u', \epsilon) \tag{30}$$

Since $f'(u^*) = f(x_{s+1})$ and $\uparrow_{\mathcal{Z}_{s+1}}(u^*) = u''$ we have:

$$\tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_{s+1}}(u^*)),f'(u^*)}(\mathcal{Z}_{s+1}, u^*, \epsilon) = \tilde{\tau}_{f'(u''),f(x_{s+1})}(\mathcal{Z}_{s+1}, u^*, \epsilon) \tag{31}$$

Now let:

$$\zeta^* := \tilde{\tau}_{f(n(x_{s+1})),f(x_{s+1})}(\mathcal{Z}_{s+1}, u^*, \epsilon) \quad ; \quad \zeta' := \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_s}(u')),f(n(x_{s+1}))}(\mathcal{Z}_s, u', \epsilon)$$

Define:

$$g(f') := \prod_{u \in \mathcal{Z}_s \setminus r(\mathcal{Z}_s)} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_s}(u)),f'(u)}(\mathcal{Z}_s, u, \epsilon)$$

and:

$$g'(f') := \prod_{u \in \mathcal{Z}_{s+1} \setminus r(\mathcal{Z}_{s+1})} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_{s+1}}(u)),f'(u)}(\mathcal{Z}_{s+1}, u, \epsilon)$$

Combining equations (29), (30) and (31) gives us:

$$\prod_{u \in \{u^*,u',u''\}} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_{s+1}}(u)),f'(u)}(\mathcal{Z}_{s+1}, u, \epsilon) = [\![f'(u'') = f(n(x_{s+1}))]\!]\zeta^*\zeta' \tag{32}$$

For all $u \in \mathcal{Z}_{s+1} \setminus \{u^*, u', u''\}$ we have $\uparrow_{\mathcal{Z}_{s+1}}(u) = \uparrow_{\mathcal{Z}_s}(u)$ so that:

$$\tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_{s+1}}(u)),f'(u)}(\mathcal{Z}_{s+1}, u, \epsilon) = \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_s}(u)),f'(u)}(\mathcal{Z}_s, u, \epsilon)$$

and hence, since $f(n(x_{s+1})) = f'(u')$, we have:

$$g'(f') = \frac{g(f')}{\zeta'} \prod_{u \in \{u^*,u',u''\}} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_{s+1}}(u)),f'(u)}(\mathcal{Z}_{s+1}, u, \epsilon)$$

Substituting in Equation (32) gives us:

$$g'(f') = g(f')[\![f'(u'') = f(n(x_{s+1}))]\!]\zeta^* \tag{33}$$

We have $\tilde{\kappa}_{f'(u'')}(\hat{\lambda}_{s+1}, u'') = 1$ and for all $u \in \mathcal{Z}_s$ we have $\tilde{\kappa}_{f'(u)}(\hat{\lambda}_{s+1}, u) = \tilde{\kappa}_{f'(u)}(\hat{\lambda}_s, u)$. Substituting into Equation (33) gives us:

$$g'(f') \prod_{u \in \mathcal{Z}_{s+1}} \tilde{\kappa}_{f'(u)}(\hat{\lambda}_{s+1}, u) = [\![f'(u'') = f(n(x_{s+1}))]\!]\tilde{\kappa}_{f'(u^*)}(\hat{\lambda}_{s+1}, u^*)\zeta^* g(f') \prod_{u \in \mathcal{Z}_s} \tilde{\kappa}_{f'(u)}(\hat{\lambda}_s, u)$$

Summing over all $f' \in \mathcal{N}(f)$ and noting that:

$$\tilde{\kappa}_{f'(r(\mathcal{Z}_{s+1}))}(\hat{\lambda}_{s+1}, r(\mathcal{Z}_{s+1})) = \tilde{\kappa}_{f'(r(\mathcal{Z}_s))}(\hat{\lambda}_s, r(\mathcal{Z}_s)) = 1$$

gives us:

$$\hat{\nu}(f) = \tilde{\kappa}_{f'(u^*)}(\hat{\lambda}_{s+1}, u^*)\zeta^*\hat{\nu}(f^*)$$

By the inductive hypothesis we then have:

$$\hat{\nu}(f) = \tilde{\kappa}_{f'(u^*)}(\hat{\lambda}_{s+1}, u^*)\zeta^*\hat{w}(f^*)$$

which by Equation (28) is equal to $\hat{w}(f)$. We have hence shown that if the inductive hypothesis holds for $t = s$ then it holds for $t = s + 1$ also. An identical argument shows that the inductive hypothesis holds for $t = 2$. We have hence shown that the inductive hypothesis holds for all $t \in [T] \setminus \{1\}$. $\square$

We now define a bijection $\mathcal{G} : \{0,1\}^{\mathcal{X}} \to 2^{\mathcal{X}}$ by:

$$\mathcal{G}(f) := \{x \in \mathcal{X} \mid f(x) = 1\} \quad \forall f \in \{0,1\}^{\mathcal{X}}$$

Note that for all $f : \mathcal{X} \to \{0,1\}$ and all $x \in \mathcal{X}'$ we have:

$$\sigma(x, \mathcal{G}(f))\epsilon + (1 - \sigma(x, \mathcal{G}(f)))(1-\epsilon) = [\![f(x) \neq f(n(x))]\!]\epsilon + [\![f(x) = f(n(x))]\!](1-\epsilon)$$

and:

$$\prod_{x \in \mathcal{G}(f)} \lambda'(x) = \prod_{t' \in [T] : f(x_{t'}) = 1} \lambda'(x_t)$$

so that:

$$\tilde{w}(\lambda, \epsilon, \mathcal{G}(f)) = \hat{w}(f)$$

and hence, by Lemma F.2, we have:

$$\tilde{w}(\lambda, \epsilon, \mathcal{G}(f)) = \hat{\nu}(f)$$

so that:

$$\sum_{\mathcal{S} \in 2^{\mathcal{X}}} [\![\gamma(\hat{u}) \in \mathcal{S}]\!] \tilde{w}(\lambda, \epsilon, \mathcal{S}) = \sum_{f \in \{0,1\}^{\mathcal{X}}} [\![f(\gamma(\hat{u})) = 1]\!] \hat{\nu}(f) \tag{34}$$

Since:

$$\bigcup \{\mathcal{N}(f) \mid f \in \{0,1\}^{\mathcal{X}}, f(\gamma(\hat{u})) = 1\} = \{f' \in \{0,1\}^{\mathcal{Z}_T} \mid f'(\hat{u}) = 1\}$$

and all sets in this union are disjoint, the right hand side of Equation (34) is equal to:

$$\sum_{f' \in \{0,1\}^{\mathcal{Z}_T}} [\![f'(\hat{u}) = 1]\!] \prod_{u \in \mathcal{Z}_T \setminus r(\mathcal{Z}_T)} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_T}(u)), f'(u)}(\mathcal{Z}_T, u, \epsilon) \tilde{\kappa}_{f'(u)}(\hat{\lambda}_T, u) \tag{35}$$

Given a vertex $u \in \mathcal{Z}_T \setminus \{r(\mathcal{Z}_T)\}$ define:

$$\mathcal{H}(u) := \Downarrow_{\mathcal{Z}_T}(u) \cup \{\uparrow_{\mathcal{Z}_T}(u)\}$$

and for all $f : \mathcal{H}(u) \to \{0,1\}$ define:

$$\hat{\zeta}(u, f) := \prod_{u' \in \Downarrow_{\mathcal{Z}_T}(u)} \tilde{\tau}_{f(\uparrow_{\mathcal{Z}_T}(u')), f(u')}(\mathcal{Z}_T, u', \epsilon)$$

Lemma F.3. *Given a vertex $u' \in \mathcal{Z}_T \setminus \{r(\mathcal{Z}_T)\}$ and an index $i \in \{0,1\}$ we have:*

$$\sum_{f \in \{0,1\}^{\mathcal{H}(u')}} [\![f(\uparrow_{\mathcal{Z}_T}(u')) = i]\!] \hat{\zeta}(u', f) = 1$$

Proof. We prove by induction on the height of $\Downarrow_{\mathcal{Z}_T}(u')$. If this height is equal to zero then $\mathcal{H}(u') = \{u', \uparrow_{\mathcal{Z}_T}(u')\}$ and for all $f : \mathcal{H}(u) \to \{0,1\}$ we have:

$$\hat{\zeta}(u', f) = \tilde{\tau}_{f(\uparrow_{\mathcal{Z}_T}(u')), f(u')}(\mathcal{Z}_T, u', \epsilon)$$

Since:

$$\tilde{\tau}_{i,0}(\mathcal{Z}_T, u', \epsilon) + \tilde{\tau}_{i,1}(\mathcal{Z}_T, u', \epsilon) = 1 \tag{36}$$

we immediately have the result for the case that the height of $\Downarrow_{\mathcal{Z}_T}(u')$ is zero. Now suppose that the result holds whenever the height of $\Downarrow_{\mathcal{Z}_T}(u')$ is equal to j (for some $j \in \mathbb{N}$). We will now show that it holds whenever the height of $\Downarrow_{\mathcal{Z}_T}(u')$ is equal to $j+1$ which will prove that the result holds always. By the inductive hypothesis we have, for all $i' \in \{0,1\}$

$$\sum_{f \in \{0,1\}^{\mathcal{H}(\triangleleft(u'))}} [\![f(u') = i']\!] \hat{\zeta}(\triangleleft(u'), f) = 1$$

and

$$\sum_{f \in \{0,1\}^{\mathcal{H}(\triangleright(u'))}} [\![f(u') = i']\!] \hat{\zeta}(\triangleright(u'), f) = 1$$

so:

$$\sum_{f \in \{0,1\}^{\mathcal{H}(u')}} [\![f(\uparrow_{\mathcal{Z}_T}(u')) = i]\!] [\![f(u') = i']\!] \hat{\zeta}(\triangleleft(u'), f) \hat{\zeta}(\triangleright(u'), f) = 1$$

and hence:

$$\sum_{f \in \{0,1\}^{\mathcal{H}(u')}} [\![f(\uparrow_{\mathcal{Z}_T}(u')) = i]\!] [\![f(u') = i']\!] \hat{\zeta}(u', f) = \tilde{\tau}_{i,i'}(\mathcal{Z}_T, u, \epsilon)$$

Summing over $i' \in \{0,1\}$ and noting Equation (36) then shows us the result holds for this case and hence, by induction, holds always. $\square$

Given $u', u'' \in \mathcal{Z}_T$ with $u'' \in \Downarrow_{\mathcal{Z}_T}(u')$ we define $\hat{\mathcal{H}}(u', u'')$ to be the maximal subtree of $\mathcal{Z}_T$ which has u' and u'' as leaves. Given, in addition, $f : \hat{\mathcal{H}}(u', u'') \to \{0, 1\}$ we define:

$$\tilde{\zeta}(u', u'', f) := \prod_{u \in \hat{\mathcal{H}}(u', u'') \setminus \{u'\}} \tilde{\tau}_{f(\uparrow_{\mathcal{Z}_T}(u)), f(u)}(\mathcal{Z}_T, u, \epsilon)$$

and:

$$\delta(u', u'') := d(u'') - d(u')$$

We now have the following lemma.

Lemma F.4. *Given $u', u'' \in \mathcal{Z}_T$ with $u'' \in \Downarrow_{\mathcal{Z}_t}(u') \setminus \{u'\}$ and indices $i', i'' \in \{0, 1\}$ we have that:*

$$\sum_{f \in \{0,1\}^{\hat{\mathcal{H}}(u', u'')}} [\![f(u') = i']\!][\![f(u'') = i'']\!]\tilde{\zeta}(u', u'', f)$$

is equal to

$$[\![i' \neq i'']\!]\phi_{\delta(u', u'')}(\epsilon) + [\![i' = i'']\!](1 - \phi_{\delta(u', u'')}(\epsilon))$$

Proof. We prove by induction on the distance from u' to u'' in $\mathcal{Z}_T$. If this distance is one then we have $u' = \uparrow_{\mathcal{Z}_T}(u'')$ and $\hat{\mathcal{H}}(u', u'') = \{u', u''\}$ so we have:

$$\sum_{f \in \{0,1\}^{\hat{\mathcal{H}}(u', u'')}} [\![f(u') = i']\!][\![f(u'') = i'']\!]\tilde{\zeta}(u', u'', f) = \tilde{\tau}_{i', i''}(\mathcal{Z}_T, u'', \epsilon)$$

which immediately implies that the inductive hypothesis holds in this case. Now suppose that the inductive hypothesis holds whenever the distance from u' to u'' is j. We now consider the case that the distance from u' to u'' is $j + 1$. Let u^* be the child of u' that lies in $\hat{\mathcal{H}}(u', u'')$. Without loss of generality assume that u'' is a descendant of $\triangleleft(u^*)$. Now choose any $i^* \in \{0, 1\}$. Given $f : \hat{\mathcal{H}}(u', u'') \to \{0, 1\}$ let:

$$h(i^*, f) = [\![f(u') = i']\!][\![f(u'') = i'']\!][\![f(u^*) = i^*]\!]$$

and let f' and f'' be the restriction of f onto the sets $\hat{\mathcal{H}}(u^*, u'')$ and $\mathcal{H}(\triangleright(u^*))$ respectively. Note that

$$\tilde{\zeta}(u', u'', f) = \tilde{\tau}_{f(u'), f(u^*)}(\mathcal{Z}_T, u^*, \epsilon)\tilde{\zeta}(u^*, u'', f')\hat{\zeta}(\triangleright(u^*), f'')$$

By Lemma F.3 and the inductive hypothesis we then have that the quantity:

$$\sum_{f \in \{0,1\}^{\hat{\mathcal{H}}(u', u'')}} h(i^*, f)\tilde{\zeta}(u', u'', f)$$

is equal to the quantity:

$$\tilde{\tau}_{i', i^*}(\mathcal{Z}_T, u^*, \epsilon)([\![i^* \neq i'']\!]\phi_{\delta(u^*, u'')}(\epsilon) + [\![i^* = i'']\!](1 - \phi_{\delta(u^*, u'')}(\epsilon)))$$

Summing over $i^* \in \{0, 1\}$ gives us the result. We have hence proved the result in general. $\square$

Suppose we have some $f : \mathcal{J} \to \{0, 1\}$. Let:

$$\hat{h}(f) = \{f' \in \{0, 1\}^{\mathcal{Z}_T} \mid \forall u \in \mathcal{J}\,,\, f'(u) = f(u)\}$$

Given $u \in \mathcal{J}$ we have that:

$$[\![f(\uparrow_{\mathcal{J}}(u)) \neq f(u)]\!]\phi_{\delta(\uparrow_{\mathcal{J}}(u), u)}(\epsilon) + [\![f(\uparrow_{\mathcal{J}}(u)) = f(u)]\!](1 - \phi_{\delta(\uparrow_{\mathcal{J}}(u), u)}(\epsilon))$$

is equal to $\tilde{\tau}_{f(\uparrow_{\mathcal{J}}(u)), f(u)}(\mathcal{J}, u, \epsilon)$ and hence Lemma F.4 implies that:

$$\sum_{f' \in \hat{\mathcal{H}}(\uparrow_{\mathcal{J}}(u), u)} [\![f'(\uparrow_{\mathcal{J}}(u)) = f(\uparrow_{\mathcal{J}}(u))]\!][\![f'(u) = f(u)]\!]\tilde{\zeta}(\uparrow_{\mathcal{J}}(u), u, f') = \tilde{\tau}_{f(\uparrow_{\mathcal{J}}(u)), f(u)}(\mathcal{J}, u, \epsilon)$$

so since, by the definition of a contraction, the edge sets of the subtrees in $\{\hat{\mathcal{H}}(\uparrow_{\mathcal{J}}(u), u) \mid u \in \mathcal{J} \setminus \{r(\mathcal{J})\}\}$ partition the edge set of $\mathcal{Z}_T$ we have, by definition of $\tilde{\zeta}$, that:

$$\sum_{f' \in \hat{h}(f)} \prod_{u \in \mathcal{Z}_T \setminus \{r(\mathcal{Z}_T)\}} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_T}(u)), f'(u)}(\mathcal{Z}_T, u, \epsilon) = \prod_{u \in \mathcal{J} \setminus \{r(\mathcal{J})\}} \tilde{\tau}_{f(\uparrow_{\mathcal{J}}(u)), f(u)}(\mathcal{J}, u, \epsilon)$$

Since for all $f' \in \hat{h}(f)$ and for all $u \in \mathcal{Z}_T \setminus \mathcal{J}$ we have $\tilde{\kappa}_{f'(u)}(\hat{\lambda}_T, u) = 1$ we have now shown that the quantity:

$$\sum_{f' \in \hat{h}(f)} \prod_{u \in \mathcal{Z}_T \setminus \{r(\mathcal{Z}_T)\}} \tilde{\tau}_{f'(\uparrow_{\mathcal{Z}_T}(u)), f'(u)}(\mathcal{Z}_T, u, \epsilon) \tilde{\kappa}_{f'(u)}(\hat{\lambda}_T, u)$$

is equal to the quantity:

$$\prod_{u \in \mathcal{J} \setminus \{r(\mathcal{J})\}} \tilde{\tau}_{f(\uparrow_{\mathcal{J}}(u)), f(u)}(\mathcal{J}, u, \epsilon) \tilde{\kappa}_{f(u)}(\hat{\lambda}_T, u)$$

Summing over all $f \in \mathcal{F}(\mathcal{J}, \hat{u})$ and noting Equations (34) and (35) gives us the result.

F.8 Theorem E.1

Lemma F.5. *Given $u, u' \in \mathcal{Z}_t$ the algorithm for computing $\nu(u, u')$ is correct.*

Proof. If $u = u'$ then the proof is trivial. Otherwise we consider the following cases:

- Consider first the case that $\hat{s} \neq \triangledown(s^*)$. Without loss of generality assume $\hat{s} = \triangleleft(s^*)$. Then we have $u \in \Downarrow(\triangleleft(\xi(s^*)))$ and since $\hat{s}' \neq \triangleleft(s^*)$ we have $u' \notin \Downarrow(\triangleleft(\xi(s^*)))$. Hence $u' \notin \Downarrow(u)$ so $\nu(u, u') = \blacktriangle$ as required.
- If $u = \xi(s^*)$ then $\hat{s} = \triangledown(s^*)$ so either $\hat{s}' = \triangleleft(s^*)$ or $\hat{s}' = \triangleright(s^*)$. In the former case we have $u' \in \Downarrow(\triangleleft(\xi(s^*))) = \Downarrow(\triangleleft(u))$ so that $\nu(u, u') = \blacktriangleleft$ and similarly in the later case we have $\nu(u, u') = \blacktriangleright$ as required.
- If $\hat{s} = \triangledown(s^*)$ and $u \neq \xi(s^*)$ we invoke the process. Consider the vertex s at any stage in the process. By induction we have that if $s \in \mathcal{E}^\bullet$ then $u' \in \Downarrow(\mu'(s))$. This is because if $s \in \mathcal{E}^\bullet$ then $\mu'(s)$ is an ancestor of $\mu'(\uparrow_{\mathcal{E}}(s))$. This further implies that when $s \neq \hat{s}$ we have $u' \in \Downarrow(\mu'(\uparrow_{\mathcal{E}}(s)))$. Now suppose that $s \in \mathcal{E}^\circ$ and without loss of generality assume $s = \triangleleft(\uparrow_{\mathcal{E}}(s))$. Then $u \in \Downarrow(\triangleleft(\xi(\uparrow_{\mathcal{E}}(s))))$ and $\mu'(\uparrow_{\mathcal{E}}(s)) \in \Downarrow(\triangleright(\xi(\uparrow_{\mathcal{E}}(s))))$ so that, since $u' \in \Downarrow(\mu'(\uparrow_{\mathcal{E}}(s)))$, we have $u' \notin \Downarrow(u)$ and hence $\nu(u, u') = \blacktriangle$ as required. Suppose now that $s \in \mathcal{E}^\bullet$ and that $u = \xi(s)$. If $\triangleleft(s) \in \mathcal{E}^\bullet$ then we have $\mu'(s) \in \Downarrow(\triangleleft(\xi(s))) = \Downarrow(\triangleleft(u))$ so that, by above, $u' \in \Downarrow(\triangleleft(u))$ and hence $\nu(u, u') = \blacktriangleleft$ as required. Similarly, if $\triangleright(s) \in \mathcal{E}^\bullet$ then $\nu(u, u') = \blacktriangleright$ as required. This completes the proof.

□

Lemma F.6. *The algorithm correctly finds $\hat{u}$.*

Proof. By induction on the depth of s we have, for all vertices s in the constructed path, that:

- If $s \in \mathcal{D}^\circ$ then u_t lies in the maximal spanning tree of $\mathcal{Z}_t$ containing $\mu(s)$ and having $\uparrow_{\mathcal{J}}(\mu(s))$ as a leaf .
- If $s \in \mathcal{D}^\bullet$ then u_t lies in the maximal spanning tree of $\mathcal{Z}_t$ with $\uparrow_{\mathcal{J}}(\mu(s))$ and $\mu'(s)$ as leaves.

Let s' be the unique leaf of $\mathcal{D}$ that is on the constructed path. If $s' \in \mathcal{D}^\circ$ then $\mu(s')$ is a leaf of $\mathcal{J}$ and hence also a leaf of $\mathcal{Z}_t$. So by above we have that u_t lies in the maximal spanning tree of $\mathcal{Z}_t$ with $\uparrow_{\mathcal{J}}(\mu(s'))$ and $\mu(s')$ as leaves. If, on the other hand, $s' \in \mathcal{D}^\bullet$ then since s' is a leaf of $\mathcal{D}$ we have that $\mu(s') = \mu'(s')$ and hence, by above, we have that u_t lies in the maximal spanning tree of $\mathcal{Z}_t$ with $\uparrow_{\mathcal{J}}(\mu(s'))$ and $\mu(s')$ as leaves. In either case we have $\hat{u} = \mu(s')$ as required. □

Lemma F.7. *The algorithm correctly finds u^*.*

Proof. By induction on the depth of s we have, for all vertices s in the constructed path, that:

- If $s \in \mathcal{E}^\circ$ then $\Gamma_{\mathcal{Z}_t}(u_t, \hat{u})$ lies in $\Downarrow_{\mathcal{Z}_t}(\mu(s))$.

- If $s \in \mathcal{E}^\bullet$ then $\Gamma_{\mathcal{Z}_t}(u_t, \hat{u})$ lies in the maximal spanning tree of $\mathcal{Z}_t$ with $\mu(s)$ and $\mu'(s)$ as leaves.

Let s' be the unique leaf of $\mathcal{E}$ that is on the constructed path. If $s' \in \mathcal{E}^\circ$ then $\mu(s')$ is a leaf of $\mathcal{Z}_t$ and hence, by above, $\Gamma_{\mathcal{Z}_t}(u_t, \hat{u}) = \mu(s')$ as required. If $s \in \mathcal{E}^\bullet$ then $\mu(s) = \mu'(s)$ and hence, by above, $\Gamma_{\mathcal{Z}_t}(u_t, \hat{u}) = \mu(s')$ as required. $\square$