Hidden Markov Contour Tree: A Spatial Structured Model for Hydrological Applications

Zhe Jiang
Department of Computer Science
University of Alabama
zjiang@cs.ua.edu

Arpan Man Sainju
Department of Computer Science
University of Alabama
asainju@crimson.ua.edu

ABSTRACT

Spatial structured models are predictive models that capture dependency structure between samples based on their locations in the space. Learning such models plays an important role in many geoscience applications such as water surface mapping, but it also poses significant challenges due to implicit dependency structure in continuous space and high computational costs. Existing models often assume that the dependency structure is based on either spatial proximity or network topology, and thus cannot incorporate complex dependency structure such as contour and flow direction on a 3D potential surface. To fill the gap, this paper proposes a novel spatial structured model called hidden Markov contour tree (HMCT), which generalizes the traditional hidden Markov model from a total order sequence to a partial order polytree. HMCT also advances existing work on hidden Markov trees through capturing complex contour structures on a 3D surface. We propose efficient model construction and learning algorithms. Evaluations on real world hydrological datasets show that our HMCT outperforms multiple baseline methods in classification performance and that HMCT is scalable to large data sizes (e.g., classifying millions of samples in seconds).

CCS CONCEPTS

• **Information systems** → *Geographic information systems*; *Data mining*; • **Computing methodologies** → *Machine learning*; • **Applied computing** → *Earth and atmospheric sciences.*

KEYWORDS

Hidden Markov Contour Tree; Spatial Structured Model; Structured Prediction; 3D Surface; Flood Mapping

ACM Reference Format:
Zhe Jiang and Arpan Man Sainju. 2019. Hidden Markov Contour Tree: A Spatial Structured Model for Hydrological Applications. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330878

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

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330878

1 INTRODUCTION

Spatial structured models are predictive models that capture dependency structure between samples based on their locations. Given data samples in a spatial raster framework with explanatory feature layers and a potential field layer, as well as an independent set of training samples with class labels, the spatial structured learning problem aims to learn a model that can predict sample classes in the same framework [8]. For example, in flood extent mapping from earth imagery, data samples are imagery pixels in a regular grid, the explanatory feature layers are spectral bands, and the potential field can be elevation that controls water flow directions. The goal is to predict the classes (flood or dry) of pixels based on both the spectral features and the implicit flow directions based on elevation.

The problem is important in many societal applications such as flood extent mapping for disaster response and national water forecasting. Flood extent mapping plays a crucial role in addressing grand societal challenges such as disaster management, national water forecasting, as well as energy and food security. For example, during Hurricane Harvey floods in 2017, first responders needed to know where flood water was in order to plan rescue efforts. In national water forecasting, detailed flood extent maps can be used to calibrate and validate the NOAA National Water Model [16], which can forecast the flow of over 2.7 million rivers and streams through the entire continental U.S. [5]. In current practice, flood extent maps are mostly generated by flood forecasting models, whose accuracy is often unsatisfactory in high spatial details [5]. Other ways to generate flood maps involve sending a field crew on the ground to record high-water marks, or visually interpreting earth observation imagery [3]. However, the process is both expensive and time consuming. With the large amount of high-resolution earth imagery being collected from satellites (e.g., DigitalGlobe, Planet Labs), aerial planes (e.g., NOAA National Geodetic Survey), and unmanned aerial vehicles, the cost of manually labeling flood extent becomes prohibitive. Note that though we use flood mapping as a motivation example, the problem can potentially be applied to many other applications such as water quality monitoring and air pollution mapping in which pollutants are transmitted following flow directions [12], protein structure learning in biochemistry [7], and geometric shape analysis in computer graphics [18].

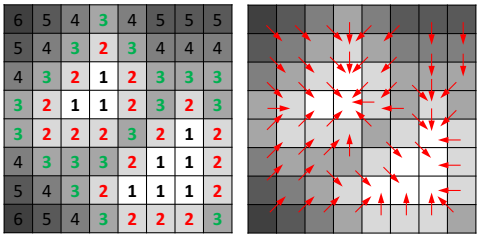
However, the problem poses several unique challenges that are not well addressed in traditional classification problems. First, implicit spatial dependency structure exists between pixel locations. For example, due to gravity, flood water tends to flow from one location to nearby lower locations. Such dependency structure is complex, following contour patterns on a 3D surface. Second, data contains rich noise and obstacles. For example, high-resolution

earth imagery often has noise, clouds and shadows. In addition, the spectral features of image pixels can be insufficient to distinguish classes (also called class confusion) due to heterogeneity. For instance, pixels of tree canopies overlaying flood water have the same spectral features with those trees in dry areas, yet their classes are different. Finally, the problem is also computationally challenging due to the cost of modeling complex spatial structure on a large data volume (e.g., billions of sample locations).

Existing spatial structured models often assume that the dependency structure is based on spatial proximity or spatial network topology. Models based on spatial proximity assume that adjacent or nearby locations have stronger dependency. The assumption comes from the first law of geography [26], "everything is related to everything else, but near things are more related than distant things". Examples include Markov random field [1] , conditional random field [13], spatial network embedding [28, 31], and convolutional neural networks [30]. Other works incorporate spatial structures by spatial regularization on loss function (to penalize difference between nearby locations) together with efficient optimization [10, 22, 23]. Models based on spatial network topology assume that dependency between sample locations only follows an underlying spatial network structure. These models are useful for samples that are only located along spatial networks (e.g., traffic accidents along road networks). Examples of such models include spatial network Kriging [1, 2] and spatial network autoregressive models [27]. In summary, related works often capture undirected spatial dependency structure based on distance or network topology. Complex dependency structure that is both directed and prevalent at all locations in the continuous space are largely unexplored. Recently, a geographical hidden Markov tree (HMT) model [29] has been proposed, which captures directed spatial dependency based on flow directions across all locations, but it is largely motivated by a one dimensional spatial view and does not consider complex spatial structures such as contours on a 3D surface.

To fill the gap, we propose a novel spatial structure model called hidden Markov contour tree (HMCT). It is a probabilistic graphical model that generalizes the common hidden Markov model (HMM) from a total order sequence to a partial order polytree. Specifically, the hidden class layer contains nodes (pixels) in a contour tree structure to reflect flow directions between all locations on a 3D surface. To speed up model learning and class inference, we propose efficient algorithms based on contour tree node collapsing and value pre-aggregation. In summary, we make the following contributions in the paper.

- We propose a novel spatial structure model called hidden Markov contour tree, which captures flow directions and contour structures on a 3D surface.
- We design algorithms for contour tree construction, as well as parameter learning and class inference for the HMCT model. We improve algorithm efficiency by tree node collapsing and pre-aggregation.
- We evaluate the proposed model on real world flood mapping datasets. Results show that HMCT outperforms multiple baseline methods in classification performance, and proposed algorithms are scalable on a large data volume (classifying millions of samples in seconds).



(a) Elevation values with contours (b) Flow directions

Figure 1: Example of raster framework and flow dependency

2 PROBLEM STATEMENT

2.1 Preliminaries

Definition 2.1. A *spatial raster framework* is a tessellation of a two dimensional plane into a regular grid of N cells. Spatial neighborhood relationship exists between cells based on cell adjacency. The framework can contain m non-spatial explanatory feature layers (e.g., spectral bands in earth imagery), one potential field layer (e.g., elevation), and one class layer (e.g., *flood*, *dry*).

Definition 2.2. Each cell in a raster framework is a *spatial data sample*, noted as $\mathbf{s_n} = (\mathbf{x}_n, \phi_n, y_n)$, where $n \in \mathbb{N}, 1 \leq n \leq N$, $\mathbf{x}_n \in \mathbb{R}^{m \times 1}$ is a vector of m non-spatial explanatory feature values with each element corresponding to one feature layer, $\phi_n \in \mathbb{R}$ is a cell's potential field value, and $y_n \in \{0, 1\}$ is a binary class label.

A raster framework with all samples is noted as $\mathcal{F} = \{\mathbf{s_n} | n \in \mathbb{N}, 1 \leq n \leq N\}$, non-spatial explanatory features of all samples are noted as $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$, the potential field layer is noted as $\mathbf{\Phi} = [\phi_1, ..., \phi_N]^T$, and the class layer is noted as $\mathbf{Y} = [y_1, ..., y_N]^T$.

Definition 2.3. A *contour* in a raster framework is a set of contiguous samples whose potential field values are equal. For example, in Figure 1(a), there are two contours for the elevation value 1. Note that our definition of contour here is discrete. Original definition based on a continuous field in topology can be found in [6, 14].

Definition 2.4. Spatial flow dependency exists between cells following the gradient on the potential field layer. Formally, a *flow dependency* $\mathbf{s}_i \rightsquigarrow \mathbf{s}_j$ exists if and only if there exist a sequence of neighboring (adjacent) cells $< \mathbf{s}_i, \mathbf{s}_{p_1}, \mathbf{s}_{p_2}, ..., \mathbf{s}_{p_l}, \mathbf{s}_j >$ such that $\phi_i \leq \phi_{p_1}, \phi_{p_l} \leq \phi_j$, and $\phi_{p_k} \leq \phi_{p_{k+1}}$ for any $1 \leq k \leq l-1$. For example, due to gravity, flood water can flow from cells with elevation 3 to neighboring cells with elevation 2 in Figure 1(a).

2.2 Formal problem definition

We now formally define the spatial structured learning problem.

Input:

- Spatial raster framework $\mathcal{F} = \{\mathbf{s}_n | n \in \mathbb{N}, 1 \leq n \leq N\}$
- Explanatory features of samples $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$
- Spatial potential field values of samples: $\mathbf{\Phi} = [\phi_1, ..., \phi_N]^T$
- Training samples $\{\mathbf{s}_k | k \in \textit{training set}\}$ outside $\mathcal{F}$

Output: A spatial structured model $f : \mathbf{Y} = f(\mathbf{X})$

Objective: minimize classification errors

Constraint:
- Explanatory feature layers contain noise and obstacles
- Sample classes follow partial order flow dependency
- Sample class is binary, $y_n \in \{0, 1\}$

3 PROPOSED APPROACH

The section introduces a novel spatial structured model called hidden Markov contour tree together with effective and efficient learning algorithms.

3.1 Modeling Spatial Contour Structure

Our goal is to explore a data structure to represent flow dependency between pixel locations on a potential field (e.g., 3D elevation surface). Such dependency structure can be represented by *contour tree* [4, 6]. A contour tree is a polytree (directed acyclic graph whose underlying undirected graph is a tree) that captures the evolution of contours (level sets) of a potential field. Figure 2(b) shows an example based on the elevation surface in Figure 1. Assume we increase an elevation threshold from 1 to 6. For threshold 1, two separate contours appear, corresponding to the two leaf nodes on the bottom of Figure 2(b). For threshold 2, the two contours both grow bigger but remain separated. For threshold 3, the two contours merge into one (corresponding to the central node with number 3), and a new separate contour appears based on the bottom right pixel in Figure 1(a) (corresponding to the tree node 3 on a side branch). For threshold 4, a contour of elevation 3 splits into three. Then, each contour grows separately as the threshold further increases. Contour tree is naturally a good representation for flow direction dependency between locations on an elevation surface. For example, in Figure 2(b), if the central node with elevation 3 is in the *flood* class, all its parent nodes (the ones with elevations 1 and 2 below it) must be in the *flood* class, since flood water flows from a high elevation to nearby lower elevations.

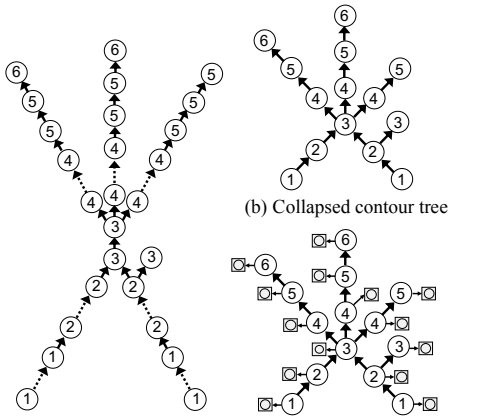


Figure 2: Illustration of hidden Markov contour tree

Existing research on contour tree often focuses on the critical nodes (such as local minimum and maximum, saddle point) [6, 14]. Tree construction algorithms often require an input elevation surface to be represented as a mesh with unique values [4] . In order to run the algorithms on an elevation surface with duplicated values, we can use perturbation to enforce an arbitrary order on locations with an equal elevation. This makes the tree unnecessarily large, as shown in Figure 2(a). To address this limitation, we propose a collapsed contour tree, which starts from a contour tree from existing algorithm but then collapses nodes in the same contour. Figure 2(a-b) shows an example. For instance, two connected nodes with elevation 3 in Figure 2(a) are merged into one in Figure 2(b). Node collapsing can dramatically reduce the size of a contour tree, potentially reducing the cost of learning and inference algorithms. Algorithm 1 shows the detailed steps for node collapsing. The main idea is to use breadth-first search to find each collapsable contour component, and then to collapse tree nodes within that component. The time complexity is $O(N)$ where N is the total number of nodes in the uncollapsed contour tree.

Algorithm 1 Collapse Contour Tree

Input:
- An uncollapsed contour tree

Output:
- A collapsed contour tree

```
Initialize all nodes as unvisited
for each tree node s_n by topological order do
    if s_n is unvisited then
        Mark s_n as visited
        Breadth First Search (BFS) from s_n to its contour
        Collapse BFS traversed nodes, mark them visited
return the collapsed contour tree
```

3.2 Hidden Markov Contour Tree (HMCT)

We propose a hidden Markov contour tree (HMCT), a probabilistic graphical model that generalizes the common hidden Markov model from a total order sequence to a partial order polytree. A HMCT model consists of two layers: a hidden class layer in the form of a collapsed contour tree, and an observation feature layer. Each contour tree node in the hidden class layer represents a same unknown hidden class shared over all pixels in the contour. Each contour tree edge represents the class transitional probability between pixels from two contours based on flow dependency. A hidden class node in contour tree is connected to a set of observed feature nodes (circle in boxes in Figure 2(c)), which correspond to the explanatory feature vectors of all pixels in that contour.

The joint distribution of all samples' features and classes can be formulated as Equation 1, where $\mathcal{P}_n$ is the set of parent samples of the nth sample in the dependency tree ($\mathcal{P}_n = \emptyset$ for a leaf node), and $y_{k\in\mathcal{P}_n} \equiv \{y_k | k \in \mathcal{P}_n\}$ is the set of parent node classes of node n. $\mathbf{x}_n = \{\mathbf{x}_{n_i} | 1 \le i \le N_n\}$ is the set of feature nodes corresponding to hidden class node n (i.e., $\mathbf{x}_{n_i}$ is the feature vector for a pixel n_i in the nth contour). Note that our notation of $\mathbf{x}_n$ here represents a set, and should not be confused with notations in Section 2.

$$P(\mathbf{X}, \mathbf{Y}) = P(\mathbf{X}|\mathbf{Y})P(\mathbf{Y}) = \prod_{n=1}^{N} \prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n) \prod_{n=1}^{N} P(y_n|y_{k\in\mathcal{P}_n}) \quad (1)$$

For simplicity, observed feature nodes are assumed conditionally independent given their hidden class node, following an i.i.d. Gaussian distribution, as shown in Equation 2, where $\boldsymbol{\mu}_{y_n}$ and Σ_{y_n} are the mean and covariance matrix of feature vector $\mathbf{x}_n$ for class y_n ($y_n = 0, 1$).

$$P(\mathbf{x}_{n_i}|y_n) \sim \mathcal{N}(\boldsymbol{\mu}_{y_n}, \Sigma_{y_n}) \tag{2}$$

Class transitional probability follows the partial order flow dependency constraint. For example, due to gravity, if any parent's class is *dry*, the child's class must be *dry*; if all parents' classes are *flood*, then the child has a high probability of being *flood* due to spatial autocorrelation. Consider *flood* as the positive class (class 1) and *dry* as the negative class (class 0), the product of all parents' classes is $y_{\mathcal{P}_n} \equiv \prod_{k \in \mathcal{P}_n} y_k$. Class transitional probability can be modeled as Table 1, where ρ is a parameter close to 1. The prior class probability of a node without parents is also shown on the right of Table 1, where π is another parameter.

Table 1: Class transition probability and prior probability

$P(y_n\|y_{\mathcal{P}_n})$	$y_{\mathcal{P}_n} = 0$	$y_{\mathcal{P}_n} = 1$
$y_n = 0$	1	$1 - \rho$
$y_n = 1$	0	ρ

	$P(y_n)$
$y_n = 0$	$1 - \pi$
$y_n = 1$	π

3.3 HMCT Learning and Inference

The parameters of hidden Markov contour tree include the mean and covariance matrix of sample features in each class, prior probability of leaf node classes, and class transition probability for non-leaf nodes. We denote the entire set of parameters as $\boldsymbol{\Theta} = \{\rho, \pi, \boldsymbol{\mu}_c, \Sigma_c | c = 0, 1\}$. Learning the set of parameters poses two major challenges: first, there exist unknown hidden class variables $\mathbf{Y} = [y_1, ..., y_N]^T$, which are non-i.i.d.; second, the number of node variables can be large.

To address these challenges, we propose to use the expectation-maximization (EM) algorithm and message (belief) propagation. Our EM-based approach has the following major steps:

(a) Initialize parameter set $\boldsymbol{\Theta}_0$
(b) Compute posterior distribution of hidden classes: $P(\mathbf{Y}|\mathbf{X}, \boldsymbol{\Theta}_0)$
(c) Compute posterior expectation of log likelihood: $LL(\boldsymbol{\Theta}) = \mathbb{E}_{\mathbf{Y}|\mathbf{X},\boldsymbol{\Theta}_0} \log P(\mathbf{X}, \mathbf{Y}|\boldsymbol{\Theta})$
(d) Update parameters: $\boldsymbol{\Theta}_0 \leftarrow \arg\max_{\boldsymbol{\Theta}} LL(\boldsymbol{\Theta})$
Return $\boldsymbol{\Theta}_0$ if it's converged, otherwise goto (b)

Algorithm 2 shows the details. First, we initialize parameters either with random values within reasonable range (for ρ and π) or with initial estimates based on training samples (e.g., the mean and covariance of features in each class for $\boldsymbol{\mu}_c$ and Σ_c). Then we conduct a breadth-first search on the tree from any start node as a root. After this, the algorithm starts the iteration till parameters converge. In each iteration, it propagates messages first from leaves to root (steps 6-7) and then from root to leaves (steps 8-9). Marginal posterior distribution of node classes are then computed (steps 10-11). Based on this, the algorithm updates parameters (step 12). Message propagation is based on the sum and product algorithm [11, 20]. Propagation of message along nodes in a graph (or tree) is equivalent to marginalizing out node variables in the overall joint distribution in Equation 1.

Algorithm 2 EM Algorithm for Hidden Markov Tree

Input:
- $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$: cell sample feature matrix
- $\mathcal{T}$: a contour tree for spatial dependency
- ϵ: parameter convergence threshold

Output:
- $\boldsymbol{\Theta} = \{\rho, \pi, \boldsymbol{\mu}_c, \Sigma_c | c = 0, 1\}$: set of model parameters

1: Initialize $\boldsymbol{\Theta}_0$, $\boldsymbol{\Theta}$
2: Find a node n_0 in $\mathcal{T}$ as the root
3: Do a breadth-first search (BFS) on $\mathcal{T}$ from root n_0
4: **while** $\|\boldsymbol{\Theta}_0 - \boldsymbol{\Theta}\|_\infty > \epsilon$ **do**
5: $\quad\boldsymbol{\Theta}_0 \leftarrow \boldsymbol{\Theta}$
6: $\quad$**for each** y_n from leaves to root in BFS **do**
7: $\quad\quad$Compute messages $f^i_n(y_n), f^o_n(y_n)$ by (3)-(6)
8: $\quad$**for each** y_n from root to leaves in BFS **do**
9: $\quad\quad$Compute messages $g^i_n(y_n), g^o_n(y_n)$ by (7)-(12)
10: $\quad$**for each** $y_n, 1 \le n \le N$ **do**
11: $\quad\quad$// Compute marginal distributions: $P(y_n|\mathbf{X}, \boldsymbol{\Theta}_0), P(y_n, y_{k\in\mathcal{P}_n}|\mathbf{X}, \boldsymbol{\Theta}_0)$
12: $\quad$Update $\boldsymbol{\Theta}$ based on marginal distributions: $\boldsymbol{\Theta} \leftarrow \arg\max_{\boldsymbol{\Theta}} \mathbb{E}_{\mathbf{Y}|\mathbf{X},\boldsymbol{\Theta}_0} \log P(\mathbf{X}, \mathbf{Y}|\boldsymbol{\Theta})$ by (13)-(16)
13: **return** Θ

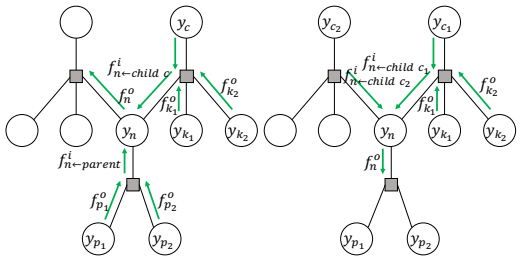
Figure 3 illustrates the forward message propagation process from leaves to root (denoted as f). Each node can have only one outgoing message (denoted as $f^o_n(y_n)$, but can have multiple incoming messages (incoming messages from a child c and the parent side are denoted as $f^i_{n\leftarrow child\ c}(y_n)$ and $f^i_{n\leftarrow parent}(y_n)$ respectively). As illustrated in Figure 3(a), computing $f^i_{n\leftarrow child\ c}(y_n)$ involves integration (sum) over the product of outgoing messages from the child c and c's other parents, together with the conditional probability between c and c's all parents, as specified in (3). The outgoing message from node n has two cases: to its child c or to its parents. In the first case, we first need to compute another incoming message from parents $f^i_{n\leftarrow parent}(y_n)$ based on (4). Then the outgoing message to a child c_0 can be computed based on (5). These are also illustrated in Figure 3(a). In the second case, we only need to compute outgoing message to parent based on (6) (also illustrated in Figure 3(b)). Note that we denote $P(\mathbf{x}_n|y_n) \equiv \prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n)$ in these equations for brevity.

$$f^i_{n\leftarrow child\ c}(y_n) = \sum_{y_c, y_{\{k\in\mathcal{P}_c, k\neq n\}}} P(y_c|y_{k\in\mathcal{P}_c}) f^o_c(y_c) \prod_{k\in\mathcal{P}_c, k\neq n} f^o_k(y_k) \tag{3}$$

$$f^i_{n\leftarrow parent}(y_n) = \begin{cases} P(y_n) & \text{if } n \text{ no parent} \\ \sum_{y_{k\in\mathcal{P}_n}} P(y_n|y_{k\in\mathcal{P}_n}) \prod_{k\in\mathcal{P}_n} f^o_k(y_k) & \text{otherwise} \end{cases} \tag{4}$$

$$f^o_n(y_n) = P(\mathbf{x}_n|y_n) f^i_{n\leftarrow parent}(y_n) \prod_{c\in C_n, c\neq c_0} f^i_{n\leftarrow child\ c}(y_n) \tag{5}$$

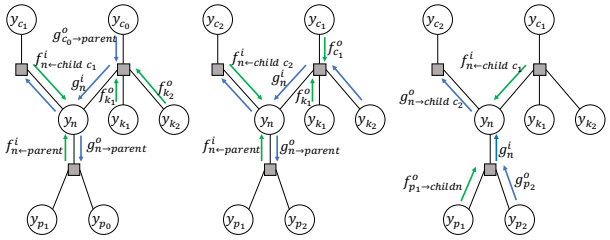
$$f^o_n(y_n) = P(\mathbf{x}_n|y_n) \prod_{c\in C_n} f^i_{n\leftarrow child\ c}(y_n) \tag{6}$$



(a) Case 1: f out to child (b) Case 2: f out to parent

Figure 3: Illustration of message propagation leaves to root

Backward message propagation from root to leaves also follows a recursive process. Each node only has one incoming message but several outgoing messages. There are three cases as shown in Figure 4(a-c): incoming message $g^i_n(y_n)$ can be from a child, a child's parent, or a parent. In the first two cases, $g^i_n(y_n)$ is computed from outgoing messages on the child side based on (7) and (8) respectively. The outgoing messages to another child or parent can be computed based on (9) and (10) for both of the first two cases. In the third case, $g^i_n(y_n)$ is computed based on outgoing messages from the parent side, as specified in (11). The outgoing message to a child can be computed based on (12). Since in the third case, the incoming message is from the parent side, we do not need to compute outgoing message to parent $g^o_{n\to parent}(y_n)$.



(a) Case 1: g from child's child (b) Case 2: g from a child's parent (c) Case 3: g from parent

Figure 4: Illustration of message propagation leaves to root

$$g^i_n(y_n) = \sum_{y_{c_0}, y_{\{p\in\mathcal{P}_{c_0}, p\neq n\}}} g^o_{c_0\to parent}(y_{c_0}) P(y_{c_0}|y_{p\in\mathcal{P}_{c_0}}) \prod_{p\in\mathcal{P}_{c_0}, p\neq n} f^o_p(y_p) \tag{7}$$

$$g^i_n(y_n) = \sum_{y_{c_0}, y_{\{p\in\mathcal{P}_{c_0}, p\neq n\}}} g^o_{p_{c_0}\to child\ c_0}(y_{p_{c_0}}) f^o_{c_0}(y_{c_0}) P(y_{c_0}|y_{k\in\mathcal{P}_{c_0}}) \prod_{p\in\mathcal{P}_{c_0}, p\neq p_{c_0}, p\neq n} f^o_p(y_p) \tag{8}$$

$$g^o_{n\to child\ c}(y_n) = g^i_n(y_n) P(\mathbf{x}_n|y_n) f^i_{n\leftarrow parent}(y_n) \prod_{c'\in C_n, c'\neq c_0, c'\neq c} f^i_{n\leftarrow child\ c'}(y_n) \tag{9}$$

$$g^o_{n\to parent}(y_n) = g^i_n(y_n) P(\mathbf{x}_n|y_n) \prod_{c\in C_n, c\neq c_0} f^i_{n\leftarrow child\ c}(y_n) \tag{10}$$

$$g^i_n(y_n) = \sum_{y_{k\in\mathcal{P}_n}} P(y_n|y_{k\in\mathcal{P}_n}) g^o_{p_0\to n}(y_{p_0}) \prod_{p\in\mathcal{P}_n} f^o_p(y_p) \tag{11}$$

$$g^o_{n\to child\ c}(y_n) = g^i_n(y_n) P(\mathbf{x}_n|y_n) \prod_{c'\in C_n, c'\neq c} f^i_{n\leftarrow child\ c'}(y_n) \tag{12}$$

After both forward and backward message propagation, we can compute marginal posterior distribution of hidden class variables. The unnormalized marginal posterior distribution of the class of a leaf node (without parents), as well as the class of a non-leaf node (with parents) can be computed by multiplying all messages coming into these nodes. We do not provide details due to space limit. Results can be normalized by dividing them over the total sum of all variable configurations.

After computing the marginal posterior distribution, we can update model parameters by maximizing the posterior expectation of log likelihood (the maximization or M step in EM). Taking the marginal posterior distributions computed above together with the prior and transitional probabilities in Table 1 into the posterior expectation, we can get the following parameter update formulas (the M step in EM).

$$\rho = \frac{\sum_{n|\mathcal{P}_n\neq\emptyset} \sum_{y_n} \sum_{y_{\mathcal{P}_n}} y_{\mathcal{P}_n} y_n P(y_n, y_{\mathcal{P}_n}|\mathbf{X}, \mathbf{\Theta_0})}{\sum_{n|\mathcal{P}_n\neq\emptyset} \sum_{y_n} \sum_{y_{\mathcal{P}_n}} y_{\mathcal{P}_n} P(y_n, y_{\mathcal{P}_n}|\mathbf{X}, \mathbf{\Theta_0})} \tag{13}$$

$$\pi = \frac{\sum_{n|\mathcal{P}_n=\emptyset} \sum_{y_n} y_n P(y_n|\mathbf{X}, \mathbf{\Theta_0})}{\sum_{n|\mathcal{P}_n=\emptyset} \sum_{y_n} P(y_n|\mathbf{X}, \mathbf{\Theta_0})} \tag{14}$$

$$\boldsymbol{\mu}_c = \frac{\sum_{n=1}^{N} \sum_{i=1}^{N_n} \mathbf{x}_{n_i} P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}{\sum_{n=1}^{N} \sum_{i=1}^{N_n} P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}, c = 0, 1 \tag{15}$$

$$\Sigma_c = \frac{\sum_{n=1}^{N} \sum_{i=1}^{N_n} (\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)^T P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}{\sum_{n=1}^{N} \sum_{i=1}^{N_n} P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}, c = 0, 1 \tag{16}$$

Class inference: After learning model parameters, we can infer hidden class variables by maximizing the overall probability. A naive approach that enumerate all combinations of class assignment is infeasible due to the exponential cost. We use a dynamic programming based method called *max-sum* [19]. The process is similar to the sum and product algorithm above. The main difference is that instead of using sum operation, we need to use max operation in message propagation, and also memorize the optimal variable values. We omit the details due to space limit.

3.4 Computational Performance Tuning

Time complexity of Algorithm 2: The cost includes contour tree traversal, message calculation, marginal probability calculation, and parameter update. Tree traversal cost is $O(N_T)$ where N_T is the number of collapsed contour tree nodes. Message calculation for the nth contour tree node is $O(N_n 2^d)$, where N_n is the number of feature nodes for tree node n, and d is the in-degree. The exponential term 2^d comes from the sum of products on all parent combinations. The term N_n comes from the aggregation over individual feature nodes in $P(\mathbf{x}_n|y_n) \equiv \prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n)$ as well as in updating $\boldsymbol{\mu}_c$ and

Σ_c. Thus, the cost per iteration is $O(N2^d)$ where N is the total number of feature nodes or pixels ($N \gg N_T$). The overall cost is $O(N2^d I)$ where I is the number of iterations.

From the analysis above, we can see that the total cost is linear to the total number of feature nodes N instead of the number of collapsed contour tree nodes N_T, due to the need of aggregating over individual feature nodes. To improve computational efficiency, we propose to **pre-aggregate** feature node values for each contour tree node (class node) once, and later to use the pre-aggregated results in message calculation. In this way, the remaining cost after pre-aggregation is $O(N_T)$ instead of $O(N)$. Pre-aggregation can be done during contour tree node collapsing.

The next question is how to do the pre-aggregation correctly (returning the same final results). In fact, there are only three parts involving aggregation over feature nodes: calculating $P(\mathbf{x}_n|y_n)$ in Equation 17, updating $\boldsymbol{\mu}_c$ in Equation 15, and updating Σ_c in Equation 16. Our goal here is to compute these three parts based on pre-aggregated feature nodes over each class node ($\mathbf{x}_{n_i}$ with $1 \le n_i \le N_n$). The specific process is described in Theorem 3.1.

$$\log P(\mathbf{x}_n|y_n) = \log \prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n) = -\frac{N_n}{2}(m\log(2\pi) + \log|\Sigma_{y_n}|) - \frac{1}{2}\sum_{i=1}^{N_n}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)^T \Sigma_{y_n}^{-1}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c) \tag{17}$$

Theorem 3.1. *Assume* $A_n = \sum_{i=1}^{N_n} \mathbf{x}_{n_i}\mathbf{x}_{n_i}^T$, $B_n = \sum_{i=1}^{N_n} \mathbf{x}_{n_i}$, $C_n = N_n$. *We can calculate Equation 17, Equation 15, and Equation 16 based on pre-aggregated results A_n, B_n and C_n on each class node n.*

Proof. Equation 18, Equation 19, and Equation 20 show how to calculate Equation 17, Equation 15, and Equation 16 based on pre-aggregated values A_n, B_n and C_n. Note that $\langle\cdot,\cdot\rangle$ is inner product between two matrices. We omit details due to space limit.

$$\sum_{i=1}^{N_n} \mathbf{x}_{n_i} = B_n \tag{18}$$

$$\sum_{i=1}^{N_n}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)^T = A_n - B_n\boldsymbol{\mu}_c^T - (B_n\boldsymbol{\mu}_c^T)^T + C_n\boldsymbol{\mu}_c\boldsymbol{\mu}_c^T \tag{19}$$

$$\sum_{i=1}^{N_n}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_{y_n})^T\Sigma_{y_n}^{-1}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_{y_n}) = \langle\Sigma_{y_n}^{-1}, A_n - B_n\boldsymbol{\mu}_{y_n}^T - (B_n\boldsymbol{\mu}_{y_n}^T)^T + N_n\boldsymbol{\mu}_{y_n}\boldsymbol{\mu}_{y_n}^T\rangle \tag{20}$$

□

Time complexity of refined algorithm: Pre-aggregation part (i.e., computing A_n, B_n, and C_n for all contour tree node n) is $O(N)$, but this can be done for only once during contour tree construction and node collapsing. After pre-aggregation, the time complexity for learning and inference part is reduced from $O(2^d N \cdot I)$ to $O(2^d N_T \cdot I)$. Since N_T is the number of contour tree nodes after node collapsing, it is significantly smaller than the total number of pixels N.

4 EXPERIMENTAL EVALUATION

In this section, we compared our proposed method with baseline methods in classification performance on two real world datasets. We also evaluated the computational scalability of our method, particularly on the effect of tree node collapsing. Experiments were conducted on a Dell workstation with Intel(R) Xeon(R) CPU E5-2687w v4 @ 3.00GHz, 64GB main memory, and Windows 10. Unless specified otherwise, we used default parameters in open source tools for baseline methods. Candidate classification methods include:

- **Non-spatial classifiers with raw features**: We tested decision tree (**DT**), random forest (**RF**), maximum likelihood classifier (**MLC**), and gradient boosted tree (**GBM**) in R packages on **raw** features (red, green, blue spectral bands).
- **Non-spatial classifiers with additional potential field feature (elev.)**: We tested **DT**, **RF** and **MLC** here.
- **Non-spatial classifier with post-processing label propagation (LP):** We tested **DT**, **RF** and **MLC** based on label propagation with 4-neighborhood [32].
- **Transductive SVM:** Since our method utilizes features of test samples, we included Transductive SVM (SVM-Light [9]), a semi-supervised tranductive method for fair comparison.
- **Markov random field (MRF):** We used open source implementation [25] based on the graph cut method [24].
- **Deep learning:** We used U-Net [21] in Python.
- **Hidden Markov Tree (HMT)**: We used HMT [29] in C++.
- **Hidden Markov Contour Tree (HMCT):** This is our proposed methods. Both collapsed (**HMCT-UC**) and uncollapsed (**HMCT-C**) versions were implemented in C++.

Dataset description: We used two flood mapping datasets from the cities of Greensville and Grimesland in North Carolina during Hurricane Mathew in 2016. Explanatory features were red, green, blue bands in aerial imagery from NOAA National Geodetic Survey [15]. The potential field was digital elevation map from the University of North Carolina Libraries [17]. All data were resampled into 2 meter by 2 meter resolution. The number of training and test samples were 5000 per class (flood, dry) in both datasets. The test region size was 1856 by 3149 in Greensville and 2757 by 3853 in Grimesland. Training samples in Greensville were drawn beyond but close to the test region. Training samples in Grimesland dataset were drawn far away from the test region to evaluate model generalizability. Note that for the deep learning method U-Net, we had to provide extra training set in form of contiguous segmented flood imagery. We used 240 image blocks (each of a size of 224 by 224) for training, and 60 blocks for validation. We used a batch size of 16, a learning rate of 0.0001, and 200 epochs.

4.1 Classification Performance Comparison

Results on the Greensville dataset were summarized in Table 2. Decision tree, random forest, gradient boosted tree, and maximum likelihood classifier achieved overall F-scores between 0.71 and 0.84 on raw features. Adding the elevation feature improved the overall F-score dramatically for decision tree and random forest, but not for maximum likelihood classifier. The improvement was due to the relatively lower elevations among *flood* class pixels. However, the recall of the flood class was still below 0.7 even after adding the elevation feature, probably because the right elevation threshold in models differ between training samples and test samples. Post-processing based on label propagation (LP) slightly improved performance of non-spatial classifiers due to the removal of salt-and-pepper noise errors. Similar results were found in Markov

Table 2: Classification on Real Dataset in Greenville NC

Classifiers	Class	Precision	Recall	F	Avg. F
DT+Raw	Dry	0.67	0.87	0.76	0.72
	Flood	0.82	0.58	0.68	
RF+Raw	Dry	0.65	0.96	0.78	0.71
	Flood	0.93	0.50	0.65	
GBM+Raw	Dry	0.76	1.00	0.86	0.84
	Flood	1.00	0.69	0.82	
MLC+Raw	Dry	0.75	0.88	0.81	0.80
	Flood	0.86	0.71	0.78	
DT+elev.	Dry	0.76	1.00	0.86	0.84
	Flood	1.00	0.69	0.82	
RF+elev.	Dry	0.75	1.00	0.86	0.83
	Flood	1.00	0.67	0.80	
MLC+elev.	Dry	0.72	0.97	0.82	0.79
	Flood	0.96	0.62	0.75	
DT+LP	Dry	0.66	0.96	0.78	0.72
	Flood	0.93	0.51	0.66	
RF+LP	Dry	0.66	0.99	0.79	0.72
	Flood	0.99	0.49	0.65	
MLC+LP	Dry	0.76	0.95	0.84	0.82
	Flood	0.93	0.69	0.80	
MRF	Dry	0.74	0.97	0.84	0.81
	Flood	0.96	0.66	0.78	
TSVM	Dry	0.74	0.83	0.78	0.77
	Flood	0.81	0.71	0.76	
U-Net	Dry	0.82	0.68	0.74	0.76
	Flood	0.72	0.85	0.78	
HMT	Dry	0.60	0.98	0.75	0.63
	Flood	0.97	0.35	0.52	
HMCT-UC	Dry	0.99	0.88	0.93	0.94
	Flood	0.89	0.99	0.94	
HMCT-C	Dry	0.99	0.89	0.94	0.94
	Flood	0.90	0.99	0.94	

Table 3: Classification on Real Data in Grimesland, NC

Classifiers	Class	Precision	Recall	F	Avg. F
DT+Raw	Dry	0.54	0.93	0.68	0.50
	Flood	0.74	0.21	0.33	
RF+Raw	Dry	0.55	0.98	0.70	0.52
	Flood	0.92	0.21	0.34	
GBM+Raw	Dry	0.87	0.06	0.12	0.40
	Flood	0.51	0.98	0.67	
MLC+Raw	Dry	0.71	0.90	0.80	0.77
	Flood	0.87	0.64	0.74	
DT+elev.	Dry	0.85	0.53	0.65	0.71
	Flood	0.66	0.91	0.76	
RF+elev.	Dry	1.00	0.25	0.40	0.56
	Flood	0.57	1.00	0.72	
MLC+elev.	Dry	0.57	0.88	0.69	0.58
	Flood	0.74	0.34	0.47	
DT+LP	Dry	0.53	0.99	0.69	0.45
	Flood	0.97	0.12	0.21	
RF+LP	Dry	0.54	0.99	0.70	0.48
	Flood	0.98	0.15	0.26	
MLC+LP	Dry	0.73	0.95	0.83	0.80
	Flood	0.93	0.66	0.77	
MRF	Dry	0.72	0.97	0.83	0.79
	Flood	0.96	0.63	0.76	
TSVM	Dry	0.75	0.85	0.80	0.78
	Flood	0.82	0.72	0.77	
U-Net	Dry	0.53	0.33	0.40	0.50
	Flood	0.52	0.72	0.60	
HMT	Dry	0.58	0.96	0.72	0.60
	Flood	0.89	0.31	0.46	
HMCT-UC	Dry	0.99	0.88	0.93	0.93
	Flood	0.89	0.99	0.94	
HMCT-C	Dry	0.99	0.89	0.94	0.94
	Flood	0.90	0.99	0.94	

random field. TSVM and deep learning method (U-Net) did not perform well on this data due to not modeling large scale spatial structure. HMT performed poorly with very low precision for the dry class and low recall for the flood class, meaning that it misclassified a significant amount of flood class samples into the dry class. Further investigation showed that many large scale obstacles (tree canopies with dry class features) in the flood area biased the learning and inference of HMT model. It seems that HMT is not robust to large obstacles in features. In contrast, HMCT models performed significantly better. The reason was that HMCT model flow dependency of locations on two sides of the river in separate tree branches (conditionally independent from each other), mitigating the impact of feature obstacles on model learning and inference.

Results on the Grimesland dataset were summarized in Table 3. In this dataset, training samples were drawn far away from the test region to evaluation model generalizability. We can see that most non-spatial classifiers performed poorly due to overfitting, except for MLC (overall F-score around 0.77) due to its simplicity. Adding elevation features improved DT and RF but degraded MLC. This is because training area was overall higher than the entire test region. Thus, models learned from training area were not applicable to test samples. Post-processing also had mixed effect on non-spatial classifiers, because it sometimes mistakenly smoothed out corrected classified samples. TSVM outperformed several non-spatial classifier due to leveraging test sample features. U-Net performed poorly due to overfitting. HMT again performed poorly due to the same reason as in Greensville dataset. HMCT outperformed others with an overall F-score of 0.94. HMCT showed better generalizability because its flow dependency was based on relative elevation values in test region. In HMCT, training samples were only used for providing a reasonably well initial model parameters $\{\boldsymbol{\mu}_c, \Sigma_c | c = 1, 2\}$.

Sensitivity of HMCT to initial parameters: We conducted sensitivity of our HMCT (with node collapsing) model to different initial parameter values on prior class probability π and class transitional probability ρ (the parameters of $\{\boldsymbol{\mu}_c, \Sigma_c | c = 1, 2\}$ were initialized based on maximum likelihood estimation on the training set). Due to space limit, we only showed results on the Greensville dataset. First, we fixed initial $\rho = 0.99$ and varied initial π from

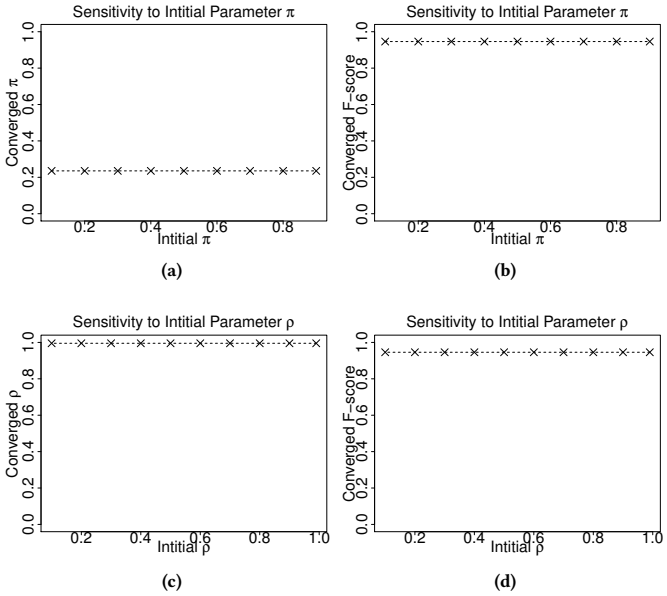


Figure 5: Sensitivity of HMCT to initial parameters π and ρ

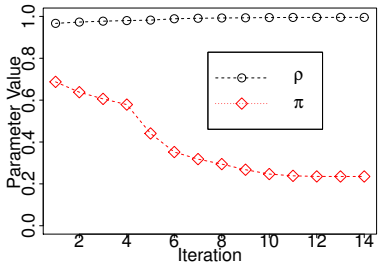


Figure 6: Parameter iterations and convergence in HMCT

0.1 to 0.9. Results of converged values of ρ together with their corresponding final F-scores were shown in Figure 5(a-b). It can be seen that our HMCT model was quite stable with different initial π values. Similarly, we fixed initial $\pi = 0.5$, and varied initial ρ from 0.2, 0.3, to 0.99. Results in Figure 5(c-d) showed the same trend. In practice, we can select an initial π value around 0.5 and a relatively high initial ρ value such as 0.9 (because *flood* pixels' neighbor is more likely to be *flood* due to spatial autocorrelation).

Parameter iterations and convergence in HMCT: Here we fixed the initial $\pi = 0.5$ and initial $\rho = 0.99$, and measured the parameter iterations and convergence of HMCT (with node collapsing) on the Greensville dataset. Our convergence threshold was set 0.001%. The values of π and ρ at each iteration were summarized in Figure 6 (we omitted $\boldsymbol{\mu}_c, \Sigma_c$ due to the large number of variables. The parameters converged after 14 iterations. The converged value of ρ was close to 1 as expected.

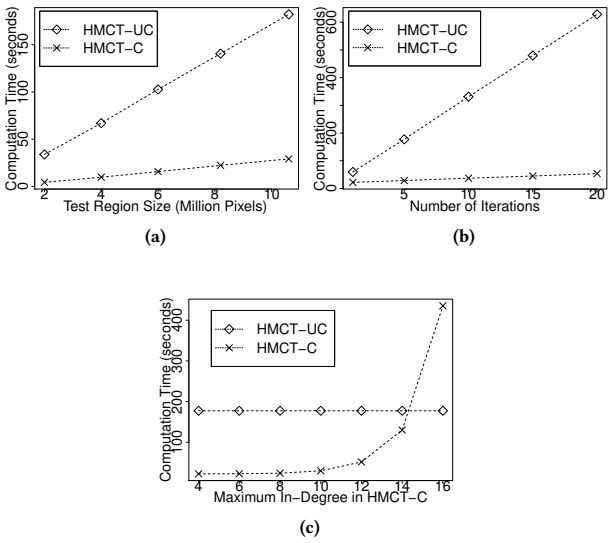


Figure 7: Computational performance of HMCT-C versus HMCT-UC on varying test region sizes (a), number of iterations (b), and maximum node in-degree (c)

4.2 Computational Performance Comparison

We evaluated the computational performance of the proposed HMCT model. We particularly compared the time costs of HMCT-UC (without node collapsing) and HMCT-C to understand the effect of node collapsing and pre-aggregation. The time cost of HMT should be very close to HMCT-UC since they both have the same number of tree nodes and same set of node operations in learning and inference. Thus, we did not include the cost of HMT to avoid redundancy. For HMCT-C, we further controlled the maximum in-degree (the number of parents) for each tree node. This was done simply by stopping the collapsing operations on a node when its in-degree reached the maximum. We compared the two methods on different input data sizes, different number of iterations in parameter learning, and different settings of maximal in-degree per node (in HMCT-C only).

Results were summarized in Figure 7. We fixed input parameters $\rho = 0.99$ and $\pi = 0.3$. First, we chose the number of iterations as 5 and the maximum allowed in-degree in HMCT-C as 5. We varied the size of the test region from around 2 million pixels to around 10.6 million pixels. Results in Figure 7(a) showed that the time costs of both HMCT-UC and HMCT-C increased almost linearly with the test region size. The rate of increase in HMCT-C was smaller. This was consistent with the $O(N_T)$ term in our cost model. Note that N_T is much smaller in HMCT-C due to node collapsing. HMCT-C ran very fast, costing less than 30 seconds on over 10 million pixels. In the second experiment, we chose a test region with around 10.6 million pixels, and set the maximum allowed in-degree in HMCT-C as 10. We varied the number of iterations in parameter learning from 1 to 20. Results in Figure 7(b) showed that the costs of both methods increased almost linearly with the number of iterations. This is consistent with our cost model. In the third experiment, we chose

Table 4: Time costs of individual components (seconds)

	HMCT-UC	HMCT-C	HMCT-C
Max in-degree	N/A	4	16
Tree construction	13.47	13.47	13.47
Node Collapsing	N/A	6.64	6.64
Pre-aggregation	N/A	0.28	0.28
Parameter learning	152.51	1.80	394.01
Class inference	10.71	0.21	21.09
Total time	176.69	22.41	435.49

a test region with around 10.6 million pixels, and set the number of iterations as 5. We varied the maximum allowed in-degree in HMCT-C from 4 to 16 (we did not choose values smaller than 4 because the maximum in-degree of uncollapsed tree nodes is already 4 due to the four-neighborhood we used). Results in Figure 7(c) showed that as the maximum allowed in-degree increases, the cost of HMCT-C grew exponentially, exceeding the cost of HMCT-UC when the maximum in-degree was 16. This is consistent with the 2^d term in our cost model.

Table 4 showed detailed time costs for individual components in HMCT algorithms. We chose an input data with a test region of 10.6 million pixels, and the number of iterations as 5, and a max in-degrees of 4 and 16 for HMCT respectively. From the table, we can see that when the max in-degree is 4, learning and inference in HMCT-C were significantly faster than HMCT-UC due to node collapsing. However, when the max in-degree is 16, the costs of HMCT-C was almost twice as that of HMCT-UC, due to the exponential cost in terms of in-degree (2^d). In practice, we can set the maximum allowed in-degree as 4.

5 CONCLUSIONS AND FUTURE WORKS

In this paper, we propose hidden Markov contour tree (HMCT), a spatial structured model that can capture flow dependency on a potential field surface. We also propose efficient algorithms for model parameter learning and class inference. Evaluations on real world data show that our HMCT algorithms are scalable to large data sizes, and can achieve higher classification performance over existing methods for hydrological applications such as flood mapping.

In future work, we plan to explore integration of deep learning framework with our HMCT.

ACKNOWLEDGEMENT

This material is based upon work supported by the Alabama Transportation Institute.

REFERENCES

[1] Luc Anselin. 2013. *Spatial econometrics: methods and models*. Vol. 4. Springer Science & Business Media.
[2] Sudipto Banerjee, Bradley P Carlin, and Alan E Gelfand. 2014. *Hierarchical modeling and analysis for spatial data*. CRC Press.
[3] PA Brivio, R Colombo, M Maggi, and R Tomasoni. 2002. Integration of remote sensing data and GIS for accurate mapping of flooded areas. *International Journal of Remote Sensing* 23, 3 (2002), 429–441.
[4] Hamish Carr, Jack Snoeyink, and Ulrike Axen. 2003. Computing contour trees in all dimensions. *Computational Geometry* 24, 2 (2003), 75–94.
[5] Don Cline. 2009. *Integrated Water Resources Science and Services: an Integrated and Adaptive Roadmap for Operational Implementation*. Technical Report. National Oceanic and Atmospheric Administration.
[6] Herbert Edelsbrunner and John Harer. 2010. *Computational topology: an introduction*. American Mathematical Soc.
[7] David Günther, Roberto A Boto, Juila Contreras-Garcia, Jean-Philip Piquemal, and Julien Tierny. 2014. Characterizing molecular interactions in chemical systems. *IEEE transactions on visualization and computer graphics* 20, 12 (2014), 2476–2485.
[8] Zhe Jiang. 2018. A Survey on Spatial Prediction Methods. *IEEE Transactions on Knowledge and Data Engineering* (2018).
[9] T. Joachims. 1999. Making large-Scale SVM Learning Practical. In *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola (Eds.). MIT Press, Cambridge, MA, Chapter 11, 169–184.
[10] Anuj Karpatne, Ankush Khandelwal, Shyam Boriah, and Vipin Kumar. 2014. Predictive Learning in the Presence of Heterogeneity and Limited Training Data. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*. 253–261.
[11] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory* 47, 2 (2001), 498–519.
[12] Alexander Kuhn, Wito Engelke, Markus Flatken, Hans-Christian Hege, and Ingrid Hotz. 2015. Topology-based analysis for multimodal atmospheric data of volcano eruptions. In *Topological Methods in Data Analysis and Visualization*. Springer, 35–50.
[13] John Lafferty, Andrew McCallum, Fernando Pereira, et al. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning (ICML), IEEE*, Vol. 1. 282–289.
[14] James R Munkres. 2000. *Topology*. Pearson.
[15] National Oceanic and Atmospheric Administration. [n.d.]. Data and Imagery from NOAA's National Geodetic Survey. https://www.ngs.noaa.gov.
[16] National Oceanic and Atmospheric Administration. 2018. National Water Model: Improving NOAA's Water Prediction Services. http://water.noaa.gov/documents/wrn-national-water-model.pdf.
[17] NCSU Libraries. 2018. LIDAR Based Elevation Data for North Carolina. https://www.lib.ncsu.edu/gis/elevation.
[18] Valerio Pascucci, Xavier Tricoche, Hans Hagen, and Julien Tierny. 2010. *Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications*. Springer Science & Business Media.
[19] Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.
[20] O Ronen, JR Rohlicek, and M Ostendorf. 1995. Parameter estimation of dependence tree models using the EM algorithm. *IEEE Signal Processing Letters* 2, 8 (1995), 157–159.
[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.
[22] Jonathan Stoeckel and Glenn Fung. 2005. SVM Feature Selection for Classification of SPECT Images of Alzheimer's Disease Using Spatial Information. In *ICDM*. 410–417.
[23] Karthik Subbian and Arindam Banerjee. 2013. Climate Multi-model Regression Using Spatial Smoothing. In *SIAM International Conference on Data Mining (SDM)*. 324–332.
[24] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. 2006. A comparative study of energy minimization methods for markov random fields. In *European conference on computer vision*. Springer, 16–29.
[25] The Middlebury Computer Vision Pages. 2018. C++ Source Code of MRF. http://vision.middlebury.edu/MRF/code/.
[26] Waldo R Tobler. 1970. A computer movie simulating urban growth in the Detroit region. *Economic geography* 46, sup1 (1970), 234–240.
[27] Philip A Viton. 2010. Notes on spatial econometric models. *City and regional planning* 870, 03 (2010), 9–10.
[28] Hongjian Wang and Zhenhui Li. 2017. Region representation learning via mobility flow. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 237–246.
[29] Miao Xie, Zhe Jiang, and Arpan Man Sainju. 2018. Geographical Hidden Markov Tree for Flood Extent Mapping. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. ACM, 2545–2554. https://doi.org/10.1145/3219819.3220053
[30] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Zhenhui Li. 2018. Deep Multi-View Spatial-Temporal Network for Taxi Demand Prediction. In *2018 AAAI Conference on Artificial Intelligence (AAAI'18)*.
[31] Zijun Yao, Yanjie Fu, Bin Liu, Wangsu Hu, and Hui Xiong. 2018. Representing Urban Functions through Zone Embedding with Human Mobility Patterns.. In *IJCAI*. 3919–3925.
[32] Xiaojin Zhu and Zoubin Ghahramani. 2002. Learning from labeled and unlabeled data with label propagation. (2002).