Classification Error Correction: A Case Study in Brain-Computer Interfacing

Hasan A. Poonawala, Mohammed Alshiekh, Scott Niekum and Ufuk Topcu

***Abstract*—Classification techniques are useful for processing complex signals into labels with semantic value. For example, they can be used to interpret brain signals generated by humans corresponding to a choice of one element out of a known finite set. However, the classifier may assign an element out of the finite set that is different from the intended one. This error in classification can lead to poor performance in tasks where the class labels are used to learn some information or to control a physical device. We propose a computationally efficient algorithm to identify which class labels may be misclassified out of a sequence of class labels, when these labels are used in a given learning or control task. The algorithm is based on inference methods using Markov random fields. We apply the algorithm to goal-learning and tracking using brain-computer interfacing (BCI), in which signals from the brain are commonly processed using classification techniques. We demonstrate that using the proposed algorithm reduces the time taken to identify the goal state in real-time control experiments.**

I. Introduction

In most dynamical systems, the output of the system is a vector of real numbers obtained through sensors. The sensed output is used in estimation and feedback control techniques for achieving various tasks, such as set-point regulation or trajectory tracking. If the sensor is noisy, the noise usually takes the form of an additive numerical term with zero mean. In some systems, the output is obtained using a classification process [1], and is a member of a discrete set instead of a continuous value. The noise in classification processes results in a feature or data sample being assigned a label different from the true label that should have been assigned to the feature. This type of error is different from noise in systems with continuous numeric outputs, since real numbers have a well-known ordering relation but a finite set of labels has no *a priori* order.

An example where classifier outputs are used in a control task arises in the shared control of semi-autonomous devices using brain-computer interfaces (BCIs) [2]. Brain-computer interface technology will enable increased effectiveness of human-machine interaction in general. Current research in BCIs is largely motivated by human-machine interaction in the context of rehabilitation devices.

Hasan A. Poonawala and Mohammed Alshiekh are with the Institute for Computational Engineering and Science, University of texas at Austin, Austin, TX 78712, USA. hasanp@utexas.edu, malshiekh@utexas.edu

Scott Niekum is with the Department of Computer Science, University of texas at Austin, Austin, TX 78712, USA. sniekum@cs.utexas.edu

Ufuk Topcu is with the Department of Aerospace Engineering, University of texas at Austin, Austin, TX 78712, USA. utopcu@utexas.edu

The human nervous system generates signals which can be recorded using techniques such as electroencephalography (EEG) or electromyography (EMG). A signicant number of BCI approaches extract signals from the human brain by recording brain activity in the form of EEG signals through the human scalp [3]–[6]. Using only EEG has the benefit of requiring the user to wear only one piece of headgear to allow interaction with the device. However, extracting meaningful signals using EEG is challenging [2]. Determination of the user's intention is achieved either by training the user to generate a fixed set of signals, or using machine learning techniques to classify recorded EEG signals [3], [5], [6].

A common task for such devices is to reach a goal state known by the user but unknown to the device, using only classified brain signals as available feedback [3]–[6]. The user can either generate brain signals corresponding to control actions, or perform the role of an expert who evaluates the actions selected by the device. Either way, the information that the human wants to transmit is extracted as complex and noisy EEGs signals in many situations, and a classifier may be required to interpret this information. A technique for learning which goal is intended by the user is presented in [5]. However, no technique to infer when misclassification occurs is attempted. The presence of misclassified commands or evaluations in the goal-learning task results in longer times taken to reach the goal. The increase in time taken to reach the goal can be due to two reasons. The first reason is that the BCI system may move *away* from the goal when it misclassifies the intended action. The second reason is that most estimation algorithms assume that the user is behaving ideally or optimally, and the misclassification becomes misleading evidence for such algorithms.

Contributions

In this paper, we propose methods to improve upon the algorithms proposed in [7]. As mentioned in [7], the algorithm therein contains a parameter β which is set arbitrarily. We propose a technique to tune the this parameter automatically in real time in order to improve estimation. It was observed that the algorithm in [7] did not perform well when particular temporal sequences of misclassification occurred in a trial. This limitation lead to poor worst-case performance. In this paper, we also propose a modification of the method in [7] that improves the worst-case performance. Through simulations and real-time experiments, we demonstrate that using the proposed algorithm to identify classification errors results in lower times taken to estimate the unknown goal intended to be reached by the user.

II. Background

A. Finite Transition Systems

A finite transition system (FTS) [8] consists of a tuple (S, A, T) where

- S is a finite set of states,
- A is a finite set of actions, and
- $T : S \times A \to S$ is a transition function.

The evolution of a FTS occurs in discrete time. At every time step $t \in \mathbb{N}$, the state is $s(t) \in S$, and an action $a(t) \in A$ is selected which results in a new state determined by $T(s(t), a(t))$.

B. Classifiers

A classifier C consists of a tuple (F, L, C^*) where

- F is a set of features,
- L is a finite set of class labels, and
- C^* is a map that assigns a label to a feature $f \in F$.

The classifier C can be associated with a confusion matrix R that models the imperfection of the classification process. The matrix R is obtained during testing of the classifier. If the classifier is perfect, then R assigns probability 1 to the true class label corresponding to a feature $f \in F$.

C. Markov Random Fields

A Markov random field (MRF) [9] is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of nodes, and $\mathcal{E}$ is a set of undirected edges in $\mathcal{V} \times \mathcal{V}$. Furthermore, each node $i \in \mathcal{V}$ is associated with a random variable x_i. We will identify the random variable x_i for a node as the node itself, in order to simplify the exposition.

The main property of a MRF is that the edge structure $\mathcal{E}$ determines the conditional independence of the nodes. That is, a node is conditionally independent of all nodes that are not its neighbors:

$$p\left(x_i \colon \{x_j\}_{(j)\in\mathcal{V}\setminus\{i\}}\right) = p\left(x_i \colon \{x_j\}_{(i,j)\in\mathcal{E}}\right). \tag{1}$$

The random variables x_i for all the nodes together for the random variable $\mathbf{x}$. The joint probabilitiy distribution of the nodes is represented by $p(\mathbf{x})$. In a discrete MRF, the random variables x_i can take on discrete values from a set. A sample of the MRF generated by $p(\mathbf{x})$ consists of an assignment of one of the discrete values to each node in $\mathcal{V}$. Therefore, we will refer to such a sample as an assignment of the MRF, or simply, an assignment.

III. Goal Learning through Brain-Computer Interfaces

The shared control of devices using brain-computer interfaces can be modeled as a classifier-in-the-loop learning and control problem. One of the common tasks to be achieved by such devices is for the state of the system to reach a desired goal state $g \in S$ determined by the human user [3]–[6], where S is the set of states that can be reached by the device.

We can model the dynamics of the device as a FTS. In this section, we will focus on a finite state space arranged in a one dimensional grid as seen below. It turns out that the case of multi-dimensional state spaces can be treated using the method for 1D spaces, which is shown in [7]. The action space A of the device is $\{a_0, a_1, a_2\}$ corresponding to moving to the left, staying in the same state, and moving to the right.

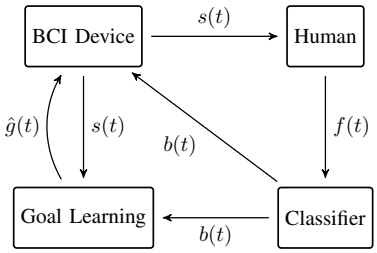
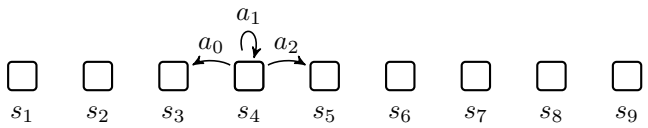


Fig. 1: Human and BCI system together as a classifier-in-the-loop system. Once the goal learning is complete, the BCI system can take over control from the human.



The transition function T of this device is given by three simple rules:

$$T(s_i, a_2) = s_{i+1}, \tag{2}$$
$$T(s_i, a_0) = s_{i-1}, \text{ and} \tag{3}$$
$$T(s_i, a_1) = s_i. \tag{4}$$

Of course, we must impose the end cases $T(s_1, a_0) = s_1$ and $T(s_{|S|}, a_2) = s_{|S|}$.

The state of the system $s(t)$ at time t is represented by a cursor. At each time step t, an action $a(t) \in \{a_0, a_1, a_2\}$ is chosen by the human. The intended action $a(t)$ to be taken in state $s(t)$ is obtained as an EEG-based feature $f(t)$ (see Figure 1). The measured EEG signal $f(t)$ is classified as $b(t)$, using a classifier C^* obtained by prior training. However, there is a non-zero probability that $a(t) \neq b(t)$. The probabilities of making mistakes is captured as the confusion matrix R for classifier C, and is obtained during testing. Therefore, the device is controlled via a classifier $C = (F, L, C^*)$, where F is the set of EEG features that can be obtained, $L = A$, and C has confusion matrix R associated with it. We refer to the pair $(s(t), b(t))$ as the state-label at time t. The elements $s(t)$ and $b(t)$ are referred to as the state and label at time t respectively. We can now state the goal learning problem.

Goal Learning problem: Given a sequence M_N of state-label pairs $(s(t), b(t))_t$ indexed by $t \in \{1, 2, \ldots, N\}$, the task for the device being controlled is to estimate the unknown goal state g selected by the agent.

A common algorithm for estimating the goal given potentially erroneous recordings of optimal actions is described in Algorithm 1. The estimate $\hat{g}(t) \in S$ of the goal g at any time t is represented by a probability mass-like function P_t defined on the finite state space S. Note that $g = s_j$ for some $j \in \{1, 2, \ldots, |S|\}$. The values of $P_t(s_i)$ for all $s_i \in S$ are updated after every action is received. The update takes the

Algorithm 1 Inverse Planning for Goal Learning

Input: M_N, $P_0(s)$, δ
Output: $\hat{g}$, t_f {Time at which goal is identified}
  goal $\leftarrow$ `false`
  $i \leftarrow 0$
  while not goal **and** $i \leq N$ **do**
    $i \leftarrow i + 1$
    $\eta_i \leftarrow 0$
    for all $s \in S$ **do**
      $P_i(s) \leftarrow k_i(s) P_{i-1}(s)$
      $\eta_i \leftarrow \eta_i + P_i(s)$
    end for
    for all $s \in S$ **do**
      $P_i(s) \leftarrow P_i(s)/\eta_i$
    end for
    $\hat{g} \leftarrow \{s : P_t(s) \geq \delta\}$
  end while
  $t_f \leftarrow i$
  return $\hat{g}$, t_f

	$i < j$	$i = j$	$i > j$
$b(t) = a_0$	k_H	k_L	k_L
$b(t) = a_1$	k_L	k_H	k_L
$b(t) = a_2$	k_L	k_L	k_H

TABLE I: Look-up table for determining $k_t(s_i)$ at time t, when $s(t) = s_j$.

form

$$P_{t+1}(s_i) = \frac{k_t(s_i)}{\eta_t} P_t(s_i), \tag{5}$$

where $s_i \in S$, and η_t is a normalizing factor given by $\eta_t = \sum_{s_i \in S} k_t(s_i) P_t(s_i)$. The term $k_t(s_i)$ acts as a multiplicative update factor of $P_t(s_i)$. The update factor $k_t(s_i)$ can only be one of the elements of $\{k_L, k_H\}$ for each $s_i \in S$, where $k_H > k_L > 0$. Let $s(t) = s_j$. Given $b(t)$, the update factor for state s_i at time t can be determined from Table I. Once a state $s_i \in S$ achieves a value of $P_t(s_i)$ greater than a threshold $\delta \in [0, 1]$, it is taken to be the goal state. If $\delta > 0.5$, then only one state can achieve this value at any time step. The time t_f at which the goal is identified is taken as a measure of performance for Algorithm 1.

If actions are perfectly classified, then $P_t(g)$ will monotonically increase and become larger than the threshold δ, and therefore g will always be identified as the goal state given enough time steps. However, due to misclassification, $P_t(g)$ may not always increase. Even if $P_t(g)$ eventually crosses the threshold, it will take more time steps to do so when compared to the classification error-free case. To prevent this increase in time taken, we want to identify which actions have been misclassified. This leads to the following problem.

Inference problem: Given a sequence M_N of state-label pairs $(s(t), b(t))_t$ indexed by $t \in \{1, 2, \ldots, N\}$ associated with a goal-learning task, estimate the sequence $(s(t), a(t))_t$ corresponding to the true actions commanded at $t \in \{1, 2, \ldots, N\}$.

The motivation of the work presented here and in [7] is that solving the inference problem and using the estimated sequence in Algorithm 1 will improve the latter's performance by decreasing t_f.

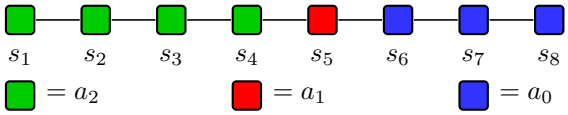


Fig. 2: Optimal actions to be taken in each state in order to reach the goal s_5 and stay there.

IV. Inference Using Markov Random Fields

Assume that a class label $x_i \in L$ is intended to be communicated through a classifier, by generating an input feature f_i. The classifier assigns the class label y_i to the feature f_i generated in order to communicate x_i, and it may occur that $y_i \neq x_i$. The class label assigned to a feature f_i can be treated as a noisy measurement of the true class label. The measurements y_i obtained at different time steps can be combined to obtain the set of measurements $\mathbf{y}$. Since the true class labels $\mathbf{x}$ cannot be observed directly, they are treated as random variables. Our goal is to estimate the most likely class labels $\hat{\mathbf{x}}$ given the measured class labels $\mathbf{y}$. The motivation for this is the premise that using $\hat{\mathbf{x}}$ instead of $\mathbf{y}$ in Algorithm 1

The posterior distribution $p(\mathbf{x}|\mathbf{y})$ can be used to compute $\hat{\mathbf{x}}$ using bayesian inference. The likelihood function $p(\mathbf{y}|\mathbf{x})$ is determined by the performance of the classifier, in other words, the error rates in classification. The prior distribution $p(\mathbf{x})$ needs to be determined.

The main insight used in [7] was that the optimal action to be taken in a state s located in a 1D grid only depends on whether the goal state is to the right or the left of state s (see Figure 2). Thus, if the classifier was perfect, we are more likely to observe the same actions commanded in adjacent states. This observation led to proposing that Markov random fields (MRFs) can be used to model a prior distribution on $\mathbf{x}$ that encodes this expected similarity of actions in adjacent states. Moreover, an MRF that encodes the property that assignments $\mathbf{x}$ with similar labels in adjacent nodes have higher probability admits efficient inference algorithms. This statement will be explained in detail below.

The Hammersley-Clifford Theorem [10] states that the prior distribution $p(\mathbf{x})$ of an MRF can be expressed as the (normalized) exponential of the sum of energy functions defined over the cliques of the graph. A clique of a graph is a subset of nodes of the graph in which all nodes in the subset are connected to all other nodes in the subset. Let $\mathcal{C}$ be the cliques in the MRF. Let $\mathbf{x_c}$ be the set of random variables corresponding to nodes in $c \in \mathcal{C}$. Then, $p(\mathbf{x}) = \frac{1}{Z} \exp\left(-\sum_{c \in \mathcal{C}} V_c(\mathbf{x_c})\right)$, where V_c is the clique energy function, and Z is an appropriate normalizing factor.

The likelihood function $p(\mathbf{y}|\mathbf{x})$ is represented through the energy function D for a single node given by

$$D(y_i, x_i) = -\log p(y_i|x_i) \tag{6}$$

where $D(y_i, x_i) \geq 0$ when $p(y_i|x_i) > 0$. Then,

$$p(\mathbf{y}|\mathbf{x}) = \exp\left(-\sum_{i \in V} D(y_i, x_i)\right) \tag{7}$$

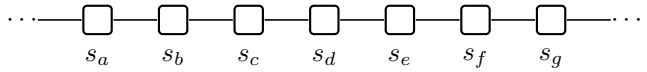


Fig. 3: The grid graph structure of the MRF with each node corresponding to a state-label. The nodes are ordered based on the state of each state-label pair, meaning that the integer indices of the states are such that $a \leq b \leq c \leq d \leq \ldots \leq g$. Equality of the indices occurs when a state is visited multiple times.

The MAP estimate is obtained from the posterior distribution as

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}). \tag{8}$$

Since $p(\mathbf{y}|\mathbf{x})$ and $p(\mathbf{x})$ can be represented by energy functions D and V_c, computing $\hat{\mathbf{x}}$ is equivalent to the following energy-minimization problem (see [9] for details):

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \sum_{i=1}^{n} D(y_i, x_i) + \sum_{c \in \mathcal{C}} V_c(\mathbf{x}_c). \tag{9}$$

If L contains only two labels, then the global minimum of the right hand side of (9) can be found exactly in polynomial time [11], [12], provided V_c is submodular [11]. The energy function for cliques with two nodes is submodular if for any $a, b \in L$,

$$V_c(a,a) + V_c(b,b) \leq V_c(a,b) + V_c(b,a). \tag{10}$$

Informally, an energy function V_c is submodular when arguments that are more similar are assigned smaller energy. The submodularity of V_c implies that $p(\mathbf{x})$ is higher when connected labels are similar. Since this property is true in our problem, we can hope to develop *efficient* inference algorithms. Computing $\hat{\mathbf{x}}$ becomes equivalent to constructing a graph $\mathcal{G}_{min}$ with edge weights determined by the energy functions, and obtaining the minimum-cut/maximum-flow solution determined by this graph [11], [12]. When L contains more than two labels, the global minimum cannot be found as easily. However, a local minimum within a known factor of the the global minimum can be found efficiently [13].

The properties of MRFs mentioned above that permit efficient inference algorithms will be used in the design of a MRF from a sequence of state-label pairs. In the next section, we will present different ways to define the MRF used in inference, including the one presented in [7].

V. Algorithms

In order to specify a MRF, we must provide the nodes, the edges, and the energy functions associated with each edge and node. Different choices of the connectivity and energy functions leads to different prior distributions on the true labels corresponding to the measured ones. We first mention some choices for the connectivity and energy functions that are common to all the algorithms we will consider later.

We will restrict ourselves to the design of a MRF for a one-dimensional state space S. As mentioned in [7], any sequence of state-label pairs in a n-dimensional grid can be reduced to a n sequences, each in a different one-dimensional grid. The procedure in this paper can be applied to these n inference problems in order to solve the goal-learning problem in the original n-dimensional grid. Therefore, this restriction is only artificial and makes the exposition easier.

Each node in the MRF $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ will correspond to a state-label pair $(s(t), b(t))_t$ for some $t \in \mathbb{N}$. The connectivity of the MRF is guided by the fact that the nodes are associated with states in a 1D grid. We connect the nodes to form a chain graph, that is, a connected graph with N nodes and $N-1$ edges (see Figure 3). The rationale for this is two-fold. First, this allows us to mimic the spatial relationships of the state-space S that the nodes are associated with. Second, the cliques of this chain graph consist of pairs of adjacent nodes. A consequence of this is that there are only $N-1$ cliques in a graph, and each one can be associated with a unique edge in the MRF. Furthermore, the clique energy functions are functions of two labels only.

We can identify the nodes in $\mathcal{V}$ with the first N natural numbers. Moreover, node i is connected to node $i+1$, where $i \in \{1, \ldots N-1\}$. Each state-label pair $t \in M_N$ obtained at some time instant $t \in \{1, \ldots, N\}$ is associated to a unique node in the MRF. This association effectively defines a permutation π from the set of time instants $t = \{1, \ldots, N\}$ (that indexes the sequence M_N) to the set of nodes $\mathcal{V} = \{1, \ldots, N\}$.

The permutation π depends on the state and action in each time step. In order to describe any permutation π precisely, we must introduce some functions that output the state and action associated with both a time $t \in M_N$ and a node $i \in \mathcal{V}$. Let $(s_i, a_j)_k$ be a state-label pair in M_N which is associated with node $l \in \mathcal{V}$. The functions $\pi_s \colon M_N \to \{1, \ldots, |S|\}$, $\pi_a \colon M_N \to \{1, \ldots, |A|\}$, and $\pi \colon M_N \to \mathcal{V}$ are then given by

$$\pi(k) = l,\ \pi_s(k) = i,\ \pi_a(k) = j \tag{11}$$

Similar functions can be defined with $\mathcal{V}$ as their domain instead of M_N:

$$\pi_s^{-1}(l) = \pi_s \circ \pi^{-1}(l) = i \tag{12}$$

$$\pi_a^{-1}(l) = \pi_a \circ \pi^{-1}(l) = j \tag{13}$$

A. Connectivity

The nodes in the MRF are always connected to form a chain graph. The nodes associated with the state-label pairs at two consecutive time instants may or may not be neighbors in this graph. The connectivity between state-label pairs is effectively determined by the ordering of the state-label pairs. Different methods for ordering the sequence of state-label pairs in a trial result in different MRFs and therefore different inference algorithms.

We always order the state-label pairs based on the state. Formally, If $\pi_s(i) < \pi_s(j)$ for any $i, j \in M_N$, then $\pi(i) < \pi(j)$. The differences lie in how state-label pairs with the same state (that is $\pi_s(i) = \pi_s(j)$) are ordered.

1) Temporal:

$$\pi_s(i) = \pi_s(j) \text{ and } i < j \Rightarrow \pi(i) < \pi(j) \tag{14}$$

The nodes with the same state are ordered based on time. This ordering was used in [7]. It was observed that if too many consecutive errors occurred when in the same state, then the

Algorithm 2 buildMRF

Input: M_N, order
Output: $\mathcal{G}$,y

```
if order = temporal then
  π ← sort(M_N, π_s, id)
end if
if order = order then
  π ← sort(M_N, π_s, π_a)
end if
if order = randomized then
  π ← sort(M_N, π_s, random)
end if
for i = 1 to N do
  add i to V
  y_i ← a_{π_a^{-1}(i)}
  if i > 1 then
    add (i, i+1) to E
  end if
end for
return G, y
```

inference was of poor quality. To overcome this, we propose the orderings in the next two subsections.

2) Ordered: We order the state-label pairs with the same value of π_s based on the value of π_a. That is, ties in the ordering based on π_s are resolved using π_a.

$$\pi_s(i) = \pi_s(j) \text{ and } \pi_a(i) < \pi_a(j) \Rightarrow \pi(i) > \pi(j) \quad (15)$$

Note that if the values of two state-label pairs under both π_s and π_a are identical (the same state and action in both time instants), then interchanging the nodes corresponding to these two state-label pairs will not change the estimate $\hat{\mathbf{x}}$. Therefore, no further step is necessary to resolve ties under π_a.

3) Randomized: We first order the nodes using (15). For state-label pairs with the same value of π_a, we randomly permute their order.

The MRF can be constructed using one of the methods above to determine the permutation π. Algorithm 2 describes this procedure. Note that the function sort(X,Y,X) sorts a list X based on criterion Y with ties resolved by criterion Z.

B. Binary Energy Function

The role of the binary energy function is to control the likelihood that adjacent nodes have similar states. Since we wish to increase that likelihood, we can choose the following function:

$$V_c(x_i, x_j) = \begin{cases} \beta & \text{if } x_i \neq x_j \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

where $\beta > 0$ is a parameter.

The binary energy function (16) was used in [7] for a fixed value of $\beta = 1.0$. There were two shortcomings to the energy function (16):

i) Two nodes i and j may be assigned different labels $\hat{x}_i$ and $\hat{x}_j$ even though $\pi_s^{-1}(i) = \pi_s^{-1}(j)$.

ii) The choice of β was made empirically

The first statement above simply says that the inference algorithm predicts that two different actions were intended by the human in the same state. However, this is impossible in a one dimensional grid with three actions. In order to capture this feature in the prior distribution, we distinguish between the case where nodes i and j are associated with the same states and the case when they are associated with different states. The binary energy function (16) can be modified to

$$V_c(x_i, x_j) = \begin{cases} \beta_{ss}\beta & \text{if } x_i \neq x_j \text{ and } \pi_s^{-1}(i) = \pi_s^{-1}(j) \\ \beta & \text{if } x_i \neq x_j \text{ and } \pi_s^{-1}(i) \neq \pi_s^{-1}(j) \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

where $\beta_{ss} > 1$ is another parameter. Since the inference is equivalent to energy minimization, choosing a large value of β_{ss} will result in higher energy for assignments where nodes associated with the same state are given different labels. This high energy makes it more likely that nodes associated with identical states are estimated to correspond to the *same* true action/label.

C. Variable β

In order to address the second issue, we describe the effect of the parameter β on the MAP estimate $\hat{\mathbf{x}}$. The possible solutions for $\hat{\mathbf{x}}$ represent a trade-off between obtaining MAP estimates where $\hat{\mathbf{x}}$ is similar to $\mathbf{y}$ and obtaining MAP estimates where the adjacent evaluations are assigned similar labels. For the case when there are two labels only so that the inference problem can be solved exactly, one can observe a phase transition in the nature of the MAP estimate as the parameter β is varied. For $\beta = 0$, the MAP estimate $\hat{\mathbf{x}}$ is identical to the measured labels $\mathbf{y}$. As β increases, at a certain value the MAP estimate changes to one where there are two connected regions of the same label separated by a single edge. At another larger value of β, all the labels of the MAP estimate become identical.

These sudden changes at specific values of β are termed as *phase transitions* [10]. The phenomenon of phase transitions in the estimate $\hat{\mathbf{x}}$ with increasing value of β can be used in designing an algorithm that allows us to choose from multiple possible values for β.

We can use the value of $\pi_a^{-1}(i)$ for all $i \in \mathcal{V}$ to identify when a phase transition has occurred by defining functions over $\hat{\mathbf{x}}$. Consider the functions h_1 and h_2 as

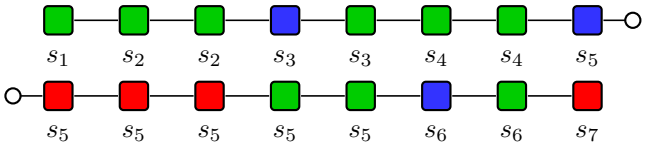
$$h_1 = \sum_{i=1}^{N-1} \pi_a^{-1}(i+1) - \pi_a^{-1}(i) \quad (18a)$$

$$h_2 = \sum_{i=1}^{N-1} |\pi_a^{-1}(i+1) - \pi_a^{-1}(i)| \quad (18b)$$

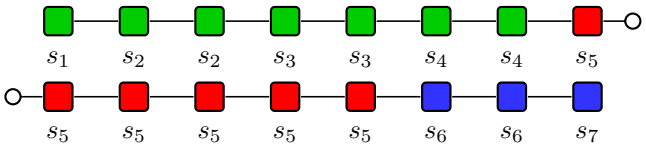
These functions are defined based on the knowledge that the correct label in a state will be a_2 for states to the left of the goal, a_1 for the goal, and a_0 for states to the right. Therefore, if the classifier is perfect, the sequence defined by the estimates $\hat{x}_i$ for $i = \{1, \ldots, N\}$ will be such that $\pi_a^{-1}(i) \geq \pi_a^{-1}(i+1)$

Algorithm 3 Inference

Input: M_N, order, R, β_0, $\delta\beta$, β_{ss}
Output: $\hat{\mathbf{x}}$
  $\beta \leftarrow \beta_0$
  repeat
    $\mathcal{G}$,$\mathbf{y} \leftarrow$ buildMRF(M_N, order)
    $\hat{\mathbf{x}} \leftarrow$ swap($\mathcal{G}$,$\mathbf{y}$, R, β, β_{ss})
    $\beta \leftarrow \beta + \delta\beta$
  until $(h_1(\hat{\mathbf{x}}) = -2$ **and** $h_2(\hat{\mathbf{x}}) <= 6)$ **or** $(h_1(\hat{\mathbf{x}}) = -1$ **and** $h_2(\hat{\mathbf{x}}) <= 4)$ **and** $\delta\beta > 0$
  return $\hat{\mathbf{x}}$



(a) An example of the MRF constructed from the state-label pairs $(s(t), b(t))$ obtained for 16 time steps, where $s(1) = s_1$. There are six time instants when the label is misclassified.



(b) The ideal assignment of the MRF in Figure 4a when the optimal action is assigned in each state. In each state, we assume that ideal classification corresponds to providing the optimal action which is commanded by the user.

$= a_2$ $= a_1$ $= a_0$

Fig. 4: Examples of MRFs obtained for a sequence of state-label pairs. The hollow circles serve to indicate that the two state-label pairs from different rows are actually connected by a single edge.

for all $i \in \{1, \ldots, N-1\}$. In other words, the sequence $\hat{x}_i$ will consist of zero or more consecutive labels a_2, followed by zero or more consecutive labels a_1 and finally by zero or more consecutive labels a_0. Such an estimate $\hat{\mathbf{x}}$ would satisfy either $h_1(\hat{\mathbf{x}}) = -2$ and $h_2(\hat{\mathbf{x}}) = 2$ or $h_1(\hat{\mathbf{x}}) = -1$ and $h_2(\hat{\mathbf{x}}) = 1$.

The algorithm that implements the search for a phase transition consists of obtaining $\hat{\mathbf{x}}$ for increasing values of β, and use logical conditions on the values of $h_1(\hat{\mathbf{x}})$ and $h_2(\hat{\mathbf{x}})$ to decide when to stop increasing β (see Algorithm 3).

D. Summary

The method proposed in [7] is simply Algorithm 3 with parameter values order = temporal, $\beta_0 = 1$, $\delta\beta = 0$, and $\beta_{ss} = 1$. We expect that choosing these parameters differently will result in improved performance for the algorithm. In the next section, we will compare the performance of algorithms with different parameters in simulated goal-inference scenarios. The algorithms are compared using the value of t_f obtained by using associated output $\hat{\mathbf{x}}$ in Algorithm 1, assuming that the goal is correctly identified.

VI. Simulations

We simulate the goal-learning scenario for a $1D$ grid with 10 states over multiple trials. Different variations of Algorithm 3 will be used to compute the estimate of the true actions/label at each time step. The variations are obtained by using different parameters for Algorithm 3, which are listed in Table II. We denote the output of Algorithm X (as named in Table II) as $\hat{\mathbf{x}}_X$.

Algorithm	order	β_0	$\delta\beta$	β_{ss}
A [7]	temporal	1.0	0	1.0
B	ordered	1.0	0	1.0
C	temporal	1.0	0	3.0
D	ordered	1.0	0	3.0
E	temporal	1.0	0.1	3.0
F	ordered	1.0	0.1	3.0
G	randomized	1.0	0.1	3.0

TABLE II: Algorithm list

A trial generates a sequence M_N of state-label pairs, where $N = 60$. At each time step, the probability of the optimal action (given the goal state) being taken is 0.7. The probability of taking one of the two non-optimal actions in a state is 0.15 each. These probability values determine the confusion matrix (and therefore the energy function D in (6)) used in the simulations. The sequence obtained in a trial is stochastic due to the stochastic nature of choosing actions. For this reason, we use a statistical approach to study the merits of the methods we propose, as was done in [7].

The estimate $\hat{\mathbf{x}}_X$ for the labels in a sequence M_N can be used in Algorithm 1 to identify the goal. The value of t_f obtained when the estimate $\hat{\mathbf{x}}_X$ is used in Algorithm 1 is denoted as $t_f(\hat{\mathbf{x}}_X)$. The time at which a state is identified as the goal using $\mathbf{y}$ is denoted as $t_f(\mathbf{y})$. We also note down the actual error rate during the trial, which varies for each trial. An error is said to have occurred at a time step when the action taken in a state is not the same as the optimal one in that state. The error rate is then the ratio of the number of errors to the value of t_f.

We can compare the the different algorithms (rather, variants of Algorithm 3) in Table II based on the value t_f associated with the algorithm. The value of δ in Algorithm 1 is chosen to be 0.95. Since each trial is randomly generated, we use the t-test to compare them.

A. Effect of β_{ss}

In order to study the effect of using a larger value of $V_c(x_i, x_j)$ for nodes i and j where $\pi_s^{-1}(i) = \pi_s^{-1}(j)$, we compare algorithms A and C. Algorithm A corresponds to the algorithm proposed in [7]. Algorithm C can be obtained by using the parameters of Algorithm A except for β_{ss}, which is increased from 1.0 to 3.0. The difference in performance (value of t_f returned by Algorithm 1) between these two inference algorithms are presented in Figures 5, 6 and 7. In Figure 5, the ordinate is the the value of t_f in a trial. The abscissa is the fraction of the total number of time steps taken for which the action was erroneous in a state. In Figure 6, the ordinate is the difference in the value of t_f between the case where $\hat{\mathbf{x}}_A$ or $\hat{\mathbf{x}}_C$ is used in Algorithm 1 and the case where the measured labels $\mathbf{y}$ are used.

The null hypothesis for our simulation experiment is that using our proposed inference algorithm to obtain $\hat{\mathbf{x}}$ and then

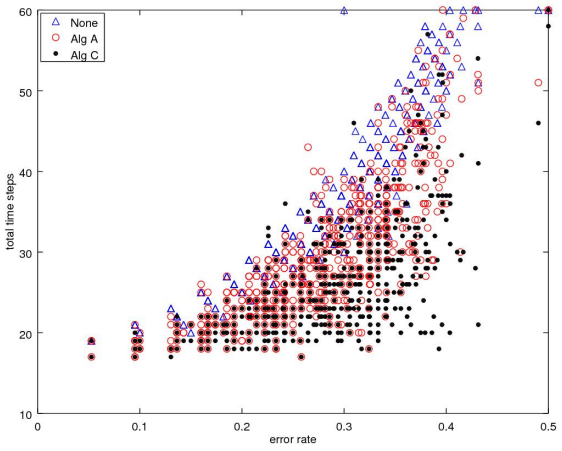


Fig. 5: Scatter plot of the values of $t_f(\mathbf{y})$, $t_f(\hat{\mathbf{x}}_A)$, and $t_f(\hat{\mathbf{x}}_C)$ versus the error rate associated with $t_f(\mathbf{y})$.

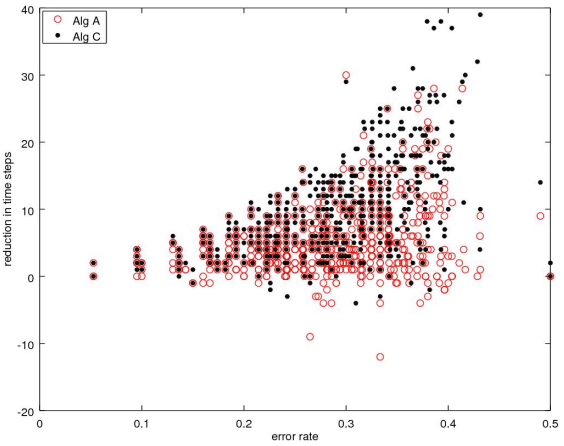


Fig. 6: Scatter plot of $t_f(\mathbf{y}) - t_f(\hat{\mathbf{x}}_A)$, and $t_f(\mathbf{y}) - t_f(\hat{\mathbf{x}}_C)$ versus the error rate associated with $t_f(\mathbf{y})$. Algorithm C is more likely to result in faster identification of the goal.

compute $P_t(s_i)$ for all $s_i \in S$ does not affect the number of time steps taken for $P_t(g)$ to become greater than $\delta = 0.95$, when compared to using the measured labels $\mathbf{y}$ to compute $P_t(s)$. In Figure 5 we see that on average, the number of time steps taken taken to identify the goal with confidence 0.95 is lower for the case when the inference algorithms A and C are used to identify misclassifications, when compared to the case of using the measured labels. This conclusion is validated by the t-statistic values (penultimate row of Table IV) for Sim 1 and Sim 2 being greater than the required paired t-test values (bottom row of Table IV) for rejecting the null hypothesis with 99.95% confidence. Furthermore, the reduction in time steps increases with the error rate, as seen in Figure 6. In order to compare Algorithms A and C, we take the difference in t_f obtained when using Algorithm C and that when using Algorithm A. This difference is plotted in Figures 7, against the error rate at the time when the goal is identified using Algorithm A. We see that Algorithm C results in a larger

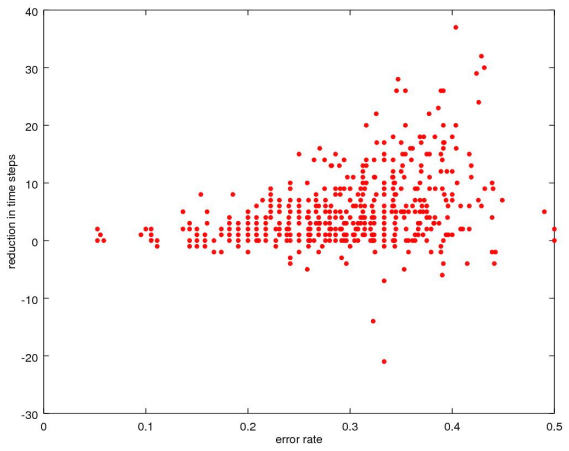


Fig. 7: Scatter plot of $t_f(\hat{\mathbf{x}}_A) - t_f(\hat{\mathbf{x}}_C)$ versus the error rate associated with $t_f(\hat{\mathbf{x}}_A)$. Since the difference is more likely to be positive, we can conclude that Algorithm C is more likely to result in faster identification of the goal, since the difference is more likely to be positive.

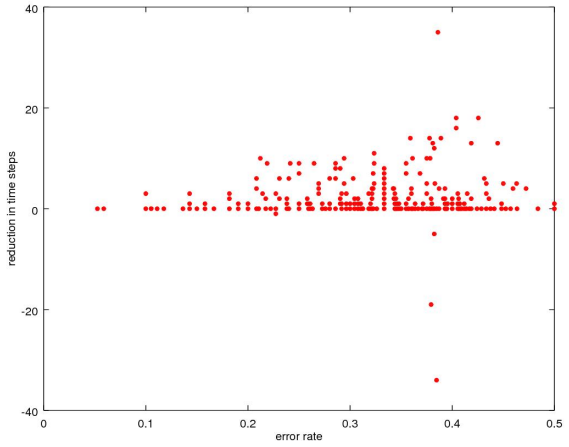


Fig. 8: Scatter plot of $t_f(\hat{\mathbf{x}}_C) - t_f(\hat{\mathbf{x}}_E)$ versus the error rate associated with $t_f(\hat{\mathbf{x}}_C)$. Algorithm E is more likely to take less time than Algorithm C, since the difference is more likely to be positive.

amount of time steps saved.

B. *Effect of Varying* β

In order to study the effect of using a variable value of β, we compare Algorithm E with Algorithm C, and also Algorithm F with Algorithm D. The results are plotted in Figures 8 and 9. In both Figures,we see that using an algorithm to search for phase transitions almost always results in the same or smaller value of t_f.

C. *Effect of* `order`

The last two sections demonstrate the using $\beta_{ss} > 1$ and varying β by searching for a phase transition results in an

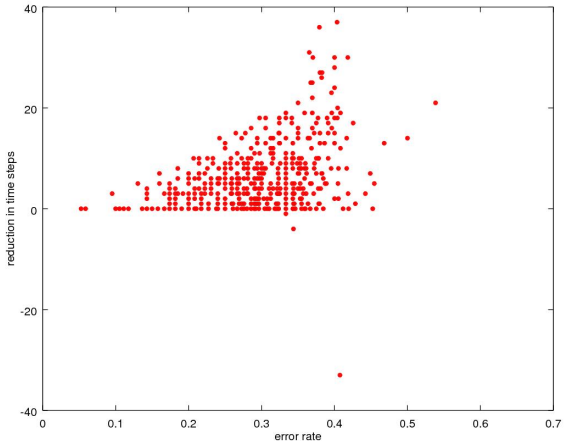


Fig. 9: Scatter plot of $t_f(\hat{\mathbf{x}}_D) - t_f(\hat{\mathbf{x}}_F)$ versus the error rate associated with $t_f(\hat{\mathbf{x}}_D)$. Algorithm F is more likely to take less time than Algorithm D, since the difference is more likely to be positive.

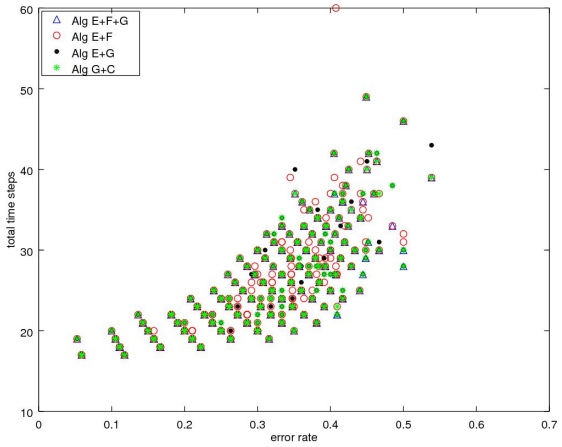


Fig. 10: Scatter plot of t_f for Algorithms $E+F+G$, $E+F$, $E+G$, and $F+G$ versus the error rate associated with t_f for Algorithm $E+F+G$.

	Sim 1	Sim 2	Sim 3	Exp 1
Mean	4.63	48.06	44.14	3.59
Std. dev.	5.46	49.91	34.77	4.14
Min	-9	-44	0	-1
Max	31	182	190	18
# of samples	285	84	483	43
t-statistic	14.56	8.83	27.899	5.67
$t_{.9995}$	3.39	3.416	3.39	3.551

TABLE III: Results of statistical analysis of the data for the simulations (Sim 1-3) and experiment (Exp 1). The unit for the first four rows is the number of time steps.

improvement in performance when compared to the algorithm first proposed in [7]. However, we observe that the ordering of state-label pairs with the same state affects the estimate. The main conclusion from the data is that there is no clear best way to order the state-label pairs.

We use all three estimates $\hat{\mathbf{x}}_E$, $\hat{\mathbf{x}}_F$, and $\hat{\mathbf{x}}_G$ in order to identify the goal. That is, we can compute the value of $P_t(s)$ for all states using each of the three estimates separately, and identify the goal when *any* of them cross the threshold $\delta = 0.95$ for some state $s \in S$. We refer to this method as Algorithm $E+F+G$. We can also identify the goal based on only two out of the three estimates. We refer to the three resulting algorithms as Algorithm $E+F$, Algorithm $E+G$, and Algorithm $F+G$.

The value of t_f using these four methods are shown in Figure 10. In order to distinguish between the four algorithms, we plot the difference between the values of t_f for Algorithm $E+F$, Algorithm $E+G$, and Algorithm $F+G$ and the value of Algorithm $E+F+G$ in Figure 11. The value is always non-positive, implying that if we leave one of the orderings out, we may not identify the goal as fast as when using all three orderings simultaneously. Since the algorithms are designed to be efficient, performing multiple inferences at each time step is feasible in real-time experiments.

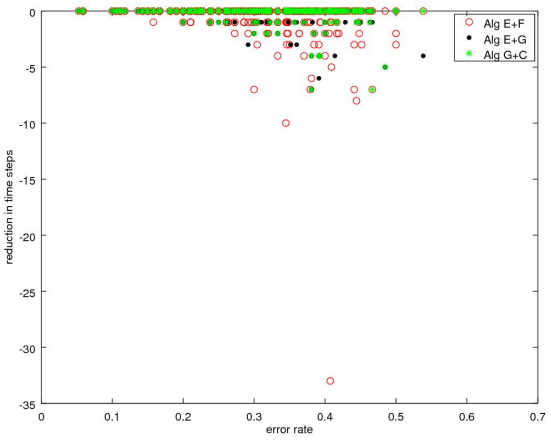


Fig. 11: Scatter plot of difference between t_f for Algorithms $E+F+G$ and t_f for Algorithms $E+F$, $E+G$, and $F+G$, versus the error rate associated with t_f for Algorithm $E+F+G$. Since this difference is never positive, and is negative for each of Algorithms $E+F$, $E+G$, and $F+G$ in at least one trial, we can conclude that using all three sorting methods together to identify the goal will be faster than the case of preferring any single sorting method.

VII. Experiments

We test the new algorithms proposed in this paper on the data collected during experiments in [7]. We recount the setup and procedure used in those experiments. Details regarding the classifier used to identify the actions intended by the human can be found in [7]. The confusion matrix obtained after testing the classifier is shown in Table IV.

A. Setup

A $1D$ grid with 10 states is presented on a computer screen to the subject (see Figure 12). The current state of the system

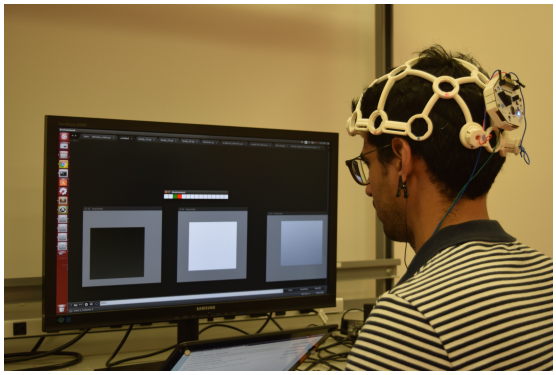


Fig. 12: Setup for experiments involving the control of a virtual device using a brain-computer interface with a classifier in the loop.

		Predicted label		
		a_0	a_1	a_2
Actual label	a_0	0.85	0.07	0.07
	a_1	0.10	0.80	0.10
	a_2	0.09	0.04	0.87

TABLE IV: Confusion matrix for C^*.

is marked by a green cursor. The goal state (taken as the state s_4 of the grid) is marked red. The cursor can move horizontally in the grid. The task for the subject is to command the cursor to move towards the goal and stay there upon reaching it. We also present three squares which flash (between black and white) at 12Hz, 15Hz, and 20Hz respectively. Each square flashing at a unique frequency corresponds to a unique action. The user selects an action by looking at the flashing square corresponding to an assigned action. The EEG signals associated with this selection are extracted using the OpenBCI R&D Kit [14]. The headset has 16 channels, and uses a 32 bit microprocessor to process the measured potentials.

The simulation and processing for the experiment is performed on a Lenovo Thinkpad with an Intel Core i7 2.1GHz CPU and 8GB of RAM running Ubuntu 14.04 as the operating system. The flashing squares are generated using the software program Psychopy [15]. The real-time processing of the labels and simulation of the virtual device is done using Robot Operating System (Indigo version).

B. Procedure

A trial consists of simulating a sequence of actions taken by the cursor at discrete instants of time. The interval between two actions is two seconds. The feature corresponds to data collected for one second just before the cursor is moved. As mentioned, the cursor takes an action based on the output of the classifier. It is assumed that the user always looks at the flashing square corresponding to the optimal action in the current state of the cursor.

The value of $P_t(g)$ is computed in real-time at every time step when an action is performed, using all state-label pairs obtained during the trial up till the current time step. The value of $P_t(g)$ is computed using both the raw measured class labels $\mathbf{y}$ and the estimated class labels $\hat{\mathbf{x}}$ obtained by using the proposed inference algorithm. The trial is stopped either when both values reach 0.95, or 60 time steps have been completed.

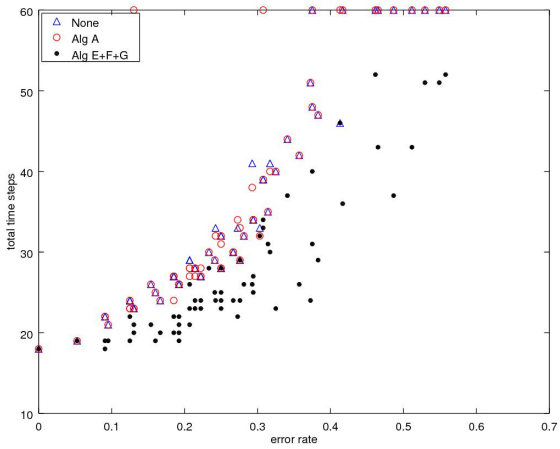


Fig. 13: Scatter plot of the number of time steps taken for $P_t(g)$ to first cross 0.95 vs the error rate, for trials in Exp 1. The red checkmark corresponds to the time taken when $\mathbf{y}$ is used to compute $P_t(s_i)$ for all $s_i \in S$, and the blue triangles correspond to the time taken $\hat{\mathbf{x}}$ is used. The dashed line and solid lines are parabolic fits to the two sets of data points.

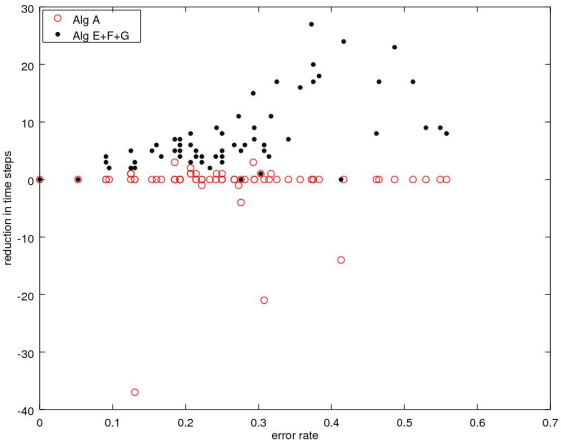


Fig. 14: Scatter plot of the difference in number of time steps taken for $P_t(g)$ to first cross 0.95 vs the error rate, for trials in Exp 1. The difference in steps is between the case when $\mathbf{y}$ is used to compute $P_t(s_i)$ for all $s_i \in S$ and the case when estimate $\hat{\mathbf{x}}$ is used. The green line represents a parabolic fit to the data points.

The trial is discarded if $P_t(g)$ does not cross 0.95 when using either $\mathbf{y}$ or $\hat{\mathbf{x}}$. We refer to this experiment as Exp 1 in Table III.

C. Results

We present the experimental data collected over 43 trials. The t-statistic for Exp 1 (5.67) is higher than that required to reject the null hypothesis with 99.95% confidence (3.55). Therefore, even in experiment, the proposed inference algorithm is effective in reducing the time taken to identify the goal. As seen in Figures 13 and 14, the reduction in time taken increases as the error rate increases.

VIII. Conclusion

In this paper we have proposed new algorithms to identify misclassifications in a sequential control task, leading to improved performance in the task. The work builds on the algorithm proposed in [7]. Two major issues affecting the performance of that algorithm in [7] have been addressed in this paper. The first issue was related to the arbitrary choice of te parameter β. The second issue was related to the ordering of a temporal sequence of data into a graph related to the inference step.

The algorithm in [7] relied on the insight that for the task of reaching a goal, the same action will be optimal in two states on the same side of the goal. Exploiting this insight was enough to result in an improvement of the performance of the goal-learning algorithm by using a novel inference algorithm.

The improvements in this paper rely on using two additional insights. The first insight is that the prior distribution used in the inference step must capture the property that in a given state in a one-dimensional grid, only one action can be optimal. The second insight is that the connectivity of the nodes in the Markov random field can affect the final estimate, and therefore using different methods to connect the nodes (which amounts to a re-ordering of the sequence of labels obtained) can overcome biases in the estimate when compared to using a fixed ordering (which was the situation in [7]).

We show how to convert these insights into a systematic algorithm to estimate the true class labels from the measured class labels related to a goal-learning task. Through simulations, we show how the insights used result in imporved performance in the goal-learning task. The simulation study leads to an algorithm that can be applied to the experimental data. We observe that the average time taken to identify the goal decreases with the new algorithm. In particular, several trials were discarded in [7] because the algorithm therein could not identify the goal in the specified time, however the goal is always identified using the new algorithm.

References

[1] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[2] G. Dornhege, J. del R. Milln, T. Hinterberger, D. J. McFarland, and K.-R. Mller, *Towards Brain Computer Interfacing*. MIT Press, 2007.

[3] J. Grizou, I. Iturrate, L. Montesano, P.-Y. Oudeyer, and M. Lopes, "Calibration-Free BCI Based Control," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, Quebec, Canada, July 2014, pp. 1–8. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00984068

[4] J. d. R. Millan, "Brain-controlled devices: the perception-action closed loop," in *2016 4th International Winter Conference on Brain-Computer Interface (BCI)*, Feb 2016, pp. 1–2.

[5] I. Iturrate, L. Montesano, and J. Minguez, "Shared-control brain-computer interface for a two dimensional reaching task using eeg error-related potentials," in *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, July 2013, pp. 5258–5262.

[6] L. J. Trejo, R. Rosipal, and B. Matthews, "Brain-computer interfaces for 1-d and 2-d cursor control: designs using volitional control of the eeg spectrum or steady-state visual evoked potentials," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 14, no. 2, pp. 225–229, June 2006.

[7] H. A. Poonawala and U. Topcu, "Classification error correction: A case study in brain-computer interfacing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[8] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[9] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[10] R. Kinderman and S. Snell, *Markov random fields and their applications*. American mathematical society, 1980.

[11] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 147–159, Feb 2004.

[12] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, Sept 2004.

[13] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1239, November 2001. [Online]. Available: http://dx.doi.org/10.1109/34.969114

[14] OpenBCI, "http://openbci.com/."

[15] J. Peirce, "Generating stimuli for neuroscience using psychopy," *Frontiers in Neuroinformatics*, vol. 2, p. 10, 2009.