Published in final edited form as:

IEEE Int Conf Healthc Inform. 2017 August ; 2017: 144–153. doi:10.1109/ICHI.2017.66.

Medical Workflow Modeling Using Alignment-Guided State-Splitting HMM

Sen Yang,

Rutgers University, Piscataway, NJ

Moliang Zhou,

Rutgers University, Piscataway, NJ

Shuhong Chen,

Rutgers University, Piscataway, NJ

Xin Dong,

Rutgers University, Piscataway, NJ

Ivan Marsic,

Rutgers University, Piscataway, NJ

Omar Ahmed, and

Children's Nat'l Med Ctr, Washington, DC

Randall S. Burd

Children's Nat'l Med Ctr, Washington, DC

Abstract

Process mining techniques have been used to discover and analyze workflows in various fields, ranging from business management to healthcare. Much of this research, however, has overlooked the potential of hidden Markov models (HMMs) for workflow discovery. We present a novel alignment-guided state-splitting HMM inference algorithm (AGSS) for discovering workflow models based on observed traces of process executions. We compared the AGSS to existing methods using four real-world medical workflow datasets and a more detailed case study on one of them. Our numerical results show that AGSS not only generates more accurate workflow models, but also better represents the underlying process. In addition, with trace alignment to guide state splitting, AGSS is significantly more efficient (by a factor of $O(n)$) than previous HMM inference algorithms. Our case study results show that our approach produces a more readable and accurate workflow model than existing algorithms. Comparing the discovered model to the hand-made expert model of the same process, we found three discrepancies. These three discrepancies were reconsidered by medical experts and used for enhancing the expert model.

Keywords

Process Mining; Medical Workflow; Hidden Markov Model; State-splitting Algorithm; Trace Alignment

I. Introduction

Process mining, including workflow discovery and analysis, has become common in recent years [1]. The application of workflow discovery and analysis in the medical field has the potential to improve patient outcomes. In the past, medical experts carefully designed medical workflow models, but actual practice often deviates from a perceived ideal process [4]. Models discovered from real process data provide information about the actual executions, and are critical for understanding process errors, e.g., omitted and duplicate activities. In addition, many clinical workflows do not have a predetermined workflow model. Process model extraction is then essential for discovering workflows more representative of their actual executions. We present a novel hidden Markov model (HMM) inference algorithm derived from existing work [2][3] to efficiently discover representative workflow models from medical processes.

Existing workflow discovery algorithms cannot provide the optimal workflow models. These methods assume that duplicate activities in a process trace are equivalent. Based on our analysis of trauma resuscitation workflow and previous work, each occurrence of an activity may have different underlying “intentions” [1][4][6]. For instance, over the course of a single trauma resuscitation, the trauma team may check the patient’s eyes at two different points in time for different reasons. First, during the primary survey they assess the patient’s pupillary response for neurological disability. Second, during the secondary survey, the team may examine the eyes in more detail, looking for injuries to the cornea, sclera and eyelids. To discover an accurate workflow model, an algorithm should be able to distinguish the first eye check from the second one, despite their identical labels. To our knowledge, no existing workflow mining algorithm can properly model such duplicate activities. Recent work has presented a strategy to refine duplicate activity labels, but only during preprocessing [6]. We address this problem by modeling duplicate activities as different hidden states in an HMM.

We believe that HMMs are particularly suitable for work-flow discovery. While HMMs are well-studied in speech, handwriting, and bioinformatics, they have been overlooked in process mining for several reasons [1]. First, HMM inference is computationally demanding due to the iterative maximization procedures (i.e., Baum-Welch (BW) algorithm). Second, classical BW HMM inferencing depends heavily on subjective and labor-intensive parameter initialization (i.e. finding the number of hidden states, observation vectors, transition matrices). Finally, the resulting HMM may be too complex for knowledge acquisition purposes; the interpretation of hidden states can be challenging even for simple processes. Few attempts have been made to use HMM for workflow mining in the last two decades. Previous process mining research using BW HMM inference methods did not address the parameter estimation problem [4]. We address these problems with AGSS and our HMM simplification algorithm.

We also studied the balance between model accuracy and generality. If a workflow discovery algorithm only pursues model accuracy, it may overfit the observed process data, making the model too specific. If the algorithm does not represent specific features of the process, the model may underfit the observed data. In both cases, the derived models are biased from the ground-truth process that generates the observed data. Consider a simple process log with

five observed traces where letters represent activities performed in the shown order: $\mathbf{O} = \{ABCDEBA, ABBDEBA, ABDEBA, ABDEA, ABCDDEA\}$. Three different representative workflow models may be discovered based on $\mathbf{O}$ (Figure 1). The most specific workflow model (a), a very general workflow model (b), and the ground-truth workflow model that was used to generate the example traces (c). Although model (a) describes the observed process traces very accurately, it lacks generality to represent unseen traces generated by the same process. Model (b) would be discovered by existing process mining algorithms [1]. It represents each activity by a single workflow node and overlooks potentially different “intentions” of the repeated activities. This model allows existence of sequences that are improbable in the underlying model (c). For example, activity A may go directly to “End”, or ABCDE may loop many times. On the other hand, the underlying model (c) used to generate the traces has repeated activities A (A_1, A_2) and B (B_1, B_2). A_2 and B_2 occur at the end of process executions, rather than in the loop shown in model (b). We present an algorithm that uncovers workflows like (c) from a process log.

Our AGSS algorithm uses trace alignment—a data-driven algorithm—to guide HMM model inference. Trace alignment can discover the consensus sequence, or the backbone procedure of the observed process traces [17]. The trace alignment algorithm alone, however, is not sufficient for model mining because it only models sequential relationships between activities, and cannot handle parallelism. This limitation is not an issue for HMMs. Instead, HMMs face problems with model parameter initialization and computational complexity. Since trace alignment is able to find the distribution of the activities, incorporating alignment into HMM inference can help avoid subjective parameter initialization, boost inference speed, and produce better HMMs. Our contributions in this study are:

- A novel alignment-guided state-splitting HMM inference algorithm (AGSS) for real-world workflow discovery. Taking advantage of alignment, AGSS can efficiently find which states to split and how many states to split. This way, the complexity of the induced HMM is controlled by both the activity distribution in the alignment matrix and the dataset size (number of observed process traces). We compared AGSS to existing state-splitting-based HMM inference algorithms. The performance was evaluated using data from four real-world medical processes, in three different aspects: (1) How representative the induced model is of the observed process traces. (2) How close the induced model is to the underlying model that generated the observed process traces. (3) The computational complexity of HMM inference.
- An HMM simplification algorithm that reduces the number of insignificant transitions and extracts a fully connected backbone workflow model from a spaghetti-like HMM model. The simplified model is more readable and allows for easier knowledge extraction.

II. Definitions and Data Formalization

We first define the terms and notations used later (Table 1):

Definition 1: A **process log** O is a set of process traces composed of activity executions. A log with T process traces is denoted $O = \{O_1, \dots, O_j, \dots, O_T\}$. A process trace O_j is represented by $O_j = \{a^{start}, a_1, \dots, a_i, \dots, a_\ell, a^{end}\}$, where a_i denotes the i -th activity in the trace and ℓ denotes the trace length. “start” (a^{start}) and “end” (a^{end}) symbols are added to mark the beginning and end of each trace.

Definition 2: Given a process log O , an **alignment algorithm** $\mathcal{A}(O)$ forms an alignment matrix with the traces in O as rows and activities of the same type as columns. If for a given trace a matching activity cannot be found, a gap symbol “-” is inserted in the corresponding column (Figure 2(a)). Alignment $\mathcal{A}(O)$ returns the consensus sequence $\mathcal{CS}$, which we use to guide the selection of splitting candidates SC . When describing alignment algorithm computational complexity, we use L to denote the average length of alignment matrices in a pairwise alignment. Alignment has previously been used for workflow and activity pattern analysis [17][18].

Definition 3: A **consensus sequence** $\mathcal{CS}$ is generated from the alignment matrix. It contains the activities in alignment matrix columns. $\mathcal{CS}$ can be considered the “average”, or “backbone” of the original traces. Due to variations in execution of complex processes, some alignment columns contain only a small number of non-gap elements. We define “column frequency” as the fraction of non-gap elements (Figure 2(a)), which correspond to how frequently the activity in this column was performed across different process executions. We apply a column frequency threshold ε to filter out columns with rare activities. We use $\mathcal{CS}'$ to denote the consensus sequence after the filtering. Splitting candidates SC are taken from $\mathcal{CS}'$. Note that we use a “relaxed” consensus sequence that considers the surrounding columns within a time window when calculating the frequency of a specific column. This is because parallel activities in the workflow may result in different alignment orderings across nearby columns, but should still be considered the same “intention”.

Definition 4: An **HMM** λ is a statistical model for modeling temporal sequences with unobserved or hidden states. An HMM is denoted by a quintuplet $\lambda = (A, B, \pi, n, m)$ where matrix A records the state transition probabilities, where $A(i, j)$ represents the transition probability between state i and state j ; B records the observation probability distribution in each state; π is a vector consisting initial state distribution; n is the number of hidden states and m is the number of distinct observation symbols per state. The **observation probability** $P(O|\lambda)$ is the probability that the HMM λ will emit the set of observed traces O . **Log-likelihood** (a.k.a. log-likelihood) $\log P(O|\lambda)$ is usually used in implementations to avoid arithmetic underflow problems.

Definition 5: The **splitting candidates** SC comprise a set of activities eligible for state splitting during the topology inferencing of HMM λ . We use a_i in SC to denote a candidate activity to split in λ . The HMMs obtained by state-splitting are placed in the set **candidate models** $\mathcal{M}$. N denotes the total number of splits done during the inferencing of λ .

III. Review of Existing HMM Inference Algorithms

Most HMM inference methods are used for supervised classification, with heuristic initialization $\lambda_0 = (\pi_0, A_0, B_0, n, m)$. Given an approximate HMM topology, these algorithms fine-tune the HMM with the Baum-Welch algorithm [16]. Without appropriate initial parameter, the inferred HMM can be heavily biased from the ground truth. Other HMM inference algorithms implement either state-merging or state-splitting to address the initial parameter problem [2][8].

The state-merging approach was first proposed by Stolcke, et al. [8]. It begins the inference with the most specific topology, containing one path for each observed trace (as in Figure 1(a)), and iteratively merges states until convergence of a model quality criterion (e.g. maximum a posteriori (MAP) [8]). But model-merging is computationally expensive [3][7][8][9]. First, as initializing the HMM with all observed traces usually requires many nodes, calculating observation probability $P(\mathbf{O}|\lambda)$ can be expensive. Second, the merging method suffers from a lack of search direction [3] and may stop too early [8], achieving local convergence far from the global optima.

The state-splitting approach, proposed by Takami and Sagayama [2], infers HMMs from the opposite direction. It begins with a general HMM and successively splits states until convergence. Compared to merging, the splitting approach is faster. Because the initial model is small, it is generally much closer to the final desired model [2][3][10][11][12][13]. Given observations $\mathbf{O}$, current state-splitting algorithms:

1. Determine the best state to split in each iteration j :

$$\lambda_j = \arg \max_{\lambda_{j-1}^s \in \mathcal{M}_{j-1}} \text{Score}(\lambda_{j-1}^s, \mathbf{O}) \quad (1)$$

1. Determine when to stop splitting:

$$\text{Score}(\lambda_j, \mathbf{O}) \leq \text{Score}(\lambda_{j-1}, \mathbf{O}) \quad (2)$$

where λ_{j-1}^s is a candidate model for splitting state s at iteration $j-1$, $\mathcal{M}_{j-1}$ includes all the candidate models at iteration $j-1$, and the scoring function $\text{Score}(\lambda, \mathbf{O})$ quantifies how well the model λ fits the observations $\mathbf{O}$, as well as the model penalty [10][11][13][14][15]. In other words, greedy splitting is done on all possible splits, and stops when further splitting does not increase the score. Both steps are dependent on the scoring function. It is difficult to analyze the greedy splitting's effect on the resulting model, but the stopping mechanism clearly influences the model's accuracy and generality. Splitting too much will overfit, sacrificing representativeness for accuracy. Splitting too little will underfit, leading to a model with poor accuracy. Maximum likelihood successive state-splitting (ML-SSS) [10] terminates splitting either when the likelihood $P(\mathbf{O}|\lambda)$ saturates, or when the model size reaches a certain threshold. ML-SSS does not penalize model complexity, so it could

theoretically keep splitting down to the most specific model. Different complexity penalties have been used in previous research. Mavromatis [13] used minimum description length (MDL). Herbst [3] used a heuristic score function. Siddiqi et al. [14] used Bayesian information criterion (BIC) for simultaneous temporal and contextual splitting (STACS). Despite the variety of scoring metrics and algorithms, none of this research was used to derive an underlying model from complex real-world workflow data.

IV. Alignment Guided State-splitting HMM

A. AGSS algorithm

AGSS (see Figure 2) first initializes a general Markov chain λ_0 . Unlike most state-splitting algorithms that start with only one state [2][10][11][13][14], AGSS begins with a Markov chain composed of one state per activity (step 1 in Algo.1), a strategy used in the heuristic state-splitting approach [3].

After initialization, AGSS determines two factors: which states to split and when to stop splitting. AGSS determines the splitting candidates from the alignment matrix and orders them by column frequency. Specifically, AGSS follows intuitions about the consensus sequence (step 2 in Algo.1): (1) activities of different columns of the alignment matrix likely have different purposes or intentions regardless of activity type; and (2) each column's frequency measures the importance or distinctiveness of the activity's intention. Having two different columns with the same activity type and frequencies above the ε threshold strongly indicates that this activity (e.g., "A and B" in Figure 2(a)) has multiple intentions, qualifying the associated state for splitting candidacy (step 3 in Algo. 1). It also intuitively follows, and our experiments have shown, that the candidates with higher column frequencies (e.g. "A" in last column of Figure 2(a)) should be tested for splitting before those with lower frequencies (e.g. "B" in column before last in Figure 2(a)). Therefore, we sorted the candidates $\mathcal{CS}'$ by descending frequency (step 4). Because each activity is already assigned a state in the initial model λ_0 , there is no need to split the first occurrence of each activity in $\mathcal{CS}'$. Therefore, we revoke the candidacy of first occurrences and return the final list $\mathcal{SC}'$ (step 4).

After calculating the splitting candidates, AGSS performs iterative splitting. In iteration j : given a splitting candidate activity $a_j \in \mathcal{SC}$, we find states S , in the model from last iteration λ_{j-1} , which have observations a_j (e.g., S_1 in Figure 2(b). Step 7 in Algo.1). For each state s_k in S , we split it into two new states s_k' and s_k'' (step 9 in Algo.1). The newly-split states (s_k' and s_k'') are assigned the same observation as their predecessor's state s_k . The transitions (i.e., transition probabilities from other states to s_k , s_k'' and from s_k' , s_k'' to other states) can be initialized either randomly or by estimating the distribution of surrounding activities in the alignment matrix (step 10 in Algo.1). After initializing the parameters of s_k' and s_k'' , λ' is trained with the Baum-Welch Algorithm to optimize the HMM parameters (step 11 in Algo.1). λ' that maximizes log-likelihood $P(\mathcal{O}|\lambda')$ is assigned λ_j (step 14 in Algo.1). Afterwards, λ_j is compared to λ_{j-1} ; λ_j is kept only if it improves model log-likelihood (steps 15-17 in Algo.1).

Algorithm 1. AGSS HMM**Input:** $\mathcal{O}, \varepsilon$ **Output:** λ

```

Step 1. Initialize HMM topology as Markov chain  $\lambda_0$ ;
/* Obtain candidate activities for splitting */
Step 2. Compute relaxed consensus sequence  $\mathcal{CS} = \mathcal{A}(\mathcal{O})$ ;
Step 3. Filter consensus sequence  $\mathcal{CS}' = \text{Filter}(\mathcal{CS}, \varepsilon)$ ;
Step 4. Find splitting candidates  $\mathcal{SC}$  by sorting  $\mathcal{CS}'$  and removing the first oc-
currences of all activities:
       $\mathcal{D} = \text{Sort}(\mathcal{CS}')$ ;  $\mathcal{SC} = \text{RemoveFirstOccurrences}(\mathcal{D})$ ;
/* Perform state splitting */
Step 5. for activity  $a_i$  in  $\mathcal{SC}$ , do
Step 6.   Create a temporal model  $\lambda' = \lambda_{i-1}$ ;
Step 7.   Find states  $\mathcal{S}$  that have observation  $a_i$  from  $\lambda'$ ;
Step 8.   for  $s_k$  in  $\mathcal{S}$ , do
Step 9.     Split  $s_k$  to  $s_k'$  and  $s_k''$  in  $\lambda'$ ;
Step10.    Initialize observation and transitions of  $s_k'$  and  $s_k''$ ;
Step11.    Train with Baum-Welch algorithm:  $\lambda' = \text{Train}(\lambda', \mathcal{O})$ ;
Step12.     $\mathcal{M} = \mathcal{M} \cup \{\lambda'\}$ ;
Step13.   end for
Step14.    $\lambda_j = \arg \max_{\lambda' \in \mathcal{M}} P(\mathcal{O}|\lambda')$ ;
Step15.   if  $\log P(\mathcal{O}|\lambda_j) < \log P(\mathcal{O}|\lambda_{j-1})$ ;
Step16.      $\lambda_j = \lambda_{j-1}$ ;
Step17.   end if
Step18.    $j = j + 1$ ;
Step19. end for
Step20.  $\lambda = \lambda_{j-1}$ ;
Step21. return  $\lambda$ ;
```

Compared with existing state-splitting HMM inference algorithms [10][11][13][14], AGSS has the following advantages. First, as we associate each state with a single activity type and assign newly-split states their predecessor's activity, each state in the inferred HMM has only one associated activity. This way, the discovered models will have easily-interpretable topologies. Second, because we calculate all splitting candidates based on alignment, AGSS does not need to search the parameter hyperspace of all possible HMMs from all possible splits. Third, as the alignment results and consensus sequence selection are tailored to characteristics of the data, our alignment-based strategy better captures the activity distribution. With these advantages, AGSS not only runs faster, but is also more robust.

B. Estimating the ε Threshold

The frequency threshold ε is important for selecting the splitting candidates which, in turn, determine when splitting will terminate. The threshold ε for the frequency of non-gap symbols in columns ranges between 0 and 1. As it approaches 0, more activities will be allowed onto the candidate list; as it approaches 1, fewer states are candidates. We propose two threshold ε selection methods. The first method finds ε by running the inference algorithm with random ε values, and then picks the threshold yielding the best model (model evaluation discussed later). The apparent limitation is the long running time required to infer

multiple models. The second method estimates ϵ from the data size (num. of traces) using Algorithm 2.

Algorithm 2. Estimating frequency threshold ϵ

Input: $\mathcal{O}$

Output: ϵ

Step 1. Randomly select half of the traces from $\mathcal{O}$, denoted as $\mathcal{O}_{half}$;

Step 2. Calculate consensus sequences: $\mathcal{CS} = \mathcal{A}(\mathcal{O})$; $\mathcal{CS}_{half} = \mathcal{A}(\mathcal{O}_{half})$;

Step 3. **for** $\epsilon = 0:r:1$, **do**

Step 4. $\mathcal{CS}' = \text{Filter}(\mathcal{CS}, \epsilon)$; $\mathcal{CS}'_{half} = \text{Filter}(\mathcal{CS}_{half}, \epsilon)$;

Step 5. **break if** $ED(\mathcal{CS}', \mathcal{CS}'_{half})$ converges

Step 6. **end for**

Step 7. **return** ϵ

where ED is the edit distance (a.k.a. Levenshtein distance), between two filtered consensus sequences and r is an increment of ϵ . Algorithm 2 is based on the law of large numbers [19], which implies that as the data size increases (more process traces), the average of the data (consensus sequence) should converge to an expected value. In other words, the elements in consensus sequences will stabilize with more data.

To evaluate whether we have enough data for producing a stable consensus sequence $\mathcal{CS}'$, we randomly select half of the input traces (step 1 in Algo. 2) and calculate their consensus sequence $\mathcal{CS}_{half}$ (step 2 in Algo. 2). $\mathcal{CS}$ and $\mathcal{CS}_{half}$ are filtered using the threshold ϵ (step 4 in Algo. 2). The edit distance between $\mathcal{CS}'_{half}$ and $\mathcal{CS}'$ will approach 0 when data size is sufficient. The convergence occurs faster for larger ϵ (Figure 3(a)). Given the observed data, Algorithm 2 returns the smallest ϵ at which the edit distance converges. The convergence threshold is set to 0.05, which is the smallest normalized edit distance (i.e., edit distance normalized by the sum of trace lengths) between $\mathcal{CS}'_{half}$ and $\mathcal{CS}'$. In other words, convergence occurs when over 95% of the two consensus sequences are equal.

This ϵ is desirable because it selects the number of splitting candidates proportional to the sufficiency of the dataset. For smaller datasets, it would be inappropriate to split too much and overfit (i.e., states split to capture characteristics or patterns that only exist in specific observed traces). If the process log $\mathcal{O}$ is small, the two consensus sequences would be unstable, forcing a higher ϵ that preserves only the most common activities in the consensus sequence. This high ϵ produces a stable but sparse consensus sequence, therefore disqualifying most states from being split. On the other hand, for large datasets ϵ would be small, allowing the model to split more. The resulting model would be more detailed, which is preferable when training with sufficient data. AGSS then adaptively adjusts the number of potential splits to the amount of data available.

C. Simplifying and Pruning the Discovered Model

Data-derived workflow models are spaghetti-like [1] because of the large number of non-zero transitions. This finding is also true for the AGSS-induced HMM. When used for

classification, spaghetti-like models are acceptable black boxes, because we only focus on classification accuracy and do not extract insights from the model structure. In workflow discovery, finding a descriptive and interpretable model is more important. This goal can be accomplished by model simplification.

Model simplification and pruning are techniques to reduce the size and complexity of workflow models by omitting insignificant information. The obvious solution is to apply transition frequency cutoffs, but this may produce a flawed workflow model with “dangling” process fragments (C in Figure 4). These dangling fragments represent impossible executions when put in the context of workflows. Removing insignificant transitions while keeping the graph a plausible workflow is a form of the NP-hard minimal spanning strong subgraph problem (where we artificially place a transition with frequency one from the end to the start). In the context of process mining, however, it is more reasonable to use heuristic methods.

In this work, we propose a heuristic HMM simplification algorithm (Algo. 3) which can extract the skeleton (the backbone structure) of the HMM. The strategy preserves the dominating incoming and outgoing transitions for each state, reduces the number of insignificant transitions, and does not produce single dangling activities. It is, however, possible for a group of states to form a subprocess isolated from the main process (i.e., the process that includes the start state s^{start} and end state s^{end}). For example, a pair of states can construct a loop where the most significant in-and-out transitions go to each other. We address this problem by first discovering isolated subprocesses (steps 9-18 in Algo. 3) and then reconnecting the subprocess to the main one through the entire subprocess’s most significant in-and-out transitions (steps 19-25 in Algo. 3). The output $\mathcal{A}^s$ from algorithm 3 is the backbone model of observed processes $\mathcal{O}$.

V. Empirical Experimental Results

We compared AGSS to existing baseline state-splitting HMM inference methods, such as ML-SSS [10], MDL [13][15], heuristic approach [11], and STACT [14]. The performance was measured based on (1) model’s quality and (2) the algorithm’s computational efficiency. Both perspectives were quantitative (the induced HMMs were not interpreted by humans), so model simplification or pruning (Section IV. C) was not applied in this section. Experiments were run on a Dell desktop (Win 10, Intel Xeon 3.7GHz CPU, 48G RAM).

A. Real World Medical Process Datasets

During the evaluation, we used activity logs from four real-world medical processes (Table 2). Obtaining and using the

Algorithm 3. HMM Simplification**Input:** $\lambda = (A, B, \pi, n, m)$ **Output:** λ^s

/* Preserve the dominating in-and-out transitions of each state */

Step 1. Initialize the simplified transition matrix $A^s = \emptyset$;Step 2. **for** $i = 0: 1: n$, **do**Step 3. **for** $j = 0: 1: n$, **do**Step 4. **if** $A(i, j) = \max(A(i, :))$, $A^s(i, j) = A(i, j)$;Step 5. **if** $A(i, j) = \max(A(:, j))$, $A^s(i, j) = A(i, j)$;Step 6. **if** $i = j$, $A^s(i, j) = A(i, j)$; /* keep self-transition */Step 7. **end for**Step 8. **end for**

/* Discover states isolated from the main process */

Step 9. Initialize a set with all states in λ except s^{start} : $S = \{s_1, \dots, s_n\}$. Initialize a stack $S^{(m)} = \{s^{\text{start}}\}$; /* s^{start} is s_0 */Step 10. **while** $S^{(m)} \neq \emptyset$ Step 11. $s_i = S^{(m)}.\text{pop}()$;Step 12. **for** $j = 0: 1: n$, **do**Step 13. **if** $A^s(i, j) \neq 0$ && $s_j \in S$ /* i is the index of s_i */Step 14. $S^{(m)}.\text{push}(s_j)$;Step 15. $S.\text{remove}(s_j)$;Step 16. **end if**Step 17. **end for**Step 18. **end while**

/* Connect isolated subprocesses to the main process */

Step 19. **while** $S \neq \emptyset$ /* S includes all states isolated from main process */Step 20. $s = S.\text{get}(0)$;Step 21. According to A^s , find all states in S directly or indirectly connected to s and insert them to $S^{(s)} = \{\dots, s, \dots\}$;Step 22. Find the most preceding state s_j or states s_j (when loop exists) in $S^{(s)}$, $A^s(i, j) = \arg\max_{i,j} A(i, j)$, find the most succeeding state s_k or states s_k (when loop exists) in $S^{(s)}$, $A^s(i, k) = \arg\max_{i,k} A(i, k)$;Step 23. Remove the states in $S^{(s)}$ from S ;Step 24. **end while**Step 25. **return** $\lambda^s = (A^s, B, \pi, n, m)$

medical data for this study was approved by the Institutional Review Board at Children's National Medical Center in Washington, DC. Three medical logs were coded from surveillance videos of trauma resuscitations: (1) The endotracheal intubation (breathing tube insertion). (2) The initial evaluation phase (i.e., primary survey) of the trauma resuscitation [21], where the team looks for immediately life-threatening injuries. (3) The head to toe examination phase (i.e., secondary survey) of the trauma resuscitation. The fourth dataset was collected at a Dutch hospital and published by 4TU (<http://researchdata.4tu.nl/home/>) [22]. After removing outliers and rare activities, there remained 24,550 activities of 18 types in the Dutch hospital data.

B. Measuring Quality of Induced HMM

The challenge for HMM inferencing in workflow mining is its unsupervised nature; labeled ground truth data is not available to test the induced model. Previous state-splitting algorithms working with supervised classification problems had labeled data to test the performance [14]. Our goal in this study is to induce a descriptive process model, as opposed to a predictive one. This descriptive model can be evaluated by how well it

represents observed and unobserved data [23]. To achieve this goal, we adopted two concepts, *model fidelity* and *model confidence*, from previous research [20]. Model fidelity (or accuracy) measures the agreement between a given workflow model and the observed process traces (the log likelihood of generating the observed process traces using the given model). The value is high if the model structure accurately describes the observed process traces. Model confidence measures how well a work-flow model represents the underlying process that generates the observed process traces. High model confidence is achieved if the model describes not only the observed traces, but also the unobserved realizations of the underlying process. Model confidence is difficult to achieve with insufficient training data, because the model may overfit the small dataset.

Given observed traces $\mathbf{O}$ and inferred model λ , model fidelity (M_f) is defined as the log-likelihood of the observed traces:

$$M_f = \log P(\mathbf{O} | \lambda) = \sum_{i \in \{1, 2, \dots, \ell\}} \log P(\mathbf{O}_i | \lambda) \quad (3)$$

where $P(\mathbf{O}_i | \lambda)$ represents the probability that the observed trace $\mathbf{O}_i$ is produced by model λ and ℓ is the average trace length.

The model confidence is hard to quantify because we have neither the underlying ground truth model, nor the entire set of possible process traces. Model confidence (M_c), can be estimated by partitioning the data into training and test sets. Let λ' denote the inferred model from training data $\mathbf{O}_{training}$. We define model confidence as $\log P(\mathbf{O}_{test} | \lambda')$. In practice, we chose to use 10-fold cross validation (Algo. 4) to reduce result variance. Real-world process logs are usually small, since they need to be coded manually by domain experts.

Algorithm 4. Calculating model confidence M_c

Input: $\mathbf{O}$

Output: M_c

Step 1. Initialize $M_c = 0$;

Step 2. Partition the observed traces $\mathbf{O}$ into 10 folds, $\mathbf{O} = \{\mathbf{O}_1, \dots, \mathbf{O}_{10}\}$;

Step 3. **for** $i = 1: 10$, **do**

Step 4. Infer λ'_i with 9 folds except $\mathbf{O}_i$

Step 5. Replay $\mathbf{O}_i$ in λ'_i and calculate log-likelihood $\log P(\mathbf{O}_i | \lambda'_i)$

Step 6. Sum up log-likelihood $M_c += \log P(\mathbf{O}_i | \lambda'_i)$

Step 7. **end for**

Step 8. **return** M_c

Some test data $\mathbf{O}_i$ may not be replay-able on the trained HMM λ' (i.e., probability $P(\mathbf{O}_i | \lambda') = 0$). This problem usually occurs with sparse training data, where an activity or transition present in the test data is not in the training data. The probability $P(\mathbf{O}_i | \lambda')$ of such a trace would then be zero, despite missing only one activity or transition. This issue is usually addressed by smoothing the model parameters, a process of flattening the probability

distribution so that all sequences can occur with some probability. Smoothing can improve HMM performance in a classification problem, but will not help mine workflow models. For model mining, adding unseen activities and transitions would only make the model. Instead of smoothing, we propose a replay rate metric R_r , the percent of test data can be replayed in λ' .

$$R_r = \frac{\sum_i \delta(P(O_i | \lambda'))}{n} \times 100 \% \quad (4)$$

where $\delta(x) = 1$ if $x \neq 0$, and $\delta(x) = 0$ otherwise; and n is the number of traces in the test set. R_r is strongly related to data size (Figure 3(b)), with R_r increasing as data size increases.

We computed model fidelity (M_f), model confidence (M_c), and replay rate (R_r) to evaluate the HMM inducing algorithms on different datasets (Table 3). As described above (Section IV. A), ML-SSS, MDL, and STACT were designed to start splitting from one node and allow different activity types to be observed at the same state. On the other hand, Heuristic and AGSS were designed to be initialized with a Markov chain. We also used tests on synthetic data to set a convergence threshold of 0.01 for ML-SSS to prevent too many splits [10] (splitting is terminated if the next iteration's log likelihood is less than 1.01 times that of the previous).

The results (Table 3) show that while AGSS does not achieve the best model fidelity on intubation and primary survey, it achieves the best model confidence on all datasets. Compared with its initial Markov chain topology, AGSS induces a model with better fidelity and confidence, indicating that the splitting improves the model. The heuristic approach, on the other hand, has higher model fidelity (−11.8 on intubation and −8.5 on primary survey) but lower model confidence (−15.8 on intubation and −10.95 on primary survey) than Markov chain. This finding indicates that the heuristic approach made unnecessary splits and overfit to the training data. Starting from a Markov chain and using the “one observation per state” constraint, both AGSS and the heuristic approach suffer from low replay rates. From a model mining perspective, however, these constraints become advantages in producing easy-interpretable workflow models. ML-SSS achieves high model fidelity using the Intubation dataset (−10.56) and Secondary Survey dataset (−48.79) but has low model confidence (−22.05 and −60.52), indicating it also overfit the data. MDL and STACT have lower model fidelity and confidence when compared to AGSS and Heuristic, indicating they penalize model complexity too severely and terminate the splitting too early. This is in agreement with the fact that MDL and STACT assume a data size much larger than the number of model parameters [15] [24],

C. Computational Complexity Comparison

Existing state-splitting algorithms (e.g. ML-SSS, MDL, Heuristic, STACT) are computationally complex because the search for splitting candidates at each iteration is expensive. Each state is considered a candidate and is therefore tested. AGSS avoids this problem by initially discovering all splitting candidates from trace alignment. The overall computational complexity of AGSS is $\mathcal{O}(Tn^2N)$, consisting of trace alignment and splitting

candidate evaluation. Trace alignment requires $\mathcal{O}(T^2\epsilon^2 + T^2 \log(T))$ to construct the guide tree and $\mathcal{O}(T^2L + TL^2)$ for progressive alignment [17][18]. State splitting requires $\mathcal{O}(Tn^2N)$ for N total state splits with $\mathcal{O}(Tn^2)$ for Baum-Welch parameter learning [16] and likelihood evaluation (“forward-backward”) at each split. In comparison, other state-splitting algorithms generally take $\mathcal{O}(Tn^3N)$, where the number of splits N depends on the convergence rate of the score function. The computational complexities of AGSS and other state-splitting algorithms strongly depend on the average number of hidden states n because of its raised power.

When comparing the number of successful splits versus time on a logarithmic scale (Figure 5(a)), we ignored the terminating criterion for each algorithm and continued splitting until a given number of states N (50 is used in Figure 5) was reached. AGSS showed a clear advantage in speed, using only 135.22 seconds. Other baseline algorithms took more than 2,000 seconds. AGSS and Heuristic have a slight advantage in that they start splitting from 17 states as opposed to one, but this is insignificant because it only takes less than 100 seconds for MLSSS, MDL, and STACT to split to 17 states. In the computation time vs. log-likelihood comparison (Figure 5(b)), AGSS also outperformed baseline algorithms in log-likelihood (model fidelity) from the start and maintained the advantage through 50 states. Also, by starting from a Markov chain, AGSS and heuristic have initial log-likelihood $-32,367$, while the others start from $-73,936$. The algorithms starting from one node did not achieve the initial log-likelihood of AGSS and heuristic even after splitting 49 times. Another observation is that the splitting speed slows (Figure 5(b)) and the log-likelihood gains per split diminishes (Figure 5(c)) as the number of states increase. Although log-likelihood kept increasing, model complexity penalties behaved differently (Figure 5(d)(e)(f)). BIC stayed close to the log-likelihood values, indicating a small penalty on the model complexity (Figure 5(d)). This small penalty leads to large HMMs. The heuristic score starts decreasing when the number of states arrives at 27 before beginning to oscillate (Figure 5(e)), indicating that further splitting could potentially produce a better model. MDL is a positive value and smaller values are rewarded. It first decreases, but then starts climbing after 15 states (Figure 5(f)). MDL converges faster than STACT and heuristic, meaning that it more heavily penalizes model complexity. Compared with baseline methods, AGSS shows its unique boosting effect on HMM inference, saving much computational time (Figure 5) while providing a higher model quality (Table 3).

VI. Medical Workflow Mining—A Case Study

AGSS is designed to discover workflow models without any prior information. To evaluate the workflow model discovered by AGSS (Figure 6(d)), we compared it to models inferred from STACT (Figure 6(c)), from Disco (Figure 6(b), a process mining tool), and by medical experts using domain knowledge (Figure 6(a)). The model discovery algorithm used in Disco (<https://fluxicon.com/disco/>) is based on a fuzzy workflow mining algorithm [25], rather than an HMM. Comparing the AGSS model to the Disco one shows that AGSS can find better work-flow models than other process mining algorithms, regardless of whether they use HMMs. We tested different process mining algorithms (e.g., alpha, heuristic, and genetic miners [1]) and chose Disco as the baseline because it is the most widely used in

process mining applications. Models discovered from highly-variable real-world processes are usually uninterpretable spaghetti-like diagrams. Models were simplified using Algorithm 3 before comparisons were made. Differences between the models have implications on both medical and model engineering interpretations of the underlying process.

First, we compare general readability. The Disco model (Figure 6(b)) is simple, with transitions between nodes with one activity per node. The reverse transition from *Rectal-Back* to *Palpation-Head* can be confusing, because the secondary survey ends after the back examination. This reverse transition can be misinterpreted as a start-over of the secondary survey, which is not true in practice. The linear structure of the model inferred by STACT is easy to follow and appears like a forward-moving workflow. The linearity comes from STACT allowing multiple observations in the same state. For example, an activity that occurs out of order, either early or late, can be simply represented by allowing it to occur at the “wrong” state. In Disco models, as each activity type is uniquely mapped to one node, such out-of-order activities are handled by adding new transitions to the “wrong” state. In AGSS models, these types of activities are handled differently based on their occurrence distribution. If one activity frequently occurs at multiple locations, then new states will be split to better describe those occurrences. Infrequent repeated occurrences would be considered outliers, and would not be granted new states. The disadvantage of the STACT model is the difficulty of interpreting multiple activity types per state. For example, state #3 in Figure 6(c) had more than four different activities. Another limitation is the large possibility of state selftransitions (i.e., transition from a state to itself), making the model implicit. AGSS explicitly describes the relationship between different activities. It also addresses the multiple occurrence problem not considered in Disco (i.e. *Palpation-H* was described using two different states, #1 and #20 (Figure 6(d))).

Second, we discover more insights after comparing data-derived models to the expert model. Aside from the reverse transitions (from *Rectal-BK* to *Palpation-H*), the Disco model follows the expert model well. This finding, however, does not mean the Disco model is better than the other two data-driven models. Subjectively created by humans, the expert model may not reflect practice. For example, both the expert model and Disco failed to capture the multiple occurrences of *Palpation-H* and *C-Spine-BK* (Figure 7). Unlike the others, STACT and AGSS captured multiple occurrences. For example, *Palpation-H* can be observed in states #1, #5, #9, #14, and #15 in the STACT model, as well as in states #1 and #20 of the AGSS model. While both STACT and AGSS were able to capture the multiple occurrences of activities, they do so very differently. AGSS balances the significance of activities’ multiple occurrences before assigning a new node for them, while the STACT model simply creates new nodes for all multiple occurrences without considering complexity-vs.-accuracy tradeoffs. This explains why we can see a large number of activity recurrences in the STACT model (Figure 6(c)). In comparison, AGSS added only six states (#19-#24 in Figure 6(d)). As an example, state #20 (*Palpation-H*) is split from state #1, and state #21 (*C-Spine-BK*) is split from #16. The splitting is conformant to the distribution of these two activities (Figure 7), as both *Palpation-H* and *C-Spine-BK* occur at different times in the secondary survey. State #1 corresponds to the first cohort of *Palpation-H* while state #20 corresponds to the one close to the end. State #16 maps to the main cohort (between time 150 to 200) of *C-Spine-BK* and state #21 maps to the secondary cohort (between time 0

to 100). A small cohort of *C-Spine-BK* occurs at the end. It is however so small that AGSS did not split it when taking model complexity into account. This way, AGSS identified several discrepancies between practice and the expert model (Table 4). These discrepancies highlighted differences between what was being taught and what is being practiced during trauma resuscitations. These deviations from the expert model may be due to errors committed by the medical team, or may be necessary based on clinical circumstance. AGSS can determine whether deviations are part of a pattern of behaviors or isolated instances. This knowledge helps determine if these differences are acceptable, or if an enhancement of the expert model is needed.

VII. Conclusion

We investigated how to discover a process workflow from process logs using Hidden Markov Models (HMM). To induce a HMM, we proposed a novel inference algorithm guided by trace alignment (AGSS) and compared it to existing state-splitting HMM inference algorithms. We tested AGSS on four real-world medical workflow datasets. Our results show AGSS is not only computationally efficient, but also produces HMMs with high model fidelity and confidence. Our case study on one medical process shows the workflow model discovered by AGSS is more readable and representative than those of existing process mining algorithms. The model induced by AGSS helped discover three discrepancies between the initial knowledge-based process model and actual practice. These discrepancies were analyzed for repairing the knowledge-based model.

Acknowledgment

This paper is based on research supported by National Institutes of Health under grant number 1R01LM011834-01A1.

References

- [1]. Van Der Aalst Wil. Process mining: discovery, conformance and enhancement of business processes. Springer Science & Business Media, 2011.
- [2]. Takami Jun-ichi, and Shigeki Sagayama. 'A successive state-splitting algorithm for efficient allophone modeling.' Acoustics, Speech, and Signal Processing, 1992. ICASSP-92, 1992 IEEE International Conference on. Vol. 1. IEEE, 1992.
- [3]. Herbst Joachim, and Dimitris Karagiannis. 'Integrating machine learning and workflow management to support acquisition and adaptation of workflow models.' Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on. IEEE, 1998.
- [4]. Clarke John R., et al. 'An objective analysis of process errors in trauma resuscitations.' Academic Emergency Medicine 711 (2000): 1303–1310. [PubMed: 11073483]
- [5]. APA Da Silva Gil Aires, and Ferreira Diogo R.. 'Applying hidden Markov models to process mining.' Sistemas e Tecnologias de Informação. AISTI/FEUP/UPF(2009).
- [6]. Lu Xixi, et al. "Handling duplicated tasks in process discovery by refining event labels' International Conference on Business Process Management. Springer International Publishing, 2016.
- [7]. Blum Tobias, et al. 'Workflow mining for visualization and analysis of surgeries.' International journal of computer assisted radiology and surgery 35 (2008): 379–386.
- [8]. Stolcke Andreas, and Omohundro Stephen M.. 'Best-first model merging for hidden Markov model induction.' arXiv preprint cmp-lg/9405017 (1994).

- [9]. Padoy Nicolas, et al. 'Statistical modeling and recognition of surgical workflow.' Medical image analysis 163 (2012): 632–641. [PubMed: 21195015]
- [10]. Singer Harald, and Mari Ostendorf. "Maximum likelihood successive state-splitting." Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on. Vol.2 IEEE, 1996.
- [11]. Ostendorf Mari, and Harald Singer. 'HMM topology design using maximum likelihood successive state-splitting.' Computer Speech & Language 111 (1997): 17–41.
- [12]. Needleman Saul B., and Wunsch Christian D.. 'A general method applicable to the search for similarities in the amino acid sequence of two proteins.' Journal of molecular biology 483 (1970): 443–453. [PubMed: 5420325]
- [13]. Mavromatis Panayotis. 'Minimum description length modelling of musical structure.' Journal of Mathematics and Music 33 (2009): 117–136.
- [14]. Siddiqi Sajid M., Gordon Geoffrey J., and .Moore Andrew W.. "Fast state discovery for HMM model selection and learning." International Conference on Artificial Intelligence and Statistics 2007.
- [15]. Tromp Maarten, Rens Bod, and Aline Honingh. 'Learning Symbolic Music through Hidden Markov Model Induction.' (2012).
- [16]. Rabiner LR. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE, vol.77, no.2, pp. 257–286, 2 1989.
- [17]. Bose RP Jagadeesh Chandra, and van der Aalst Wil MP. 'Process diagnostics using trace alignment: opportunities, issues, and challenges.' Information Systems 372 (2012): 117–141.
- [18]. Yang Sen, et al. "Duration-Aware Alignment of Process Traces' Industrial Conference on Data Mining. Springer International Publishing, 2016.
- [19]. Hsu Pao-Lu, and Herbert Robbins. 'Complete convergence and the law of large numbers.' Proceedings of the National Academy of Sciences 332 (1947): 25–31.
- [20]. Yu Lu, et al. 'Inferring statistically significant hidden Markov models.' IEEE Transactions on Knowledge and Data Engineering 257 (2013): 1548–1558.
- [21]. Kelleher Deirdre C., et al. 'Effect of a checklist on advanced trauma life support workflow deviations during trauma resuscitations without prearrival notification.' Journal of the American College of Surgeons 2183 (2014): 459–466. [PubMed: 24468229]
- [22]. Mans RS, et al. "Application of process mining in healthcare—A case study in a Dutch hospital' International Joint Conference on Biomedical Engineering Systems and Technologies. Springer Berlin Heidelberg, 2008.
- [23]. Shmueli Galit. 'To explain or to predict?.' Statistical science 253 (2010): 289–310.
- [24]. Giraud Christophe. Introduction to high-dimensional statistics. Vol. 138 CRC Press, 2014.
- [25]. Günther Christian W., and Van Der Aalst Wil MP. "Fuzzy mining–adaptive process simplification based on multi-perspective metrics' International Conference on Business Process Management. Springer Berlin Heidelberg, 2007.
- [26]. Yang Sen, et al. 'VIT-PLA: Visual Interactive Tool for Process Log Analysis.' KDD 2016 Workshop on Interactive Data Exploration and Analytics (IDEA 2016).

Figure 1.

(a) The most specific workflow model of process log O ; (b) a typical workflow model discovered based on existing Markov chain modeling methods (probabilities not shown); (c) the possible underlying workflow model that generated O .

Trace Alignment

State-Splitting

Figure 2.

Illustration of the AGSS algorithm, (a) Splitting candidates $C = \{A, B\}$ are selected from the consensus sequence. (b) The splitting starts from the initial model. (c) Activity A is split first because the column in which A is repeated has higher frequency (1.0) than that of activity B (0.6). (d) Activity B is split after A. The splitting stops when all splitting candidates are split.

Figure 3.

(a) Training dataset size plotted against edit distance between $\mathcal{ES}_{half}$ and $\mathcal{ES}$ using different frequency thresholds ϵ ; (b) replay rate on test data. The results are averaged over 30 runs using the Dutch hospital data [22].

Figure 4.

Flawed simplified workflow model with a dangling node S_3 . This model is simplified from (d) in Figure 2 with an arbitrary transition cutoff as 0.35.

Figure 5.

(a) The number of states vs. computation time in seconds; (b) the computation time vs. log-likelihood (i.e., log-likelihood); (c) the number of states vs. log-likelihood; (d) BIC and log-likelihood change as state splitting using STACT; (e) Heuristic score and log-likelihood change as state splitting using Heuristic approach; (f) MDL and log-likelihood change as state splitting using MDL. Experiments were run on Dutch hospital dataset [22].

Figure 6.

(a) Expert model designed by medical experts. (b) Simplified model discovered by Disco (redrawn for clearer view). The number in the node denotes the frequency of activity and the number on the edge denotes the frequency of the transition. (c) HMM induced by STACT where the transitions were simplified by Algorithm 3 and state observations with probability less than 0.1 were filtered. (d) HMM induced by AGSS and simplified by Algorithm 3.

Figure 7.

Trace alignment and distribution of activity “Palpation-H” and “C-Spine-BK”. In the alignment matrix, each row represents a different trace and each color represents an activity type. In the distribution plot (under the alignment), x-axis represents the column number (can be treated as “logic time”) of the alignment matrix and y-axis represents the probability. The trace alignment result is generated using our process visualization tool, VIT-PLA [26].

Table 1.

Summary of Notation

$\mathcal{O}$	A process log containing a set of traces
$\mathcal{O}_j$	A process trace composed of activities
a_i	An activity in the process trace
ℓ	Length (num. of activities) of a process trace
$\mathcal{A}(\mathcal{O})$	Alignment algorithm
$\mathcal{CS}$	A sequence of activities captured from alignment matrix columns
L	Average length of alignment
ϵ	Threshold for selecting $\mathcal{CS}$ activities from alignment matrix
$\mathcal{CS}'$	Filtered $\mathcal{CS}$ based on threshold ϵ
λ	A hidden Markov model
A	Transition probability matrix over states
B	Observation probability distribution in a state
π	Initial state distribution
n	The number of hidden states
m	The number of distinct observation symbols in states
SC	Splitting candidates derived from alignment
N	Total number of splits

Table 2.

Statistics of our four medical process datasets.

Stats \ Dataset	Intubation	Primary	Secondary	Dutch
Num. Patient Records	74	186	122	833
Num. Total Acts	900	1291	3057	24,550
Num. Act Types	15	8	17	18
Longest Trace (Num. Acts)	13	8	51	229
Shortest Trace (Num. Acts)	5	4	7	2
Avg. Num. Acts in Trace	12.16	6.94	25.06	29.47

Comparison of AGSS with existing state-splitting algorithms on four real world medical processes. Model fidelity (M_f) and model confidence (M_c) are scaled by the number of process traces in each process.

Table 3.

	Intubation			Primary Survey			Secondary Survey			Dutch Hospital		
	M_f	M_c	R_r	M_f	M_c	R_r	M_f	M_c	R_r	M_f	M_c	R_r
Markov Chain	-12.8	-12.9	84.3%	-10.00	-10.38	96.6%	-47.59	-44.09	65.8%	-38.8	-37.3	97.6%
AGSS	-12.8	-12.9	84.3%	-9.98	-10.16	96.1%	-45.22	-44.13	65.8%	-32.76	-34.03	97.6%
ML-SSS (0.01)	-10.56	-22.05	98.6%	-9.05	-10.66	100%	-48.79	-60.52	100%	-52.39	-54.66	100%
Heuristic	-11.8	-15.8	84.3%	-8.50	-10.95	96.1%	-47.59	-44.10	65.8%	-38.6	-36.9	97.6%
MDL	-19.45	-19.78	100%	-12.37	-12.97	100%	-64.44	-65.37	100%	-81.34	-82.03	100%
STACT	-12.32	-14.23	98.6%	-9.32	-10.19	100%	-49.22	-58.08	99.2%	-44.86	-47.04	100%

Table 4. Discrepancies between expert model and practical procedures. Medical explanation and model enhancement.

Discrepancies	Medical Explanation	Model Enhancement
C-Spine-Back not only occurs during the back examination but also may occur during the head examination.	To determine whether the patient needs cervical spine (c-spine) support, it may be easier to palpate the c-spine and the neck simultaneously. From a clinical perspective, it is acceptable to perform c-spine assessment at either point in the secondary survey.	C-Spine-BK should be completed during head exam or during back exam. If it occurs at head exam, it is not mandatory to repeat again at back exam.
Palpation-Head not only occurs during the head examination but may also occur during the back examination.	The medical team logrolls the patient to perform the back exam without stressing the spinal column. To perform a complete head exam, medical team members often visualize or palpate the occipital region or back of the head during the logroll, so as to reduce head manipulation.	Palpation-Head can be performed during the back exam.
Palpation-Abdomen, Lower-Extremities, Palpation-Right-Upper-Extremities, Palpation-Right-Upper-Extremities were found to repeat in the process.	To more completely assess the patient, it is acceptable to repeat activities; for example, repeated palpation of bilateral extremities may occur to assess potential swelling or deformities.	The model should be repaired to allow repetitions of activities within a particular body region.