SPARK: A Space-Efficient Smoothing Algorithm for Hidden Markov Models Cover Letter

3419

We thank all reviewers for the time they are investing to review this submission to IJCAI 2025. In this cover letter, we explain how we have addressed all remarks in the reviews to our previous submission to AAAI 2025. First, we address common concerns regarding the performance of SPARK on real-world data and the comparison to other baselines. We then address the concerns of each reviewer in Sections 1–3. We quote reviewer comments in *cyan.*

Performance on real-world data

R1O1 It is unclear how restrictive these assumptions [in synthetic data experiments] are in practice.
R2O2 . . . lacks of real-world validation . . . In the experiments, real-world data is not clear.
R3O2 The proposed algorithm is only demonstrated on toy problems, and it is unclear if it scales to more complex tasks.

We added experiments on real-world forced alignment data. We use a composite HMM for speech-text forced alignment built with the HTK software toolkit, with 5529 states out of which 3204 are emitting, aiming to align speech recordings from the TIMIT corpus. As observations, we extract a standard 39-dimensional feature vector (13 Mel-Cepstrum Cepstral Coefficients, augmented by their first and second order derivatives). We report on our results in Section 5.5.

Comparison to other baselines

R2O1 . . . lacks comparisons to modern baselines . . . neural network-based model like RNN/transformer can also solve the HMM smoothing problem . . . What's the reason author do not compare with these kind of methods?
R4O1 . . . would benefit from a more thorough discussion of related work and a comparison with state-of-the-art methods.

The Island algorithm we are comparing with is the state-of-the-art method for a space-efficient alternative to the forward-backward (F-B) smoothing algorithm; the F-B algorithm remains useful and necessary, despite the advent of other models, and is combined with other models in hybrid solutions. As examples of its continued application, see [Sun *et al.*, 2022; Yao *et al.*, 2021; Zeineldeen *et al.*, 2022]. Our algorithms produce the same exact results as the F-B algorithm. We clarify this matter in the beginning of Section 5.

1 Reviewer #1

R1O2 I still don't know what claims, if any, the authors are making about general interval graphs.

As long as the state space is divisible in halves, SPARK incurs $\mathcal{O}(S)$ space and $\mathcal{O}(S^3T)$ time (worst case). In case an even division is impossible, it divides along the time horizon, yielding a $\log T$ factor in both space and time complexity. We added further details in the analysis SPARK (Section 5.3).

R1O3 It isn't clear that space is a real practical bottleneck.

Space is a bottleneck when applying smoothing in low-memory devices; in addition, space scalability is crucial in data centers and facilitates energy-efficient computations.

R1O4 A number of parameters are only defined once.

We included more parameters, such as α and β, in Table 1.

2 Reviewer #2

R2O3 If the layer of DAGs larger than 40, the memory will getting much larger than "standard" baseline.

The memory remains lower than the standard baseline as Figures 6 & 7 in the paper show.

R2O4 The rationale behind the chosen division step.

The rationale is to evenly divide the HMM state space.

3 Reviewer #3

R3O1 . . . implementation (Alg. 2) is very complex.

The pseudo-code follows the notations for the forward-backward algorithm (Table 2).

References

[Sun *et al.*, 2022] Jinwen Sun, Akash Deep, Shiyu Zhou, and Dharmaraj Veeramani. Industrial system working condition identification using operation-adjusted hidden markov model. *Journal of Intelligent Manufacturing*, 34(6):2611–2624, April 2022.

[Yao *et al.*, 2021] Lin Yao, Yuqi Wang, Xin Wang, and Guowei WU. Cooperative caching in vehicular content centric network based on social attributes and mobility. *IEEE Trans. on Mobile Computing*, 20(2):391–402, 2021.

[Zeineldeen *et al.*, 2022] Mohammad Zeineldeen, Jingjing Xu, Christoph Lüscher, Ralf Schlüter, and Hermann Ney. Improving the training recipe for a robust conformer-based hybrid model. In *Interspeech*, page 1036–1040, 2022.

SPARK: A Space-Efficient Smoothing Algorithm for Hidden Markov Models

Anonymous submission

Abstract

The popularity of large AI models and limited-memory devices calls for space-efficient algorithm design. A prime example of task calling for space-efficient formulations is statistical inference, or *smoothing*, especially with Hidden Markov Models (HMMs), which is cardinal in understanding real-world phenomena. Standard smoothing with HMMs incurs high memory consumption, increasing with the length of the observation sequence. As such models become heavier in computational and memory demands, portable devices face a memory bottleneck in offline inference. In this paper, we propose SPARK (Space-Efficient Forward-Backward), a reformulation of the Forward-Backward smoothing algorithm that eliminates the dependence of its space complexity on the length of the observation horizon in certain classes of HMMs. We showcase SPARK's effectiveness through an experimental study on synthetic and real-world data and analyse the challenges in generalizing our formulation to generic HMMs.

1 Introduction

The Forward-Backward (F-B) algorithm is a dynamic programming technique used in statistical inference, particularly in the context of Hidden Markov Models (HMMs) (Ghahramani 2001). It efficiently computes the posterior probability distribution marginals $\mathbb{P}(X^{(t)}|O^{(1:T)})$ of hidden states given an entire sequence of observations $O^{(1:T)} = o^{(1)}, o^{(2)}, \ldots, o^{(T)}$, a task called *smoothing*. The algorithm involves two passes through the data: the *forward* pass calculates the probability of the observations up to each time step given each possible state, and the *backward* pass computes the probability of the remaining observations given each state. These probabilities are then combined to obtain the marginal probabilities of the hidden states, enabling real-world applications like decoding and parameter estimation in speech recognition (Rabiner 1989), medical treatment (Ludwig et al. 2021), computational biology (Yoon 2009), and natural language processing (Jurafsky and Martin 2000). Smoothing provides more accurate estimations of hidden states at each time step compared to *filtering*, which only considers past observations, and *prediction*, which only considers future observations.

While the F-B algorithm is time-efficient, its space demands grow with respect to the sequence length, or *horizon* T. Prior work proposed the *Island algorithm* (Binder, Murphy, and Russell 1997), which enhances the standard F-B algorithm's memory usage by trading smaller memory for longer running times, yielding space complexity logarithmic in the horizon. We propose a method with space complexity independent of the horizon on a specific class of directed acyclic graphs which commonly appear in real-world applications, and discuss potential future directions in this area.

Table 1: Notations for HMMs.

Symbol	Meaning
X	set of hidden states
S	size of state space
E	number of edges in HMM
$X^{(t)}$	set of reachable states at time t
O	set of observed (emitted) symbols
M	number of observable symbols
$O^{(n:m)}$	observed sequence from index n to m
$o^{(t)}$	observed symbol o at time t
T	horizon length

2 Hidden Markov Models

We introduce HMMs using the notations listed in Table 1. Formally, an HMM is defined by:

1. A set of variables $X = \{x_1, x_2, \ldots, x_S\}$ representing states, also known as *latent variables* or *hidden variables*. A state can be either *emitting* or *non-emitting*. An emitting state can observe some or all symbols, while a non-emitting state cannot observe any symbols. Without loss of generality, we set every state to be emitting.
2. A set of symbols $O = \{o_1, o_2, \ldots, o_M\}$ that are observable at an emitting state.
3. A transition probability distribution $\mathbb{P}(x'|x)$ that expresses the distribution of next state variable given the current state. HMMs conform to the Markov property, by which a transition from one state depends only on that current state. Transition probabilities are represented by a matrix A, where $A_{x_i,x_j} \coloneqq \mathbb{P}(x_i|x_j)$.
4. An observation (*emission*) probability distribution $\mathbb{P}(o|x)$ describing the distribution of outputs given a current state. We define $B_x(o) \coloneqq \mathbb{P}(o|x)$.
5. An initial probability distribution $\pi(X)$ over X, expressing the likelihood that a sequence starts at each state.

To produce a T-step observation sequence, we draw the initial state $x^{(1)}$ from π. In each step $1 \le t \le T$, hidden state $x^{(t)} \in X$ yields observation $o^{(t)} \in O$ with probability $B_{x^{(t)}}(o^{(t)})$; the next state $x^{(t+1)}$ is drawn from $A_{\cdot, x^{(t)}}$.

3 The Forward-Backward Algorithm

The Forward-Backward algorithm computes the posterior distribution $\mathbb{P}(X^{(t)}|O^{(1:T)})$, or smoothing, by propagating *forward* and *backward* messages and relying on Bayes' rule:

$$\begin{aligned} \mathbb{P}(X^{(t)}|O^{(1:T)}) &= \mathbb{P}(X^{(t)}|O^{(1:t)}, O^{(t+1:T)}) \\ &\propto \mathbb{P}(X^{(t)}|O^{(1:t)})\, \mathbb{P}(O^{(t+1:T)}|X^{(t)}) \\ &\propto \mathbb{P}(X^{(t)}, O^{(1:t)})\, \mathbb{P}(O^{(t+1:T)}|X^{(t)}) \end{aligned} \tag{1}$$

The joint (forward) probability, $\alpha_t(X^{(t)}) = \mathbb{P}(X^{(t)}, O^{(1:t)})$, is given by the recursive update (Rabiner 1989):

$$\begin{aligned} &\alpha_t(X^{(t)}) = \mathbb{P}(X^{(t)}, O^{(1:t)}) \\ &= \sum_{X^{(t-1)}} \mathbb{P}(X^{(t)}, X^{(t-1)}, O^{(1:t)}) \\ &= \mathbb{P}(o^{(t)}|X^{(t)}) \sum_{X^{(t-1)}} \mathbb{P}(X^{(t)}|X^{(t-1)}, O^{(1:t)}) \mathbb{P}(X^{(t-1)}|O^{(1:t-1)}) \\ &= \mathbb{P}(o^{(t)}|X^{(t)}) \sum_{X^{(t-1)}} \mathbb{P}(X^{(t)}|X^{(t-1)}) \alpha_{t-1}(X^{(t-1)}) \end{aligned} \tag{2}$$

At the base case, $\alpha_1(X^{(1)}) = \pi(X)\, \mathbb{P}(O^{(1)}|X^{(1)})$. On the other hand, the conditional (backward) probability, $\beta_t(X^{(t)}) = \mathbb{P}(O^{(t+1:T)}|X^{(t)})$, is given as (Rabiner 1989):

$$\begin{aligned} &\beta_t(X^{(t)}) = \mathbb{P}(O^{(t+1:T)}|X^{(t)}) \\ &= \sum_{X^{t+1}} \mathbb{P}(O^{(t+1:T)}, X^{(t+1)}|X^{(t)}) \\ &= \sum_{X^{t+1}} \mathbb{P}(O^{(t+2:T)}|X^{(t+1)})\, \mathbb{P}(o^{(t+1)}|X^{(t+1)})\, \mathbb{P}(X^{(t+1)}|X^{(t)}) \\ &= \sum_{X^{t+1}} \beta_{t+1}(X^{(t+1)})\, \mathbb{P}(o^{(t+1)}|X^{(t+1)})\, \mathbb{P}(X^{(t+1)}|X^{(t)}) \end{aligned} \tag{3}$$

At the base case, $\beta_T(X^{(T)}) = 1$. Putting the two equations together, $\mathbb{P}(X^{(t)}|O^{(1:T)}) \propto \alpha_t(X^{(t)})\beta_t(X^{(t)})$. Besides, when computing the summation terms in Equations (2) and (3), we may encounter underflow by multiplication. To prevent such numerical instability, we use the logarithm scale (Mann 2006), yielding log-probabilities.

Using the recursions in Equations (2) and (3), we compute and store α_t, β_t given the boundary conditions α_0, β_T and combine them to get the posterior marginals. This approach requires $\mathcal{O}(S^2T)$ time, which becomes $\mathcal{O}(ET)$ in case all HMM edges are known in advance; the space requirement is $\mathcal{O}(ST)$, as we need to tabulate the forward and backward tables to produce marginals upon request.

4 The Island Algorithm

Here, we outline the Island algorithm (Binder, Murphy, and Russell 1997), which lowers the memory usage of the standard F-B algorithm to a logarithmic factor in horizon length, and conduct an experiment that illustrates its behavior.

4.1 Design

In an effort to render the F-B algorithm space-efficient as T grows, Binder, Murphy, and Russell (1997) proposed a formulation that trades off runtime for working space. Given the boundary conditions α_1 and β_T, the algorithm iteratively computes α_t and β_t, keeps k vectors of each as checkpoints in $\mathcal{O}(kS)$ space, and recursively uses those checkpoints as boundary conditions for further recursion. For example, consider $T = 24$ and $k = 1$; knowing the boundary conditions α_1, β_{24}, we iteratively compute $\alpha_2, \alpha_3, \ldots, \alpha_{24}$ and $\beta_{23}, \beta_{22}, \ldots, \beta_1$, keep α_{12} and β_{12} in the working space, calculate $\mathbb{P}(X^{(12)}|O^{(1:T)})$, and use each of the pairs α_1, β_{12} and α_{12}, β_{24} similarly in the next level of recursion in a depth-first fashion. Using checkpoints that split the observation sequence in almost equal parts, the recursion has depth $\mathcal{O}(\log_k T)$. To manage the depth-first recursion, we need to keep at most $k \log_k T$ checkpoints in memory, hence $\mathcal{O}(Sk \log_k T)$ space. Still, due to the recursive recomputations, the runtime grows to $\mathcal{O}(S^2 T \log_k T)$.

Algorithm 1 shows a pseudocode, which applies the forward-backward computation using a log-scale for the sake of numerical stability, with α_1 and β_N as in Section 3.

Algorithm 1: Island Algorithm

```
1:  procedure ISLAND($F_1, B_N, O, \ell, h$)
2:      $T \leftarrow h - \ell + 1$                       ▷ number of observations in a segment
3:      if $T = 1$ then                                            ▷ Base case
4:          $p \leftarrow normalize(F_1 * B_N)$           ▷ compute probability distribution
5:          $handle(p)$                              ▷ pass the message for further handle
6:          delete $p$                                         ▷ discard immediately
7:          return
8:      $k \leftarrow getNumChckpts(T)$   ▷ for constant $k$, function independent of $T$
9:      $s \leftarrow \lceil \frac{T}{k} \rceil$                                           ▷ chunk size
10:     $Forw \leftarrow [F_1], Forw\_idx \leftarrow [\ell], \alpha \leftarrow F_1$
11:     for $t \leftarrow \ell + 1 \rightarrow h$ do                           ▷ forward computation
12:         $\alpha_j \leftarrow \sum_{i \in X} \alpha_i * A_{i,j} * B_j(o^{(t)}),\ \forall j \in X$
13:         if $(t-\ell+1) \% s = 0$ and $t \neq h$ or $s = 0$ or $T = 2$ then ▷ store checkpoints
14:             $append(Forw, \alpha), append(Forw\_idx, t)$
15:     $Back \leftarrow [], Back\_idx \leftarrow [], \beta \leftarrow B_N, prev \leftarrow B_N$
16:     for $t \leftarrow h - 1 \rightarrow \ell$ do                          ▷ backward computation
17:         $\beta_i \leftarrow \sum_{j \in X} \beta_j * A_{i,j} * B_j(o^{(t+1)}),\ \forall i \in X$
18:         if $(t - \ell + 1) \% s = 0$ and $t \neq \ell$ or $s = 0$ or $T = 2$ then
19:             $append(Back, \beta), append(Back\_idx, t)$
20:     $reverse(Back), reverse(Back\_idx)$
21:     $append(Back, B_N), append(Back\_idx, h)$
22:     for $i \leftarrow 0 \rightarrow |Forw| - 1$ do                          ▷ recursive calls
23:         ISLAND($Forw_i, Back_i, O, Forw\_idx_i, Back\_idx_i$)
```

4.2 Experiments and Results

We experimented with random synthetic HMMs to illustrate the Island Algorithm's behavior. We evaluate four cases of k:

- $k = T$ (linear): this set up is equivalent to the standard Forward-Backward algorithm.
- $k = 1$ (logarithmic): this variant splits the observation sequence into two segments in each recursive call, raising the time complexity to $\mathcal{O}(S^2 T \log_2 T)$ and reducing the working space to $\mathcal{O}(S \log_2 T)$.

- $k = \sqrt{T}$ (radical): this variant reduces space to $\mathcal{O}(S\sqrt{T})$ and increases the runtime by a factor of 2.
- $k = \sqrt{t}$ (dynamic radical): This variant dynamically attunes the number of checkpoints to the length of the observation sequence at each depth of recursion, aiming to reduce memory usage as the recursion deepens while slightly sacrificing the running time.

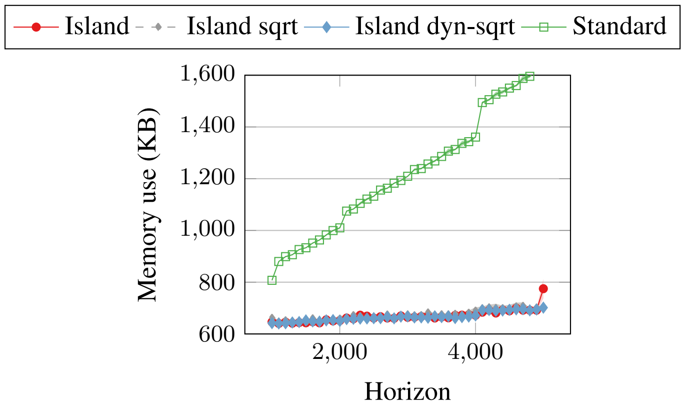

Figure 1: Memory use on random 10-state HMM.

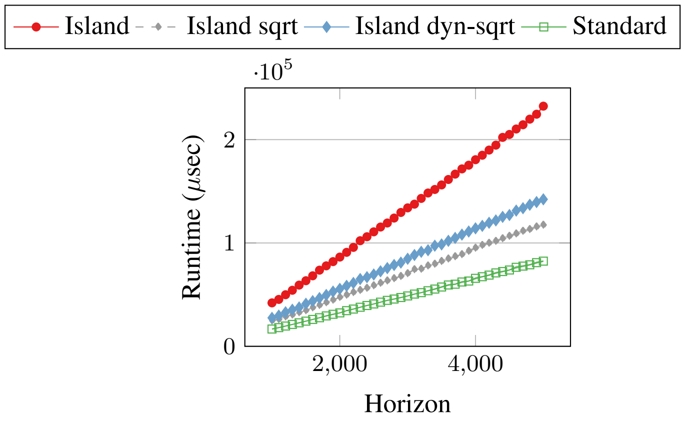

Figure 2: Runtime on random 10-state HMM.

Figure 1 plots the memory use with a 10-state randomized HMM whose transition probability distribution is randomly generated. Each setup ran 20 times independently. Observably, the memory use of the linear version is significantly higher than that of the others. Figure 2 plots the runtime on the same configurations. As expected, the linear version yields the lowest runtime, while the logarithmic one is slowest. As the number of states is small, different numbers of checkpoints k do not yield a large difference in memory, yet runtimes are distinctly separated, with the dynamic radical variant performing a bit worse than the radical one. We conduct further experiments on larger networks in Section 5.

5 The SPARK Algorithm

While eschewing the linear dependence on horizon length, the space complexity of the Island algorithm retains a factor logarithmic in horizon length. The questions arises, can we fully eliminate this dependence? We propose a method that is *horizon-independent* on Directed Acyclic Graphs (DAGs) under some constraints. We introduce DAG variants, we present our method and analyze its space-time complexity. Lastly, we evaluate the proposed algorithm in juxtaposition to the Standard algorithm and the Island algorithm.

5.1 Directed Acyclic Graphs

HMMs forming Directed Acyclic Graphs (DAGs) appear in real-world applications such as *forced alignment* in speech recognition (Moreno and Alberti 2009). In a DAG $G = (V, E, W)$, if there is a directed path from a node u to another node v, then there is no directed path from v to u. *Layer* $\mathcal{L}_\ell$ of a DAG $G = (V, E, W)$ with respect to a set of initial states $\mathcal{S}$ is the set of all nodes (i.e., states) v in G that are reachable by a directed path of *at most* ℓ steps from any $s \in \mathcal{S}$. We distinguish the following classes of DAGs:

- In a *well-layered* graph, each node v in layer $\mathcal{L}_\ell$ with respect to a set of initial states $\mathcal{S}$ only connects to nodes (i.e., states) in layers $\mathcal{L}_{\ell-1}$ and $\mathcal{L}_{\ell+1}$.
- In an *interval-i* graph, each node v in layer $\mathcal{L}_\ell$ may connect to nodes (i.e., states) in layer $\mathcal{L}_{\ell-i}$ to $\mathcal{L}_{\ell+i}$. Well-layered graphs are interval-1 graphs.

We use the term *band* to denote a set of states reachable in a number of steps. A *band* differs from a *layer* used to define interval-i graphs as it considers *any* number of steps to reach a node, whereas a layer considers the maximum.

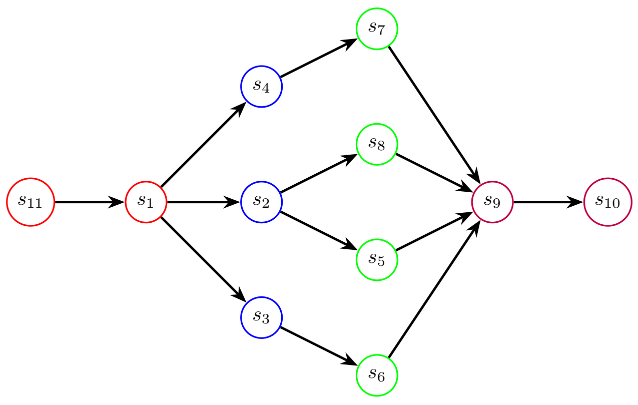

Figure 3: Bands $\mathcal{S}_3$ (blue) and $\mathcal{S}_4$ (green) band split the HMM into their ancestors in bands $\mathcal{S}_1$ and $\mathcal{S}_2$ (red) and descendants in bands $\mathcal{S}_5$ and $\mathcal{S}_6$ (purple).

5.2 Formulation

Instead of recursively splitting T, we find a timestamp t such that the band of states reachable in t steps, $\mathcal{S}_t$, divides the HMM graph into two subgraphs comprising roughly the same number of states. Let A_t and D_t be the sets of ancestors and descendants of $\mathcal{S}_t$:

$$A_t = \{v \in V : v \text{ can be reached in } i \text{ steps}, i \le t\}$$
$$D_t = \{v \in V : v \text{ can be reached in } i \text{ steps}, t < i \le T\}$$

To achieve balance, we pick a t that minimizes the quantity $\max\{|A_t|, |D_t|\}$. A similar method has been used in Viterbi decoding in (Ciaperoni et al. 2022), yet Viterbi decoding aims to find a path, hence suffices with a medially positioned edge from which to traverse two state space regions that enclose the respective sub-paths, whereas the F-B algorithm aims to compute probability distributions over all states, hence requires a band of nodes to precisely slice the state space in two components from which it can derive missing probabilities. Figure 3 shows an example of such balanced division. This division is justified as the posterior probability comprises $\mathbb{P}(X^{(t)}, O^{(1:t)})$, which only depends on nodes reachable in $i \leq t$ steps, and $\mathbb{P}(O^{(t+1:T)}|X^{(t)})$, which only depends on nodes reachable in $i > t$ steps.

Algorithm 2: SPARK

```
1: procedure SPARK($X, O, \pi, F_0, B_n$)
2:     $T \leftarrow |O|, \alpha_s \leftarrow 0, \forall s \in X$
3:     $\alpha_s \leftarrow \pi_s * B_s(o^{(1)}), \forall s \in X$ if $\pi \neq$ NULL else $\alpha \leftarrow F_0$
4:     $t_{opt} \leftarrow -1, v_{opt} \leftarrow \infty, d_1 \leftarrow \varnothing, d_2 \leftarrow \varnothing, \alpha_1, \alpha_2$
5:     for $t \leftarrow 2 \rightarrow T$ do
6:         $anc \leftarrow \varnothing, des \leftarrow \varnothing, d'_1 \leftarrow \varnothing, d'_2 \leftarrow \varnothing, \alpha'_s \leftarrow 0, \forall s \in X$
7:         for $j \in X$ do
8:             for $i \in X$ do
9:                 $\alpha'_j \leftarrow \alpha'_j + \alpha_i * A_{i,j} * B_j(o^{(t)})$
10:                if $\alpha_i > 0$ and $A_{i,j} > 0$ then    ▷ update ancestors & descendants
11:                    $anc \leftarrow anc \cup (A(i) \cap X)$
12:                    $des \leftarrow des \cup (D(j) \cap X)$
13:                    $d'_1 \leftarrow d'_1 \cup \{i\}, d'_2 \leftarrow d'_2 \cup \{j\}$
14:        $v \leftarrow$ max$(|anc|, |des|)$
15:        if $v < v_{opt}$ then    ▷ update best choice
16:            $v_{opt} \leftarrow v, t_{opt} \leftarrow t-1, d_1 \leftarrow d'_1, d_2 \leftarrow d'_2, \alpha_1 \leftarrow \alpha, \alpha_2 \leftarrow \alpha'$
17:        $\alpha \leftarrow \alpha'$
18:    $\beta_2 \leftarrow B_n$
19:    for $t \leftarrow T-1 \rightarrow t_{opt}+1$ do    ▷ only compute up to chosen band
20:        $\beta_{2_i} \leftarrow \sum_{j \in X} \beta_{2_j} * A_{i,j} * B_j(o^{(t+1)}), \forall i \in X$
21:    $\beta_{1_i} \leftarrow \sum_{j \in X} \beta_{2_j} * A_{i,j} * B_j(o^{(t_{opt}+1)}), \forall i \in X$
22:    $O_{anc} \leftarrow O^{(1:t_{opt})}, O_{des} \leftarrow O^{(t_{opt}+1:T)}$
23:    $X_{anc} \leftarrow \bigcup_{s \in d_1} collect\_anc(s, t_{opt}) \cup d_1$    ▷ collect ancestors
24:    $X_{des} \leftarrow \bigcup_{s \in d_2} collect\_desc(s, T-t_{opt}) \cup d_2$    ▷ collect descendants
25:    if $|O_{anc}| > 1$ then    ▷ recursion call
26:        SPARK($X_{anc}, O_{anc}, \pi, F_0, \beta_1$)
27:    $p_1 \leftarrow normalize(\alpha_1, \beta_1), p_2 \leftarrow normalize(\alpha_2, \beta_2)$
28:    if $|O_{des}| > 1$ then
29:        SPARK($X_{des}, O_{des}$, NULL, $\alpha_2, B_n$)
```

In each recursive call, we perform standard F-B, find the timestamp t_{opt} that minimizes $\max\{|A_t|, |D_t\}$, and store the computed forward and backward probability values, $\alpha_{t_{opt}}$, $\beta_{t_{opt}}$, $\alpha_{t_{opt}+1}$, $\beta_{t_{opt}+1}$, only at t_{opt} and its successor $t_{opt}+1$. We use these values to compute the posterior probability at t_{opt} and $t_{opt}+1$, which act as boundary conditions for the subsequent level of recursion. In the recursion's base case, the HMM contains states belonging to a single band. Algorithm 2 lists the pseudocode of the ensuing *Space-Efficient Forward-Backward* (SPARK) algorithm and Table 2 gathers the used notations. We update forward probabilities α as the standard solution does (Line 9), yet only keep the values at the preceding timestamp to compute those in the next one, instead of tabulating all of them. We update the chosen step t_{opt} while computing probabilities and keep the probability vectors at that band and its successor (Line 16). Then we only need to compute backward probabilities up to timestamps $t_{opt}+1$ (Line 20) and t_{opt} (Line 21). Eventually, the recursion proceeds to the sets of ancestors (Line 26) and descendants (Line 29). To facilitate the computation, we precompute, for each state, the full sets of its T-hop-ancestors and T-hop-descendants, which we use to maintain the current band's ancestors and descendants (Lines 11–12), as well as the bands of its ancestors and descendants reachable in exactly b steps, $b \in \{1, \ldots, T\}$, which we use to maintain bands reachable in t and $t+1$ steps (Lines 23–24).

Table 2: Notations for SPARK.

Symbol	Meaning
α_s	forward probability of state s
β_s	backward probability of state s
t_{opt}	chosen band step
T	length of considered observation sequence
anc	set of ancestors of the current band
des	set of descendants of the current band
d_1	band of states reachable in t_{opt} steps
d_2	band of states reachable in $t_{opt}+1$ steps
α_1	forward probabilities at t_{opt}
α_2	forward probabilities at $t_{opt}+1$
β_1	backward probabilities at t_{opt}
β_2	backward probabilities at $t_{opt}+1$
π_s	initial probability of the state s
X	set of considered states
O	observation sequences under consideration
F_0	boundary forward probability
B_n	boundary backward probability
X_{anc}	set of ancestor states for left recursion
X_{des}	set of descendant states for right recursion

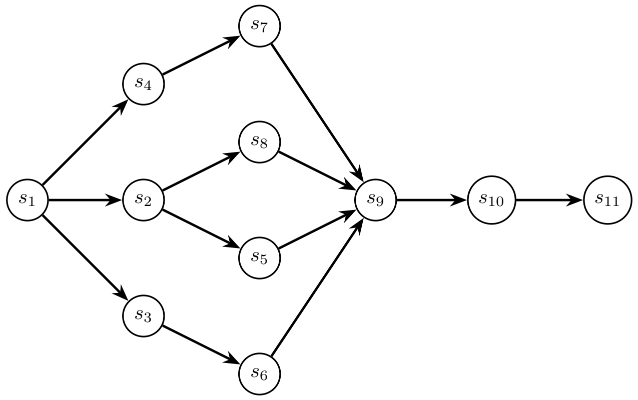

Figure 4: Division; bands $\mathcal{S}_2$–$\mathcal{S}_3$ yield 1 ancestor and 3 descendants; bands $\mathcal{S}_3$–$\mathcal{S}_4$ yield 4 ancestors and 2 descendants.

5.3 Analysis

Each recursion in SPARK requires $\mathcal{O}(S)$ space as it maintains two instances of α and β values per state. On a well-layered graph with initial states at nodes without incoming edges, we can divide the set of latent variables into two parts, so that the number of states before and after the dividing bands are of size at most $\frac{S}{2}$. For instance, in Figure 4, splitting either at the 2nd and 3rd or at the 3rd and 4th bands yields subsets of less than half the initial size.

However, on an interval-i graph, the division may lead to overlaps as a state may participate in both subproblems. For example, in Figure 5, a division at the 3rd and 4th bands yields a subproblem comprising states $\{s_{11}, s_1, s_4, s_2, s_3, s_7, s_9, s_{10}\}$ and one with states $\{s_7, s_8, s_5, s_6, s_9, s_{10}\}$. As s_9, s_{10} and s_7 appear in both subproblems, the state space sizes add up to more than S.

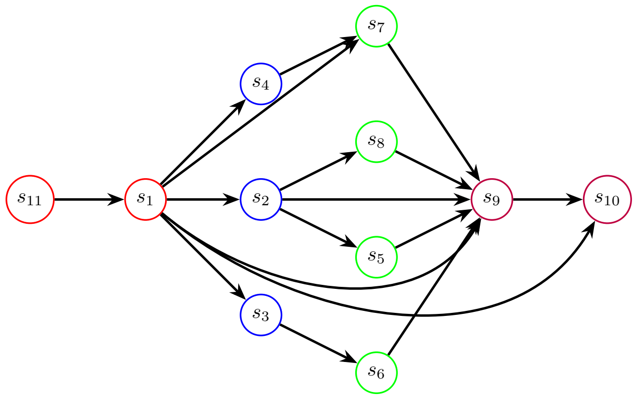

Figure 5: Division band containing overlapping states.

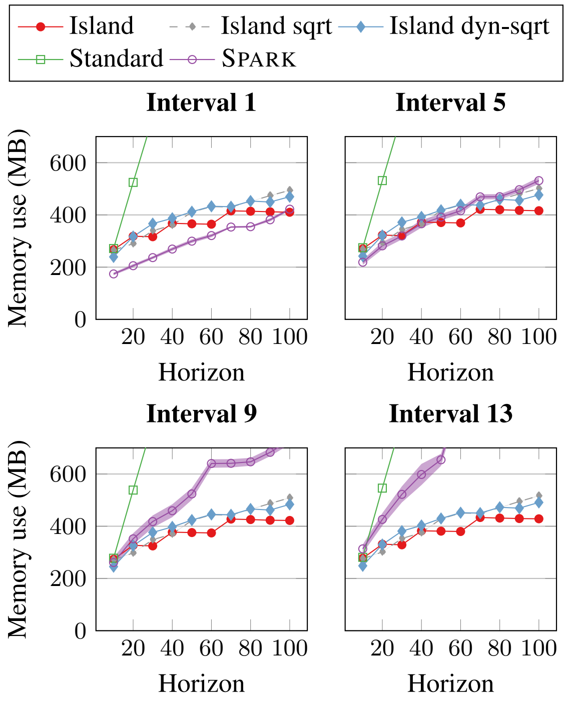

Figure 6: Memory use, 300-state inteval-i graphs.

The recursion depth is $\log S$ while in each iteration we spend $\mathcal{O}(S^3T)$ time. The additional S factor compared to the standard solution arises due to set-union operations. The runtime then is $\mathcal{O}\left(\sum_{i=1}^{\log S} 2^i \left(\frac{S}{2^i}\right)^3 T\right) = \mathcal{O}(S^3T)$ ($\mathcal{O}(SET \log S)$ in the edge-aware case). Further, time complexity is affected by the set of initial states. In an extreme case, if each state is an initial state, every state with an incoming edge is in band S_2. In deeper levels of the recursion, there will be more than $\frac{S}{2}$ states and the divide-and-conquer approach at best degenerates to an Island-like algorithm with $\mathcal{O}(S \log T)$ space and $\mathcal{O}(S^3T \log T)$ time.

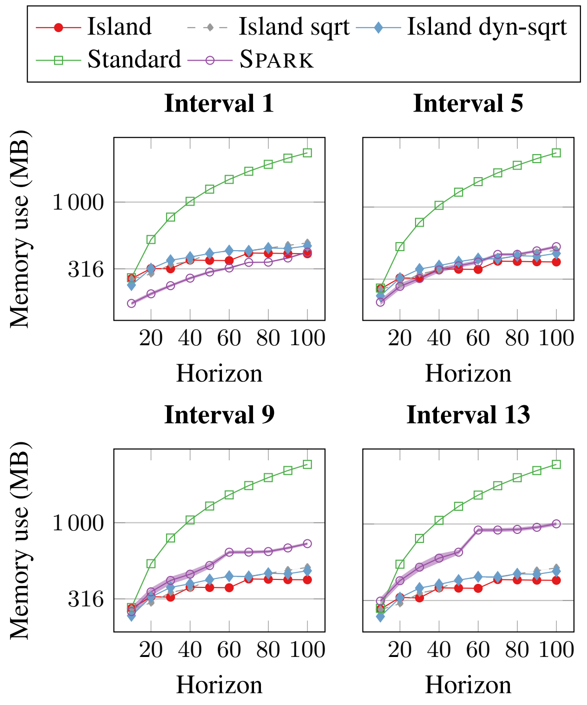

Figure 7: Memory use, 300-state inteval-i graphs (log scale).

5.4 Experiments and Results

To create a well-layered graph, we first allocate each state uniformly at random to a layer, while guaranteeing that each layer has at least one state; that results in layers of 2–3 states; then we connect each state to each state in the following layer with probability 0.9. The first layer has no incoming edges and the last has no outgoing edges. The high probability of 0.9 ensures a path from the first to the last layer exists. To form an interval-i graph for $i > 1$, we iterate through the layers in a well-layered graph and add cross-layer edges within the interval range. To do so, we assign a variable d to each state s, initially set to its distance from layer $\mathcal{L}_1$. Given a state u in layer $\mathcal{L}_k$, we add an edge from a state v in a layer from $\mathcal{L}_{k-i}$ to $\mathcal{L}_{k-1}$ to u, with probability 0.5, if $d_v \geq k - i$, and update d_u to $d_v + 1$; thus we ensure u is reached in at least $k - i + 1$ steps. We set all states in $\mathcal{L}_1$ to be initial states with uniform probability. We compare Spark to standard Forward-Backward and the variants of the Island algorithm we have introduced on so formed synthetic interval-i graphs of 300 states and 120 layers. We repeat each measurement 20 times and we plot 95% confidence intervals. Figures 6–7 plot memory usage and Figures 8–9 runtime vs. horizon length, in linear and logarithmic axes.

On well-layered (interval-1) graphs, Spark uses the least memory while its runtime is close to the standard. Still, as the interval grows, a state may appear in several bands, hence the performance of Spark worsens. On the other hand, with a relatively small interval (e.g., 5), Spark's memory use is as the same level as that of Island, while requiring much less runtime. Notably, despite its horizon-independent space complexity, Spark consumes more memory as the horizon grows, as it explores a larger part of the sate space. Among Island algorithm variants, the dynamic radical solution consumes slightly less memory than plain radical while incurring a more notable runtime disadvantage; thus, the dynamic approach fails to improve much on the radical solution.

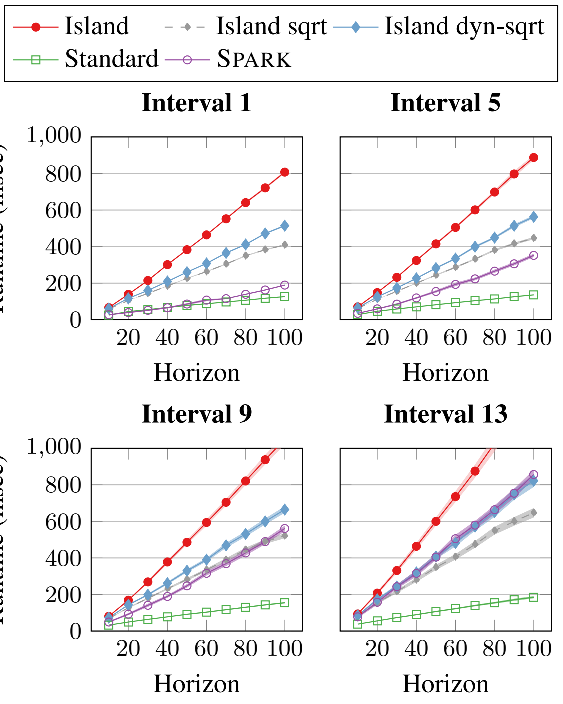

Figure 8: Runtime, 300-state inteval-i graphs.

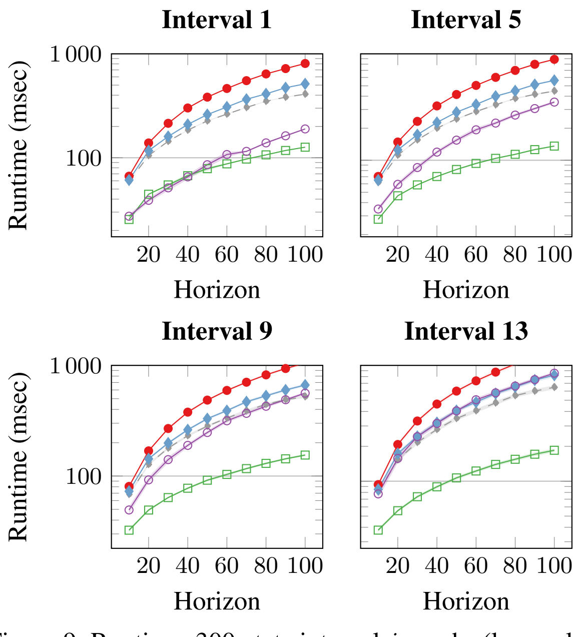

Figure 9: Runtime, 300-state interval-i graphs (log scale).

In terms of runtime, while the Island algorithm has time complexity $\mathcal{O}(S^2 T \log T)$ while SPARK has $\mathcal{O}(S^3 T)$, the observed runtime of SPARK is much better. This is due to the fact that the initial states are only those with no incoming edges. Therefore, only a few states are participate in the union operations at each timestamp, which can be treated as constant. Next, we investigate how the algorithm performs when there are more starting vertices.

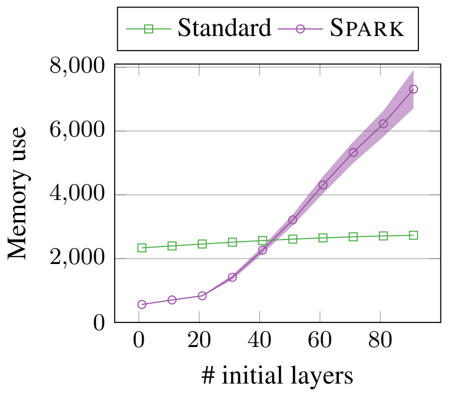

Figure 10: Memory use vs. # of initial-state layers.

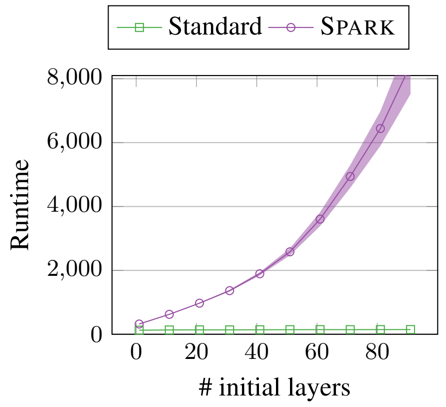

Figure 11: Runtime vs. # of initial-state layers.

On a 300-state well-layered graph divided into 120 layers, we let the first k layers serve as initial states and fix the horizon at 10. Figures 10 and 11 visualize the memory usage and runtime vs. k. As k rises, the union operator at each timestamp t involves more states, hence the runtime grows as a quadratic function. The memory usage starts worsening relatively to the standard solution when k is around 40 layers and presents linear growth thereafter.

6 Additional Considerations

In this section, we discuss some concerns that arise from this work and their potential remedies.

Additional S factor. SPARK incurs an additional time-complexity factor of S compared to standard Forward-Backward to maintain dividing state sets. Still, we may drop the time complexity of the union operator to $\alpha(S)$ where α is the inverse Ackermann (Ackermann 1928) function, using a disjoint-set data structure (Tarjan and van Leeuwen 1984) instead of a hash table (Mehta and Sahni 2004).

Pre-processing Space Complexity. To ensure set computations that avoid overlapping nodes, SPARK maintains sets of ancestors and descendants for each state, yielding $\mathcal{O}(S^2)$

space complexity if all are materialized. We may use counters and addition instead of set union operations, at the cost of ignoring overlapping nodes.

Self-loops and Cyclic Graphs. A self-loop state is a state with an edge to itself. If such a state is reachable at time t, it is also reachable at time $t+1, t+2, \ldots$, hence appears in bands $\mathcal{S}_t, \mathcal{S}_{t+1}, \ldots$; then t_{opt} is likely to be at t, since sets of ancestors (and descendants) are accumulated. In the worst case, no t_{opt} divides the state space satisfactorily. Cycles longer than self-loops incur a similar effect, yet the accumulation occurs non-consecutively, with a gap depending on cycle length; as cycles get longer, the problem attenuates.

An Attempt to Delimit Revisits. In the real world, each state emits sparingly—e.g., in a spoken utterance, the same phoneme is repeated sporadically. To facilitate divide-and-conquer in the presence of cycles, we tried constraining the number of times each state is visited using counters. Yet that does not suffice. Figure 12 shows an example that explains why, with initial states s_1, s_2 and s_3 and two visits allowed per state. Whenever we encounter a state during the forward-backward recursion, we increase its counter. At $t = 3$ the counter of s_{10} increases to 1 due to path $\{s_1, s_4, s_{10}\}$; at $t = 4$ the same counter grows to 2 due to paths $\{s_2, s_5, s_7\}$ and $\{s_1, s_4, s_{10}\}$. Then, at $t = 5$, path $\{s_3, s_6, s_8, s_9\}$ cannot extend to s_{10}, as its counter has reached 2.

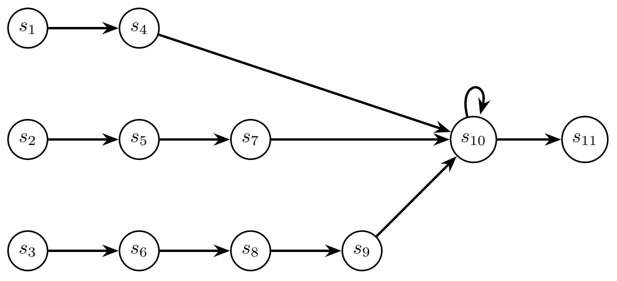

Figure 12: Example with revisit constraint.

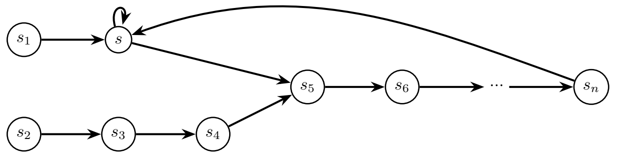

Figure 13: Example with revisit constraint—paths.

Given the insufficiency of state counters, we considered examining each path to a state separately. However, the F-B algorithm only maintains aggregate information about partially overlapping paths reaching the same state; thus, to compute the probability at a state, we should reconsider previous timestamp values. For instance, in Figure 13, with a maximum visit constraint 2, we can reach s_n at time t via two paths: $\{s_2, s_3, \ldots, s_n\}$ and $\{s_1, s, s, s_5, \ldots, s_n\}$. At time $t+1$, revisiting s from s_n violates the visit constraint on the latter path, but not on the former. Since we derived α in the forward pass to s_n using both paths, we should recompute $\alpha_3[s_5]$ excluding path $\{s_1, s, s, s_5, \ldots, s_n\}$ to derive $\alpha_n[\mathrm{S}]$; such reconsideration contradicts the Markovian property (Markov and Nagorny 1988).

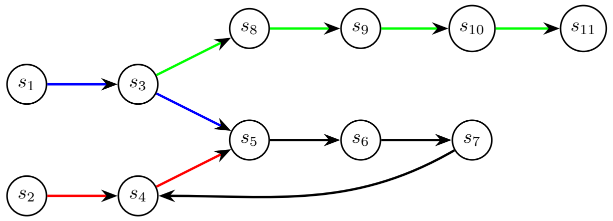

Figure 14: Example with revisit constraint—Viterbi.

Viterbi Decoding with a Visit Constraint. Since multiple paths reaching a same state pose a problem, we contemplated adding a visit constraint to the Viterbi decoding algorithm (Forney 2005), which finds a path in an HMM that best explains an observed sequence; instead of summing over paths to a state, it picks the best. However, as Figure 14 illustrates, the Viterbi algorithm also fails to admit a visit constraint. Starting from s_1 or s_2 with visit constraint 1, assume path $\{s_2, s_4\}$ is preferable to $\{s_1, s_3\}$, thus at $t = 3$, state s_5, we pick the latter (red) path over the former. At $t = 6$, we cannot revisit s_4, as it has reached one visit, hence opt for path $\{s_1, s_3, s_8, s_9, s_{10}, s_{11}\}$. Path $\{s_1, s_3, s_5, s_6, s_7, s_4\}$ may be the best, yet was dismissed at $t = 3$. In hindsight, the problem we try to address generalizes the HAMILTONIANPATH problem, which is to decide whether a path exists that visits each vertex of a graph exactly once. We reduce HAMILTONIANPATH to Viterbi decoding with a visit constraint by setting all transition and emission probabilities to 1, path length $T = |V| - 1$, and visit constraint to 1. Then a Viterbi path is a Hamiltonian path, and vice versa. As HAMILTONIANPATH is NP-complete (Karp 1972), no polynomial-time algorithm solves Viterbi decoding with a visit constraint unless P=NP.

7 Conclusion

The space complexity of the standard forward-backward algorithm for HMM-based statistical inference grows linearly in horizon length, restricting the algorithm's use in low-memory devices and applications with large horizons. An alternative, the Island algorithm, attains space complexity with logarithmic dependence on horizon length via a divide-and-conquer strategy. We proposed the Space-Efficient Forward-Backward (SPARK) algorithm, which divides the HMM state space instead of the observation sequence, yielding space complexity independent of horizon length on directed acyclic graphs. We have shown that SPARK achieves significant space savings without compromising runtime under the realistic assumption that the HMM model is well-layered with a few states serving as initial states of the observation sequence. We also explored SPARK's applicability on HMMs with cross-layer connections and the challenges of its application to general cyclic graphs.

References

Ackermann, W. 1928. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99: 118–133.

Binder, J.; Murphy, K. P.; and Russell, S. 1997. Space-Efficient Inference in Dynamic Probabilistic Networks. In *15th Intl Joint Conference on Artificial Intelligence, IJCAI*, 1292–1296.

Ciaperoni, M.; Gionis, A.; Katsamanis, A.; and Karras, P. 2022. SIEVE: A Space-Efficient Algorithm for Viterbi Decoding. In *Intl Conf. on Management of Data, SIGMOD*, 1136–1145.

Forney, G. D. 2005. The Viterbi Algorithm: A Personal History. *ArXiv*, abs/cs/0504020.

Ghahramani, Z. 2001. *An introduction to hidden Markov models and Bayesian networks*, 9–42. World Scientific Publishing Co., Inc.

Jurafsky, D.; and Martin, J. H. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. USA: Prentice Hall PTR, 1st edition.

Karp, R. M. 1972. *Reducibility among Combinatorial Problems*, 85–103. Springer US.

Ludwig, R.; Pouymayou, B.; Balermpas, P.; and Unkelbach, J. 2021. A hidden Markov model for lymphatic tumor progression in the head and neck. *Scientific Reports*, 11(1).

Mann, T. P. 2006. Numerically Stable Hidden Markov Model Implementation. *An HMM scaling tutorial*, 1–8.

Markov, A.; and Nagorny, N. 1988. *The Theory of Algorithms*. Mathematics and its Applications. Springer Dordrecht. Originally published in Russian.

Mehta, D. P.; and Sahni, S. 2004. *Handbook Of Data Structures And Applications (Chapman & Hall/Crc Computer and Information Science Series.)*. Chapman & Hall/CRC. ISBN 1584884355.

Moreno, P. J.; and Alberti, C. 2009. A factor automaton approach for the forced alignment of long speech recordings. In *IEEE Intl Conf. on Acoustics, Speech, and Signal Processing, ICASSP*, 4869–4872.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2): 257–286.

Tarjan, R. E.; and van Leeuwen, J. 1984. Worst-case Analysis of Set Union Algorithms. *J. ACM*, 31(2): 245–281.

Yoon, B.-J. 2009. Hidden Markov Models and their Applications in Biological Sequence Analysis. *Current Genomics*, 10: 402 – 415.

SPARK: A Space-Efficient Smoothing Algorithm for Hidden Markov Models

openreview.net/forum

Primary Keyword: Machine Learning (ML) -> ML: Scalability of ML Systems

Secondary Keywords: Planning, Routing, and Scheduling (PRS) -> PRS: Planning with Markov Models (MDPs, POMDPs), Reasoning under Uncertainty (RU) -> RU: Stochastic Models & Probabilistic Inference

TL;DR: We develop a reduced-space reformulation of the forward-backward HMM smoothing algorithm.

Abstract:
The popularity of large AI models and limited-memory devices calls for space-efficient algorithm design. A prime example of task calling for space-efficient formulations is statistical inference, or smoothing, especially with Hidden Markov Models (HMMs), which is cardinal in understanding real-world phenomena. Standard smoothing with HMMs incurs high memory consumption, increasing with the length of the observation sequence. As such models become heavier in computational and memory demands, portable devices face a memory bottleneck in offline inference. In this paper, we propose SPARK (Space-Efficient Forward-Backward), a reformulation of the Forward-Backward smoothing algorithm that eliminates the dependence of its space complexity on the length of the observation horizon in certain classes of HMMs. We showcase SPARK's effectiveness through an experimental study on synthetic and real-world data and analyse the challenges in generalizing our formulation to generic HMMs.

Supplementary Material: zip
iThenticate Agreement: Yes, I agree to iThenticate's EULA agreement version: v1beta

Reproducibility Checklist: I certify all co-authors of this work have read and completed the Reproducibility Checklist.

Submission Number: 2784

Paper Decision

Decisionby Program Chairs09 Dec 2024, 23:10 (modified: 10 Dec 2024, 00:04)Program Chairs, Area Chairs, Senior Program Committee, AuthorsRevisions
Decision: Reject

Comment:
The paper presents SPARK, a new space efficient algorithm for statistical inference in Hidden Markov Models. The algorithm is essentially a reformulation of the classical Forward-Backward algorithm in order to render its working space requirements

independent of the sequence length. It can also be viewed as an improved version of a previous space efficient method called the island algorithm. However, the proposed SPARK only works as expected on a specific class of directed acyclic graphs called well-layered graphs. The experimental results seem to validate the proposed approach.

Pros:

- the paper presents a new variation of the Forward-Backward algorithm for HMMs that achieves impressive space savings on a particular class of directed acyclic graphs.
- the quality of the presentation is fairly good and the paper is relatively easy to follow.

Cons:

- there is little evidence that the proposed method works in real-world scenarios as the experimental evaluation considers synthetically generated data.
- comparison with other existing baselines is also missing from the evaluation

A more space-efficient inference procedure for certain HMMs

Official Reviewby Program Committee nKNu06 Nov 2024, 03:18 (modified: 09 Dec 2024, 23:30)Program Chairs, Area Chairs, Senior Program Committee, Program Committee, AuthorsRevisions
Review:
The authors describe a new algorithmic approach, dubbed SPARK, to smoothing in HMMs that essentially aims to apply a divide and conquer strategy on the state space instead of the time horizon (the approach taken by the island algorithm in previous work). The algorithmic results seem to require the following assumptions in order to yield an improvement:

1. The state transition diagram of the HMM is "well-layered" DAG.
2. There are not "too many" initial states. It is unclear how restrictive these assumptions are in practice. Further, the complexity arguments are not formal proofs and do not contain a formal statement of the authors' claims. For example, I had to reread this section multiple times, and I still don't know what claims, if any, the authors are making about general interval graphs.

The paper concludes with an experimental comparison of the standard forward-backward algorithm, the island algorithm, and SPARK on synthetic tasks. The paper could be made stronger by making a practical case for SPARK and tightening up the theoretical discussion.

Pros:

- HMMs are a classical model with a wide variety of applications.
- The algorithmic changes may be of independent interest.

Cons:

- The practical applications are a bit limited as it isn't clear that space is a real practical bottleneck.
- The space improvements are limited to HMMs where the state transition diagram is a directed acyclic graph.
- The complexity arguments (for SPARK) are a bit messy, i.e., they could be made more formal.
- The paper was somewhat difficult to read as a number of parameters are only defined once (and it isn't easy to refer back to them).

Rating: 5: Marginally below acceptance threshold

Confidence: 4: The reviewer is confident but not absolutely certain that the evaluation is correct

Rebuttal by Authors

Rebuttal:
Thank you for the timely and comprehensive review.

> It is unclear how restrictive these assumptions are in practice.

Practical tasks such as forced alignment in speech recognition indeed work with a given initial state on a well-layered graph. We conducted experiments with real-world forced alignment data. We use a composite HMM for speech-text forced alignment built with the HTK software toolkit, with 5529 states out of which 3204 are emitting, aiming to align speech recordings from the TIMIT corpus. As observations, we extract a standard 39-dimensional feature vector (13 Mel-Cepstrum Cepstral Coefficients, augmented by their first and second order derivatives). The table below presents memory consumption in KB vs. observation length T.

Number of Observations	SPARK	Standard	Island
10	3,862,409	8,722,010	7,892,457
20	4,465,161	15,915,712	9,331,722
30	4,810,520	23,114,885	9,335,651
50	11,188,668	37,541,629	10,800,642

Notably, the advantage of SPARK remains on real-world data.

> I still don't know what claims, if any, the authors are making about general interval graphs.

As long as the state space is divisible in halves, SPARK incurs $O(S)$ space and $O(S^3T)$ time. In case an even division is impossible, it divides along the time horizon, yielding a $\log T$ factor in both space and time complexity.

> it isn't clear that space is a real practical bottleneck.

Space is a bottleneck when applying smoothing in low-memory devices; in addition, space scalability is crucial in data centers and facilitates energy-efficient computations.

> a number of parameters are only defined once (and it isn't easy to refer back to them).

To address this concern, we will include more parameters, such as α and β, in Table 1.

SPARK introduces a memory-efficient F-B reconfiguration using state-splitting on well-layered DAGs, but lacks real-world validation and comparisons to modern baselines like rnn or transformer. Improved readable explanations for the algorithm and division choices would also help clarify the approach.

Official Reviewby Program Committee SRYZ02 Nov 2024, 03:02 (modified: 09 Dec 2024, 23:30)Program Chairs, Area Chairs, Senior Program Committee, Program Committee, AuthorsRevisions

Review:

This paper proposed SPARK, which has a novel reconfiguration of the F-B algorithm and significantly reduces memory usage for HMM smoothing by splitting HMM states instead of observation sequences and modelling the state into a well-layered DAGs.

In the experiments, real-world data is not clear, it seems there is only synthetic data and some 'real-world applications such as forced alignment' but still use synthetic data.

The memory efficient result is limited on the well-layered DAGs and real-wold data is usually not just a well-layered DAGs. Besides, if the layer of DAGs larger than 40, the memory will getting much larger than "standard" baseline.

For the algorithm for Island and spark, the more readable description and the rationale behind the chosen division step need better explanation if possible, especially for readers less familiar with advanced HMM configurations.

The baseline is referring the Island Algorithm, which is a old baseline in 1997. The neural network-based model like RNN/transformer can also solve the HMM smoothing problem by modeling as a sequence modeling task and these method can also be memory efficient. What's the reason author do not compare with these kind of methods?

Rating: 5: Marginally below acceptance threshold

Confidence: 2: The reviewer is willing to defend the evaluation, but it is quite likely that the reviewer did not understand central parts of the paper

Rebuttal by Authors

Rebuttal:

> lacks comparisons to modern baselines ... neural network-based model can also solve the HMM smoothing problem ... What's the reason author do not compare with these kind of methods?

The forward-backward algorithm remains useful and necessary, despite the advent of other models, and is combined with other models in hybrid solutions. As examples of its continued present-day application, see [A, B, C].

[A] Yao et al.: Cooperative Caching in Vehicular Content Centric Network Based on Social Attributes and Mobility. IEEE Trans. Mob. Comput. 20(2): 391-402 (2021)

[B] Zeineldeen et al.: Improving the Training Recipe for a Robust Conformer-based Hybrid Model. INTERSPEECH 2022: 1036-1040

[C] Sun et al.: Industrial system working condition identification using operation-adjusted hidden Markov model. J. Intell. Manuf. 34(6): 2611-2624 (2023)

> lacks real-world validation ... In the experiments, real-world data is not clear ... there is only synthetic data and some 'real-world applications such as forced alignment'.

Our memory consumption results are orthogonal to accuracy: our algorithms produce **exactly** the same results as the standard forward-backward smoothing algorithm on real-world data. Yet we conducted experiments with real-world forced alignment data. We use a composite HMM for speech-text forced alignment built with the HTK software toolkit, with 5529 states out of which 3204 are emitting, aiming to align speech recordings from the TIMIT corpus. As observations, we extract a standard 39-dimensional feature vector (13 Mel-Cepstrum Cepstral Coefficients, augmented by their first and second order derivatives). The table below presents memory consumption in KB vs. observation length T.

Number of Observations	SPARK	Standard	Island
10	3,862,409	8,722,010	7,892,457
20	4,465,161	15,915,712	9,331,722
30	4,810,520	23,114,885	9,335,651
50	11,188,668	37,541,629	10,800,642

Notably, the advantage of SPARK remains on real-world data.

> if the layer of DAGs larger than 40, the memory will getting much larger than "standard" baseline.

The memory remains lower than the standard baseline as Figures 6 & 7 in the paper show.

> the rationale behind the chosen division step need better explanation

The rationale is to evenly divide the HMM state space.

Review

Official Reviewby Program Committee rjjW20 Sept 2024, 04:23 (modified: 09 Dec 2024, 23:30)Program Chairs, Area Chairs, Senior Program Committee, Program Committee, AuthorsRevisions
Review:

Summary

The authors propose an algorithm for inferring the state in an HMM, based on the forward-backward (FB) smoothing algorithm. Unlike FB, their method does not scale in complexity with the length of the sequence. After introducing FB, they introduce the island algorithm which reduces the space complexity of FB.

The authors' proposed SPARK algorithm models the HMM as a directed acyclic graph, which it cuts in two based on the distance from the current state. They perform a number of studies, showing the strengths and weaknesses of their approach

Strengths

- The authors introduce a number of technical concepts and algorithms in an easy-to-digest manner
    The authors provide a concise and intuitive explanation of forward-backward
- The authors provide a thorough evaluation of their algorithm, compared with the original FB algorithm and variants of the island algorithm

Weaknesses

- The SPARK algorithm implementation (Alg. 2) is very complex, making it difficult to understand
- The proposed algorithm is only demonstrated on toy problems, and it is unclear if it scales to more complex tasks
- Typo in 5.4: sate => state

Rating: 7: Good paper, accept

Confidence: 2: The reviewer is willing to defend the evaluation, but it is quite likely that the reviewer did not understand central parts of the paper

Rebuttal by Authors

Rebuttal:

> The SPARK algorithm implementation (Alg. 2) is very complex, making it difficult to understand

Admittedly, the pseudo-code has to follow the notations for the forward-backward algorithm.

> The proposed algorithm is only demonstrated on toy problems, and it is unclear if it scales to more complex tasks

We use these stylized problems, which include highly challenging cases, to investigate the limits and potential of SPARK. Yet we conducted experiments with real-world forced alignment data. We use a composite HMM for speech-text forced alignment built with the HTK software toolkit, with 5529 states out of which 3204 are emitting, aiming to align speech recordings from the TIMIT corpus. As observations, we extract a standard 39-dimensional feature vector (13 Mel-Cepstrum Cepstral Coefficients, augmented by their first and second order derivatives). The table below presents memory consumption in KB vs. observation length T.

Number of Observations	SPARK	Standard	Island
10	3,862,409	8,722,010	7,892,457
20	4,465,161	15,915,712	9,331,722
30	4,810,520	23,114,885	9,335,651
50	11,188,668	37,541,629	10,800,642

Notably, the advantage of SPARK remains on real-world data.

> Typo in 5.4: sate => state

Thank for noting our typo.

Addresses an Important Challenge, but Lacks Clear Contribution to the Research Community

Official Reviewby Program Committee EJu806 Sept 2024, 13:47 (modified: 09 Dec 2024, 23:30)Program Chairs, Area Chairs, Senior Program Committee, Program Committee, AuthorsRevisions
Review:
The paper proposes SPARK (Space-Efficient Forward-Backward) to address the high memory demands of statistical inference using Hidden Markov Models. Traditional smoothing with the Forward-Backward algorithm requires space proportional to the length of the observation sequence, making it unsuitable for devices with limited memory. SPARK reformulates this algorithm, specifically for HMMs represented as DAGs, to eliminate the

space complexity's dependence on the sequence length. The paper demonstrates SPARK's effectiveness with experimental results comparing its memory usage and runtime against standard methods like the Island algorithm.

This work tackles an important problem, making HMMs more viable for real-time applications on resource-constrained devices. The paper is well-organized, with a clear explanation of the problem, background, proposed solution, and experiments. That said, to elevate the paper's impact, it would benefit from a more thorough discussion of related work and a comparison with state-of-the-art methods. Expanding on these areas would not only emphasize the novelty of SPARK but also situate it within the broader research context, helping readers better appreciate its significance.
Rating: 4: Ok but not good enough - rejection

Confidence: 3: The reviewer is fairly confident that the evaluation is correct

Rebuttal by Authors

Rebuttal:

> would benefit from a more thorough discussion of related work and a comparison with state-of-the-art methods.

The Island algorithm we are comparing with is the state-of-the-art method for a space-efficient alternative to the forward-backward algorithm. The forward-backward algorithm remains highly useful as a state-of-the-art solution to this day and is also combined with other models in hybrid solutions. As examples of its continued present-day application, see [A, B, C].

[A] Lin Yao, Yuqi Wang, Xin Wang, Guowei Wu: Cooperative Caching in Vehicular Content Centric Network Based on Social Attributes and Mobility. IEEE Trans. Mob. Comput. 20(2): 391-402 (2021)

[B] Mohammad Zeineldeen, Jingjing Xu, Christoph Lüscher, Ralf Schlüter, Hermann Ney: Improving the Training Recipe for a Robust Conformer-based Hybrid Model. INTERSPEECH 2022: 1036-1040

[C] Jinwen Sun, Akash Deep, Shiyu Zhou, Dharmaraj Veeramani: Industrial system working condition identification using operation-adjusted hidden Markov model. J. Intell. Manuf. 34(6): 2611-2624 (2023)