Beam-search SIEVE for low-memory speech recognition

Martino Ciaperoni[1], *Athanasios Katsamanis*[2], *Aristides Gionis*[3], *Panagiotis Karras*[4]

[1]Aalto University, Finland
[2]Athena R.C., Greece
[3]KTH Royal Institute of Technology, Sweden
[4]University of Copenhagen, Denmark

martino.ciaperoni@aalto.fi, nkatsam@athenarc.gr, argioni@kth.se, piekarras@gmail.com

Abstract

A capacity to recognize speech offline eliminates privacy concerns and the need for an internet connection. Despite efforts to reduce the memory demands of speech recognition systems, these demands remain formidable and thus popular tools such as Kaldi run best via cloud computing. The key bottleneck arises form the fact that a bedrock of such tools, the Viterbi algorithm, requires memory that grows *linearly* with utterance length even when contained via beam search. A recent recasting of the Viterbi algorithm, SIEVE, eliminates the path length factor from space complexity, but with a significant practical runtime overhead. In this paper, we develop a variant of SIEVE that lessens this runtime overhead via beam search, retains the decoding quality of standard beam search, and waives its linearly growing memory bottleneck. This space-complexity reduction is orthogonal to decoding quality and complementary to memory savings in model representation and training.

Index Terms: speech recognition, memory efficient algorithms

1. Introduction

Decoding in speech recognition finds the sequence of states in a model — Hidden Markov Model (HMM) or Weighted Finite State Transducer (WFST) — that best explains a signal [1, 2].

The standard algorithm for this task is the Viterbi algorithm, formulated in terms of *dynamic programming* (DP) [3] or *token passing* [4]. Its memory requirements grow linearly with both the length of the observation sequence (i.e., acoustic signal) and number of states, hindering its use in devices with hard memory constrains [5]. However, it is nowadays necessary to perform speech recognition locally in limited-memory devices, such as smartphones or Raspberry Pis, while avoiding cumbersome data transfers that introduce latency and raise privacy issues.

A recently introduced paradigm, SIEVE (Space Efficient Viterbi) [6], reformulates Viterbi decoding to eliminate the linear memory dependence in the observation sequence length. However, this space efficiency comes with a significant runtime overhead, while it is unclear whether this reformulation also applies to the token-passing and beam-search instances of Viterbi decoding. In this paper, we advance the main idea underlying the SIEVE paradigm to those instances of Viterbi decoding, and thereby achieve space efficiency with a reduced runtime overhead. Figure 1 illustrates the kind of space-efficiency advantage our work brings; while other works merely reduce the memory for model representation and number of states, hence the size of inputs such as initial probability vector π, transition probability matrix A, and emission probability matrix B, we reduce the *working space* complexity of the decoding algorithm by completely eliminating the dependence of the dynamic programming array $\mathcal{T}$ on the length of the decoded sequence. We support this claim with results that show a difference of *one to three orders of magnitude* in memory requirements between SIEVE variants and the state-of-the-art Kaldi toolkit [7]. The beam-search variant of SIEVE achieves runtime competitive with Kaldi, while consuming over 700 times less memory, and produces exactly the same result as standard beam search, as it only alters the algorithm to judiciously recompute tokens instead of storing them. We argue that bringing the SIEVE paradigm into Kaldi would be an important step towards cloud-independent speech recognition, shaping future smart environments.

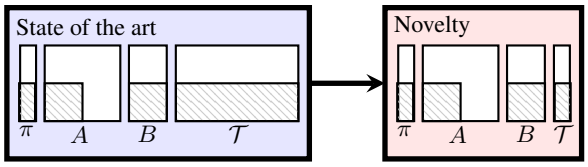

Figure 1: *Prior works reduce memory for model representation, illustrated by white areas; we reduce* working-space *memory used for decoding in a manner* orthogonal *to model representation, other memory savings, and decoding quality.*

2. Background and related work

A Hidden Markov Model (HMM) [8], used to recover the finite set of hidden states that generated a sequence of observations, is described by the triplet (π, A, B), where π is the initial probability distribution over hidden states, elements of A are transition probabilities between states, and elements of B are probabilities of *emitting* an observation from a hidden state. Probabilities in B may be discrete or continuous and are traditionally described by a Gaussian Mixture Model or by a Deep Neural Network [9]. We henceforth consider the directed graph G defined by transitions A and we use such graph-theoretic notions as *predecessor* and *successor* nodes (i.e., states).

2.1. HMM-based decoding

HMM-based speech recognition calls to find, given a sequence of observations $Y = \{y_1, y_2, \ldots, y_T\}$, the sequence of hidden states $Q = \{s_1, s_2, \ldots, s_T\}$ that maximizes the probability:

$$P(Q, Y) = \pi_{s_1} \cdot B_{s_1 y_1} \prod_{i=2}^{T} A_{s_{i-1} s_i} \cdot B_{s_i y_i}, \qquad (1)$$

where s_i and y_i denote the i-th hidden state and observation, respectively. For computational convenience, we replace products of probabilities with equivalent sums of log-probabilities. States correspond to phonemes and, to capture context, there are typically three states per phoneme. Phoneme-specific HMMs

are composed and embedded into a large *Weighted Finite-State Transducer* (WFST) [10]. Some states are *non-emitting*, i.e., associated with transition probabilities but not with emission probabilities. Observations arise from a speech recording in 10–30 ms intervals and comprise spectral energy measurements (or variants) in selected frequency bands [11].

2.2. Viterbi algorithm

The Viterbi decoding algorithm [3] finds the most likely path of hidden states in an HMM given a sequence of observations. In a *forward pass*, the algorithm populates two DP tables, storing the log-likelihood and the predecessor associated with the most likely path ending at each state for each frame. The former table is used to identify the most likely path upon reaching the final frame, and the latter to reconstruct the optimal path by backtracking. Both tables have size $O(KT)$, where K is the number of hidden states and T is the number of frames (i.e., the observation sequence length), hence space complexity is $O(KT)$. Time complexity is $O(K^2T)$, or, by an edge-aware implementation, $O((K + |E|)T)$, where $|E|$ is the number of edges in G. The deficiency of space complexity in the Viterbi algorithm, and decoding algorithms for speech recognition in general, has received scant attention, hindering locally performed speech recognition in constrained-memory devices. Speech recognition practitioners utilize cloud computing to resolve the problem. Still, cloud computing necessitates data transfer, which may be undesirable when one is interested to perform speech recognition while not connected to the internet or when one may not want to share sensitive data. A preliminary attempt to reduce space complexity in Viterbi decoding, *Checkpoint Viterbi* [12], stores the values in a subset of (by default) $\sqrt{T}$ equidistant rows, or checkpoints, of the DP table in a forward pass and locally recomputes the Viterbi path between checkpoints, reducing space complexity to $O(K\sqrt{T})$.

2.3. Token passing

One formulation of the Viterbi algorithm follows *token passing* [4]; thereby, at time t, each state s_i holds a *token*, which stores the optimal log-likelihood of a path ending at state s_i at frame t, and passes it to each of its outgoing connected states s_j, increasing the log-likelihood score by the associated edge transition log-likelihood $\log A_{s_i,s_j}$ and emission log-likelihood $\log B_{s_i,y_t}$, if s_i is an *emitting state*. In the last time frame, we retrieve the state associated with the highest log-likelihood, which is the last state in the optimal Viterbi path. To reconstruct the path followed up to the last state, we need to augment basic tokens with the list of words (or states) that a path goes through, using a considerable amount of memory.

2.4. Beam search

Token passing allows to prune, at any frame, tokens deemed to be unlikely to lead to the optimal path, judging from their current log-likelihood; thereby, we forfeit optimality for the sake of efficiency. A beam width parameter $\mathcal{B}$ specifies the number of tokens to preserve in each frame. By such *beam search* [13], the time complexity for decoding becomes $O(\mathcal{B}KT)$, or $O((\mathcal{B} + |E|)T)$ in the edge-aware case. To store tokens per frame, a priority queue of the top $\mathcal{B}$ tokens needs $O(\mathcal{B}T)$ space, whereas storing all generated tokens takes $O(KT)$ space. While beam-search decoding has been extensively studied [14, 15, 16], its memory usage has never undergone a thorough investigation.

2.5. Kaldi

Beam search decoding and token passing are implemented in Kaldi [7], a widely adopted speech recognition toolkit written in C++. Kaldi utilizes a decoding graph, represented as a WFST combining a grammar model, a lexicon, context-dependencies, and HMM definitions. Kaldi progressively propagates tokens from one frame to the next along the edges of the decoding graph and across states, which may be *emitting* or *non-emitting*; each token is associated with a set of links, used to reconstruct the path corresponding to a hypothesis for decoding the utterance. Kaldi prunes tokens whose log-likelihood differ more than a user-specified threshold θ from the current optimum in each frame. Kaldi also implements a decoding algorithm generating a lattice of paths that are alternative transcriptions of an utterance rather than a single path. Unlike the Viterbi algorithm, Kaldi beam search is not guaranteed to retrieve an optimal path; it focuses on quickly finding a set of highly likely hypotheses.

2.6. SIEVE

SIEVE is a space-efficient reformulation of Viterbi decoding recently developed [6]; it discards and recomputes parts of the DP solution for the sake of space efficiency, adopting a divide-and-conquer algorithmic design that has been also used in the context of data summarization [17, 18].

In a nutshell, SIEVE runs the forward pass of the Viterbi algorithm without storing the dynamic programming tables, but keeps in memory only the log-likelihood values associated with the current and previous frame. At the same time, it identifies, for each path, a *median pair* of states, defined as a pair of consecutive states s_{m^-} and s_{m^+} along the path, which minimize the maximum among the number of *predecessors* of s_{m^-} and the number of *successors* of s_{m^+} in the transition graph G; we obtain counts of predecessors and successors in a pre-processing stage. When the forward pass terminates, we access the median pair associated with the optimal Viterbi path. To find the remaining edges in such optimal path, we recursively perform forward passes and identify new medians in the predecessors of s_{m^-} and successors of s_{m^+}. Apart from numbers of predecessors and successors, SIEVE also stores the number of observations N_p that lie in the optimal Viterbi path before the median pair, from which it derives the number N_s of observations after the median pair, which it uses to prune, via breadth-first search, states that are not reachable from s_{m^-} in N_p hops and from s_{m^+} in N_s hops. When the recursion terminates, the set of retrieved median pairs reconstructs the entire optimal Viterbi path. Under the assumption of an acyclic G, SIEVE achieves space complexity $O(K)$ and time complexity $O\left(\sum_{i=1}^{\log K} 2^i \left(\frac{K}{2^i}\right)^2 T\right) = O\left(\sum_{i=1}^{\log K} \frac{K^2}{2^i} T\right) = O(K^2T)$, i.e., the same as the standard Viterbi algorithm. The presence of small, highly localized cycles that are typically found in speech recognition applications does not bear a significant impact on the performance of SIEVE. However, in the worst case, in presence of cycles, time complexity rises to $O(K^2T^2)$.

A different theoretical result holds for general topologies of G by a different application of the SIEVE paradigm, SIEVE-Middlepath, which we present next.

2.7. SIEVE-Middlepath

SIEVE-Middlepath uses a *middle pair* of states, which divide the observation sequence into two halves, in place of the *median pairs* as defined in Section 3. Thereby it divides, in each

level of the recursion, the optimal Viterbi path, rather than the entire state space, in two parts of length at most $T/2$. The space complexity of SIEVE-Middlepath is $O(K)$, while its time complexity is $O(K^2 T \log T)$, regardless of the topology of G. In practice, SIEVE-Middlepath is less memory-demanding than SIEVE as it requires neither counts of predecessors and successors, nor numbers of observations before a middle pair. Furthermore, as we will see in Section 3.2, SIEVE-Middlepath confers an even higher advantage when combined with beam search.

2.8. Orthogonal efforts for space efficiency

Several prior works strive for space-efficient speech recognition on low-memory devices [19] in ways orthogonal to our work. Such works contain codebook requirements [20, 21], limit the search space when *rescoring* [22], quantize parameters [23, 24, 25], eliminate intermediate steps when constructing a decoding graph [26], trim less likely unigram graphones during training [27], avoid fully expanding the transducer during modeling [28], prospectively control the beam width [29], reduce the word embedding matrix [30], train fewer domain-embedding parameters for adaptation to a domain [31], stabilize the memory needed for training multi-step denoising models [32], avoid fetching weights from off-chip memory for each input speech frame [33], and reduce the memory needed to compute loss [34] and the vocabulary subset required [35] while training RRN-Transducers. An open-source toolkit for low-memory speech recognition is available [36]. Still, these efforts either prune the decoding search space or contain the memory footprint for model representation and training; they *do not* reduce the *working space complexity* of the decoding algorithm. Thus, they are *orthogonal* and *complementary* to our work.

3. SIEVE with beam search

Here, we discuss extensions of the SIEVE paradigm to the token passing model and beam search heuristic.

3.1. SIEVE-TP

Like the Viterbi algorithm, SIEVE is amenable to a token-passing formulation. To attain a space-efficient reformulation of token passing, we omit the costly list of words (or states) stored within each token. Instead, we store in each token its log-likelihood ℓ, the corresponding *median value* m (i.e., maximum among the number of predecessors and successors), median pair m_p, and the number of observations n that precede that median pair in the optimal path. The ensuing algorithm, SIEVE Token Passing (for short, SIEVE-TP), shares the structure of standard SIEVE, the difference lying in that it performs token passing instead of dynamic programming.

3.2. SIEVE-BS

Here, we integrate SIEVE-TP with *beam search*, which preserves only the best $\mathcal{B} \in \mathbb{Z}^+$ tokens in each frame and expands $\mathcal{B}$ paths to the next frame, yielding the SIEVE-Beam-Search algorithm (for short, SIEVE-BS). In each frame, we define a set of $\mathcal{B}$ *active states* that hold tokens to be propagated further. Algorithm 1 lists the pseudocode, assuming, for simplicity, that all states are emitting and hold tokens in the first frame; in practice, a single *initial state* may be active in the first frame. We pre-compute the arrays N^- and N^+, which hold the numbers of predecessors and successors of nodes in G. The algorithm resembles SIEVE [6], with some cardinal differences. First, it operates by token passing. Second, in each frame, it updates the set of $\mathcal{B}$ active states holding the tokens of highest likelihood; by default we assemble that set in an $O(K)$-space one-off partial-sorting operation before moving to the next frame in Line 19, which we empirically found to strike a better time- vs. space-efficiency tradeoff than maintaining an $O(\mathcal{B})$-space priority queue of best tokens, as discussed in Section 2.4.

Algorithm 1 SIEVE-BS

Input $lastSt$, π, A, B, y, N^-, N^+, $\mathcal{B}$, $ActSt$;
1: $t \leftarrow y.size(); k \leftarrow \pi.size(); \mathcal{T}_p \leftarrow []; ActSt \leftarrow \{\}$;
2: **for** $i = 1, \ldots, k$ **do** ▷ initialization at first call only
3: $\mathcal{T}_p[i] \leftarrow Tok(\ell = \pi + B_{i,y_1}, m = \infty)$;
4: $ActSt.add(i)$; ▷ store active states
5: **for** $step = 2, \ldots, t$ **do** ▷ main loop
6: $\mathcal{T}_c \leftarrow []$; $M_{new} \leftarrow \{\}$;
7: **for** $i \in ActSt$ **do**
8: **for** $j \in outNeighbours(i)$ **do**
9: $\ell \leftarrow \mathcal{T}_p[i].\ell + A_{s_i,s_j} + B_{s_j,y_{step}}$;
10: **if** $\mathcal{T}_c[j] = \emptyset$ **then** $\mathcal{T}_c[j] \leftarrow Tok(\ell = -\infty, m{=}\infty)$;
11: **if** $\ell > \mathcal{T}_c[j].\ell$ **then** ▷ new best path
12: $\mathcal{T}_c[j].\ell \leftarrow \ell$; $m \leftarrow \max\{N^-[i], N^+[j]\}$;
13: **if** $m < \mathcal{T}_p[i].m$ **then** ▷ new median
14: $\mathcal{T}_c[j].m \leftarrow m$; $\mathcal{T}_c[j].m\_p \leftarrow (i, j)$;
15: $\mathcal{T}_c[j].n \leftarrow step$; $M_{new}.add(j)$;
16: **else** ▷ inherited median
17: $\mathcal{T}_c[j].m \leftarrow \mathcal{T}_p[i].m$; $\mathcal{T}_c[j].m\_p \leftarrow \mathcal{T}_p[i].m\_p$;
18: $\mathcal{T}_c[j].n \leftarrow \mathcal{T}_p[i].n$; $\mathcal{T}_c[j].a \leftarrow \mathcal{T}_p[i].a$;
19: $ActSt \leftarrow PartialSort(\mathcal{T}_c, \max\{\mathcal{B}, |\{s|\mathcal{T}_c[s]{\neq}\emptyset\}|\})$;
20: $\mathcal{T}_c[M_{new}].a \leftarrow ActSt$; $\mathcal{T}_p \leftarrow \mathcal{T}_c$;
21: **if** $lastSt = \emptyset$ **then** $lastSt \leftarrow \arg\max \mathcal{T}_c.\ell$;
22: $s_{m^-}, s_{m^+} \leftarrow \mathcal{T}_c[lastSt].m\_p$; ▷ extract median
23: $N_p \leftarrow \mathcal{T}_c[lastSt].n$; $y_p \leftarrow y[:N_p]$;
24: $ActSt_m \leftarrow \mathcal{T}_c[lastSt].a$;
25: **if** $N_p > 1$ **then** ▷ continue recursion in predecessors
26: $s_{pred} \leftarrow$ FIND-T-HOPPRED(s_{m^-}, N_p); ▷ BFS
27: $A_p \leftarrow A[s_{pred}]$; $B_p \leftarrow B[s_{pred}]$; $\pi_p \leftarrow \pi[s_{pred}]$;
28: update $N^-[s_{pred}]$, $N^+[s_{pred}]$;
29: SIEVE-BS$(s_{m^-}, \pi_p, A_p, B_p, y_p, N^-, N^+, \mathcal{B}, ActSt)$;
30: $N_s \leftarrow t - N_p$; $y_s \leftarrow y[N_s:]$;
31: **print** (s_{m^-}, s_{m^+}); ▷ in-order print
32: **if** $N_s > 1$ **then** ▷ continue recursion in successors
33: $s_{succ} \leftarrow$ FIND-T-HOPSUCC(s_{m^+}, N_s); ▷ BFS
34: $A_s \leftarrow A[s_{succ}]$; $B_s \leftarrow B[s_{succ}]$;
35: $\pi_s[i = 1, \ldots, s_{succ}.size()] \leftarrow -\infty$; $\pi_s[s_{m^+}] \leftarrow 0$;
36: update $N^-[s_{succ}]$, $N^+[s_{succ}]$;
37: SIEVE-BS$(lastSt, \pi_s, A_s, B_s, y_s, N^-, N^+, \mathcal{B}, ActSt_m)$;

If we relax the requirement that SIEVE-BS returns the same solution as standard beam search (BS), SIEVE-BS has space complexity $O(K)$ and, assuming acyclic G, time complexity $O(\mathcal{B}KT)$. Yet to ensure that SIEVE-BS retrieves the same solution as BS, we need to initialize each subproblem with the set of *active states* ($ActSt$) BS has used. To that end, SIEVE-BS stores in each token the set $ActSt$ in the frame immediately after the median. Unfortunately, this storage incurs space complexity $O(K\mathcal{B})$. On the other hand, the extension of SIEVE-Middlepath to beam search (hereafter, SIEVE-BS-Mp) needs to store a set of active states *only* for the frame at the middle of the path, rather than in each token; the subsequent subproblem always starts at that frame by the Middlepath solution; thus, SIEVE-BS-Mp requires only $O(K)$ space and $O(\mathcal{B}KT \log T)$ time for any decoding graph G, $O((\mathcal{B} + |E|)T \log T)$ in the edge-aware case, to retrieve the exact same solution as BS. In effect, SIEVE-Middlepath adapts seamlessly to beam search and therefore SIEVE-BS-Mp is preferable to SIEVE-BS.

3.3. SIEVE-Kaldi

Following up on SIEVE-TP and SIEVE-BS, we can extend the SIEVE paradigm to confer space efficiency to the Kaldi *beam search* algorithm. As with SIEVE-BS, the tokens should contain the information concerning median pairs or middle pairs, but no path information. Once the lattice-generating decoding terminates, we obtain a set of high-probability tokens, and run SIEVE-BS to reconstruct the path they went through. Thereby, we sidestep storing links, Kaldi's main memory bottleneck. Further, to minimize the recomputation-related overhead of SIEVE-Kaldi with respect to standard Kaldi, we may resort to parallelization, multithreading and GPU accellerators.

4. Experiments

Here, we present an empirical comparison of SIEVE, SIEVE-BS and Kaldi [7] in terms of memory requirements and execution time. Experiments ran on a 2×10 CORE XEON E5 2680 V2 2.80 GHZ, 256 GB machine. Code is available[1] online.

Baselines. We compare SIEVE, SIEVE-BS, and their Middlepath (Mp) variants against (i) standard Viterbi, (ii) Checkpoint Viterbi, described in Section 2.2, (iii) the Kaldi beam search decoder, and (iv) a naïve Beam Search (BS) method.

Data. We experimented with the Resource Management (RM) corpora of digitized and transcribed speech [37]. We present representative results on utterances of $T = 250$ and $T = 866$ frames. The decoding graph comprises $K = 25,333$ states and $175,428$ edges. The experiment can be reproduced by running the example script[2] provided by Kaldi until line 54, keeping only the first and the longest utterance in the test data.

Metrics. We report runtime in seconds and memory use in bytes (B) or megabytes (MB), reflecting the memory occupied by all data structures used, without considering the memory for the input data, common to all algorithms. For beam search methods, we also report the relative error in log-likelihood, obtained as $\eta = \frac{|\ell_{OPT} - \ell|}{\ell_{OPT}}$, where ℓ_{OPT} and ℓ are the log-likelihoods of the optimal and beam search solution path, respectively. We average results over ten post-warm-up runs.

Table 1: *Comparison of resource requirements with* $T = 250$.

Algorithm	Runtime (s)	Peak Memory (MB)
Vanilla Viterbi	47.35	76.02
Checkpoint Viterbi	86.74	9.93
Kaldi	1.00	11.10
BS	0.19	1.00
SIEVE	164.83	1.87
SIEVE-Middlepath	121.10	1.01
SIEVE-BS	14.15	0.12
SIEVE-BS-Mp	1.18	0.01

Table 2: *Comparison of resource requirements with* $T = 866$.

Algorithm	Runtime (s)	Peak Memory (MB)
Vanilla Viterbi	165.39	262.98
Checkpoint Viterbi	537.31	18.24
Kaldi	2.00	12.32
BS	0.56	2.36
SIEVE	682.64	1.87
SIEVE-Middlepath	460.86	1.01
SIEVE-BS	20.28	0.12
SIEVE-BS-Mp	4.65	0.01

[1] https://github.com/beamsearchsieve/SIEVE-BS

[2] https://github.com/kaldi-asr/kaldi/blob/master/egs/rm/s5/run.sh

Results. Tables 1–2 show decoding memory requirements and runtime. We use default parameters in Kaldi and $\mathcal{B} = 100$ in SIEVE-BS(-Mp). SIEVE-BS-Mp requires the smallest memory of all algorithms and time in the same order of magnitude as Kaldi. SIEVE-BS uses more memory than SIEVE-BS-Mp, as expected from the analysis (Section 3.2). SIEVE requires over 100× more memory than SIEVE-BS-Mp and is slower than SIEVE-BS(-Mp) and Kaldi. Still, it uses about 10 times less memory than Kaldi. Vanilla Viterbi requires significantly more memory than SIEVE, while Checkpoint Viterbi consumes memory similar to Kaldi and runtime similar to SIEVE.

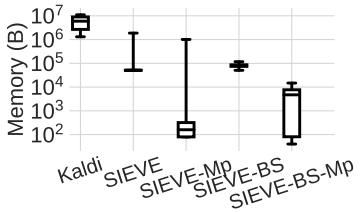

Figure 2: *Memory usage in decoding (log scale) with* $T = 250$.

Figure 2 shows the memory consumption distribution for decoding the utterance of $T = 250$, over different frames for Kaldi, recursive calls for SIEVE and SIEVE-Middlepath, and both recursive calls and frames for SIEVE-BS variants. Other baselines consume constant memory.

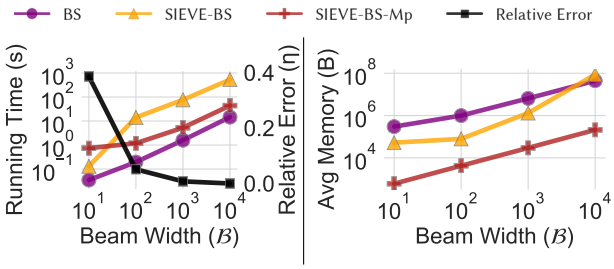

Figure 3: *Rutime vs.* $\mathcal{B}$ *(log scale) with* $T = 250$.

Figure 3 shows that, with the utterance of $T = 250$, SIEVE-BS-Mp attains an apt tradeoff between runtime, memory use, and quality across values of $\mathcal{B}$ and yields optimal accuracy using beam width in the order of 10^3 and memory in the order of 10KB, 3 orders of magnitude lower than Kaldi. We obtained similar results with other utterances; SIEVE-BS-Mp thus combines *correctness*, time-efficiency, and space-efficiency. We provide results on correctness to show that optimal accuracy is feasible with beam search, for the sake of completeness; we stress that the space-efficiency we achieve is *orthogonal* to decoding quality; beam search by SIEVE achieves, by design, the same decoding quality as standard beam search.

5. Conclusion

The SIEVE paradigm reduces the space complexity for Viterbi decoding by eliminating its dependence in observation sequence length without a time-complexity overhead, yet with a significant practical runtime overhead. To address this overhead, we invented a beam-search variant of SIEVE, SIEVE-BS-Mp, that *drastically reduces* the memory footprint of beam search. We stress that this result is orthogonal to decoding quality and complementary to memory savings in model representation and training. Nonetheless, with sufficient beam width, SIEVE-BS-Mp attains optimal decoding in runtime comparable to Kaldi, a popular speech recognition toolkit; thus it heralds efficient local speech recognition in memory-constrained devices.

6. References

[1] M. J. F. Gales and S. J. Young, "The application of hidden markov models in speech recognition," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, 2007.

[2] H. Braun, J. Luitjens, R. Leary, T. Kaldewey, and D. Povey, "Gpu-accelerated viterbi exact lattice decoder for batched online and offline speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7874–7878.

[3] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[4] S. J. Young, N. Russell, and J. Thornton, *Token passing: a simple conceptual model for connected speech recognition systems*. University of Cambridge, Department of Engineering, 1989.

[5] A. Georgescu, A. Pappalardo, H. Cucu, and M. Blott, "Performance vs. hardware requirements in state-of-the-art automatic speech recognition," *EURASIP J. Audio Speech Music. Process.*, vol. 2021, no. 1, p. 28, 2021.

[6] M. Ciaperoni, A. Gionis, A. Katsamanis, and P. Karras, "SIEVE: A space-efficient algorithm for Viterbi decoding," in *ACM SIGMOD*, 2022, pp. 1136–1145.

[7] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.

[8] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[9] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[10] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[11] M. Benzeghiba, R. de Mori, O. Deroo, S. Dupont, T. Erbes, D. Jouvet, L. Fissore, P. Laface, A. Mertins, C. Ris, R. Rose, V. Tyagi, and C. Wellekens, "Automatic speech recognition and speech variability: A review," *Speech Communication*, vol. 49, no. 10-11, pp. 763–786, 2007.

[12] C. Tarnas and R. Hughey, "Reduced space hidden markov model training," *Bioinformatics*, vol. 14, no. 5, pp. 401–406, 1998.

[13] B. T. Lowerre, *The harpy speech recognition system.* Carnegie Mellon University, 1976.

[14] S. Kobashikawa, T. Hori, Y. Yamaguchi, T. Asami, H. Masataki, and S. Takahashi, "Efficient beam width control to suppress excessive speech recognition computation time based on prior score range normalization," in *Interspeech*, 2012, pp. 1011–1014.

[15] T. Hori, S. Watanabe, and A. Nakamura, "Improvements of search error risk minimization in viterbi beam search for speech recognition," in *Interspeech*, 2010, pp. 1962–1965.

[16] I. Williams and P. Aleksic, "Rescoring-Aware Beam Search for Reduced Search Errors in Contextual Automatic Speech Recognition," in *Proc. Interspeech 2017*, 2017, pp. 508–512.

[17] S. Guha, "Space efficiency in synopsis construction algorithms," in *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, 2005, pp. 409–420.

[18] P. Karras and N. Mamoulis, "Hierarchical synopses with optimal error guarantees," *ACM Transacation on Database Systems*, vol. 33, no. 3, pp. 18:1–18:53, 2008.

[19] A. Waibel, A. Badran, A. W. Black, R. Frederking, D. Gates, A. Lavie, L. Levin, K. Lenzo, L. M. Tomokiyo, J. Reichert *et al.*, "Speechalator: two-way speech-to-speech translation in your hand," in *Companion Volume of the Proceedings of HLT-NAACL 2003-Demonstrations*, 2003, pp. 29–30.

[20] K. Funaki, K. Yoshida, and K. Ozawa, "4kb/s speech coding with small computational amount and memory requirement: UL-CELP," in *The 3rd Intl Conf. on Spoken Language Processing*, 1994, pp. 2059–2062.

[21] S. Astrov, "Memory space reduction for hidden markov models in low-resource speech recognition systems," in *7th Intl Conf. on Spoken Language Processing - INTERSPEECH*, 2002, pp. 1585–1588.

[22] M. Schuster, "Nozomi - a fast, memory-efficient stack decoder for LVCSR," in *The 5th Intl Conf. on Spoken Language Processing*, 1998.

[23] K. Filali, X. Li, and J. A. Bilmes, "Data-driven vector clustering for low-memory footprint ASR," in *7th Intl Conf. on Spoken Language Processing - INTERSPEECH*, 2002, pp. 1601–1604.

[24] S. Jeong, I. Han, E. Jon, and J. Kim, "Memory and computation reduction for embedded ASR systems," in *INTERSPEECH - ICSLP, 8th Intl Conf. on Spoken Language Processing*, 2004, pp. 2325–2328.

[25] I. McGraw, R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, H. Sak, A. Gruenstein, F. Beaufays *et al.*, "Personalized speech recognition on mobile devices," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 5955–5959.

[26] M. Novak and V. Bergl, "Memory efficient decoding graph compilation with wide cross-word acoustic context," in *INTERSPEECH - ICSLP, 8th Intl Conf. on Spoken Language Processing*, 2004, pp. 281–284.

[27] J. Huang, L. Olorenshaw, G. H. Ábrego, and L. Duan, "A memory efficient grapheme-to-phoneme conversion system for speech processing," in *INTERSPEECH - ICSLP, 8th Intl Conf. on Spoken Language Processing*, 2004, pp. 1237–1240.

[28] E. Stoimenov and J. W. McDonough, "Memory efficient modeling of polyphone context with weighted finite-state transducers," in *INTERSPEECH, 8th Annual Conf. of the Intl Speech Communication Association*, 2007, pp. 1457–1460.

[29] M. Price, A. P. Chandrakasan, and J. R. Glass, "Memory-efficient modeling and search techniques for hardware ASR decoders," in *Interspeech, 17th Annual Conf. of the Intl Speech Communication Association*, 2016, pp. 1893–1897.

[30] K. Shi and K. Yu, "Structured word embedding for low memory neural network language model," in *Interspeech, 19th Annual Conf. of the Intl Speech Communication Association*, 2018, pp. 1254–1258.

[31] S. Dingliwal, A. Shenoy, S. Bodapati, A. Gandhe, R. T. Gadde, and K. Kirchhoff, "Domain prompts: Towards memory and compute efficient domain adaptation of ASR systems," in *Interspeech, 23rd Annual Conf. of the Intl Speech Communication Association*, 2022, pp. 684–688.

[32] J. Huang and C. Wu, "Memory-efficient multi-step speech enhancement with neural ODE," in *Interspeech, 23rd Annual Conf. of the Intl Speech Communication Association*, 2022, pp. 961–965.

[33] G. Venkatesh, A. Valliappan, J. Mahadeokar, Y. Shangguan, C. Fuegen, M. L. Seltzer, and V. Chandra, "Memory-efficient speech recognition on smart devices," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 8368–8372.

[34] F. Kuang, L. Guo, W. Kang, L. Lin, M. Luo, Z. Yao, and D. Povey, "Pruned RNN-T for fast, memory-efficient ASR training," in *Interspeech, 23rd Annual Conf. of the Intl Speech Communication Association*, 2022, pp. 2068–2072.

[35] J. Lee, L. Lee, and S. Watanabe, "Memory-efficient training of rnn-transducer with sampled softmax," in *Interspeech, 23rd Annual Conf. of the Intl Speech Communication Association*, 2022, pp. 4441–4445.

[36] Alpha Cephei, "VOSK speech recognition toolkit: Accurate speech recognition for Android, iOS, Raspberry Pi and servers with Python, Java, C#, Swift and Node," https://github.com/alphacep/vosk-api, 2020.

[37] P. Price, W. Fisher, J. Bernstein, and D. Pallett, "Resource management rm2 2.0," *DVD. Philadelphia: Linguistic Data Consortium*, 1993.