On the complexity of exact maximum likelihood decoding on tail-biting trellises

Priti Shankar
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560012, India
email: priti@csa.iisc.ernet.in

Abstract — **We analyze an algorithm for exact maximum likelihood decoding on tail-biting trellises and show that under certain conditions exact maximum likelihood decoding has time complexity $O(S_t\ log(S_t))$ and where S_t is the total number of states of the tail-biting trellis.**

I. Our Main Result

We show that an algorithm proposed by us in [1] needs at most $O(S_t)$ Viterbi updates in the tail-biting trellis under certain assumptions. The algorithm needs to use a heap for keeping track of intermediate results. The $S_t \log S_t$ term comes from the overheads of maintaining this heap. In experiments we observe that the actual execution time is dominated by the number of Viterbi updates.

II. The Algorithm

We view the tail-biting trellis as a superposition of S_0 subtrellises each with its own start and final state. We denote the start state of a trellis T_i by s_i and the final state by f_i. The algorithm works in two passes. Since the Viterbi algorithm is essentially a shortest path algorithm we refer to the metric of a path as the cost of the path. The first pass is just a Viterbi algorithm run on the tail-biting trellis. The survivors at each final node are inspected. Some of these are codewords and the others are what we term *semi-codewords*, that is those that start at some state s_i and terminate at a state f_j, $i \neq j$. We define a winning path as one corresponding to a codeword or semicodeword survivor at the final index with the least cost. If the overall winning path is a codeword then decoding is complete. If this is not the case we make use of the fact that the cost of a semicodeword ending at state f_j is actually a lower bound on the cost of the winning codeword in trellis T_j. We can thus disqualify all sub-trellises that have a winning semicodeword path whose cost is greater than the best winning codeword path. What we are left with are trellises that have winning semicodewords;we call these residual trellises and only these participate in the second phase. The second phase is an adaptation of the A^* algorithm for computing the shortest path from a source to a goal node in a graph. The A^* algorithm uses the sum of the path length $l(s,u)$ from the source s to the node u, and an *underestimate* $e(u,f)$ of the shortest path length from the node u to the goal node f in guiding the search.Let us call this sum, the *estimate* at a node. Since this is a lower bound on the path length, it is an over-estimate of the belief that this is indeed the best path. At each instant in the second phase the algorithm is executing on a single subtrellis. This is the trellis with the least estimate. The A^* algorithm expands only one path at a time, and after each expansion it checks the cost of the path against the best estimate among all others expanded so far in all trellises. If at any stage the current path ceases to be the best path as measured by its estimate the path estimate is placed on a data structure called a heap and the best path which may perhaps lie in some other trellis is taken up. The first path to reach a final node is guaranteed by the A^* algorithm to be the winner under certain conditions satisfied by our algorithm.

We now define certain terms:

Intersection Property A tail-biting trellis satisfies the intersection property if all subtrellises share all states at at least one time index

Non-merging semicodewords A non-merging semicodeword is a semicodeword which does not touch the all-zero codeword path.
For a BSC the following theorem holds. Let $w(\mathbf{v})$ be the Hamming weight of vector $\mathbf{v}$ and S_0 the number of start states of the tail-biting trellis.

Theorem
If the tail-biting trellis satisfies the intersection property, the algorithm needs at most $O(S_t)$ Viterbi updates for all error patterns $\mathbf{e}$ for which there are at most $O(S_0)$ non-merging semicodewords $\mathbf{s}$ satisfying $w(\mathbf{s}+\mathbf{e}) < w(\mathbf{e}) < w(\mathbf{c}+\mathbf{e})$ for *all* non-zero codewords $\mathbf{c}$.

References

[1] P. Shankar, A. Dasgupta, K. Deshmukh and B.S. Rajan, On Viewing Block Codes as Finite Automata, Theoretical Computer Science, **290**(2003) 1775-1797.

0-7803-8280-3/04/$20.00 ©2004 IEEE