A Best-First Tree-Searching Approach for ML Decoding in MIMO System

Chung-An Shen*, Ahmed M. Eltawil*, Sudip Mondal † and Khaled N. Salama ‡
*EECS Department, University of California, Irvine, CA, USA
† Cypress Semiconductors Corporation, San Jose, CA, USA
‡ EE Program, King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia
Email: *{chungans, aeltawil}@uci.edu, ‡khaled_salama@ieee.org

Abstract—**In MIMO communication systems maximum-likelihood (ML) decoding can be formulated as a tree-searching problem. This paper presents a tree-searching approach that combines the features of classical depth-first and breadth-first approaches to achieve close to ML performance while minimizing the number of visited nodes. A detailed outline of the algorithm is given, including the required storage. The effects of storage size on BER performance and complexity in terms of search space are also studied. Our result demonstrates that with a proper choice of storage size the proposed method visits 40% fewer nodes than a sphere decoding algorithm at signal to noise ratio (SNR) = 20dB and by an order of magnitude at 0 dB SNR.**

I. Introduction

In Multiple Input Multiple Output (MIMO) systems the maximum-likelihood (ML) decoder is the optimum scheme in the sense of minimizing bit-error-rate (BER) [1]. However, its computational overhead makes a direct implementation of ML decoders practically infeasible. Therefore, from an implementation perspective, tree-searching schemes such as the *sphere decoding algorithm (SDA)* and its variant the *K-Best decoding algorithm* [2], [3], [4] have been adopted as decoders of choice, because of their ability to implement ML (or close to ML) decoding with significantly reduced complexity. These approaches operate to find the shortest path in a tree structure, while only visiting a subset of the tree and thus reducing the complexity. Recent research activity has results in innovative system architectures including those reported in [4], [7-13] with different tradeoffs in terms of area, power, complexity etc.

In general tree-searching schemes can be classified as *depth-first* and *breadth-first* [5]. The *depth-first* scheme, e.g. *SDA* outlined in [3], traverses the tree sequentially until hitting a dead end or reaching the bottom level, where it backtracks to examine unvisited nodes in previous levels. This approach results in varying throughput based on the received SNR. On the other hand, *breadth-first* search, such as the *K-Best algorithm* outlined in [4], traverses the tree in a level-by-level fashion, thus lending itself to fixed throughput and complexity at the price of visiting more nodes on average as well as deviating from the ML solution at higher SNRs.

In this work we present a tree-searching approach that combines the features of depth- and breadth- first schemes. In this approach, every visited node is stored in a list and the most promising node in this list (determined by a metric) is selected by the decoder for further expansion. This node is then enumerated to both its sibling as well as to its child, such that the decoder has more *scope* to decide on the traversing direction (either breadth or depth) and to reach the shortest path with a minimal search space. Therefore the decoding throughput and implementation complexity can be further improved.

The key contributions of this paper are:

- We demonstrate optimality of this approach when size of the list is infinity.
- We investigate effects of the list size on BER and complexity in terms of the number of search points. We also show that with a proper choice of the list size this algorithm searches fewer points than that of the *SDA*, while only suffering fractional performance degradation.
- We discuss different approaches to handle a full list and illustrate their performance-complexity trade-off.

II. Background

In this section, we briefly review ML decoding of MIMO communication and the sphere decoding/K-Best decoding algorithm. In the MIMO system with $\boldsymbol{M}$ transmit antennas and $\boldsymbol{N}$ receive antennas, the $\boldsymbol{M}$-dimensional transmit signal vector $\boldsymbol{s}$ is mapped independently from the constellation of a modulation scheme. Hence the $\boldsymbol{N}$-dimensional received signal vector $\boldsymbol{y}$ is given by

$$\boldsymbol{y} = \boldsymbol{Hs} + \boldsymbol{n} \tag{1}$$

where the matrix $\boldsymbol{H}$ denotes the $\boldsymbol{N}\times\boldsymbol{M}$ channel matrix. After real-value decomposition the modified $\boldsymbol{s}$ vector contains $m = 2M$ elements, and the $\boldsymbol{y}$ vector contains $n = 2N$ elements. In this system model with ML decoding, the estimated transmitted signal vector $\boldsymbol{s}_{ML}$ is determined by

$$\boldsymbol{s}_{ML} = \arg\min_{s\in\Omega^M} \|\boldsymbol{y} - \boldsymbol{Hs}\|^2 \tag{2}$$

where $\boldsymbol{\Omega}$ denotes the set containing all possible modulation points. Considering the **QR**-decomposed channel model, that is, $\boldsymbol{H} = \boldsymbol{QR}$, the summation term of (2) can be re-written in a recursive process as the following

$$\sum_{i=1}^{m} \left| \hat{y}_i - \sum_{j=i}^{m} r_{ij} s_j \right|^2 \tag{3}$$

This work is partially funded by the Department of Justice, National Institute of Justice DOJ/NIJ, under grant number 2006-IJ-CX-K044

The revised problem equation (3) can be well-represented by a tree structure with m levels, and each node in the tree contains Ω child nodes. Therefore the ML solution can be acquired by finding the path with smallest path metric in the tree constructed by (3). We can further rewrite (3) to:

$$T_i(s) = T_{i+1}(s) + \left| \hat{y}_i - \sum_{j=i+1}^{m} r_{ij} s_j - r_{ii} s_i \right|^2 \quad (4)$$

and

$$\left| \hat{y}_i - \sum_{j=i+1}^{m} r_{ij} s_j - r_{ii} s_i \right|^2 = |r_{ii}|^2 |c_i - s_i|^2 \quad (5)$$

where T_{i+1} is the accumulated path metric to level $i+1$, T_i is the accumulated path metric at the level i, and C_i is usually called the "search center" and is given by

$$c_i = (1/r_{ii})\left(\hat{y}_i - \sum_{j=i+1}^{m} r_{ij} s_j \right) \quad (6)$$

The *SDA* traverses the tree and computes path metric (4) for each node in the tree. It only visits a subset of the tree by discarding any branch that delivers a metric exceeding the predefined *pruning constraint* R, thus resulting in a reduced complexity. The *SDA* typically performs a *depth-first* search and selects the child based on Schnorr-Euchner [6] strategy. Once reaching the lowest level, it updates R to path metric of the newly discovered path. A detailed description of this algorithm can be found in [3].

III. Proposed Tree-Search Scheme

The *SDA* performs well when the channel condition is good where R is shrunk rapidly and a large part of the tree is pruned. However, when the SNR is low, many redundant tree nodes are visited before determining the ML solution. This is mostly due to its insufficient capability to determine if the branch it is currently traversing on is *good* enough to lead to the ML point. This can be explained by the fact that when the SNR is low, the metric of a *bad* path can be temporarily small (and the metric of a *good* path can be temporarily large) thus misleading the decoder. Therefore the search space can be significantly reduced if the decoder possesses sufficient *scope* to decide, as early as possible, whether it should still traverse along the branch or shift to an adjacent one.

To this end, we propose that for each visited node, we enumerate both to its immediate sibling node (breadth-extension) as well as to its best child node (depth-extension). The decoder then extends to the direction that contains a smaller path metric. Therefore the decoder will traverse along a branch into a deeper level if it is promising; otherwise it will attempt another branch. Moreover, in order to guarantee visiting the node with the lowest metric in the tree, extended nodes should be stored in a list and the decoder should expand the branch from the node with smallest path metric therein, which is the most favorable path being visited. The basic idea of this proposed approach can be illustrated graphically in Fig. 1.

We note here that the proposed approach, while reducing the search space, may increase implementation area due to the requirement of extra storage. It is also noted that the well known Fano algorithm [5] in the context of sequential decoding for convolutional codes aims to achieve similar goals by dynamically updating the pruning threshold and allowing a node to be visited multiple times. However, since the Fano algorithm does not utilize a list, it could result in an excessive number of operations due to recomputation of metrics for multiple visits of a node. Furthermore, in the context of ML decoding of lattice codes, an analogous idea that proposes the use of a list was proposed in [14] strictly from a signal processing perspective. In this paper, we extend state of the art by studying the impact of the length of the list, as well as the sorting of the stored nodes on the BER performance and complexity of the decoder.

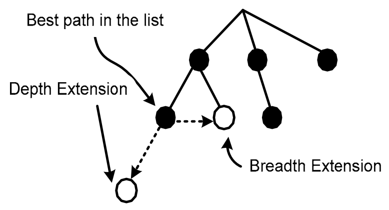

Figure 1. Idea of proopsed tree-searching approach

A. Algorithm

A flowchart to outline this algorithm is shown in Fig. 2. The decoder starts at the root node with path metric = 0. It then extends to its best node at the first level of the tree and stores this node into the list with size $\boldsymbol{L}$, while the root node is then removed from the list. In the following, the decoder selects the node with smallest path metric in the list, which is denoted as $\boldsymbol{w}$. The decoder will be further expanding from it to its best child node as well as to its next sibling. These two extended nodes are also stored in the list, while $\boldsymbol{w}$ will be removed from the list, since all states are already kept in the extended nodes. The decoder once again selects a new $\boldsymbol{w}$ from the list and performs another extension from it. This *select-and-expand* operation will be performed iteratively until $\boldsymbol{w}$ is a leaf node where it is selected as the decoding solution.

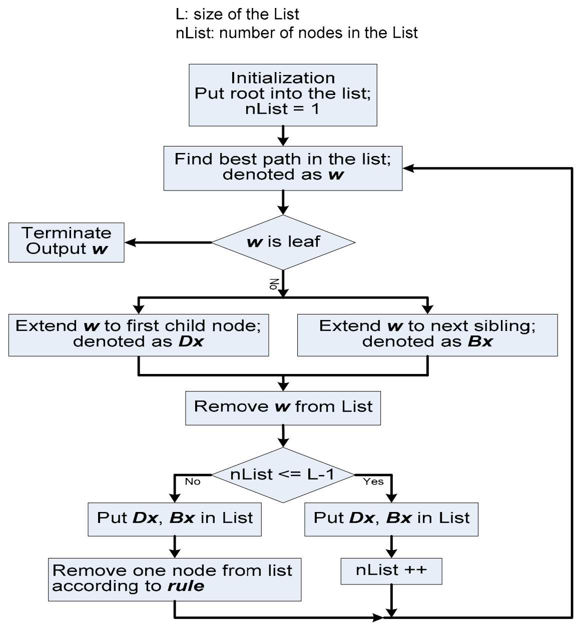
L: size of the List
nList: number of nodes in the List
Initialization Put root into the list; nList = 1
Find best path in the list; denoted as w
w is leaf
Terminate Output w
No
Extend w to first child node; denoted as Dx
Extend w to next sibling; denoted as Bx
Remove w from List
nList <= L-1
No
Yes
Put Dx, Bx in List
Put Dx, Bx in List
Remove one node from list according to rule
nList ++

Figure 2. Flow chart of the algorith

Moreover, while the list is full the decoder must abandon a node in the list, which is selected according to a ***rule***. In the following subsection we will discuss different approaches of forming the ***rule*** which incurs a performance-complexity trade-off.

B. Performance-Complexity Tradeoffs

In the algorithm outlined in Fig. 2 there are two critical factors that determine a trade-off between performance and complexity, which are:

- Size of the list, ***L***
- The ***rule*** that decides the node to be dropped while list is full

If L is assumed to be unlimited (i.e. no node is dropped) the output path of this algorithm is guaranteed to be the ML point, that is, the optimal ML performance can be achieved. The reason behind this is as follows. The decoder always selects the best path that has been visited in the list, that is, the path metric of w is always smaller than any other path in the list. If w is a leaf node, it must indicate the shortest path in the tree; since for any other node to become a leaf node, it has to increase its path metric (i.e. either traverses one more level or one more sibling node).

However, in practice, ***L*** must be a limited number and some nodes must be removed from the list when it is full. When ***L*** is sufficiently large the probability of dropping a node and missing a path is very low, thus a close-to ML performance can be achieved. However the number of search points and requirements of hardware implementations (e.g. storage components, control circuits, etc.) also increase with larger ***L***.

For the ***rule*** highlighted in Fig. 2, we consider two approaches:

- ***rule 1***: *sorting*. In this approach the decoder removes the node with largest path metric in the list. This node is the one that is most unlikely to lead to the ML point (or to be selected as ***w***), thus the impact on BER performance should be smaller. However, this method requires extra storage which increases the complexity of the decoder.
- ***rule 2***: *no sorting*. In this approach the decoder drops a node that is randomly selected from the list (e.g. the node in the final location of the list), which is not necessarily the node with the largest path metric. This approach might accidently prune a path that could have been selected as ***w***, thus performance degradation is incurred. The advantage of this method is its lack of sorting operations which reduces the implementation overhead. Furthermore, this approach results in visiting fewer nodes than ***rule 1*** since there is higher chance that a favorable path is pruned, which may be visited when ***rule 1*** is applied.

IV. Simulation Results

In this section we study and quantify the BER performance as well as the complexity in terms of number of search points of the proposed algorithm, and compare with that of the SDA outlined in [3]. The simulations are all setup for a 4×4 64-QAM MIMO communication system.

A. Bit Error Rate

Fig. 3 depicts the BER performance of the proposed algorithm with *rule* = *sorting* for different list size. The comparisons of performance degradation at BER = 10^{-3} and BER = 10^{-4} are further summarized in Table I. From the figure and the table, it is shown that the proposed algorithm with ***L***=24 provides similar to ML performance until SNR ranges up to 20dB. When the SNR grows beyond 20 dB one can identify a slight degradation in performance, which is 0.2 dB at BER = 10^{-3} and 0.3 dB at BER = 10^{-4}. Furthermore, as the list size ***L*** is reduced, one can observe increasing performance degradation, especially in the high SNR region (SNR>20). For example it is 1.3 dB at BER = 10^{-4} when ***L*** = 16.

In Fig. 4 and Table I we further illustrate and summarize the BER performance of the proposed approach with different ***rules***. It is shown that to maintain close-to ML performance the *no sorting* approach requires larger ***L***. To further clarify

Table I. BER Comparisons

rule	List Size (L)	Deviation from ML at BER = 10^{-3}	Deviation from ML at BER = 10^{-4}
sorting	24	0.2 dB	0.3 dB
	20	0.3 dB	0.4 dB
	16	0.6 dB	1.3 dB
no sorting	64	0.2 dB	0.3 dB
	24	1.2 dB	4 dB

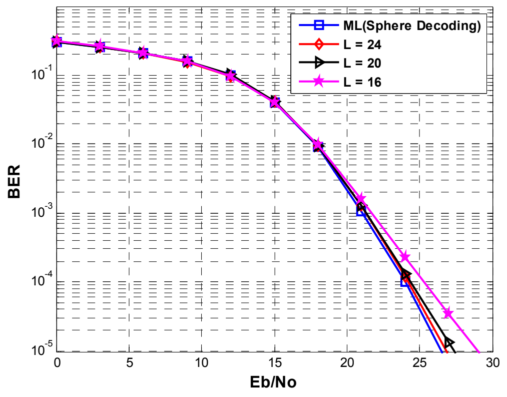

Figure 3. Bit Error Rate for proposed algorithm

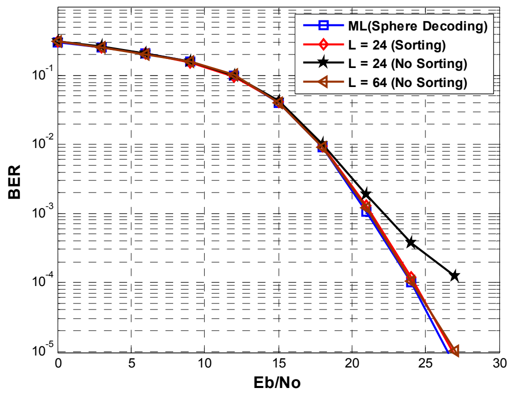

Figure 4. Bit Error Rate with different ***rules***

this point, consider the case where ***L=24*** in Fig. 4, it is clear that the algorithm with ***rule*** = *sorting* achieves much better performance than the approach with ***rule*** = *no sorting*. Moreover Fig. 4 also depicts that the *no sorting* approach with ***L***=64 achieves similar performance to the *sorting* approach with ***L*** = 24.

B. Complexity

Fig. 5 depicts the number of search points for the proposed algorithm with different ***L***. From the figure, we can observe that the proposed algorithm search space is significantly reduced as compared to that of the *SDA*. It is also shown, that decreasing ***L*** reduces the number of search points, especially in the low SNR region. We further compare the effect of different ***rules*** on the complexity and demonstrate the result in Fig. 6. It is shown that, with the same list size, the *no sorting* approach visits less nodes than the *sorting* approach. This is because that in the *no sorting* approach promising nodes might be deleted from the list such that some paths are never considered, whereas the *sorting* approach keeps them as viable candidates. The speed-area trade-off of different approaches is summarized in Table II.

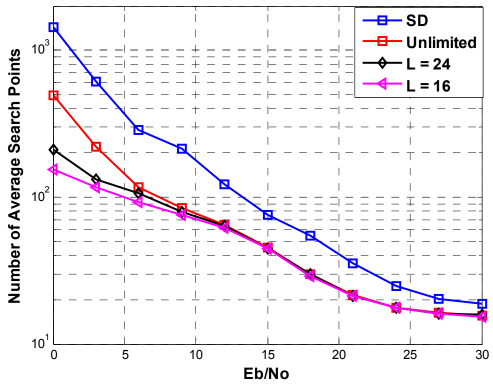

Figure 5. Number of Search Points

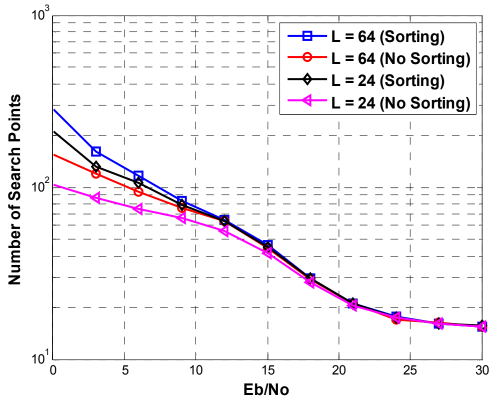

Figure 6. Number of Search Points with different ***rules***

Table II. Comparisons of speed-area trade-offs

	SDA	Proposed (sorting)	Proposed (no sorting)
Speed	Low	Medium	High
Area	Small	Medium	Large

V. Conclusion

This paper presents a storage-based tree-searching approach that combines the features of depth- and breadth-first schemes. The basic idea of this approach is discussed and a detailed flowchart of the algorithm is illustrated graphically. The effect of list size on the BER performance and number of search points is investigated and quantified through simulations. Our result demonstrates that with a proper choice of storage size the proposed method visits 40% fewer nodes than a sphere decoding algorithm at signal to noise ratio (SNR) = 20dB and by an order of magnitude at 0 dB SNR, while the performance loss is marginal. Moreover, we discussed two different approaches to mange the situation of memory overflow and illustrated the performance-complexity trade-off.

References

[1] D. Gesbert, M. Shafi, Da-Shan Shiu, P. J. Smith, and A. Naguib, “From theory to practice: An overview of MIMO space-time coded wireless systems,” *IEEE Journal on Selected Areas in Comm.*, vol. 21, no. 3, Apr. 2003.

[2] B. Hassibi and H. Vikalo, "On the sphere decoding algorithm. Part I: The expected complexity," *IEEE Trans. Sig. Proc.*, vol. 53, no. 8, pp. 2806-2818, Aug 2005

[3] M. O. Damen, E. Gamal, and G. Caire, “On maximum-likelihood detection and the search for the closet lattice point,” *IEEE Trans. Inform. Theory*, vol. 49, no.10, Oct 2003

[4] K. Wong, C. Ysui, S. Cheng, and W. Mow, “A VLSI architecture of a K-Best lattice decoding algorithm for MIMO channels,” *ISCAS-2002*, vol. 3, 2002

[5] J. B. Anderson and S. Mohan, “Sequential coding algorithms: a survey and cost analysis,” *IEEE Trans. Commun.,* vol. com-32, no.1, Feb 1984.

[6] C. Schnorr and M. Euchner, “Lattice basis reduction: improved practical algorithms and solving subset sum problems,” *Math. Programming*, vol. 66, 1994

[7] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, “VLSI implementation of MIMO detection using the sphere decoding,” *IEEE Jour. Solid-State Circuits,* vol. 40, no. 7, Jul 2005

[8] L. G. Barbero and J. S. Thompson, "A Fixed-Complexity MIMO Detector Based on the Complex Sphere Decoder", in IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC '06), Cannes, France, Jul. 2006

[9] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection," *IEEE J. Selected Areas in Comm.*, vol. 24, no. 3, pp. 491-503, Mar 2006

[10] M. Wenk, M. Zellweger, A. Burg, N. Felber, W. Fichtner, "K-Best MIMO detection VLSI architectures achieving upto 424 Mbps," *Proc. IEEE-ISCAS'06*, pp. 1151-1154, May 2006

[11] S. Chen, T. Zhang, and Y. Xin, "Relaxed K-Best MIMO signal detector design and VLSI Implementation," *IEEE Tran. very Large Scale Integ. (VLSI) Syst.*, vol. 15, no. 3, pp. 328-337, Mar 2007

[12] S. Modal, A. Eltawil, and K. Salama, "Architectural Optimizations for Low-Power K-Best MIMO Decoders," *IEEE Trans. on Vehicular Technologies*, vol. 58, issue 7, pp. 3145-3153, Sep 2009

[13] S. Mondal, A. Eltawil, C. Shen, K. Salama, "Design and Implementation of a Sort Free K-Best Sphere Decoder " *Accepted to IEEE Transactions on Very Large Scale Integration Systems*, June 2009

[14] K. Su, I. Berenguer, I. J. Wassell, and X. Wang, “Efficient maximumlikelihood decoding of spherical lattice space-time codes,” in *Proc. IEEE Int. Conf. Communications*, Istanbul, Turkey, Jun. 2006.