# A Unification of ML-Optimal Tree-Search Decoders

Christoph Studer, Andreas Burg, and Wolfgang Fichtner
Integrated Systems Laboratory
ETH Zurich, Switzerland
Email: {studer,apburg,fw}@iis.ee.ethz.ch

*Abstract*— **In this paper, a generalization of tree-search algorithms with maximum likelihood (ML) error rate performance for the detection in multiple-input multiple-output (MIMO) systems is presented. The Schnorr-Euchner sphere decoder [1], Djikstra's algorithm [2] applied to MIMO detection [3], and an ML-optimal variant of the K-best [4] detector are shown to be special cases of the presented unification. The complexity of the detection procedure is characterized by three different measures, relevant for VLSI implementations, and the corresponding trade-offs between hardware complexity and throughput are investigated. Finally, efficient detector configurations are identified which are suitable for MIMO detection of higher-order modulation alphabets and for large numbers of transmit antennas.**

## I. Introduction

*Multiple-input multiple-output* (MIMO) technology constitutes the basis for next-generation wireless communication systems. Spatial multiplexing allows to achieve high spectral efficiency, but comes at cost of increased signal processing complexity, most prominently in the detection unit. The optimum detector in terms of vector error rate is the *maximum likelihood* (ML) detector. Unfortunately, a straightforward implementation of this scheme with an *exhaustive search* (EMLD) over all possible transmitted vector symbols has an exponentially growing complexity in the number of transmit antennas. However, the ML criterion can be reformulated in such a way that the unstructured brute-force exhaustive search can be carried out by efficient tree-traversal and tree-pruning algorithms which ultimately lead to a considerable reduction of the average number of operations (computational complexity).

Recent publications have produced a variety different tree-search algorithms, constantly aiming at a reduction of the computational complexity. Limiting our scope to ML-optimal schemes, it can be observed that mainly two strains of algorithms have been reported: the *Schnorr-Euchner sphere decoder* (SESD) [1] which performs a depth-first tree traversal and *Dijkstra's search* applied to MIMO detection (DSD) [3], [5] which follows a metric-first strategy. The SESD has a slightly higher complexity than the DSD, but requires only a small amount of memory. The DSD performs the tree-traversal more efficiently in terms of computational complexity [6], but an implementation requires an exponentially growing amount of memory in the number of spatial streams. Since the complexity of a VLSI implementation is ultimately determined by both, the *number of operations* and by *worst-case memory requirement*, we claim that there is a fundamental trade-off between these two measures.

This work was supported by the STREP project No. IST-026905 (MASCOT) within the sixth framework programme (FP6) of the European Commission.

*Contributions:* We present a unified tree-search strategy for ML-optimal MIMO detection which allows to realize a variety of trade-offs between computational complexity, memory requirements and the ability to provide early estimates of the result. The presented algorithm contains the SESD, the DSD, the exhaustive search ML decoder, and an *ML-optimal K-best decoder* (MLKBD) as special cases and provides intermediate solutions between these well-known algorithms. It is shown how the detector configuration with the lowest complexity can be identified within the corresponding design-space.

*Outline:* The remainder of this paper is organized as follows: Chapter II introduces the system model and explains the transformation of ML-detection into a tree-search problem. Chapter III describes the algorithm in detail and its pseudocode is given. In Chapter IV the properties of our unified view are studied and finally, conclusions are drawn in Chapter V.

## II. System Model and Detection

We consider a MIMO system with $M_T$ transmit and $M_R$ receive antennas and employ spatial multiplexing, where the transmitter sends $M_T$ independent streams and chooses the components $s_i$ of the transmit vector $\mathbf{s}$ from a complex-valued set $\mathcal{O}$ of constellation points. The transfer function of the system is given by

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n} \tag{1}$$

where the $M_R$-dimensional vector $\mathbf{n}$ represents the i.i.d. circular symmetric complex Gaussian noise, the $M_R \times M_T$ channel matrix is denoted by $\mathbf{H}$, and $\mathbf{y}$ is the $M_R$-dimensional received vector.

### A. ML Detection

The task of a MIMO detector is to estimate $\mathbf{s}$ from the received vector $\mathbf{y}$ using an estimate of the channel $\mathbf{H}$. The optimum in terms of vector error probability is achieved by the ML detection rule [7]

$$\hat{\mathbf{s}} = \underset{\mathbf{s}\in\mathcal{O}^{M_T}}{\arg\min}\ \|\mathbf{H}\mathbf{s} - \mathbf{y}\|^2. \tag{2}$$

A straightforward approach to solve (2) is an exhaustive search over all possible candidate vectors $\mathcal{O}^{M_T}$. However, such an implementation becomes quickly prohibitive as the number of transmit antennas increases.

### B. Tree Representation

Tree-search algorithms have been shown to be able to solve (2) with a low average computational complexity, while still being ML optimal. To this end, the corresponding schemes start by factorizing the channel matrix into $\mathbf{H} = \mathbf{QR}$, such that $\mathbf{R}$ is upper-triangular and $\mathbf{Q}$ is unitary. Left multiplied by $\mathbf{Q}^H$ leads to

$$\hat{\mathbf{s}} = \underset{\mathbf{s} \in \mathcal{O}^{M_T}}{\arg\min}\, d(\mathbf{s}) \quad \text{with} \quad d(\mathbf{s}) = \|\mathbf{Rs} - \tilde{\mathbf{y}}\|^2 \qquad (3)$$

where $\tilde{\mathbf{y}} = \mathbf{Q}^H \mathbf{y}$ and where $d(\mathbf{s})$ is the *Euclidean distance*, which can be computed recursively by using *partial Euclidean distances* (PEDs)

$$d_i\left(\mathbf{s}^{(i)}\right) = d_{i+1}\left(\mathbf{s}^{(i+1)}\right) + e_i\left(\mathbf{s}^{(i)}\right) \qquad (4)$$

with the initialization $d_{M_T+1} = 0$ and with the *partial symbol vectors* (PSVs) $\mathbf{s}^{(i)} = \left[\, s_i \ \cdots \ s_{M_T} \,\right]^T$. The *distance increments* (DIs) are computed according to

$$e_i\left(\mathbf{s}^{(i)}\right) = \Big| \sum_{j=i}^{M_T} R_{i,j} s_j - \tilde{y}_i \Big|^2 \qquad (5)$$

and the PED of level $i$ is only dependent on the previously chosen $\mathbf{s}^{(i+1)}$, the corresponding $d_{i+1}(\mathbf{s}^{(i+1)})$, and the DI which requires a new $s_i \in \mathcal{O}$. Thus, $d(\mathbf{s})$ can be computed by computing the DIs (5) and PEDs (4) for $i = \{M_T + 1, \ldots, 1\}$ recursively. This transforms the ML detection problem (2) into a *weighted tree search*. In that tree, PEDs and PSVs are associated with *nodes* and *branches* correspond to DIs. Each path from top-to-bottom through the tree, i.e., from the root node down to the leaves, corresponds to a particular symbol vector $\mathbf{s} \in \mathcal{O}^{M_T}$.

### C. Tree-Search Algorithms

With the above described tree representation, efficient ML detection resorts to finding the leaf associated with the smallest metric. The silicon implementation complexity of such a search is characterized by the amount of memory required to store intermediate results and by the computational complexity of the tree traversal. Several tree-search strategies exist and two efficient algorithms are explained in the following.

*1) Schnorr-Euchner Sphere Decoder:* The SESD [1] performs a depth-first best-expand procedure in the tree, constraining its search to nodes which lie in a hypersphere with radius $r$ around $\tilde{\mathbf{y}}$ (sphere constraint). Starting from the root with $r = \infty$, the decoder traverses the tree depth-first, always giving preference to the child associated with the smallest PED. When a leaf is reached, the ED of that leaf is kept as new radius $r$ and the depth-first search continues. If none of the children of the current parent node meets the sphere constraint the search continues at another unexplored child branching off the path from the root to the current node until no more valid children can be found. The path to the leaf with the smallest PED then corresponds to the ML solution.

*2) Dijkstra's Search:* The DSD mainly performs a purely metric-first procedure in a list of nodes which constitutes the *boundary* between the explored and the unexplored nodes of the tree. First, this list is initialized with only the root node. Then, the following three steps are performed iteratively: The decoder considers all children of all nodes in the boundary in order to identify the child associated with the smallest PED (note that explicit consideration of all children is usually not required). This best child is expanded and is added to the boundary. A node is removed from the boundary, if all of its children have been expanded. When a leaf is reached (expanded), the algorithm stops and the ML solution corresponds the path from the root to this leaf. Since the DSD only expands the best (most promising) available candidates, the number of expanded nodes is minimized [6], however, the boundary may become very large, requiring a considerable amount of memory.

## III. Unification

### A. Notation

A *node* on the $i$th level is described by $\mathcal{N}_p\left(\mathbf{s}_p, d_i(\mathbf{s}_p), \mathcal{E}_p\right)$, where $\mathbf{s}_p$ is the corresponding PSV that uniquely identifies this node in the tree. The associated PED $d_i(\mathbf{s}_p)$ is calculated according to (4) and $\mathcal{E}_p = \left\{e_1, e_2, \ldots, e_{|\mathcal{O}|}\right\}$ is the set of DIs leading to all $|\mathcal{O}|$ children of this node on level $i-1$. The PED of the $m$th child of $\mathcal{N}_p$ can be calculated by $d_{i-1}(\mathbf{s}_c) = d_i(\mathbf{s}_p) + e_m$, where $e_m \in \mathcal{E}_p$ and $\mathcal{E}_p \in \mathcal{N}_p$. The level of a node $\mathcal{N}$ in the tree is defined as

$$L(\mathcal{N}) = i = M_T + 1 - \text{length}(\mathbf{s}) \qquad \mathbf{s} \in \mathcal{N}.$$

Leaves are on level 1 and the *root*, which is defined as $\mathcal{N}_1(\emptyset, 0, \mathcal{E}_1)$, is on level $M_T + 1$. Expanding a parent node $\mathcal{N}_p$ into a new child node $\mathcal{N}_c$, can be described as

$$\mathcal{N}_c\left(\mathbf{s}_c = \left[\{\mathcal{O}\}_m \quad \mathbf{s}_p^T\right]^T, d_{i-1}(\mathbf{s}_c) = d_i(\mathbf{s}_p) + e_m, \mathcal{E}_c\right) \quad (6)$$

where $\{\mathcal{O}\}_m$ denotes the $m$th symbol in $\mathcal{O}$. After the computation of (6), the corresponding DI $e_m \in \mathcal{E}_p$ is set to infinity to remember that the associated child has already been expanded. In summary, expanding a parent node into its $m$th child corresponds to extending the partial symbol vector, adding the DI to the PED of the parent node, and to calculating the DIs for the children of the new child node. Note that during expansion, not all child DIs have to be calculated at once. A possible method which saves a significant amount of computational complexity and lowers the memory consumption has been described in [8].

### B. Algorithm

The scheme, summarized by Algorithm 1, is a combination of the computationally efficient DSD algorithm and the memory efficient SESD. The main goal is to provide trade-offs between memory consumption and computational complexity.

To this end, our unification distinguishes between two sets of nodes: The first set $\mathcal{S}$ contains nodes, obtained from the expansion of other nodes, which can not yet be pruned from the tree since these nodes may still lead to a leaf with a PED that is smaller than the PED of the best leaf

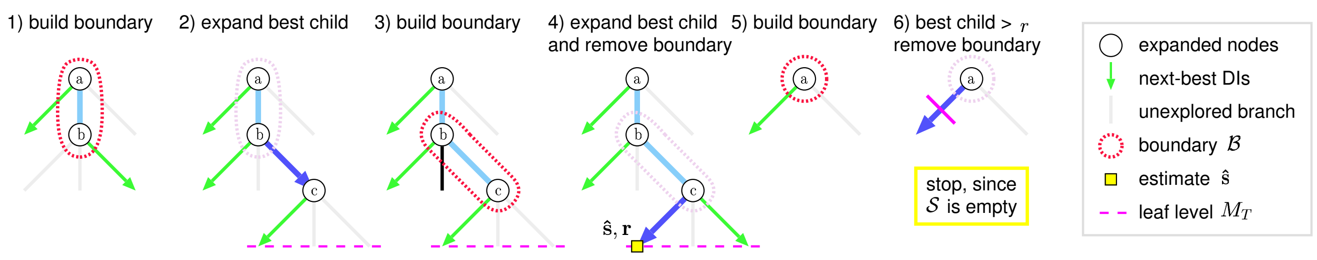
1) build boundary
2) expand best child
3) build boundary
4) expand best child and remove boundary
5) build boundary
6) best child > $r$ remove boundary
stop, since $\mathcal{S}$ is empty
$\hat{\mathbf{s}}, \mathbf{r}$
a
b
c
○ expanded nodes
next-best DIs
unexplored branch
boundary $\mathcal{B}$
estimate $\hat{\mathbf{s}}$
leaf level $M_T$

Fig. 1. Decoding example of a partially explored ternary tree ($M_T = 3$) by using Algorithm 1 with $B = 2$ and $K = 1$.

**Algorithm 1** $[\hat{\mathbf{s}}, n_e, n_s] = \text{Detect}(\mathbf{R}, \tilde{\mathbf{y}}, B, K)$

```
1: initial radius: $r := \infty$
2: number of searched nodes: $n_s := 0$
3: number of expanded nodes: $n_e := 1$
4: initialize $\mathcal{S}$ with the root node: $\mathcal{S} := \{\mathcal{N}_1\}$
5: while $|\mathcal{S}| > 0$ do
6:   build the boundary $\mathcal{B}$ out of $\mathcal{S}$
7:   for $k := 1$ to $K$ do
8:     search in $\mathcal{B}$ for the node $\mathcal{N}_b$ containing the smallest
       child PED: $d := d_b + e_m$ where $d_b, e_m \in \mathcal{N}_b$
9:     $n_s := n_s + |\mathcal{B}|$
10:    if $d < r$ then
11:      if $L(\mathcal{N}_b) > 1$ then
12:        $n_e := n_e + 1$
13:        expand $\mathcal{N}_b$ using $\{\mathcal{O}\}_m$ into the child $\mathcal{N}_c$
14:        insert $\mathcal{N}_c$ into $\mathcal{S}$
15:      else
16:        new estimate: $\hat{\mathbf{s}} := \left[\{\mathcal{O}\}_m \quad \mathbf{s}^T\right]^T$ with $\mathbf{s} \in \mathcal{N}_b$
17:        reduce the search radius: $r := d$
18:        remove $\mathcal{B}$ from $\mathcal{S}$ and break
19:      end if
20:    else
21:      remove $\mathcal{B}$ from $\mathcal{S}$ and break
22:    end if
23:  end for
24: end while
```

identified so far. The members of $\mathcal{S}$ are ordered according to their level in the tree so that $\text{L}(\{\mathcal{S}\}_{i-1}) \geq \text{L}(\{\mathcal{S}\}_i)$. The *active boundary* $\mathcal{B}$ – from which new nodes are expanded – is extracted from the $B$ lowest-level nodes in $\mathcal{S}$ according to $\mathcal{B} = \left\{\mathcal{N}_{|\mathcal{S}|-B+1}, \ldots, \mathcal{N}_{|\mathcal{S}|}\right\}$. If $B$ is chosen larger than $|\mathcal{S}|$ we set $\mathcal{B} = \mathcal{S}$. Since $\mathcal{B} \subseteq \mathcal{S}$ no additional memory needs to be allocated for $\mathcal{B}$ and we shall see that pruning can keep the size of the list $\mathcal{S}$ much smaller than the size of the unconstrained boundary in the DSD. With these definitions, the decoding procedure iterates between the extraction of an *active boundary* $\mathcal{B}$ from $\mathcal{S}$ and the expansion of $K$ children with the smallest PEDs of parent nodes in the active boundary. The resulting new nodes are then added to the set $\mathcal{S}$ and nodes in $\mathcal{S}$, which can not lead to a leaf with a PED that is smaller than the PED of the best leaf identified so far, are removed from $\mathcal{S}$ using a sphere constraint before a new boundary is extracted. We shall refer to the parameter $K$ which determines the number of expansion operations per boundary extraction as the *expand effort*.

*Example for $B = 2$ and $K = 1$:* Assume that the nodes $a$ and $b$ have already been expanded ($\mathcal{S} = \{a, b\}$) as shown on the left of Fig. 1. In step 1 the decoder builds the boundary with $B = 2$ nodes. Then, the best child node $c$ is expanded (step 2). The new node $c$ is added to $\mathcal{S}$. In step 3 a new boundary is extracted, here containing $b$ and $c$. The algorithm searches the best child (step 4) which turns out to be a leaf. Thus, a symbol estimate $\hat{\mathbf{s}}$ is found and the search radius can be updated as $r = d_1(\hat{\mathbf{s}})$. All nodes in the active boundary can now be removed from the $\mathcal{S}$ since all of their children are known to have a PED that exceeds $r$ and can thus not lead to a better solution than $\hat{\mathbf{s}}$ (Algorithm 1 line 18). In step 5 a new boundary is extracted from the sole remaining node $a$ in $\mathcal{S}$. However, in the example, the PED of the best child of $a$ is assumed to exceed the radius $r$ so that $a$ can also be pruned from $\mathcal{S}$ which is now empty, leading to the termination of the algorithm (Algorithm 1 line 5).

## IV. Properties

### A. ML-Optimality

Since the proposed algorithm only excludes nodes from the search whose PED is known to exceed the PED of the currently best estimate (i.e., the leaf with the smallest PED), it is guaranteed to find the ML solution for any combination of $K > 0$ and $B > 0$. In the following, we shall thus focus only on the impact of these two parameters on the memory requirements and on the computational complexity.

### B. Memory Requirements

VLSI implementations are unable to use dynamic memory allocation and must therefore be built to handle the worst-case memory consumption. During detection, the set $\mathcal{S}$, which governs the memory requirement of the algorithm, grows to a certain maximum size $N_{\mathcal{S}}$, before it gets emptied and refilled again (cf. lines 14, 18, and 21 of Algorithm 1). Since the the pace of the con-/destruction of $\mathcal{S}$ depends on the choice of $K$ and $B$, the upper bound $N_{\mathcal{S}}$ on the cardinality of $\mathcal{S}$ is also determined by these two design parameters. Fig. 2 shows $N_{\mathcal{S}}$ as a function of $K$ and $B$.

The configuration with $B = K = 1$ corresponds to the SESD with radius reduction [1] (cf. Section II-C.1). Since $B = 1$, the search for the smallest child to be expanded is limited to the children of a single node. The corresponding memory requirement for the SESD is given by $N_{\mathcal{S}} = M_T$ which is the lowest among all possible configurations. Two other configurations using low memory are: $K = 1$ and $zK = B$, where $z$ is a positive integer. Counting the maximum number of nodes in $\mathcal{S}$ for these cases results in

$$N_{\mathcal{S}} \leq (M_T - 1)\,B + 1 \tag{7}$$

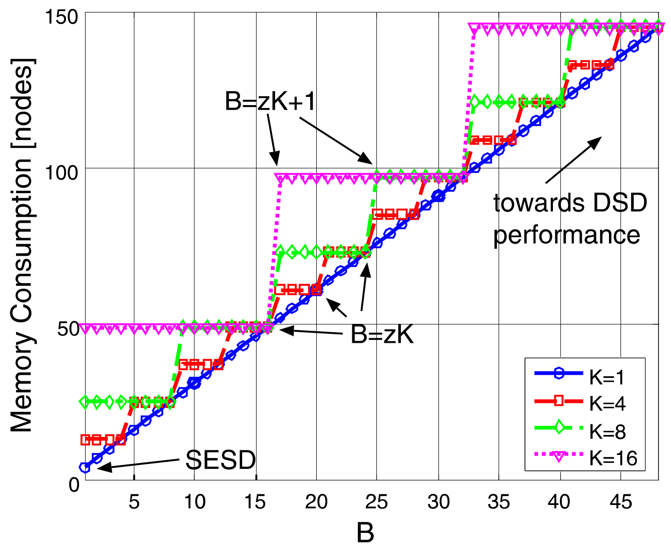

Fig. 2. Memory consumption $N_{\mathcal{S}}$ [nodes] dependent on $K$ and $B$ in a $4 \times 4$ MIMO system using a 64-QAM symbol alphabet.

which holds with equality if $B \leq |\mathcal{O}|$. Hence, in these special cases, memory requirements grow at most linearly with the boundary size for a given $M_T$.

Inefficient configurations are $B = zK + 1$, where the number of expansions of nodes closer to the root is maximized, since the size of the boundary is at most stages larger than the number of nodes in $\mathcal{S}$ that are on the same lowest level of the tree. The worst case in terms of memory consumption occurs, when the decoder advances as slow as possible from the root down to the leaves, i.e., when it always expands the nodes in the boundary which are closest to the root.

The global upper bound for memory consumption (over all choices for $B$ and $K$) is

$$N_{\mathcal{S},\max} = \frac{|\mathcal{O}|^{M_T} - 1}{|\mathcal{O}| - 1} \tag{8}$$

which is attained by the DSD ($B = N_{\mathcal{S},\max}$ and $K = 1$) and by an exhaustive traversal of the tree ($B = K = N_{\mathcal{S},\max}$).

## C. Decoding Complexity

The proposed algorithm may be implemented by using different strategies. To remain as generic as possible in our analysis, two fundamental decoding complexities are identified: The *search complexity* $n_s$ given by the aggregate effort required to identify which children should be expanded from the nodes in the boundary. The *expand complexity*[1] $n_e$ is given by the number of nodes which are actually expanded according to (6). Since $n_s$ and $n_e$ are both random variables, we are interested in the average taken over many symbols, noise-, and channel realizations. To obtain this average, the required accounting is included in the pseudocode in Algorithm 1. To characterize the *true detection complexity* (denoted by $n$), the search and expand complexities have to be weighted according to the chosen implementation strategy.

[1]The expand complexity is equal to the number of visited nodes which has been used in [8] to characterize the throughput of depth-first sphere decoding implemented with a one-node-per-cycle VLSI architecture.

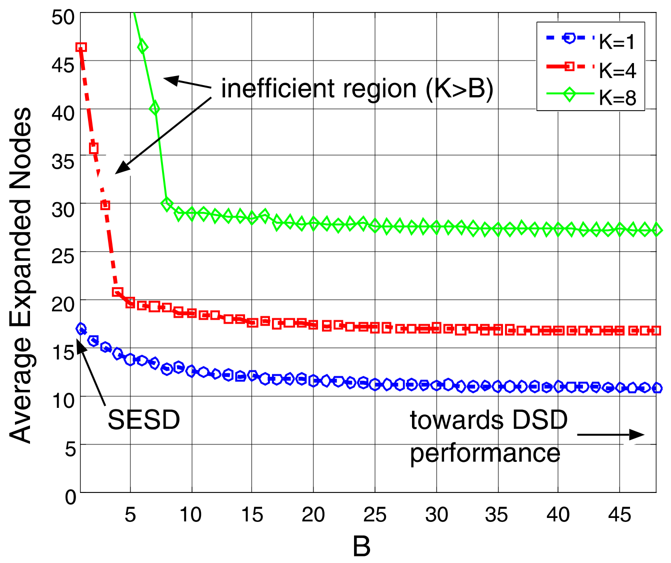

Fig. 3. Average expanded nodes $\overline{n_e}$, simulated for 320k channel realizations in a $4 \times 4$ MIMO system using 64-QAM at 25dB SNR.

*1) Expand Complexity:* Fig. 3 shows the *average expand complexity* $\overline{n_e}$. For small $K$ and increasing $B$ the average expand complexity decreases which corresponds to the result in [6] where it has been proven that the *automatic sphere decoder* – which corresponds to the DSD – has the lowest $\overline{n_e}$ among all sphere decoders since it performs an unconstrained metric-first search. Unfortunately, this configuration requires a large ammount of memory (cf. Section IV-B and Fig. 2) and is therefore not suited for practical implementations.

Since $K$ determines the expand effort, i.e., the maximum number of nodes which are expanded for the current boundary $\mathcal{B}$, the expand complexity grows with $K$. In the region where $K > B$, more nodes are expanded than searched, which quickly becomes inefficient in terms of $\overline{n_e}$.

*2) Search Complexity:* Fig. 4 shows the *average search complexity* $\overline{n_s}$. Note that the search complexity is larger than the expand complexity, since each node may be searched more than once. However for small boundaries $B$, the overall complexity is mostly dominated by the expand complexity as the search in the boundary can be computed in parallel to expanding nodes. Additionally, search units are usually smaller than an expand circuit.

Observing Fig. 4 it can be concluded that the optimum decoder in terms of $n_s$ is achieved by the configuration $K = B = 1$, which is equal to the SESD algorithm. However, this property has not been proven so far.

For large $K$, the search effort becomes prohibitive. Expanding many nodes from a boundary, also expands nodes which have large PED. Thus, the search for the lowest PED becomes difficult since the boundary size $B$ is limited and good PEDs may lie outside. Additional sorting of the nodes on equal levels in $\mathcal{S}$ according to their PEDs reduces this problem only slightly and causes additional sorting complexity.

## D. Trade-offs

*1) Search vs. Expand Complexity:* Comparing Fig. 3 and Fig. 4, we observe (especially for low $K$) that a tradeoff between search and expand complexity exists. For an increasing active boundary size (i.e., increased metric-first ability) $B$ the

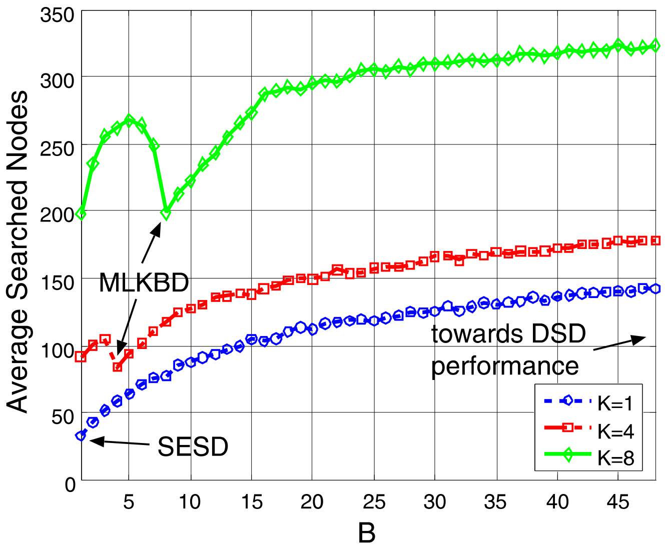

Fig. 4. Average searched nodes $\overline{n_s}$, simulated for 320k channel realizations in a $4 \times 4$ MIMO system using 64-QAM at 25dB SNR.

average search complexity grows, while the expand complexity decreases.

Increasing the boundary size lowers the expand complexity since potentially better children can be found earlier. Additionally, as searching might be done in parallel to expanding, increasing $B$ appears to be a valuable technique to speed up the total detection complexity $n$. The optimum choice of $B$ in terms of runtime effort, can be found by minimizing $n$. We claim from experience that in practical systems, the optimum choice of $B$ is either one or only slightly larger.

*2) Decoding Complexity vs. Memory Consumption:* If $B$ is increased, e.g., to lower the total detection complexity, the memory consumption grows (cf. Fig. 2). Thus, $B$ has to be chosen such that the overall complexity of the decoding unit is low and the memory consumption remains moderate. As shown before, a small increase in metric-first ability, already reduces the average expand complexity, especially for higher order modulation schemes (e.g. 64-QAM). Therefore, the actual implementation of the search and expand units decides on the optimal trade-off.

Starting with an SESD implementation, e.g. [8], and slightly increasing the metric-first ability $B$, can reduce the overall complexity of the MIMO ML detection unit. However, the expand effort of the decoder should be set to $K = 1$ to avoid high computational complexity.

### *E. Early Estimates*

The presented algorithm may find intermediate estimates, while searching for the ML solution. Such *early estimates* can be used if the search must be terminated before completion. This technique is called *early termination* [9] and is no longer ML-optimal. The parameters $K$ and $B$ determine the *delivery window* for early estimates, which is essential for scheduling techniques [9], [10].

The maximum number of $n_e$ until the first estimate is available equals to the case where the most memory is consumed (cf. Section IV-B). The SESD delivers its first estimate always after expanding $M_T$ nodes and the cases where $zK = B$ and $K = 1$ deliver estimates corresponding to (7). The configuration $B = K$ corresponds to the MLKBD which expands all nodes in the active boundary. With early termination after the first leaf has been found, the MLKBD corresponds to the *K-best decoder* (KBD) [4], which has shown to achieve close-to ML error performance. The DSD delivers no early estimates and might require exponential growing expand complexity which is not suited for scheduling.

The minimum $n_e$ to find an estimate, occurs if the detector advances as fast as possible from the root to the leaves and depends on $K$. If $K = 1$ the first estimate may be available after $M_T$ expanded nodes and thus, the SESD and the DSD may deliver their first estimate after expanding $M_T$ nodes. However, the SESD *always* delivers its first estimate after expanding $M_T$ nodes (since $B = 1$). The maximum expand complexity of the DSD is equal to the worst-case computational complexity.

## V. Conclusion

The presented algorithm allows a unified view over ML-optimal tree-search decoders by exploring its adjustable metric-first ability and adjustable expand effort. It has been shown that the Schnorr-Euchner sphere decoder (SESD) requires the lowest memory and is already very efficient in terms of average search complexity. Dijkstra's search in the symbol tree outperforms the SESD in terms of expand complexity. However, this approach requires a prohibitively large amount of memory and is therefore not well-suited for VLSI implementation. It is shown that there exists a trade-off region between computational complexity and memory consumption. Exploiting this trade-off region reveals that slightly enhancing the metric-first ability of the SESD might be desirable. Depending on the chosen implementation strategy, the overall detection complexity of the decoder can be reduced, especially if decoding higher order symbol alphabets (e.g. 64-QAM) or employ a large number of transmit antennas.

## References

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, “Closest point search in lattices,” *IEEE Transactions on Information Theory*, vol. 48, no. 8, pp. 2201–2214, Aug. 2002.

[2] E. W. Dijkstra, “A note on two problems in connexion with graphs,” in *Numerische Mathematik*. Mathematisch Centrum, Amsterdam, The Netherlands, 1959, vol. 1, pp. 269–271.

[3] T. Fukatani, R. Matsumoto, and T. Uyematsu, “Two methods for decreasing the computational complexity of the MIMO ML decoder,” *IEICE Trans. Fundamentals*, vol. E87-A, no. 10, pp. 2571–2576, Oct. 2004.

[4] K. Wong, C. Tsui, R.-K. Cheng, and W. Mow, “A VLSI architecture of a $K$-best lattice decoding algorithm for MIMO channels,” in *Proc. IEEE ISCAS'02*, vol. 3, May 2002, pp. 273–276.

[5] W. Xu, Y. Wang, Z. Zhou, and J. Wang, “A computationally efficient exact ML sphere decoder,” in *Global Telecommunications Conference*, vol. 4, Nov. 2004, pp. 2594–2598.

[6] K. Su, “Efficient maximum likelihood detection for communication over multiple input multiple output channels,” Ph.D. dissertation, University of Cambridge, Feb. 2005.

[7] A. Paulraj, R. Nabar, and D. Gore, *Introduction to Space-Time Wireless Communications*, 1st ed. Cambridge University Press, 2003.

[8] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bölcskei, “VLSI implementation of MIMO detection using the sphere decoding algorithm,” *IEEE Journal of Solid-State Circuits*, July 2005.

[9] C. Studer, “Sphere decoding with resource constraints,” Master's thesis, ETH Zurich, Switzerland, Aug. 2005.

[10] A. Burg, M. Borgmann, M. Wenk, C. Studer, and H. Bölcskei, “Advanced receiver algorithms for MIMO wireless communications,” in *Proc. Design, Automation, and Test in Europe (DATE)*, Mar. 2006.