Complexity Reduction for Sphere Decoding using Unum-Type-II-Based SORN-Arithmetic

Simon Knobbe, Moritz Bärthel*, Steffen Paul* and Jochen Rust*
*Institute of Electrodynamics and Microelectronics (ITEM.me)
University of Bremen, Bremen, Germany, +49(0)421/218-62553
Email: sknobbe@uni-bremen.de, {baerthel, steffen.paul, rust}@me.uni-bremen.de

***Abstract*—In this paper the complexity of a Sphere Decoding algorithm is reduced by using a Unum type-II-based SORN preprocessor, which excludes a number of possible solutions in advance. Two different methods for permuting the reduced symbol tree are discussed and evaluated for different SORN datatypes, sorting algorithms and MIMO system sizes. It is shown that the processing time of the Sphere Decoding algorithm by means of the visited nodes depending on the SNR can be reduced by 17% to 52%.**

***Index Terms*—Digital Signal Processing, MIMO, Sphere Decoding, Unum, SORN**

I. Introduction and Related Work

In communication technologies, MIMO systems (Multiple Input Multiple Output) with multiple antennas at the transmitter and receiver stations are often used [1] in order to increase channel capacity and expand the data rate [2]. During the data transmission the signals of the different antennas are superimposed and then have to be separated again at the receiver station. For this detection, different methods like Zero-Forcing [3] or Sphere Decoding (SD) [4] can be exploited.

Beside the classical digital number formats fixed-point and IEEE floating-point, the Universal Number (Unum) format offers new possibilities for implementing those algorithms. While the initial Unum format (type-I) was developed as an extension to classical floating-point to handle numerical instabilities and provide runtime variable datawidths [5], further developments led to type-II Unums. This approach is based on low complexity and fast computing resulting in Sets-Of-Real-Numbers (SORNs) which use open intervals and exact values and provide a coarse quantization of the reals [6]. This number format can be utilized in order to reduce the amount of possible solutions for an optimization problem as exemplary shown in [6]. In [7] the number of possible symbol combinations in order to determine the maximum likelihood solution (ML) for a MIMO transmission problem is reduced by using a Unum type-II-based SORN preprocessor.

In this paper we apply SORN based signal processing for non-linear SD-based signal detection in MIMO wireless communication systems. In detail, we reduce the search tree of the Sphere Decoder using the SORN preprocessor and then permute it with two different approaches in order to reach a lower overall computing time.

Our paper is organized as follows: In sec. II, the basics of Sphere Decoding and the SORN preprocessing are introduced. Sec. III explains how to change the tree structure of the Sphere Decoder by permuting the symbol vectors. In Sec. IV, our approaches are evaluated on a simulation basis. Finally, the paper is concluded in sec. V.

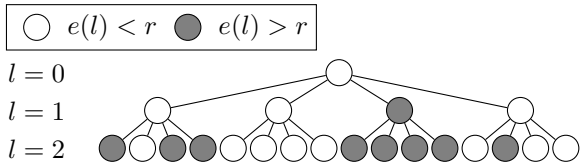


Fig. 1. Idea of Sphere Decoding: The decoder observes only the white nodes with an error smaller than r.

II. Reduction of the Solution Set

A. MIMO-Transmission Setup

For a discrete modulation alphabet (e.g. m-PSK, QAM) the set of possible combinations for transmitting signals $\boldsymbol{X}$ at the transmitter side is finite. As transmitting signal, the symbol vector $\boldsymbol{x} \in \mathbb{C}^{N_t}$ is assumed with N_t as number of transmitting antennas. Assuming an Additive White Gaussian Noise-channel (AWGN) with a channel matrix $\boldsymbol{H} \in \mathbb{C}^{N_r \times N_t}$ the received signal vector can be described by

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n} \tag{1}$$

with $\boldsymbol{y} \in \mathbb{C}^{N_r}$ and N_r as number of receiving antennas. In this paper the same number of antennas $N = N_t = N_r$ on the transmitter and receiver side are assumed. For symbol detection the channel matrix $\boldsymbol{H}$ is assumed to be known at the receiver side. Hence the receiver has to detect the best symbol combination

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \boldsymbol{X}} ||\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}||_2^2 \tag{2}$$

either via exhaustive search or using linear or non-linear solvers like Zero-Forcing or Sphere Decoding.

B. Sphere Decoding

The main idea of Sphere Decoding is to detect all solutions in a given radius r. By exploiting an ordinary QR-decomposition [8], equation (2) can be written as

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \boldsymbol{X}} ||\underbrace{\boldsymbol{Q}^{\boldsymbol{H}}\boldsymbol{y}}_{=:\hat{\boldsymbol{y}}} - \boldsymbol{R}\boldsymbol{x}||_2^2 \tag{3}$$

with a unitary matrix $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ and an upper triangular matrix $\boldsymbol{R} \in \mathbb{C}^{N \times N}$. This allows to compute the error element

TABLE I
9 BIT SORN

decimal:	$-\infty$	$(-\infty,-1)$	-1	$(-1,0)$	0	$(0,1)$	1	$(1,\infty)$	∞
SORN:	100000000	010000000	001000000	000100000	000010000	000001000	000000100	000000010	000000001

wise by starting with the last element of $\boldsymbol{x}$. Each possible symbol yields to a finite number of new branches with error nodes as shown in figure 1. Based on equation (3) the error at a node at tree level l can be computed recursively by

$$e(l) = \Big| \sum_{i=N-l+1}^{N} (R_{N-l+1,i}\ x_i) - \hat{y}_{N-l+1} \Big|^2 + e(l-1) \quad (4)$$

with $e(l-1)$ as error of the previous level and $e(0) = 0$. According to the Schnorr-Euchner algorithm [9], at each node the branch with the lowest error will be taken first. For each branch the error increases at each node, such that branches with a higher error than the search radius r can be neglected. After reaching the lowest level of the tree the search radius can be adapted as described in [4].

C. SORN Preprocessing

The Sets-Of-Real-Numbers (SORN) datatype is derived from the Unum type-II representation which displays the real numbers with exact values and open intervals in between. Two possible SORN configurations taken from the original work[1] [6] are composed of the values

$$\begin{aligned} \mathcal{L}_9 &= \{0,\ 1,\ \infty\} \\ \mathcal{L}_{17} &= \{0,\ 0.5,\ 1,\ 2,\ \infty\}, \end{aligned} \quad (5)$$

the corresponding negative values and the open intervals in between. Tab. I shows the mapping of the SORN datatype for the first configuration, the second one is built in a similar way. For SORN arithmetic at first inputs are converted into the respective SORN datatype. The arithmetic operations are then executed using pre-computed look-up tables (LUTs). These LUTs can be efficiently implemented in hardware using simple Boolean Logic. Applied to the ML-Estimation problem from equation. (2), the SORN-error $E(\boldsymbol{x}_i) = ||\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}_i||_2^2$ is computed for every possible symbol vector $\boldsymbol{x}_i$ from the set $\boldsymbol{X}$ via exhaustive search. Depending on the result $E(\boldsymbol{x}_i)$, a certain amount of symbol vectors $\boldsymbol{x}_i$ can be considered as possible solutions for equation (2). They build the reduced solution set $\boldsymbol{X_s}$ which is the result of the preprocessing and is passed to the Sphere Decoder.

Since the SORN preprocessing is not the main focus of this work, a more detailed description of SORN arithmetic, the structure of the LUTs and the application to the MLE can be found in [7].

Analog to the implementation in [7], figure 2 shows the mean number of remaining symbol vectors after SORN preprocessing for 4×4 and 8×8 MIMO-Systems with 20000 simulations. Using the Unum type-II based 17-bit-SORNs the reduction is much higher than using the 9-bit-SORNs.

[1]The only difference are the endpoints ∞ and $-\infty$ which are separated here while the original work uses a combined value $\pm\infty$.

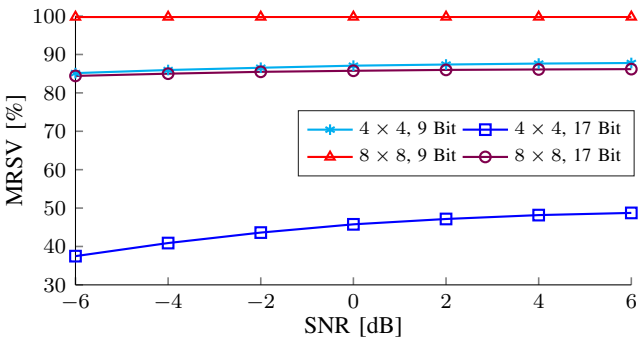
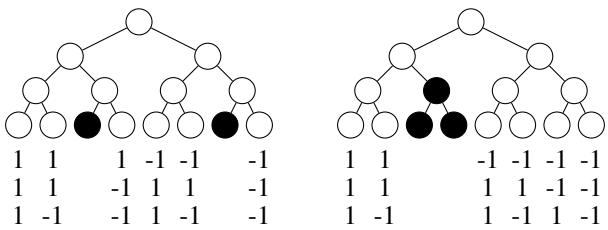


Fig. 2. Mean remaining symbol vectors (MRSV) in % of 20000 simulations for a 4×4 and a 8×8 MIMO-System.

Similarly, the 4×4 MIMO-system leads to a higher reduction than the 8×8 system. In the best case reduction rates of about 40% are reached, a 8×8 system with 9-bit-SORNs on the other hand leads to an insignificant reduction.

III. Permutation of the Solution Set

Excluding solutions with the SORN preprocessor has the effect of cutting off branches from the search tree. This leads to different subtrees with smaller branches including less solutions at the lowest level and greater branches including many solutions. Smaller branches have lower information contents but faster computing times than greater branches. The reduced solution set $\boldsymbol{X_s}$ can be reordered to concentrate or to offset this effect. Practically, the reordering has the meaning of interpreting the signals of the different antennas in a new order. In this paper we will discuss two possibilities of permuting the solution set $\boldsymbol{X_s}$: One with a balanced ratio of subtree sizes and one with an unbalanced ratio. In figure 3 an example of a 3-dimensional BPSK-system is shown. The original solution set in figure 3(a) was reduced from 8 to 6 symbol vectors. Both subtrees from the point of view of the first level results into possible solutions, the ratio of the subtrees is balanced. By exchanging the first and third line in all symbol vectors, the solution set changes as shown in figure 3(b). The right subtree includes four solutions, the left one two solutions, which is an unbalanced ratio. Also



(a) Balanced Sorting (b) Unbalanced Sorting

Fig. 3. 3-Dimensional BPSK with different permutations: The black circles represent excluded nodes. In this example for the Balanced Sorting (a) 2 nodes are deleted. The Unbalanced Sorting (b) leads to 3 deleted nodes.

the number of nodes changed: Without the root node the tree from figure 3(a) includes 12 nodes, the tree from figure 3(b) decreases to 11 nodes. The effects of the two different permutation methods on Sphere Decoding will be discussed in section IV.

A. Computing the permutation

In the following steps an algorithm to find a permutation is explained. The main idea of the sorting algorithm is to count the number of solutions included in each branch at each tree level and to find a suitable balanced or an unbalanced sorting. The algorithm presented in code 1 determines the permutation order.

For description the symbol vectors $\boldsymbol{s}_l(k) \in \mathbb{C}^l$ for $k = 1, ..., m^l$ are introduced. The symbol vector $\boldsymbol{s}_l$ describes all possible combinations of the given modulation for a l-Dimensional set. As an example for a QPSK ($m = 4$) with $l = 2$:

$$\boldsymbol{s}_2(1) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1+j \\ 1+j \end{pmatrix}, ..., \boldsymbol{s}_2(16) = \frac{1}{\sqrt{2}} \begin{pmatrix} -1-j \\ -1-j \end{pmatrix} \quad (6)$$

The counter c_{lik} specifies the occurrence of the l-dimensional symbol vector $\boldsymbol{s}_l(k)$ in the N-dimensional solution set $\boldsymbol{X}_{\boldsymbol{s}}$ with regard to a certain line i. To provide a measure for the size of the subtrees the standard deviation is used:

$$\sigma_{li} := \text{std}(c_{li}) := \sqrt{\frac{1}{m^l} \sum_{k=1}^{m^l} (c_{lik} - \mu_{c_{li}})^2} \quad (7)$$

Therefore $\mu_{c_{li}}$ is the average value and m the modulation size, which leads overall to m^l symbols at level l. The approach of the algorithm in code 1 is to start with the highest level and find the line from $\boldsymbol{X}_{\boldsymbol{s}}$, that minimizes or maximizes the standard deviation for a balanced or unbalanced tree sorting. With regards to the hardware complexity of the sorting algorithm the square root and the division with m^i are not relevant for the comparison. Also the mean value is identical for all lines, because $\sum_{k=1}^{m^l} c_{lik}$ is constant for all lines i.

$$\sum_{k=1}^{m^l} (c_{lik} - \mu_{c_{li}})^2 = \underbrace{\sum_{k=1}^{m^l} c_{lik}^2}_{\text{relevant part}} \underbrace{- 2\mu_{c_{li}} \sum_{k=1}^{m^l} c_{lik} + \sum_{k=1}^{m^l} \mu_{c_{li}}^2}_{\text{constant part}} \quad (8)$$

An equivalent comparison is to compute the squared sum of the occurrence $T_{li} := \sum_{k=1}^{m^l} c_{lik}^2$. At the following levels a similar approach is taken, but the number of symbols increases to m^l combinations. The parameter T_{di} determines the measure for the subtree ratio depending on the previous lines.

Note that the symbols of $\boldsymbol{x}$ in equation (4) are detected upside down, such that the permutation order is turned. Permuting the lines of $\boldsymbol{x}$ also leads to permuting the lines of $\hat{\boldsymbol{y}}$ and the columns of $\boldsymbol{H}$. Consequently the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ of the QR-decomposition are changed.

In Tab. II the values for the standard deviation and the squared sum for the examples from figure 3 are presented.

TABLE II
STANDARD DEVIATION OF FIGURE 3

Tree	Level l	σ_{li}	T_{li}	c_{li1}	c_{li2}	c_{li3}	c_{li4}
(a) balanced	1	0	18	3	3	-	-
	2	0.5	10	2	1	2	1
(b) unbalanced	1	1	20	2	4	-	-
	2	0.87	12	2	0	2	2

It can be seen that an unbalanced sorting leads to greater values of squared sum and standard deviation than a balanced sorting.

B. Approximation of the permutation

The problem of the sorting algorithm in code 1 is that the complexity increases exponentially with the tree level. As an example for an 8-Dimensional QPSK $4^7 = 16384$ possible symbol-vectors have to be counted. Especially at the lower levels the optimal sorting might not be required. To avoid the complexity increase in code 2 an alternative sorting algorithm is presented. The sorting of this algorithm based only on the knowledge of the occurrence of the 1-dimensional symbols $\boldsymbol{s}_1(k)$. By using the multidimensional Taylor series first order [10], it can be shown that the squared sum can be approximated as follows:

$$T_{li} = \sum_{k=1}^{m^l} c_{lik}^2 \approx \frac{1}{m^{l-1}} \sum_{k=1}^{m} c_{1ik}^2 = \frac{T_{1i}}{m^{l-1}} \quad (9)$$

Using this approximation the permutation can be determined by sorting the squared sum of the first level. To compare

Code 1 Sort Tree Exact Version

Input: $\boldsymbol{X}_{\boldsymbol{s}}, \boldsymbol{s}_l$ ▷ matrix with all remaining symbol vectors, vector of symbol combinations

Output: $\boldsymbol{p} \in \mathbb{N}^N$ ▷ permutation order

1: **for** $l = 1, ..., N-1$ **do** ▷ Determine permutation elements
2: Set $\boldsymbol{C} \in \mathbb{N}^{(N-l+1) \times m^l}$ as
3: $c_{lik} := \sum_{\kappa} \left(\boldsymbol{X}_{\boldsymbol{s}\{\underbrace{p_N, ...p_{N-l+2}}_{\text{not existent for } l=1}, i\}\kappa} = \boldsymbol{s}_l(k) \right)$ ▷ count symbol vectors $s_l(\kappa)$
4: **for** $i = 1, ..., N$ **do**
5: Set $\boldsymbol{T}_l \in \mathbb{N}^N$ with $T_{li} := \sum_{k=1}^{m} c_{lik}^2$ ▷ Compute squared sum (eq. (8))
6: **end for**
7: $p_{N-l+1} = \arg\max_i(T_{li})$ ▷ use max for unbalanced and min for balanced branches
8: **end for**

Code 2 Sort Tree Approximative Version

Input: $\boldsymbol{X}_{\boldsymbol{s}}, \boldsymbol{s}_l$ ▷ matrix with all remaining symbol vectors, vector of symbol combinations

Output: $\boldsymbol{p} \in \mathbb{N}^N$ ▷ permutation order

1: Set $\boldsymbol{C} \in \mathbb{N}^{N \times m}$ as $c_{ik} := \sum_{\kappa} \left(\boldsymbol{X}_{\boldsymbol{s}\kappa}(i) = \boldsymbol{s}_1(k) \right)$
2: Set $\boldsymbol{T} \in \mathbb{N}^N$ with $T_i := \sum_{j=1}^{m} c_{ij}^2$
3: $\boldsymbol{p} = \arg\text{sort}_i(T_i)$ ▷ use *ascend* for balanced and *descend* sorting for unbalanced branches

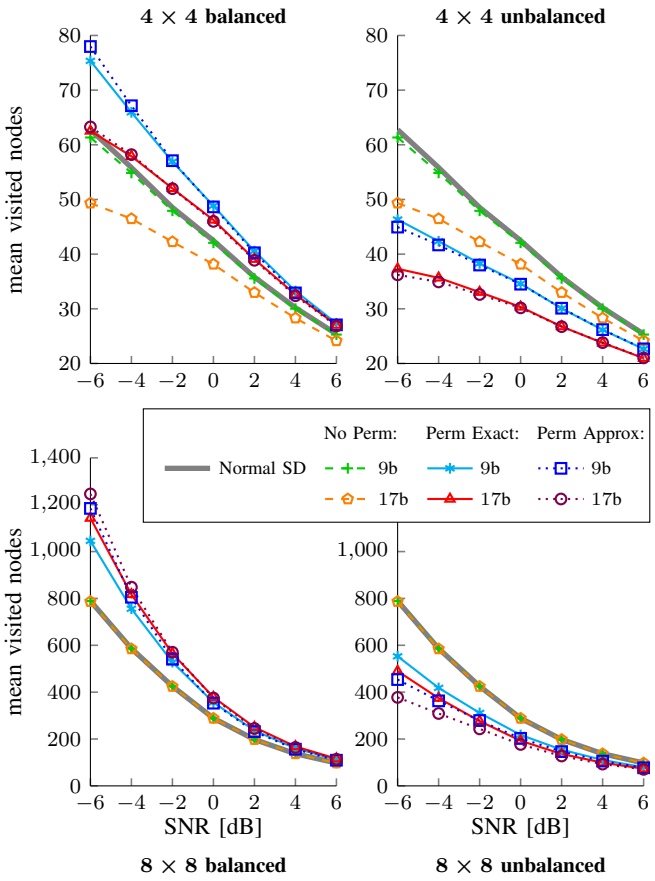


Fig. 4. Mean visited nodes of 20000 simulations for 4×4 and 8×8 MIMO systems.

the different lines, the factor $\frac{1}{m^{l-1}}$ is irrelevant and can be disregarded. Because of the turned computing order of the Sphere Decoding, a descend sorting of T_{1i} leads to an unbalanced permutation, an ascend sorting to a balanced permutation. The results of the approximation will be shown in the following section.

IV. Simulation Results

To evaluate the presented approach, a simulation with a conventional Sphere Decoder and the explained different permutations is executed. In this simulation every Sphere Decoder uses the Schnorr-Euchner algorithm with an adaptive search radius $r_0 = \infty$. For reduction of the solutions Unum type-II-based 9-Bit and 17-Bit SORN-preprocessors [6][7] are used.

As metric, the number of *visited nodes* [4] is used. Every node error from equation (4) which is calculated is counted as a *visited node*. In contrast to floating-point-operations (FLOPs) the greater computation effort of lower tree levels is not considered by this metric. But in reference to a hardware implementation this measure might be useful, because the same hardware could be used for different tree levels. In figure 4 the results of 20000 simulations for a 4×4 and 8×8 MIMO-system are presented. The average number of visited nodes is visualized depending on the SNR. Reducing the symbol tree using 9-bit-SORNs without any permutation leads to a very slight reduction of the visited nodes. Using 17-bit SORNs leads to a significant reduction in particular in a negative SNR-range. For a balanced permuting all curves lie above the normal Sphere Decoding, which means a deterioration of the performance. An unbalanced permutation results in a performance improvement. The different sorting algorithms from code 1 and 2 differ minimally but the approximated algorithm which is easier to implement leads to slightly better results. Overall the computing time of the Sphere Decoder is reduced to values between 42.30% (SNR $= -6\,\mathrm{dB}$) and 17.19% (SNR $= 6\,\mathrm{dB}$).

The results of the 8×8 MIMO-system also show reductions for an unbalanced tree permutation. The approximative version of the permutation again leads to better results than the exact one. Particularly, for negative SNR (52.15% for SNR $= -6\,\mathrm{dB}$) the profit is much higher in comparison to the 4×4-system.

In relation to the overall effort of the Sphere Decoder it should be noted, that only the number of visited notes is decreased. The cost of the approximative version of the permutation can be determined by line 2-3 in code 2 and are estimated by $N\times(m-1)$ additions, $N\times m$ square operations and $N/2\,(N-1)$ comparisons which all are integer operations. The SORN preprocessing also leads to a small increase in complexity which is considered in [7].

V. Conclusion

This paper presents the application of reducing the solution set of a Sphere Decoder via 9-bit and 17-bit SORN-arithmetics. It is shown that the processing time of the Sphere Decoder can be reduced by 17% to 52% using 4×4 and 8×8 QPSK-MIMO-Systems and an unbalanced sorting algorithm. For future work, it must be considered that the SORN preprocessing and the permutation algorithm reduce the achieved performance gain by means of complexity and computing time.

References

[1] E. G. Larsson, "MIMO detection methods: How they work," *IEEE signal processing magazine (Print)*, vol. 26, no. 3, pp. 91–95, 2009.

[2] L. G. Barbero and J. S. Thompson, "Fixing the complexity of the sphere decoder for MIMO detection," *IEEE Transactions on Wireless communications*, vol. 7, no. 6, pp. 2131–2142, 2008.

[3] Q. H. Spencer, A. L. Swindlehurst, and M. Haardt, "Zero-forcing methods for downlink spatial multiplexing in multiuser MIMO channels," *IEEE transactions on signal processing*, vol. 52, no. 2, pp. 461–471, 2004.

[4] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of solid-state circuits*, vol. 40, no. 7, pp. 1566–1577, 2005.

[5] J. L. Gustafson, *The end of error: Unum computing*. Boca Raton: CRC Press, 2015.

[6] ——, "A radical approach to computation with real numbers," *Supercomputing frontiers and innovations*, vol. 3, no. 2, pp. 38–53, 2016.

[7] M. Bärthel, P. Seidel, J. Rust, and S. Paul, "SORN Arithmetic for MIMO Symbol Detection - Exploration of the Type-2 Unum Format," in *2019 17th IEEE International New Circuits and Systems Conference (NEWCAS)*, June 2019, pp. 1–4.

[8] E. Süli and D. F. Mayers, *An introduction to numerical analysis*. Cambridge university press, 2003.

[9] C.-P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Mathematical programming*, vol. 66, no. 1-3, pp. 181–199, 1994.

[10] J. J. Duistermaat and J. A. C. Kolk, *Distributions : Theory and Applications*, ser. Cornerstones. New York, NY: Birkhauser, 2010, chapter 6, page 61.