ARCHITECTURE DESIGN AND IMPLEMENTATION OF THE INCREASING RADIUS - LIST SPHERE DETECTOR ALGORITHM

Markus Myllylä, Markku Juntti*

Centre for Wireless Communications
FIN-90014 University of Oulu, Finland
{markus.myllyla, markku.juntti}@ee.oulu.fi

Joseph R. Cavallaro

Dept. of Electrical & Computer Engineering
Rice University, Houston, TX 77251,USA
cavallar@rice.edu

ABSTRACT

A list sphere detector (LSD) is an enhancement of a sphere detector (SD) that can be used to approximate the optimal MAP detector. In this paper, we introduce a novel architecture for the increasing radius (IR)-LSD algorithm, which is based on the Dijkstra's algorithm. The parallelism possibilities are introduced in the presented architecture, which is also scalable for different multiple-input multiple-output (MIMO) systems. The novel architecture is implemented on a Virtex-IV field programmable gate array (FPGA) chip using high-level ANSI C++ language based Catapult C Synthesis tool from Mentor Graphics. The used word lengths, the latency of the design, and the required resources are presented and analyzed for 4×4 MIMO system with 16- quadrature amplitude modulation (QAM). The detector implementation achieves a maximum throughput of 12.1Mbps at high signal-to-noise ratio (SNR).

Index Terms— Sphere, LSD, architecture, implementation

1. INTRODUCTION

Multiple-input multiple-output (MIMO) channels offer improved capacity and significant potential for improved reliability compared to single antenna channels. Sphere detector (SD) calculates the hard output maximum likelihood (ML) solution with reduced complexity compared to full-complexity ML detectors [1]. A list sphere detector (LSD) [2] is a variant of the sphere detector that can be used to approximate a MAP detector, which is the optimal detector for forward error coded (FEC) systems with lower complexity [2, 3, 4].

The sphere algorithms are often divided into breadth-first search algorithms, such as the K-best algorithm [4], and sequential search algorithms, such as the Schnorr-Euchner enumeration (SEE) based algorithms [3]. The architecture design is important for an efficient implementation, and architecture solutions with different levels of parallelism have been introduced for the most common K-best and SEE sphere algorithms, e.g., in [3, 4, 5]. In this paper, we consider a sequential search algorithm, namely the increasing radius (IR)-LSD algorithm, which is a modification of Dijkstra's algorithm [6] to a LSD algorithm and optimal in the sense of visited number of nodes in the sphere search tree structure [6, 7]. We identify and introduce the key functional units of the IR-LSD algorithm, and design a novel, highly parallel and scalable architecture for the algorithm. The designed architecture is implemented on a Virtex-IV field programmable gate array (FPGA) chip for 4×4 MIMO system with 16- quadrature amplitude modulation (QAM). The implementation is done using Mentor Graphics' Catapult ® C Synthesis tool with high-level ANSI C++ language, which is then completely synthesized through the tool to produce the resulting RTL. We present the complexity and latency of the implementation and describe the major challenges.

The paper is organized as follows. The MIMO signal detection, and the IR-LSD algorithm are presented in Section II. The designed architecture is presented in Section III. The implementation of the algorithm is presented in Section IV. The conclusions are drawn in Section V.

*This work was done in MITSE project which was supported by Elektrobit, Nokia, Nokia-Siemens Networks, Texas Instruments and the Finnish Funding Agency for Technology and Innovation, Tekes. The authors would like to thank Mentor Graphics for the possibility to evaluate Catapult C ® Synthesis tool.

2. MIMO SIGNAL DETECTION

A narrowband system with N_T transmit and N_R receive antennas is considered with assumption $N_R \geq N_T$ and QAM constellation. The received signal can be expressed in real domain as [1]

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \boldsymbol{\eta}, \tag{1}$$

where the received signal vector $\mathbf{y} \in \mathbb{R}^{2N_R \times 1}$, the transmit symbol vector $\mathbf{x} \in \Omega_R^{2N_T} \subset \mathbb{R}^{2N_T \times 1}$ and the Gaussian noise vector $\boldsymbol{\eta} \in \mathbb{R}^{2N_R \times 1}$ are defined in the frequency domain. The channel matrix $\mathbf{H} \in \mathbb{R}^{2N_R \times 2N_T}$ contains real Gaussian coefficients with unit variance. The complex QAM constellation Ω is transformed into real symbol alphabet with Q_R bits per symbol $\Omega_R \subset \mathbb{Z}$, e.g., $\Omega_R = \{-3, -1, 1, 3\}$ in the case of 16-QAM. We assume a practical case of system with FEC and with separate soft-output detector and decoder at the receiver, where the detector generates soft output information $L_{\mathrm{D1}}(b_k)$ of each transmitted bit b_k [2].

The SDs find the ML solution of $\mathbf{x}$ with reduced complexity compared to exhaustive search algorithms. Then the sphere search is done by limiting the search to the points inside a $2N_R$-dimensional hyper-sphere $S(\mathbf{y}, \sqrt{C_0})$ centered at $\mathbf{y}$. After QR decomposition (QRD) of the channel matrix $\mathbf{H}$, the condition can be written as [1]

$$||\tilde{\mathbf{y}} - \mathbf{R}\mathbf{x}||_2^2 \leq C_0, \tag{2}$$

where C_0 is the squared radius of the sphere, $\mathbf{R} \in \mathbb{R}^{2N_R \times 2N_T}$ is an upper triangular matrix with positive diagonal elements, $\mathbf{Q} \in \mathbb{R}^{2N_R \times 2N_R}$ is an orthogonal matrix, and $\tilde{\mathbf{y}} = \mathbf{Q}^{\mathrm{T}}\mathbf{y}$. Due to the upper triangular form of $\mathbf{R}$ the values of $\mathbf{x}$ can be solved from (2) level by level using the back-substitution algorithm. Let $\mathbf{x}_i^{2N_T} = (x_i, x_{i+1}, \ldots, x_{2N_T})^{\mathrm{T}}$ denote the last $2N_T - i + 1$ components of the vector $\mathbf{x}$. The squared partial Euclidean distance (PED) of $\mathbf{x}_i^{2N_T}$

can be calculated as [3]

$$d(\mathbf{x}_i^{2N_T}) = d(\mathbf{x}_{i+1}^{2N_T}) + |\tilde{y}_i - \sum_{j=i}^{2N_T} R_{i,j}x_j|^2 \\ = d(\mathbf{x}_{i+1}^{2N_T}) + |b_{i+1}(\mathbf{x}_{i+1}^{2N_T}) - R_{i,i}x_i|^2, \quad (3)$$

where $d(\mathbf{x}_{2N_T}^{2N_T}) = 0$, $b_{i+1}(\mathbf{x}_{i+1}^{2N_T}) = \tilde{y}_i - \sum_{j=i+1}^{2N_T} R_{i,j}x_j$, $R_{i,j}$ is the (i,j)th term of $\mathbf{R}$ and $i = 2N_T, \ldots, 1$. Depending on the search strategy and the channel realization, the SD searches a variable number of nodes in the tree structure, and aims to find the point $\mathbf{x} = \mathbf{x}_1^{2N_T}$, also called a leaf node, for which the ED $d(\mathbf{x}_1^{2N_T})$ is minimum.

The list sphere detector (LSD) [2] can be used for obtaining a list of candidates of the transmitted symbol vectors and the corresponding EDs $\mathcal{L} \in \mathbb{Z}^{N_{\text{cand}} \times 2N_T}$ as an output, where N_{cand} is the size of the candidate list so that $1 \leq N_{\text{cand}} \leq 2^{Q_R 2N_T}$. The LSD output candidate list can then be used to approximate the log-likelihood ratio (LLR) of the transmitted data as a soft output. The increasing radius (IR) - LSD algorithm is listed as Algorithm 1. The algorithm operates in a sequential fashion starting from the root layer, and extends the partial candidate $\mathbf{s} = \mathbf{x}_{i+1}^{2N_T}$ with the next best admissible node x_i. The father candidate $\mathbf{s}_f = \mathbf{x}_{i+2}^{2N_T}$ is also, if admissible node x_{i+1} exists, extended. The variables n_1 and n_2 indicate the order number of the next best node, i.e., inform how many nodes have been checked, for the child and father candidates, respectively. The algorithm uses two memory sets: the final candidate memory $\mathcal{L}$, which is the size of N_{cand} candidates and the partial candidate memory $\mathcal{S}$, which size depends on the algorithm iterations, i.e., $while$ loop repetitions. The stored partial candidate information $\mathcal{N}(\mathbf{s}, d(\mathbf{s}), n_2, i)$, which is stored to $\mathcal{S}$, includes the partial candidate, the PED, the number of extended father nodes, and the current layer, respectively. The information stored to the final list $\mathcal{L}$ includes only the candidate and the ED, and the C_0 is updated according to the largest ED in $\mathcal{L}$. After each iteration, the algorithm continues with the partial candidate $\mathcal{N}$ with the minimum PED from $\mathcal{S}$ until $d(\mathbf{s}) < C_0$.

Algorithm 1 $[\mathcal{L}]$ = IR-LSD$(\tilde{\mathbf{y}}, \mathbf{R}, \mathbf{N}_{\text{cand}}, \mathbf{\Omega}_{\mathbf{R}}, \mathbf{N}_{\mathbf{T}})$

1: Initialize sets $\mathcal{S}$ and $\mathcal{L}$, and set $C_0 = \infty$, $m = 0$, $n_1 = 1$
2: Initialize $\mathcal{N}(\mathbf{s} = \mathbf{x}_{2N_T}^{2N_T}, d(\mathbf{s}) = 0, n_2 = 2, i = 2N_T - 1)$
3: **while** $d(\mathbf{s}) < C_0$ **do**
4: Determine the n_1th best node x_i for $\mathbf{s}_c = (x_i, \mathbf{x}_{i+1}^{2N_T})^{\mathrm{T}}$ and calculate $d(\mathbf{s}_c)$
5: Determine the n_2th best node x_{i+1} for father candidate $\mathbf{s}_f = (x_{i+1}, \mathbf{x}_{i+2}^{2N_T})^{\mathrm{T}}$ and calculate $d(\mathbf{s}_f)$ if $n_2 \leq |\Omega_R|$
6: **if** $d(\mathbf{s}_c) < C_0$ **then**
7: **if** $\mathbf{s}_c$ is a leaf node **then**
8: Store $\mathcal{N}_F(\mathbf{s}_c, d(\mathbf{s}_c))$ in $\{\mathcal{L}\}_m$
9: Set $m = m + 1$ or, if the set is full, set m according to $\{\mathcal{L}\}_m$ with max ED and $C_0 = d(\mathbf{s})_m$
10: Continue with $\mathcal{N}(\mathbf{s}, d(\mathbf{s}), n_1 + +, 1)$ if $n_1 + 1 \leq |\Omega_R|$
11: **else**
12: Store $\mathcal{N}_c(\mathbf{s}_c, d(\mathbf{s}_c), n_2 = 2, i - -)$ in $\mathcal{S}$
13: **end if**
14: **end if**
15: **if** $\mathcal{N}_f$ calculated and $d(\mathbf{s}_f) < C_0$ **then**
16: Store $\mathcal{N}_f(\mathbf{s}_f, d(\mathbf{s})_f, n_2 + +, i)$ in $\mathcal{S}$
17: **end if**
18: Continue with $\mathcal{N}$ with min PED from $\mathcal{S}$ and set $n_1 = 1$
19: **end while**

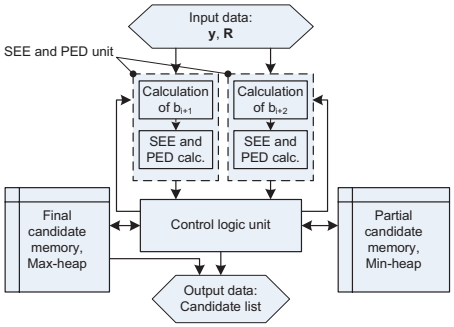

Fig. 1. An architecture for the IR-LSD algorithm.

3. ARCHITECTURE

A list sphere detector consists of the QRD, the LSD algorithm, and the log-likelihood ratio (LLR) calculation units. The QRD unit decomposes the channel matrix **H** into **R** and **Q**, which are given as an input with **y** to the LSD algorithm. The LSD algorithm unit executes the sphere tree search and determines the output candidate list $\mathcal{L}$. The approximation of $L_{\text{D}}(b_k)$ is calculated in the LLR calculation unit using the candidate list $\mathcal{L}$. In this paper, we focus our attention to the architecture of the IR-LSD algorithm.

The architecture for IR-LSD algorithm is shown in Figure 1, and it consists of two SEE and PED units, a partial candidate memory unit, a final candidate memory unit, and a logic unit. The SEE and PED units define and extend the selected partial candidate and its father node with the next best admissible nodes and calculates the PEDs of the updated candidates. The partial candidate memory unit is used to store the already extended partial candidates while the leaf candidates are stored to the final candidate memory unit. The logic unit defines the candidate(s) to be extended and stored in the next algorithm iteration.

The IR-LSD algorithm architecture, which is presented in Figure 1, is designed to have as much parallel processing as possible to decrease the overall latency of one algorithm iteration. In one algorithm iteration, the algorithm studies one or two new nodes of the sphere search tree depending if the father node is extendable or not. The two SEE and PED units are designed to execute the algorithm description lines 4 and 5 in parallel. The control logic unit executes the logic between lines 6−18, and defines the candidates to be stored and the candidate to be extended in the next iteration. This means that the candidates extended in the iteration round 1 are stored to the memory in iteration round 2. The storing of the partial candidates in lines 12 and 16 to the partial memory unit and the storing of final candidate in line 8 to the final memory unit is then done in parallel with the SEE and PED units. Thus, the total latency of one algorithm iteration is equal to the latency of the highest latency parallel unit plus the latency of the control unit. The total latency of one signal vector detection process, i.e., one algorithm run, is dependent on the required number of algorithm iterations, which is also relative to the number of checked nodes in the search tree. The required number of iterations depends on the system configuration and the channel environment.

3.1. SEE and PED Unit

There are two similar SEE and PED units in the IR-LSD architecture as shown in Figure 1. The units execute the partial candidate extension in lines 4 and 5 in the algorithm description separately in

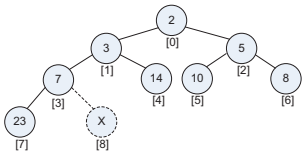

Fig. 2. A min-heap memory architecture.

parallel. Both units are not used in all of the cases, but in practice both units are occupied $> 90\%$ of the time. Each SEE and PED unit is divided into two subunits as shown in Figure 1.

The first unit calculates the $b_{i+1}(\mathbf{x}_{i+1}^{2N_T})$, which is the part of PED calculation that is independent from the new symbol x_i, as in (3). The unit can be implemented with different levels of parallelism to get faster calculation of the multiplication (MUL) operations. The number of required multiplications in the calculation of $b_{i+1}(\mathbf{x}_{i+1}^{2N_T})$ is $2N_T - i - 1$, where i is the current layer in the search tree and $i_{max} = 2N_T$. It should be noted that the average layer in the search process is less than half of the tree hight N_T, because a larger ratio of the search process is done in the upper part of the tree.

The second unit executes the SEE, i.e., determines the n_1th best node x_i, and calculates the PED of the extended candidate accordingly. The SEE is done in a slightly modified fashion from the way presented in (14) in [1]. Instead of calculating the costly and high latency division operation, we calculate the (3) with Ω_R different symbols x_i, what can be implemented with $1 - |\Omega_R|$ parallel MUL and subtraction (SUB) operations. Then the $|\Omega_R|$ values are sorted and the PED is calculated with the n_1th best node. The architecture could be designed to determine first the symbol with minimum PED and then determine the n_1th best node using logic the same way as presented in [1].

3.2. Memory Units

The memory units are designed as binary heap data structures [8], which keep the stored elements in order according to selected definition. The partial candidate memory set $\mathcal{S}$ is implemented as min-heap, where the stored elements $\mathcal{N}(\mathbf{s}, d(\mathbf{s}), n_2, i)$ are ordered so that the candidate with minimum PED is always sorted to be at the top of the heap. The final memory set $\mathcal{L}_F$ is implemented as max-heap, where the stored elements $\mathcal{N}(\mathbf{s}, d(\mathbf{s}))$ are sorted according to the maximum PED. A binary min-heap tree structure is illustrated in Figure 2, where the value of the memory slot is illustrated inside the circle and the memory address underneath the circle.

The used operations with heap memory are read min/max, extract-min/max, and insert new. The running time of read min/max operation is $O(1)$ [8],i.e., it requires just a memory read of the base address. The insert and extract-min/max operations running time is $O(\log_2(k))$ in the worst case [8], where k is size of the memory and $\lceil \log_2(k) \rceil$ is the height of the tree. In the insert operation, we store the information to the next available memory slot, which is illustrated as inserting the value X to the address 8 in Figure 2. The value is swapped to correct level with up-heap operation. The operation requires at least one read and write operation of the memory, and it is repeated maximum of $\lfloor \log_2(k) \rfloor$ times until the new element is in its correct place. The extract-max/min operation extracts the base address element and replaces it with either a new element or the last element of the memory. Then the down-heap operation is executed to move the element in the right position. The down-heap operation, which requires at least two memory reads and one memory write, is repeated also a maximum of $\lfloor \log_2(k) \rfloor$ times until the new added element is at the correct position.

Table 1. Determined word lengths for the real IR-LSD algorithm.

Signal	$\tilde{\mathbf{y}}$	$\tilde{\mathbf{R}}$	Ω	$\mathbf{b}_i(\mathbf{s})$	$d(\mathbf{s})$
(W,I)	(10,4)	(9,3)	(8,1)	(12,5)	(10,5)

In the IR-LSD architecture, the sizes of $\mathcal{L}_F$ and $\mathcal{S}$ are equal to the required list size N_{cand} and to the maximum number of algorithm iterations. In the worst case, one extract-min and one insert operations are required to the partial memory unit in each iteration. The partial memory size can be decreased by introducing a separate sphere constraint for the stored candidates. Therefore, with a proper choice of the sphere constraint, the required memory size is decreased without any performance loss. The sphere constraint can be determined to be, e.g., relative to the previous largest candidate in the final list or relative to the partial candidate search level.

3.3. Scalability

The architecture can be used as such in systems with different numbers of transmit antennas N_T and constellation size Ω. The number of transmit antennas N_T and constellation Ω effect the size of the search tree and, thus, effect the required number of algorithm iterations to detect the transmitted signal vector $\mathbf{x}$. The maximum number of required iterations is equal to the number of required elements in the partial candidate memory. Also the required operations by the SEE in SEE and PED unit depend on the constellation size Ω. The proper final list size also varies with system configuration.

4. IMPLEMENTATION

The IR-LSD algorithm architecture was implemented with real signal model for 4×4 MIMO system with 16-QAM. The performance of a turbo coded system was studied with a real IR-LSD, sorted QRD (SQRD) [9] preprocessing and log-MAP LLR calculation in an uncorrelated (UNC) channel as shown in the left subplot in Figure 3. The LSD candidate list size was selected as $N_{\text{cand}} = 15$ and the used fixed-point word lengths are listed in Table 1, where **W** and the **I** refer to the total number of bits and the number of bits used for the integer part representation, respectively. We also studied the required maximum and average number of iterations D_{max} and D_{avg} for 10% target frame error rate (FER) by the LSD algorithm with different SNR as shown in the right subplot in Figure 3. It can be seen that the required maximum number of iterations D_{max} decreases with increasing SNR and the IR-LSD algorithm requires as low as $D_{min} = 9$ iterations, i.e., 18 studied nodes, in high SNR environment to reach target FER. The size of the partial candidate memory is selected as $D_{max} = 80$ to support the lower SNR operation.

4.1. Complexity and Latency

The Catapult C ® Synthesis tool output RTL was synthesized with Mentor Graphics Precision RTL and the FPGA place and route operation was done with Xilinx ISE software for Xilinx Virtex-IV chip with $f_s = 150$ MHz clock frequency. The device utilization of Xilinx Virtex-IV chip and the latencies of the units in terms of clock cycles (cc:s) Δ_{tot} are shown in Table 2. The latencies of the memory units are calculated according to the average number of heap operations, and the units are implemented in dual port RAM memory, which enable two parallel read/write operations. The total latency of the detector iteration consists of the latencies of the slowest parallel unit, which is the SEE and PED unit, and of the control logic unit.

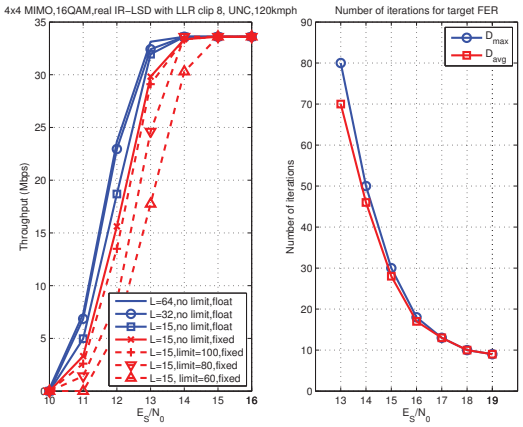

Fig. 3. Throughput vs SNR (left) and alg. iter. vs SNR (right).

Table 2. Device utilization for Xilinx Virtex-IV chip and latencies.

Resource	CLB Slices	BRAMs	DSP48s	Latency
SEE&PED x2	536	0	3	17cc
Partial mem.	522	4	0	16cc
Final mem.	291	2	0	11cc
Control logic	144	0	0	5cc
Total	2029	6	6	22cc

The total guaranteed throughput of the implementation can be calculated as

$$\text{Throughput} = \frac{2N_T Q_R f_s}{\Delta_{tot} D_{avg}} \text{bits/s}. \quad (4)$$

The maximum guaranteed throughput is then 12.1 Mbps at $\gamma = 21$ dB for 10% target FER with $D_{avg} = 9$. However, the guaranteed implementation throughput is 1.6 Mbps at $\gamma = 13$ dB, which can be considered as the worst case scenario with $D_{avg} = 70$. Thus, the throughput is mainly dependent on the number of iterations D_{avg}.

4.2. Discussion and Comparison to Other Work

The main limiting factor for higher throughput is the latency of one algorithm iteration. The total number of iterations can only be lowered with some lattice reduction techniques or by sacrificing the performance of the detector. The latency of one algorithm iteration is currently limited by the SEE and PED unit, and it could be lowered by, e.g., ASIC implementation. As far as the authors know, there has not been any architecture designs or implementations of the IR-LSD algorithm in the literature.

The parallel nature of the introduced architecture makes the algorithm implementation competitive against the current state of the art depth first algorithm or soft output detectors [10, 11]. Some parallelism can be added by dividing the search into separate real and imaginary branches as in the hard output work in [10]. However, the increased throughput results in approximately double the complexity and decreased performance. The complex signal model leads to higher maximum throughput in [11], but results in more complex units and higher average number of visited nodes. The main interest in practice is the performance and the complexity of the implementation in the worst case scenario, because the implementation has to be able to work in those conditions. As the other works typically present the maximum throughput results, a direct comparison of other work is difficult, but the authors believe that with possible ASIC implementation the IR-LSD algorithm is a very good competing alternative.

5. CONCLUSIONS

We designed and introduced a novel and scalable architecture for the IR-LSD algorithm with parallel processing units. The architecture was designed so that the main operations of the algorithm can be run in parallel and it is scalable to different configurations with minor changes. An implementation of the architecture was presented for 4×4 system with 16-QAM on a Virtex-IV FPGA chip. The complexity and the latency of the implementation were presented and analyzed. The throughput of the current implementation could be enhanced, e.g., with an ASIC implementation.

6. REFERENCES

[1] M. O. Damen, H. El Gamal, and G. Caire, "On maximum–likelihood detection and the search for the closest lattice point," *IEEE Trans. Inform. Theory*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[2] B. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. Commun.*, vol. 51, no. 3, Mar. 2003.

[3] Andreas Burg, Moritz Borgmann, Markus Wenk, Martin Zellweger, Wolfgang Fichtner, and Helmut Bölcskei, "VLSI Implementation of MIMO Detection Using the Sphere Decoding Algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, July 2005.

[4] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection," *IEEE J. Select. Areas Commun.*, vol. 24, no. 3, Mar. 2006.

[5] B. Widdup, G. Woodward, and G. Knagge, "A Highly-Parallel VLSI Architecture for a List Sphere Detector," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, 20-24 June 2004, pp. 2720–2725.

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

[7] W. Xu, Y. Wang, Z. Zhou, and J. Wang, "A Computationally Efficient Exact ML Sphere Decoder," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov. 29–Dec. 3 2004, vol. 4, pp. 2594–2598.

[8] D. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition*, Addison-Wesley, 1997.

[9] D. Wübben, R. Böhnke, V. Kühn, and K. Kammeyer, "MMSE extension of V-BLAST based on sorted QR decomposition," in *Proc. IEEE Veh. Technol. Conf. (VTC)*, Orlando, Florida, Oct. 6–9 2003, vol. 1, pp. 508–512.

[10] X. Huang, C. Liang, and J. Ma, "System architecture and implementation of MIMO sphere decoders on FPGA," *IEEE Trans. VLSI Syst.*, vol. 16, no. 2, pp. 188 – 197, Feb. 2008.

[11] C. Studer, A. Burg, and H. Bolcskei, "Soft-output sphere decoding: algorithms and VLSI implementation," *IEEE J. Select. Areas Commun.*, vol. 26, no. 2, pp. 290 – 300, Feb. 2008.