Best-First Tree Search with Probabilistic Node Ordering for MIMO Detection: Generalization and Performance-Complexity Tradeoff

Ronald Y. Chang and Wei-Ho Chung, *Member, IEEE*

***Abstract*—The tree representation of the multiple-input multiple-output (MIMO) detection problem is illuminating for the development, interpretation, and classification of various detection methods. Best-first detection based on Dijkstra's algorithm pursues tree search according to a sorted list of tree nodes. In the first part of the paper, a new probabilistic sorting scheme is developed and incorporated in a modified Dijkstra's algorithm for MIMO detection. The proposed sorting exploits the statistics of the problem and yields effective tree exploration and truncation in the proposed algorithm. The second part of the paper generalizes the results in the first part and removes some limitations. A generalized Dijkstra's algorithm is developed as a unified tree-search detection framework. The proposed framework incorporates a parameter triplet that allow the configuration of the memory usage, detection complexity, and sorting dynamic associated with the tree-search algorithm. By tuning different parameters, desired performance-complexity tradeoffs are attained and a fixed-complexity version can be produced. Simulation results and analytical discussions demonstrate that the proposed generalized Dijkstra's algorithm shows abilities to achieve highly favorable performance-complexity tradeoffs.**

***Index Terms*—Maximum likelihood (ML) decoding, multiple-input multiple-output (MIMO) systems, tree-search detection, Dijkstra's algorithm.**

I. Introduction

THE multiple-input multiple-output (MIMO) detection problem can be graphically interpreted as a tree-search problem [1]. Maximum likelihood (ML) detection, relying on exhaustive search over the entire tree, is theoretically optimal yet computationally prohibitive in practical systems. To remedy the infeasibility of ML detection, various optimal and suboptimal detectors with reduced complexity have been proposed. These detection methods can be broadly classified into one of the three main categories according to the order in which tree-nodes are visited, with self-explanatory names: depth-first search, breadth-first search, and best-first search [2]. Sphere decoding (SD) and its variants [3]–[9] belong to the depth-first search, which have fixed memory requirements yet varying search complexity and throughput. The K-best detector [10]–[12] and its predecessor M-algorithm [13] belong to the breadth-first search, which have fixed memory requirements and fixed search complexity, and can be efficiently implemented in a pipelined fashion. The stack or Dijkstra's algorithm [2], [14], [15] belongs to the best-first search, which requires a minimum number of searched tree-nodes to establish the ML solution [14], but has varying search complexity and potentially large memory requirements. Improvements have been suggested to address associated issues for each category of schemes. For example, a fixed-complexity version of SD was proposed in [16] and a memory-constrained Dijkstra's algorithm was proposed in [17]–[19]. Hybrid schemes were also developed to leverage the advantages of different tree-search schemes, e.g., one that combines the memory-efficient SD and computationally-efficient Dijkstra's algorithm [20], [21]. Furthermore, unification frameworks building on the relation between different categories of schemes were suggested, such as the generic branch-and-bound algorithm in [2] and the unified algorithm that augments the SD with the best-first ability to improve search complexity in [21].

Dijkstra's algorithm-based best-first detector is attractive for its computational efficiency. In the meantime, its memory usage, varying complexity, and sorting scheme need to be addressed to support use in practice and/or achieve enhanced performance. The memory issue was addressed in [17]–[19], where an explicit memory constraint was assigned to the algorithm. While the resulting performance penalty was quantified, the algorithm's potential for achieving improved performance given a pre-specified memory size was not discussed. The concept of *bias* was introduced to the sorting scheme in [2]; however, the choice of bias was largely heuristic and its effect on the resulting algorithm was not examined.

In this paper, the application of Dijkstra's algorithm to solving the MIMO detection problem is thoroughly investigated. The paper contains two logically related parts. In the first part, a probabilistic sorting scheme that specifically consults the statistics of the problem is developed to enhance the original algorithm. The properties of the new sorting scheme are studied analytically, and the performance of the modified algorithm, termed Dijkstra's Search with Probabilistic Sorting (DSPS) algorithm, is demonstrated numerically. The proposed algorithm is shown advantageous over the original algorithm in terms of complexity and performance for the cases of highly constrained memory.

The second part of the paper tackles several issues observed

Manuscript received March 29, 2011; revised August 11, 2011 and October 31, 2011; accepted November 24, 2011. The associate editor coordinating the review of this paper and approving it for publication was H. Ochiai.

This work was presented in part at the IEEE International Conference on Communications (ICC), Kyoto, Japan, June 5–9, 2011. This work was supported in part by the National Science Council of Taiwan under Grant NSC 100-2221-E-001-004, and in part by the Industrial Technology Research Institute of Taiwan under the Ministry of Economic Affairs project.

R. Y. Chang and W.-H. Chung (corresponding author) are with the Research Center for Information Technology Innovation, Academia Sinica, Taipei 115, Taiwan (e-mail: yjrchang@gmail.com; whc@citi.sinica.edu.tw).

Digital Object Identifier 10.1109/TWC.2011.121911.110568

in the DSPS algorithm. Specifically, the DSPS algorithm 1) inherits the varying complexity and latency of the original Dijkstra's algorithm, 2) does not guarantee the ML solution even with unbounded memory, and 3) does not flexibly produce desired performance-complexity tradeoffs. To address these issues, we first propose a new configurable sorting scheme that includes the conventional and probabilistic sorting as special cases. Then, we develop a unified framework, in the form of Generalized Dijkstra's Search (GDS) algorithm, that incorporates a parameter triplet used for controlling the different facets of the algorithm. By tuning different parameters, the aforementioned issues such as extensive memory requirements, varying search complexity, and inflexibility associated with Dijkstra's algorithm are properly addressed, and improved performance-complexity tradeoffs can be attained.

The rest of the paper is organized as follows. In Sec. II, the system model and problem formulation are presented, and Dijkstra's algorithm for tree-search detection is reviewed. The DSPS algorithm and its properties and performance are studied in Sec. III. The GDS framework and thorough discussions are presented in Sec. IV. Finally, concluding remarks are given in Sec. V.

II. System Model and Best-First Tree Search Detection

A. System Model and Problem Definition

We consider an uncoded MIMO transmission system with N_T transmit antennas and N_R receive antennas ($N_R \geq N_T$). The baseband signal model is given by

$$\mathbf{y}_c = \mathbf{H}_c\tilde{\mathbf{x}}_c + \mathbf{v}_c \tag{1}$$

where $\mathbf{y}_c$ is the $N_R \times 1$ received signal containing the $N_T \times 1$ transmitted signal $\tilde{\mathbf{x}}_c$ perturbed by the $N_R \times N_T$ flat-fading channel $\mathbf{H}_c$ and the $N_R \times 1$ noise $\mathbf{v}_c$. The transmitted symbol vector $\tilde{\mathbf{x}}_c$ contains uncorrelated entries drawn equally probably from the squared quadrature amplitude modulation (QAM) alphabet $\mathbb{S} = \{a + ib \mid a, b \in \mathbb{Q}\}$, where $\mathbb{Q}$ is the pulse amplitude modulation (PAM) alphabet, and has zero mean and covariance matrix $\sigma_x^2\mathbf{I}_{N_T}$, where $\mathbf{I}_{N_T}$ is the $N_T \times N_T$ identity matrix. The complex-valued channel matrix $\mathbf{H}_c$ has independent and identically distributed (i.i.d.) Gaussian entries with zero mean and covariance matrix $\sigma_H^2\mathbf{I}_{N_R}$, where $\sigma_H^2 = 1$. The channel information $\mathbf{H}_c$ is assumed known perfectly at the receiver, but not at the transmitter. The noise $\mathbf{v}_c$ is additive white Gaussian noise (AWGN) with i.i.d. complex elements and has zero mean and covariance matrix $\sigma_v^2\mathbf{I}_{N_R}$, where $\sigma_v^2 > 0$.

The complex signal model in (1) can be transformed into an equivalent real signal model by defining

$$\mathbf{y} = \begin{bmatrix}\Re(\mathbf{y}_c)\\ \Im(\mathbf{y}_c)\end{bmatrix}, \tilde{\mathbf{x}} = \begin{bmatrix}\Re(\tilde{\mathbf{x}}_c)\\ \Im(\tilde{\mathbf{x}}_c)\end{bmatrix}, \mathbf{v} = \begin{bmatrix}\Re(\mathbf{v}_c)\\ \Im(\mathbf{v}_c)\end{bmatrix},$$
$$\mathbf{H} = \begin{bmatrix}\Re(\mathbf{H}_c) & -\Im(\mathbf{H}_c)\\ \Im(\mathbf{H}_c) & \Re(\mathbf{H}_c)\end{bmatrix}$$

where $\Re(\cdot)$ and $\Im(\cdot)$ denote the real and imaginary parts of its argument, respectively. The real signal model is given by

$$\mathbf{y} = \mathbf{H}\tilde{\mathbf{x}} + \mathbf{v} \tag{2}$$

where $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{H} \in \mathbb{R}^{n\times m}$, $\tilde{\mathbf{x}} \in \mathbb{Q}^m$, $\mathbf{v} \in \mathbb{R}^n$ with $n = 2N_R$ and $m = 2N_T$. Given the signal model in (2), optimal ML detection is equivalent to solving a constrained least-square problem, i.e.,

$$\tilde{\mathbf{x}}_{\text{ML}} = \arg\min_{\mathbf{x}\in\mathbb{Q}^m} \|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2 \tag{3}$$

where $\|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2$ is the *likelihood metric* for some $\mathbf{x}$ given $\mathbf{y}$ and $\mathbf{H}$, and $\|\cdot\|$ is the l_2-norm of a vector.

B. Dijkstra's Algorithm for Tree-Search Detection

The detection problem in (3) can be represented as a tree search problem [1]. This can be seen by first performing QR decomposition on $\mathbf{H}$ such that $\mathbf{H} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper-triangular matrix with positive-valued diagonal entries (assuming $m = n$ and $\mathbf{H}$ has full rank), and then formulating ML detection into an equivalent expression $\tilde{\mathbf{x}}_{\text{ML}} = \arg\min_{\mathbf{x}\in\mathbb{Q}^m} \|\mathbf{y}' - \mathbf{R}\mathbf{x}\|^2$, where $\mathbf{y}' = \mathbf{Q}^T\mathbf{y}$ and $(\cdot)^T$ denotes the matrix transpose. Due to the upper-triangular nature of $\mathbf{R}$, $\|\mathbf{y}' - \mathbf{R}\mathbf{x}\|^2$ can be expanded as

$$(y'_m - r_{m,m}x_m)^2 + \Big(y'_{m-1} - \sum_{i=m-1}^{m} r_{m-1,i}x_i\Big)^2 + \cdots + \Big(y'_1 - \sum_{i=1}^{m} r_{1,i}x_i\Big)^2 \tag{4}$$

where y'_i is the ith element of $\mathbf{y}'$, x_i is the ith element of $\mathbf{x}$, and $r_{i,j}$ is the (i,j)-entry of $\mathbf{R}$. Denote the $(m-k+1)$th term in (4) by $B_k(\mathbf{x}_k^m)$ and the summation of the first $m-k+1$ terms by $D_k(\mathbf{x}_k^m)$, $k = 1, 2, \ldots, m$, i.e.,

$$B_k(\mathbf{x}_k^m) = \Big(y'_k - \sum_{i=k}^{m} r_{k,i}x_i\Big)^2 \tag{5}$$

$$D_k(\mathbf{x}_k^m) = \sum_{j=k}^{m} B_j(\mathbf{x}_j^m) \tag{6}$$

where $\mathbf{x}_k^m \triangleq (x_k, \ldots, x_m)^T$ represents the partial symbol vector. With $D_{m+1}(\mathbf{x}_{m+1}^m) \triangleq 0$, $D_k(\mathbf{x}_k^m)$ can be obtained recursively via the following relation

$$D_k(\mathbf{x}_k^m) = D_{k+1}(\mathbf{x}_{k+1}^m) + B_k(\mathbf{x}_k^m). \tag{7}$$

The upper-triangular structure of $\mathbf{R}$ creates the detection tree, which consists of a virtual root node and m layers of nodes where each non-leaf node has $|\mathbb{Q}|$ child nodes, $|\cdot|$ denoting the cardinality of a set. Each node in layer $m-k+1$ ($k = 1, 2, \ldots, m$) uniquely represents an $\mathbf{x}_k^m$ and has an associated *path metric* $D_k(\mathbf{x}_k^m)$ and *branch metric* $B_k(\mathbf{x}_k^m)$. In particular, $D_1(\mathbf{x}_1^m)$ of a leaf node in layer m equals the likelihood metric $\|\mathbf{y}' - \mathbf{R}\mathbf{x}\|^2$ evaluated for the particular $\mathbf{x} = \mathbf{x}_1^m$ represented by the node. Hence, the objective of optimal detection is to find the leaf node with the smallest path metric among all leaf nodes. Fig. 1 illustrates an example of the detection tree.

Dijkstra's algorithm for finding the shortest path in a graph can be applied to the detection tree to solve the detection problem [14], [15], [17]–[19]. Dijkstra's algorithm maintains a list of nodes sorted in ascending order of their defined cost and explores the nodes in such order (thus "best-first"). The cost

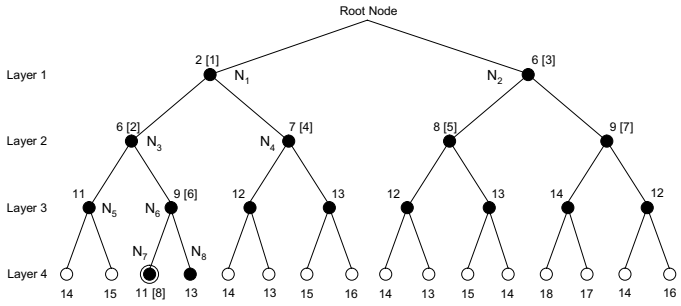


Fig. 1. An example of MIMO detection based on the DS(∞) algorithm for a 2×2 system with 4-QAM ($m = 4$ and $|\mathbb{Q}| = 2$). The number labeled next to a node is its path metric, and the number in the brackets is the order in which the node is selected as the best node and thus expanded. Filled nodes represent the total visited nodes. In this example, the number of expanded and visited nodes are 8 and 16, respectively. The circled leaf node is the optimal (ML) solution.

(denoted by c) of a node is defined as the node's path metric D in the conventional algorithm,[1] which is termed Dijkstra's Search (DS) algorithm in this paper. The DS algorithm with a list size (memory) constraint L ($L \in \mathbb{Z}^+$, where $\mathbb{Z}^+$ is the set of positive integers) [17]–[19] is described as follows.

Algorithm: DS(L)

Initially, the node list $\mathbb{N}$ contains only the root node.

1) Select the best (first) node from $\mathbb{N}$. If this node is in layer m (i.e., a leaf node), stop the algorithm and output it as the solution.
2) Expand the best node by adding all its child nodes to $\mathbb{N}$ and removing itself from $\mathbb{N}$.
3) Order the nodes in $\mathbb{N}$ in ascending order of their cost $c = D$.[2] Retain the first $\min(|\mathbb{N}|, L)$ nodes and discard others. Go to step 1.

An illustrative example of tree-search detection based on the DS(∞) algorithm is shown in Fig. 1, where the algorithm iterations are depicted. Visited (expanded) nodes are defined by those that ever occupy a position (become the best node) in $\mathbb{N}$ during the execution of the algorithm. Note that DS(L) guarantees to find the ML solution when $L = \infty$ and a near-ML solution with L sufficiently large [18].

III. Dijkstra's Search with Probabilistic Sorting (DSPS) Algorithm

The DS algorithm with a memory constraint does not guarantee to find the ML solution. Nodes on the perfect decoding path may be discarded if they are not among the "good" nodes preserved by the algorithm at an intermediate time. Since the path metric of nodes from different layers constitute different partial sums in the likelihood metric according to (7), the path metric does not objectively represent the "cost" of nodes from the detection point of view. In the following, we present the proposed modified algorithm adopting a new cost function that leverages the statistical properties of the path metric to achieve more efficient node maintenance and tree exploration.

A. Algorithm Description

In the tree representation, in each layer k there is exactly one node that corresponds to the actually transmitted $\tilde{\mathbf{x}}_{m-k+1}^m$. The path metric for this particular node, $D_{m-k+1}(\tilde{\mathbf{x}}_{m-k+1}^m)$, is the summation of the square of k random variables that are i.i.d. $\mathcal{N}(0, \sigma_v^2/2)$ and is distributed according to the chi-square distribution with k degrees of freedom. Its cumulative distribution function (cdf) is given by

$$F(x;k) = \frac{\gamma(k/2, x/\sigma_v^2)}{\Gamma(k/2)}, \quad x \geq 0, k = 1, 2, \ldots, m \tag{8}$$

where Γ is the Gamma function and γ is the incomplete Gamma function defined as $\gamma(s,x) = \int_0^x t^{s-1}e^{-t}dt$ [22]. We let F^{-1} be the inverse function of F, such that $x = F^{-1}(y;k)$ if $y = F(x;k)$.

The statistics of the path metric have been used to control different tree pruning strategies for SD in [4]. The sphere radius effectively decreases for lower layers, resulting in a different pruning behavior in the modified SD algorithm. Here, we propose to leverage the statistics of the path metric from a different perspective, namely, to yield more effective node ordering in Dijkstra's algorithm-based detection. We first define the *normalized path metric* of a node $\mathbf{x}_{m-k+1}^m$ in layer k, $k = 1, 2, \ldots, m$, as[3]

$$G_N(\mathbf{x}_{m-k+1}^m) \triangleq F\big(D_{m-k+1}(\mathbf{x}_{m-k+1}^m); k\big). \tag{9}$$

Then, we adopt a new cost function defined according to the normalized path metric in the proposed DSPS algorithm. Since the normalized path metric better represents the "cost" of nodes from different layers, improved performance and/or complexity results can be achieved in the case of limited memory. Substituting $c = G_N$ in the DS(L) algorithm gives the DSPS(L) algorithm.

An illustrative example of tree-search detection based on the DSPS(∞) algorithm is shown in Fig. 2. To exemplify the difference between DS(∞) and DSPS(∞) algorithms, we trace the first several detection iterations. The beginning iterations are identical for both algorithms until node N_3 is selected as the best node and expanded. At this point, the node list contains four nodes N_2, N_4, N_5, N_6, which are ordered as N_2, N_4, N_6, N_5 by the DS(∞) algorithm and N_6, N_4, N_5, N_2 by the DSPS(∞) algorithm.[4] As a result, in the next iteration, N_2 (N_6) is selected as the best node by the DS(∞) (DSPS(∞)) algorithm, as illustrated in Figs. 1–2. This representative example also signifies the visited node reduction property of DSPS, which will be verified in Sec. III-D.

B. Properties of Probabilistic Sorting

The probabilistic sorting according to $c = G_N$ inherits many nice properties of the conventional sorting according to $c = D$ and introduces a new, favorable one. Specifically, two sorting schemes are equivalent in the following three cases: 1) sorting two nodes from the same layer, 2) breaking

[1]Throughout this paper, we use D to refer to the path metric generally and $D_k(\mathbf{x}_k^m)$ to denote the path metric of a specific node; same for c and, later, G_N and $G^{(T)}$.

[2]Ties are broken by ordering nodes in higher layers (closer to the bottom of the tree) before nodes in lower layers, but otherwise arbitrarily.

[3]It is termed *normalized* because a node's path metric relative to the distribution of the path metric of the transmitted partial symbol vector in the same layer is considered.

[4]The ordering by the DSPS algorithm shown here is an example. In practice, G_N is computed and used for ordering the nodes.

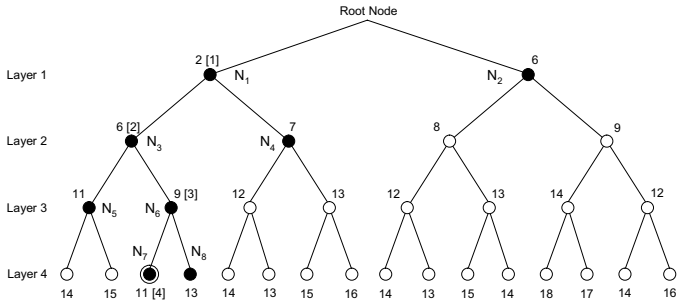


Fig. 2. An example of MIMO detection based on the DSPS(∞) algorithm for a 2×2 system with 4-QAM ($m = 4$ and $|\mathbb{Q}| = 2$). Notations follow those in Fig. 1. In this example, the number of expanded and visited nodes are 4 and 8, respectively.

ties, and 3) sorting two nodes from different layers where the node in the higher layer has a smaller path metric. This can be easily verified by noting the properties of F, namely, $F(x;k) < F(x';k)$ for $x < x'$, $F(x;k) < F(x;k')$ for $k > k'$, and $F(x;k) < F(x';k')$ for $k > k'$ and $x < x'$. There is however one case in which these two sorting strategies may produce different results. When sorting two nodes from different layers where the node in the higher layer has a bigger path metric (e.g., N_2 and N_6 in Figs. 1–2), the probabilistic sorting may or may not order the node with a smaller path metric before the node with a bigger path metric. In fact, this is exactly the case when the path metric of two nodes cannot unambiguously determine which node is better. The probabilistic sorting makes sensible decisions by referencing the statistics of the path metric.

Note that the path metric of layer-k nodes that do not correspond to the actually transmitted partial symbol vector $\tilde{\mathbf{x}}_{m-k+1}^m$ is distributed according to the *noncentral* chi-square distribution, whose cdf is denoted by $F_{nc}(x;k,\lambda)$, where $\lambda \geq 0$ is a node-dependent noncentrality parameter. Thus, it may appear more reasonable to use $F_{nc}(x;k,\lambda)$ in defining G_N for these nodes and in sorting accordingly. However, by adopting $F(x;k)$ for *all* nodes, a disadvantage is posed against "bad" nodes in sorting since $F(x;k) \geq F_{nc}(x;k,\lambda)$ for any given values of $x \geq 0$, $\lambda \geq 0$, and $k \in \mathbb{Z}^+$. This is a highly desirable property in the presence of a memory constraint, as the objective of a detector is to preserve nodes that are on the perfect decoding path and discard those that are not.

C. Complexity Analysis

An analytical exposition is presented in this subsection to offer insights on the complexity differences between the DS and DSPS algorithms as a result of different sorting strategies. Similar to [14], we use the number of visited nodes as a useful abstraction of the actual complexity involved. Denote the number of visited nodes by $J^{(v)}$, the number of expanded nodes by $J^{(e)}$, and the number of expanded nodes in layer $m-k+1$ by $J_k, k = 1, 2, \ldots, m$. By observing the DS and DSPS algorithms, we have $J^{(v)} = J^{(e)} \times |\mathbb{Q}|$ and $J^{(e)} = \sum_{k=1}^{m} J_k$. Thus, analyzing $J^{(e)}$ is sufficient to understand the complexity represented by $J^{(v)}$. A straightforward lower-bound on $J^{(v)}$ is given by $J^{(v)} \geq m|\mathbb{Q}|$, since $J_k \geq 1$ and $J^{(e)} \geq m$. To gain further insights on $J^{(v)}$, we first introduce some properties.

Proposition 1: (DS(L) algorithm) Given fixed $\mathbf{H}$, $\tilde{\mathbf{x}}$, and $\mathbf{v}$, in the case of infinite L, if a node $\mathbf{x}_k^m$ is expanded, then

$$D_k(\mathbf{x}_k^m) \leq D_1(\tilde{\mathbf{x}}_1^m).^5 \tag{10}$$

Conversely, a node $\mathbf{x}_k^m$ is expanded if

$$D_k(\mathbf{x}_k^m) < D_1(\tilde{\mathbf{x}}_1^m). \tag{11}$$

In the case of finite L, only the necessary condition in (10) holds true.

Proof: We first show that the necessary condition is true. If $D_k(\mathbf{x}_k^m) > D_1(\tilde{\mathbf{x}}_1^m)$, then node $\tilde{\mathbf{x}}_1^m$ is expanded before node $\mathbf{x}_k^m$ is expanded. Since node $\tilde{\mathbf{x}}_1^m$ is a leaf node, the algorithm terminates when node $\tilde{\mathbf{x}}_1^m$ is expanded. Thus, node $\mathbf{x}_k^m$ is not expanded. For the sufficient condition, since $D_k(\mathbf{x}_k^m) < D_1(\tilde{\mathbf{x}}_1^m)$, node $\mathbf{x}_k^m$ will be expanded before node $\tilde{\mathbf{x}}_1^m$ is expanded, a time when the algorithm terminates. In the case of finite L, the necessary condition holds true by the same argument. To see why the sufficient condition does not hold, consider a special case with $L = |\mathbb{Q}| - 1$. Suppose all the $|\mathbb{Q}|$ nodes in layer 1 have a path metric satisfying (11). However, before the next iteration, one of these $|\mathbb{Q}|$ nodes will be discarded due to the memory constraint. Thus, the sufficient condition in (11) does not hold for finite L. ■

Proposition 2: (DSPS(L) algorithm) Given fixed $\mathbf{H}$, $\tilde{\mathbf{x}}$, and $\mathbf{v}$, in the case of both infinite and finite L, if a node $\mathbf{x}_k^m$ is expanded, then

$$G_N(\mathbf{x}_k^m) \leq \max\Big(G_N(\tilde{\mathbf{x}}_m^m), \ldots, G_N(\tilde{\mathbf{x}}_1^m)\Big). \tag{12}$$

The converse is not true.

Proof: The same proof for Proposition 2 in Appendix B in [2] can be applied here to show the validity of the necessary condition after letting $h(\mathbf{x}_k^m) = G_N(\mathbf{x}_k^m)$. To see the converse is not true, note that the normalized path metric, unlike the path metric, is not monotonically nondecreasing towards the bottom of the tree. Specifically, consider a node $\mathbf{x}_k^m$ whose $G_N(\mathbf{x}_k^m)$ satisfies (12) and its parent node $\mathbf{x}_{k+1}^m$ whose $G_N(\mathbf{x}_{k+1}^m)$ does not satisfy (12) (an example can be easily drawn numerically). Then, due to the necessary condition, node $\mathbf{x}_{k+1}^m$ will not be expanded and, as a result, node $\mathbf{x}_k^m$ is not expanded even if its normalized path metric satisfies (12). ■

To give a sense of the complexity differences between DS and DSPS, we consider the case $L = \infty$. The complexity results will form an upper-bound for the case of finite L, which is much more difficult to quantify due to the dynamic of node elimination. Given fixed $\mathbf{H}$ and $\tilde{\mathbf{x}}$, it follows from Proposition 1 that for DS(∞),

$$J_k \quad = \quad \sum_{\mathbf{x}_k^m} \mathbf{1}_{D_k(\mathbf{x}_k^m) \leq D_1(\tilde{\mathbf{x}}_1^m)} \tag{13}$$

where $\mathbf{1}$ is the indicator function, equal to one if its argument is true and zero otherwise. Therefore,

$$\mathrm{E}_{\mathbf{x}}[J_k] = 1 + \sum_{\mathbf{x}_k^m \neq \tilde{\mathbf{x}}_k^m} Pr\Big(D_k(\mathbf{x}_k^m) \leq D_1(\tilde{\mathbf{x}}_1^m)\Big) \tag{14}$$

[5]Here we have assumed that the actually transmitted symbol vector $\tilde{\mathbf{x}}_1^m$ is the ML solution yielded by the DS(∞) algorithm. In a rigorous sense, this may not always be true; however, from a detection point of view, this is a reasonable assumption.

where the expectation $\mathrm{E}_{\mathbf{x}}$ is taken over all search paths. Likewise, it follows from Proposition 2 that for DSPS(∞),

$$J_k \leq \sum_{\mathbf{x}_k^m} \mathbf{1}_{G_N(\mathbf{x}_k^m) \leq \max\left(G_N(\tilde{\mathbf{x}}_m^m),\ldots,G_N(\tilde{\mathbf{x}}_1^m)\right)}. \quad (15)$$

Therefore,

$$\begin{aligned}\mathrm{E}_{\mathbf{x}}[J_k] &\leq 1 + \sum_{\mathbf{x}_k^m \neq \tilde{\mathbf{x}}_k^m} Pr\Big(G_N(\mathbf{x}_k^m) \\ &\qquad \leq \max\left(G_N(\tilde{\mathbf{x}}_m^m),\ldots,G_N(\tilde{\mathbf{x}}_1^m)\right)\Big) \\ &\leq 1 + \sum_{\mathbf{x}_k^m \neq \tilde{\mathbf{x}}_k^m} Pr\Big(G_N(\mathbf{x}_k^m) \leq Y\Big) \\ &= 1 + \sum_{\mathbf{x}_k^m \neq \tilde{\mathbf{x}}_k^m} Pr\Big(D_k(\mathbf{x}_k^m) \leq F^{-1}(Y; m-k+1)\Big)\end{aligned} \quad (16)$$

where Y is the largest order statistic of m i.i.d. standard uniform random variables, which is beta distributed with parameters m and 1, denoted by $Y \sim \mathrm{Beta}(m, 1)$. The second inequality in (16) follows from the fact that $G_N(\tilde{\mathbf{x}}_k^m), k = 1,\ldots,m$, are *dependent* standard uniform random variables with positive correlation (recall the definition of normalized path metric in (9) and the fact that $D_1(\tilde{\mathbf{x}}_1^m),\ldots,D_m(\tilde{\mathbf{x}}_m^m)$ are dependent from (6)), and the maximum sample drawn from i.i.d. random variables is statistically greater than that drawn from dependent ones following the same distribution.

Comparing (14) and (16), it is seen that the difference between $D_1(\tilde{\mathbf{x}}_1^m)$ and $F^{-1}(Y; m-k+1)$, or equivalently, $F(D_1(\tilde{\mathbf{x}}_1^m); m-k+1)$ and Y, reveals the difference in $\mathrm{E}_{\mathbf{x}}[J_k]$, the expected number of expanded nodes in layer $m-k+1$. Fig. 3 plots the cdf of $X_k \triangleq F\big(D_1(\tilde{\mathbf{x}}_1^m); m-k+1\big), k = 1,2,\ldots,m$, and Y, for $m = 8$ (4×4 system). It is seen that X_k's corresponding to the complexity in lower layers (e.g., X_7, X_8) are predominantly larger than Y, and therefore, after taking $F^{-1}(\cdot; m-k+1)$, $D_1(\tilde{\mathbf{x}}_1^m)$ can be significantly larger than $F^{-1}(Y; m-k+1)$. This suggests that the expected number of expanded nodes in lower layers is larger for DS than for DSPS in this 4×4 system. Although reverse relations between X_k's corresponding to the complexity in higher layers (e.g., X_1, X_2) and Y are observed, nodes in higher layers will not be expanded if their parent nodes in lower layers are not expanded, as discussed previously. In other words, (16) becomes an increasingly loose upper bound in higher layers.

D. Numerical Results and Discussions

Here we compare the proposed DSPS algorithm with the DS algorithm [17]–[19], the DS algorithm adopting the l_∞-norm approximation for the path metric[6] [23], the K-best decoding algorithm [10], and the minimum-mean-square-error (MMSE) linear detector by computer simulation. The symbol error rate (SER) performance and the complexity results are

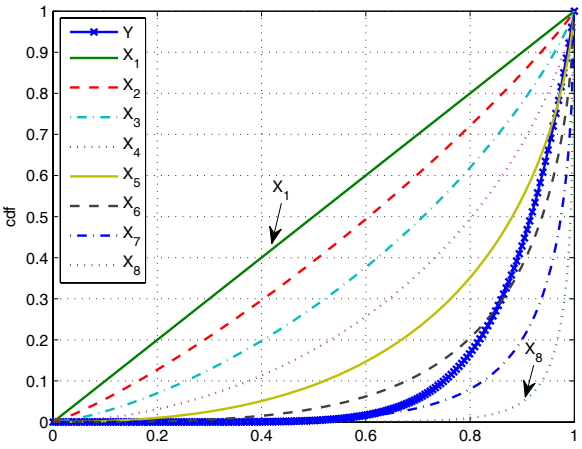


Fig. 3. Cumulative distribution function of $X_k = F\big(D_1(\tilde{\mathbf{x}}_1^m); m-k+1\big), k = 1,2,\ldots,m$, and Y, for $m = 8$.

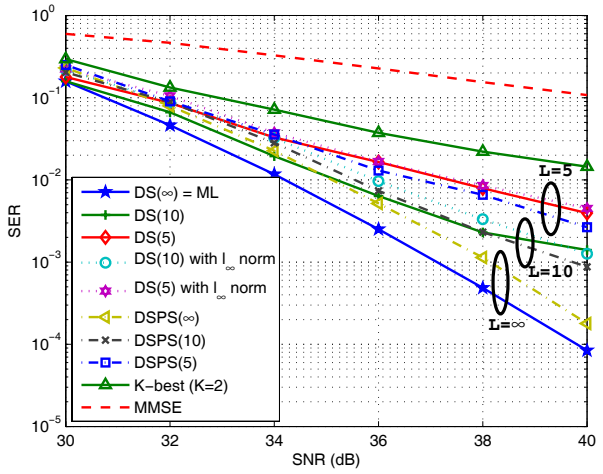
demonstrated for different signal-to-noise ratios (SNRs)[7] for a 4×4 system with 64-QAM in Fig. 4. It is seen in Fig. 4(a) that when there is no memory constraint ($L = \infty$), DSPS does not achieve the ML performance (1 dB short). This is a direct result of the normalized path metric not being monotonically nondecreasing towards the bottom of the tree, as discussed previously. When there is a memory constraint, e.g., $L = 5$ and $L = 10$, DS, DS with l_∞-norm, and DSPS all demonstrate comparable performance, with DSPS slightly better at high SNRs. The most noticeable advantage of DSPS lies in its visited node reduction capability, as shown in Fig. 4(b). It is seen that, for $L = 10$, DS with l_∞-norm and DSPS reduce the number of visited nodes by 21% and 39% at SNR = 30 dB, respectively, as compared with DS. As SNR increases, all schemes converge to the lower bound $m|\mathbb{Q}| = 64$. The above performance and complexity results suggest that DSPS maintains the node list efficiently in the presence of a memory constraint. Furthermore, it is observed that the complexity increases more moderately for DSPS than for other schemes as L increases, suggesting that a larger L can be adopted in DSPS to yield improved performance at little additional complexity cost. The K-best scheme, with K chosen to show comparable complexity, achieves worse performance than all other schemes except MMSE, at a higher yet fixed complexity.

Fig. 5 shows the average latency and the amount of variability in latency for different schemes, with $L = 10$. As can be clearly seen, the K-best detector with any K has a fixed iteration count of m (8 in this case), while all other schemes show different degrees of variability.[8] Among schemes with variable latency, DSPS has the smallest average latency as well as the least amount of variability, both at a given operating SNR and across different SNRs. As a result, DSPS needs to maintain the smallest number of I/O buffers which could limit the detection throughput and pose an extra overhead in

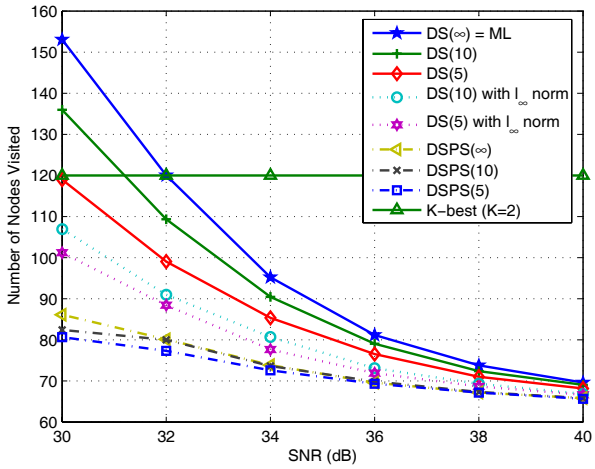
[6]The path metric defined on l_∞-norm is given by the recursive relation $D_k(\mathbf{x}_k^m) = \max\left(D_{k+1}(\mathbf{x}_{k+1}^m), \left|y_k' - \sum_{i=k}^m r_{k,i}x_i\right|\right)$. Alternatively, l_1-norm approximation with $D_k(\mathbf{x}_k^m) = D_{k+1}(\mathbf{x}_{k+1}^m) + \left|y_k' - \sum_{i=k}^m r_{k,i}x_i\right|$ may be used, which reports better performance and higher complexity in [23] and in our simulation. For simplicity, in this paper we show results for l_∞-norm only.

[7]The SNR is defined as $\mathrm{E}[\|\mathbf{H}\tilde{\mathbf{x}}\|^2]/\mathrm{E}[\|\mathbf{v}\|^2]$.

[8]While the K-best detector has a fixed iteration count independent of K, the time required for each iteration to complete is proportional to K [11]. Adopting iteration counts in Fig. 5, however, does not compromise its implications on throughput variability and hardware implementation requirements such as the number of input/output (I/O) buffers needed.



(a)



(b)

Fig. 4. Performance and complexity of MIMO detection schemes for a 4×4 system with 64-QAM. (a) SER. (b) Complexity.

a practical system [11].

While we have seen some desirable properties of DSPS in terms of performance, complexity, and latency, two observations made from Figs. 4–5 motivate a further enhancement and generalization of the DSPS scheme: 1) Even with unbounded memory space, DSPS does not always find the ML solution; 2) DSPS has a varying complexity and latency, with no throughput guarantee. To retain the advantage of probabilistic sorting as employed by DSPS while removing these limitations, we have developed a general framework for Dijkstra's algorithm-based detection incorporating probabilistic node ordering. This is presented in the next section.

IV. Generalized Dijkstra's Search (GDS) Framework

The GDS framework generalizes Dijkstra's algorithm with probabilistic node ordering by introducing three configurable input parameters. By tuning these parameters, memory usage, expansion complexity, and sorting dynamic associated with the detection algorithm can flexibly be customized, and, as a result, the aforementioned limitations can be removed. Furthermore, the GDS framework encompasses the DS algorithm, the DSPS algorithm, and other existing schemes as special cases.

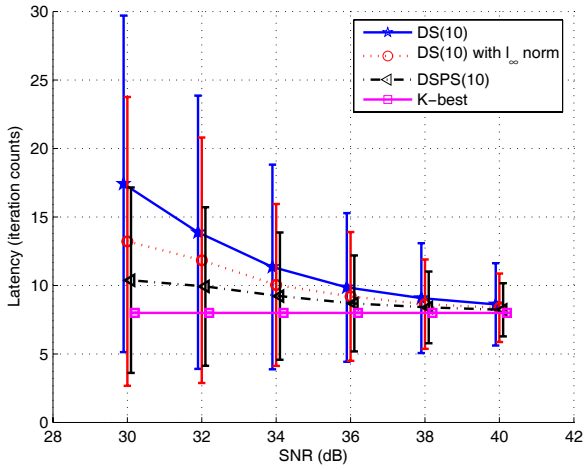


Fig. 5. Latency of MIMO detection schemes for a 4×4 system with 64-QAM. Both mean (markers) and one standard deviation (error bars) are shown.

A. Algorithm Description

We first define the *generalized path metric* for a node $\mathbf{x}_{m-k+1}^m$ in layer k, $k = 1, 2, \ldots, m$, as

$$G^{(T)}(\mathbf{x}_{m-k+1}^m) \triangleq F^{-1}\Big(F\big(D_{m-k+1}(\mathbf{x}_{m-k+1}^m); k\big); k+T\Big) \tag{17}$$

where $T \in \{0, \mathbb{Z}^+\}$ is a configurable parameter. As will be seen, different settings of T produce different sorting dynamic, and $G^{(T)}$ is closely related to G_N and D. The proposed GDS framework, presented as a general algorithm with three parameters $L \in \mathbb{Z}^+$, $K \in \mathbb{Z}^+$, and $T \in \{0, \mathbb{Z}^+\}$ for configuring memory usage, expansion complexity, and sorting dynamic, respectively, is described as follows.

Algorithm: GDS(L, K, T)

Initially, the node list $\mathbb{N} = \{N_{(1)}, N_{(2)}, \ldots\}$ contains only the root node, as $N_{(1)}$.

1) Select the leading nodes $N_{(1)}, N_{(2)}, \ldots, N_{(\min(|\mathbb{N}|,K))}$ from $\mathbb{N}$. If $N_{(1)}$ is in layer m (i.e., a leaf node), stop the algorithm and output it as the solution.
2) Expand each of the leading nodes $N_{(i)}, i = 1, 2, \ldots, \min(|\mathbb{N}|, K)$, by adding its child nodes to $\mathbb{N}$ and removing itself from $\mathbb{N}$. If any of the $\min(|\mathbb{N}|, K)$ leading nodes is in layer m, leave it in $\mathbb{N}$ and do not expand it.
3) Order the nodes in $\mathbb{N}$ in ascending order of their cost $c = G^{(T)}$. Retain the first $\min(|\mathbb{N}|, L)$ nodes and discard others. Go to step 1.

B. Properties of the GDS Algorithm

The GDS algorithm offers rich configurability through the parameter triplet (L, K, T). Its properties are discussed as follows.

1) Memory usage: GDS retains use of L to limit the memory usage in Dijkstra's algorithm-based detection, as in DS and DSPS algorithms. Given some L and K, the GDS algorithm requires a fixed memory size of $L + K(|\mathbb{Q}| - 1)$, each unit for storing the (partial) symbol vector and the defined cost for a node. Compared to the SD algorithm, which requires a fixed memory size of $(m-1)|\mathbb{Q}|+1$ [20], the GDS algorithm is comparably memory-efficient since $K = 1$ and a moderate-valued L are sufficient to yield near-optimal performance, as presented in Sec. IV-C.

2) Fixing the complexity: GDS determines the number of expanded nodes at each iteration through K. Both DS and DSPS algorithms employ $K = 1$. A larger K increases the expansion complexity while may or may not improve the performance in the case of limited memory (see Sec. IV-C). However, this parameter enables fixing the complexity of GDS. The fixed-complexity version of GDS is produced by setting $L = K$ with an arbitrary T, where the GDS algorithm expands all nodes in the node list at any iteration and effectively becomes a K-best detector that searches the tree uniformly towards the bottom of the tree. The detection latency is a fixed number of m iterations, and efficient implementation in a pipelined fashion is feasible [11].

3) Sorting dynamic: GDS controls the sorting dynamic through T, the parameter of the generalized path metric in (17). The effect of T on the defined cost of nodes and consequently on sorting is illustrated in Fig. 6. Consider two example nodes, $\mathbf{x}_m^m$ in layer 1 and $\mathbf{x}_{m-1}^m$ in layer 2, with generalized path metric denoted by $a_T = G^{(T)}(\mathbf{x}_m^m)$ and $b_T = G^{(T)}(\mathbf{x}_{m-1}^m)$, respectively. As can be seen in Fig. 6, the relative magnitude of these two nodes' generalized path metric depends on T: for small T (e.g., $T = 0, 1, 2$), $a_T < b_T$, with diminishing differences as T increases; for some intermediate T (e.g., $T = 4$), they are nearly identical; and for larger T (e.g., $T = 7, 11, 17$), $a_T > b_T$. It would be interesting to see the effect of T on the sorting dynamic as well as the relation between $G^{(T)}$ and $\{G_N, D\}$. Therefore, we have the following proposition.

Proposition 3: Sorting in step 3 of the GDS algorithm according to cost $c = G^{(T)}$ is equivalent to sorting according to $c = G_N$ and $c = D$ when $T \to \infty$ and $T = 0$, respectively.

Proof: See the Appendix. ∎

As can be inferred from Fig. 6, an intermediate value of T is expected to yield a sorting dynamic between the two extreme cases of $T = 0$ (achieving the sorting employed in DS) and $T = \infty$ (achieving the sorting employed in DSPS). We have seen in Sec. III that a modified sorting dynamic affects the performance and complexity. Therefore, it is reasonable to expect that choosing an intermediate-valued T will give a performance-complexity tradeoff between what is attained with $T = 0$ and $T = \infty$ when other parameters are fixed, as will be verified by simulation in the next subsection.

C. Numerical Results and Discussions

Here we study the performance of the proposed GDS algorithm by computer simulation. First, we plot the SER and complexity results for different configurations of the GDS algorithm for a 4×4 system with 16-QAM in Fig. 7. It is

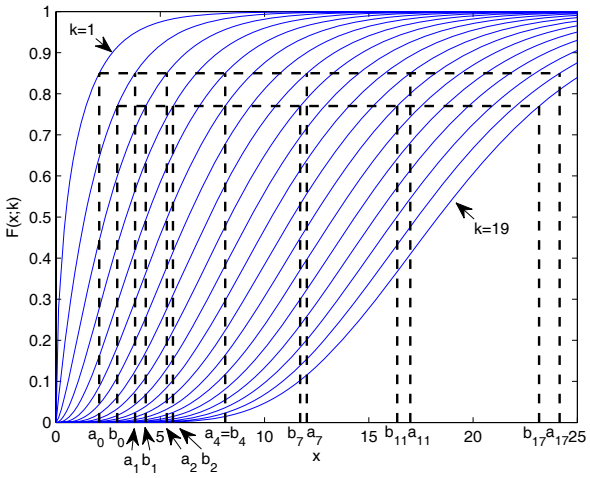
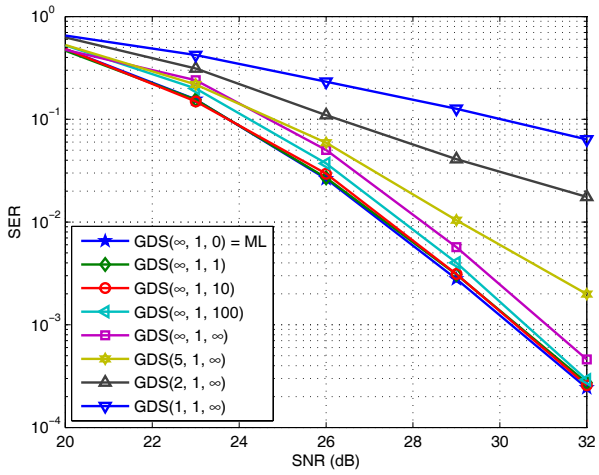


Fig. 6. Plot of $F(x;k)$ in (8) for $k = 1, 2, \ldots, 19$ ($\sigma_v^2 = 2$), with the generalized path metric a_T and b_T of two example nodes shown for $T = 0, 1, 2, 4, 7, 11,$ and 17.

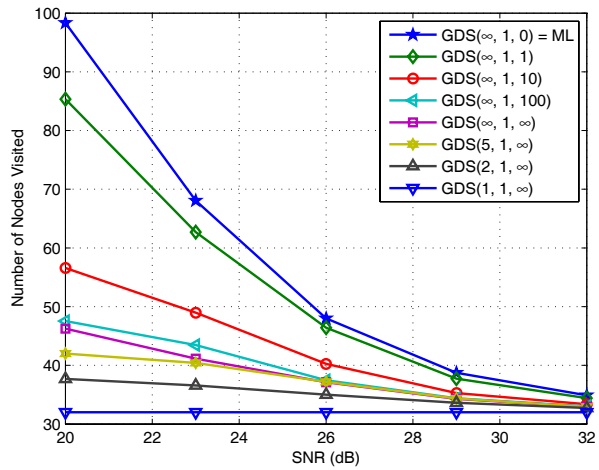
seen that different performance and complexity tradeoffs are attained by customizing parameters. Specifically, beginning with the ML-achieving GDS(∞, 1, 0), which corresponds to the DS(∞) scheme, first increasing T from 0 to ∞ and then decreasing L from ∞ to 1 with a fixed $K = 1$ gradually trades the performance for lower complexity in the resulting scheme.

To demonstrate the performance-complexity tradeoffs more explicitly and examine the effect of different parameter combinations, we plot the SER against the complexity at fixed SNR for two scenarios: 4×4 system with 16-QAM (Fig. 8(a)) and 8×8 system with 4-QAM (Fig. 8(b)). The modified Dijkstra's algorithm [17]–[19] and the probabilistic tree pruning SD (PTP-SD) scheme with pruning probability P [4] (employing the Schnorr-Euchner (SE) enumeration [8], [24] and an infinite initial sphere radius) are considered in our comparison. It is seen in both figures that, while the modified Dijkstra's algorithm (the GDS(L, 1, 0) line) can produce different points, it is unable to produce points in the lower-left region of the plot, which are however attainable by the GDS algorithm through a joint configuration of memory usage (L) and sorting (T), as seen in the GDS(L, 1, ∞) and GDS(∞, 1, T) lines. In fact, interestingly, while both the GDS algorithm and the modified Dijkstra's algorithm attain the same end-points (zero-forcing decision-feedback-equalization (ZF-DFE) point at one end and the optimal point at the other), the GDS algorithm produces different intermediate points. Specifically, as the modified Dijkstra's algorithm goes from GDS(1, 1, 0) to GDS(∞, 1, 0), the GDS algorithm goes from GDS(1, 1, ∞) to GDS(∞, 1, ∞) to GDS(∞, 1, 0) in the lower-left region. Note that the line section between GDS(∞, 1, ∞) and GDS(∞, 1, 0) is not achievable by DSPS but by GDS. By jointly designing L and T in the GDS algorithm, various desired tradeoff points can be produced.

In the heuristic performance regime, the GDS algorithm can potentially achieve lower complexity and better performance than the modified Dijkstra's algorithm for the same L, e.g., GDS(5, 1, ∞) vs. GDS(5, 1, 0). Similar results are observed

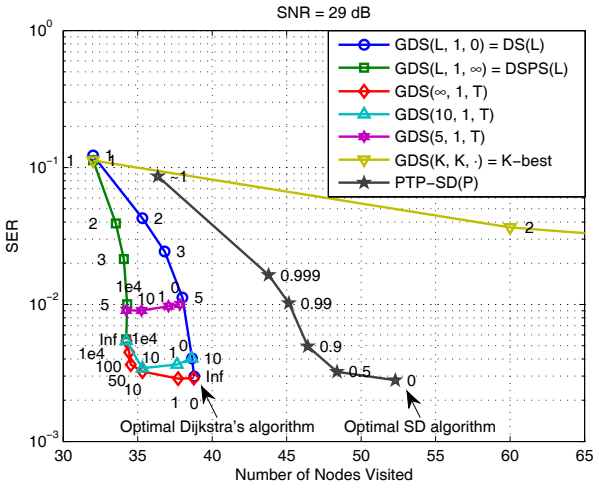


(a)

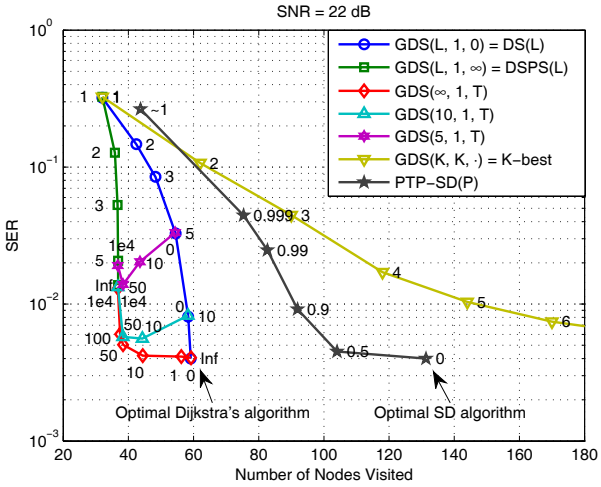


(b)

Fig. 7. Performance and complexity of the GDS algorithm with different configurations for a 4×4 system with 16-QAM. (a) SER. (b) Complexity.



(a)



(b)

Fig. 8. Performance-complexity tradeoffs for MIMO detection schemes. The numbers labeled next to a line represent the values of the configurable parameter (L, K, T, or P) for each line. (a) 4×4 system with 16-QAM. (b) 8×8 system with 4-QAM.

in the near-optimal performance regime, e.g., along the line GDS(10, 1, T). Furthermore, in the heuristic regime, GDS(L, 1, ∞) exhibits a steeper slope than the modified Dijkstra's algorithm (and PTP-SD, too), indicating that the required number of nodes visited increases more moderately as the performance increases, a property we have also commented in Sec. III-D.

By setting $L = K$ with an arbitrary T (say, zero), the GDS algorithm becomes the K-best scheme, as indicated by the GDS(K, K, $\cdot$) line. It is seen that such a configuration generally searches more nodes than needed to attain a certain SER performance, a price paid for the benefit of fixed complexity. The effect of K on the performance is revealed by comparing GDS(K, 1, 0) and GDS(K, K, 0). As seen in Fig. 8(b), for small values of L, increasing K improves the performance at some cost of complexity (comparing GDS(3, 1, 0) and GDS(3, 3, 0), for example). As L increases, the performance gain from a greater K diminishes, and even parishes in the case of $L = \infty$, as both GDS(∞, 1, 0) and GDS(∞, ∞, 0) achieve the optimal performance.

Note that some computational costs (such as evaluating F and F^{-1}) associated with the GDS and PTP-SD algorithms are not accounted for in the comparison in Figs. 8(a)–8(b). However, we believe the number of searched nodes is a meaningful and important abstraction of the actual complexity involved to distinguish between different schemes which would otherwise be difficult to compare side-by-side. Besides, the results present a theoretical interest, showing that a more favorable tradeoff region can be achieved. To reduce the computational cost of evaluating the cdf function and its inverse, closed-form approximations [25] and implementation of table lookups [26] are suggested for an online operation of the proposed algorithm.

While the discussions on the proposed modified and generalized Dijkstra's algorithms have focused on application to hard-output detection, these algorithms, without modification, enable soft-output detection since the computation of log likelihood ratios (LLRs) for each bit entails the computation of the likelihood metric of the ML solution (employing the max-log approximation [27]). Furthermore, with simple modifica-

tions, the proposed algorithms can also accomplish soft-output detection. For example, to obtain enough candidate symbols that constitute each counter-hypothesis for soft detection, the stopping criterion of the DSPS or GDS algorithm can be modified such that a counter is implemented to ensure enough leaf nodes (i.e., full-length paths) are collected before the algorithm terminates. With these full-length paths, and also the information contained in partial-length paths, the likelihood metric of the ML solution and counter-hypotheses can be calculated exactly or approximately [28]. Note that this soft-output augmentation requires extra iterations for the algorithm, and thus the demonstrated latency advantage of the proposed detection scheme can also benefit Dijkstra's algorithm-based soft detection. A more detailed account of the complexity and various issues associated with Dijkstra's algorithm-based soft detection with probabilistic node ordering is relegated to future work.

V. Conclusion

In the first part of the paper, a new tree-search detection scheme for MIMO systems was proposed. The proposed scheme modifies Dijkstra's algorithm developed from a pure graph perspective by acknowledging the statistical properties associated with the problem at hand. Performance and complexity characteristics of the proposed scheme were discussed analytically and numerically. In the second part of the paper, Dijkstra's algorithm was generalized such that its memory usage, expansion complexity, and sorting behavior are made configurable. New properties of the generalized framework were presented, including its unification of several schemes. Computer simulation has shown that the proposed framework presents noticeable advantages in terms of performance-complexity tradeoff as compared to a prior Dijkstra's algorithm-based scheme and the SD algorithm.

Appendix
Proof of Proposition 3

First, it is straightforward to see that sorting by $G^{(T)}$ reduces to sorting by D when $T = 0$, as $G^{(T)}$ returns the path metric of a node in this case. Secondly, to see that sorting by $G^{(T)}$ reduces to sorting by G_N as $T \to \infty$, it suffices to show that the function F^{-1} in the definition of $G^{(T)}$ does not alter the sorting dynamic when $T \to \infty$. Mathematically, proof of Proposition 3 is equivalent to showing that for any finite $k_1, k_2 \in \mathbb{Z}^+$ and for any $\Delta\alpha > 0$, $F^{-1}(\alpha + \Delta\alpha; k_1 + T) > F^{-1}(\alpha; k_2 + T)$ as $T \to \infty$. Note that for $k_1 = k_2$, this is straightforwardly true; and for $k_1 > k_2$, this is also true since $F^{-1}(\alpha + \Delta\alpha; k_1 + T) > F^{-1}(\alpha + \Delta\alpha; k_2 + T) > F^{-1}(\alpha; k_2 + T)$ by the properties of F discussed in Sec. III-B. Therefore, it remains only to show that this holds true for $k_1 < k_2$, which will be proved through the following lemmas.

Lemma 1: Define $f(x) = \partial F(x; k + T)/\partial x$ for $x > 0$ and $k \in \mathbb{Z}^+$. Then, $f(x) \to 0$ as $T \to \infty$.

Proof: Using the definition of F in (8) we have

$$f(x) = \frac{1}{\Gamma(\frac{k+T}{2})\sigma_v^2}\Big(\frac{x}{\sigma_v^2}\Big)^{\frac{k+T}{2}-1} e^{-\left(\frac{x}{\sigma_v^2}\right)}. \tag{18}$$

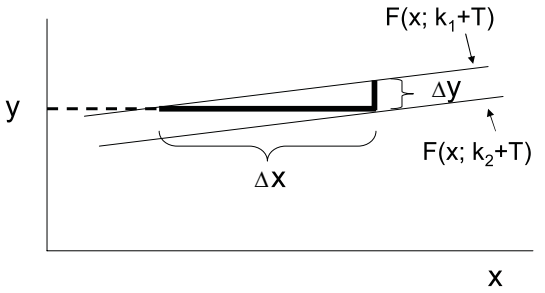


Fig. 9. Illustration of a section of $F(x; k_1 + T)$ and $F(x; k_2 + T)$ with $k_1 < k_2$, for sufficiently large T.

Since $(k + T)/2$ is either a positive integer or positive half-integer, we can derive $f(x)$ in each case by substituting the known closed-form expressions for Γ into (18), i.e.,

$$f(x) = c_1 \cdot \frac{\Big(\frac{x}{\sigma_v^2}\Big)^{\frac{k+T}{2}-1}}{\Big(\frac{k+T}{2}-1\Big)!}, \quad \text{if } \tfrac{k+T}{2} \text{ is an integer} \tag{19}$$

$$f(x) = c_2 \cdot \frac{\Big(\frac{4x}{\sigma_v^2}\Big)^{\frac{k+T-1}{2}}}{(k+T-1)!/\Big(\frac{k+T-1}{2}\Big)!}, \quad \text{if } \tfrac{k+T}{2} \text{ is a half-integer} \tag{20}$$

where $c_1 = e^{-(x/\sigma_v^2)}/\sigma_v^2$ and $c_2 = e^{-(x/\sigma_v^2)}/\sqrt{\pi x \sigma_v^2}$. Since x/σ_v^2 is of finite values, from (19) and (20) we have $f(x) \to 0$ as $T \to \infty$. ∎

Lemma 1 states that the slope of $F(x; k + T)$ approaches zero everywhere when $T \to \infty$.

Lemma 2: For $0 < y < 1$ and $k_1 < k_2$, $F\big(F^{-1}(y; k_2 + T); k_1 + T\big) - y \to 0$ as $T \to \infty$.

Proof: Let Z_1 and Z_2 be random variables whose cdf is given by $F(x; k_1 + T)$ and $F(x; k_2 + T)$, respectively, where $k_1 < k_2$. By the central limit theorem, as $T \to \infty$, the distribution of Z_1 and Z_2 converges to a Gaussian distribution. In addition, by Lemma 1, $F(x; k_1 + T)$ and $F(x; k_2 + T)$ can be infinitesimally approximated by the straight-line section of fixed length for sufficiently large T, as depicted in Fig. 9. For any given $0 < y < 1$, $F(x; k_1 + T)$ and $F(x; k_2 + T)$ are offset horizontally by $\Delta x \triangleq F^{-1}(y; k_2 + T) - F^{-1}(y; k_1 + T)$ and vertically by $\Delta y \triangleq F\big(F^{-1}(y; k_2 + T); k_1 + T\big) - y$. Note that Δx is equal to the difference of means of Z_1 and Z_2 (i.e., $k_2 - k_1$) for $y = 0.5$, and of finite values elsewhere. Since $f(x) = \Delta y/\Delta x$ for $x \in \big[F^{-1}(y; k_1 + T), F^{-1}(y; k_2 + T)\big]$, and $f(x) \to 0$ as $T \to \infty$ by Lemma 1, we have $\Delta y \to 0$ as $T \to \infty$. ∎

Let $\Delta\alpha = \Big[F\big(F^{-1}(\alpha; k_2 + T); k_1 + T\big) - \alpha\Big] + \epsilon$, where $\epsilon > 0$. Then, we have $F^{-1}(\alpha + \Delta\alpha; k_1 + T) > F^{-1}(\alpha; k_2 + T)$ for this particular $\Delta\alpha$. By Lemma 1 and Lemma 2, $F^{-1}(\alpha + \Delta\alpha; k_1 + T) > F^{-1}(\alpha; k_2 + T)$ holds for any $\Delta\alpha > 0$ and $k_1 < k_2$ as $T \to \infty$, which completes the proof.

References

[1] E. G. Larsson, "MIMO detection methods: how they work," *IEEE Signal Process. Mag.*, vol. 26, pp. 91–95, May 2009.

[2] A. D. Murugan, H. El Gamal, M. O. Damen, and G. Caire, "A unified framework for tree search decoding: rediscovering the sequential decoder," *IEEE Trans. Inf. Theory*, vol. 52, pp. 933–953, Mar. 2006.

[3] R. Gowaikar and B. Hassibi, "Statistical pruning for near-maximum likelihood decoding," *IEEE Trans. Signal Process.*, vol. 55, pp. 2661–2675, June 2007.

[4] B. Shim and I. Kang, "Sphere decoding with a probabilistic tree pruning," *IEEE Trans. Signal Process.*, vol. 56, pp. 4867–4878, Oct. 2008.

[5] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I: expected complexity," *IEEE Trans. Signal Process.*, vol. 53, pp. 2806–2818, Aug. 2005.

[6] L. Azzam and E. Ayanoglu, "Reduced complexity sphere decoding via a reordered lattice representation," *IEEE Trans. Commun.*, vol. 57, pp. 2564–2569, Sep. 2009.

[7] R. Y. Chang and W.-H. Chung, "Reduced-complexity sphere decoding with dimension-dependent sphere radius design," in *Proc. 2011 IEEE WCNC*.

[8] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Trans. Inf. Theory*, vol. 48, pp. 2201–2214, Aug. 2002.

[9] W. Zhao and G. B. Giannakis, "Reduced complexity closest point decoding algorithms for random lattices," *IEEE Trans. Wireless Commun.*, vol. 5, pp. 101–111, Jan. 2006.

[10] K.-W. Wong, C.-Y. Tsui, R. S.-K. Cheng, and W.-H. Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *Proc. 2002 IEEE Int. Symp. Circuits Syst.*, pp. III-273–III-276.

[11] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection," *IEEE J. Sel. Areas Commun.*, vol. 24, pp. 491–503, Mar. 2006.

[12] Q. Li and Z. Wang, "Improved K-Best sphere decoding algorithms for MIMO systems," in *Proc. 2006 IEEE ISCAS*, pp. 1159–1162.

[13] J. B. Anderson and S. Mohan, "Sequential coding algorithms: a survey and cost analysis," *IEEE Trans. Commun.*, vol. 32, pp. 169–176, Feb. 1984.

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

[15] 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, pp. 2571–2576, Oct. 2004.

[16] L. G. Barbero and J. S. Thompson, "Fixing the complexity of the sphere decoder for MIMO detection," *IEEE Trans. Wireless Commun.*, vol. 7, pp. 2131–2142, June 2008.

[17] T.-H. Kim and I.-C. Park, "High-throughput and area-efficient MIMO symbol detection based on modified Dijkstra's search," *IEEE Trans. Circuits Syst. I*, vol. 57, pp. 1756–1766, July 2010.

[18] A. Okawado, R. Matsumoto, and T. Uyematsu, "Near ML detection using Dijkstra's algorithm with bounded list size over MIMO channels," in *Proc. 2008 IEEE International Symp. on Inform. Theory*, pp. 2022–2025.

[19] M. Rachid and B. Daneshrad, "Iterative MIMO sphere decoding throughput guarantees under realistic channel conditions," *IEEE Commun. Lett.*, vol. 14, pp. 342–344, Apr. 2010.

[20] Y. Dai and Z. Yan, "Memory-constrained tree search detection and new ordering schemes," *IEEE J. Sel. Topics Signal Process.*, vol. 3, pp. 1026–1037, Dec. 2009.

[21] C. Studer, A. Burg, and W. Fichtner, "A unification of ML-optimal tree-search decoders," in *Proc. 2006 Asilomar Conference on Signals, Systems and Computers*, pp. 2185–2189.

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd edition. Cambridge University Press, 2007.

[23] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bölcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE J. Solid-State Circuits*, vol. 40, pp. 1566–1577, July 2005.

[24] C. P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems," *Mathematical Programming*, vol. 66, pp. 181–199, 1994.

[25] M. Sankaran, "Approximations to the noncentral chi-square distribution," *Biometrika*, vol. 50, pp. 199–204, 1963.

[26] H. L. Harter and D. B. Owen, *Selected Tables in Mathematical Statistics, Volume 1*. Markham Publishing Company, 1970.

[27] C. Studer, A. Burg, and H. Bölcskei, "Soft-output sphere decoding: algorithms and VLSI implementation," *IEEE J. Sel. Areas Commun.*, vol. 26, pp. 290–300, Feb. 2008.

[28] J. Hagenauer and C. Kuhn, "The list-sequential (LISS) algorithm and its application," *IEEE Trans. Commun.*, vol. 55, pp. 918–928, May 2007.

Ronald Y. Chang received the B.S. degree in electrical engineering from the National Tsing Hua University, Hsinchu, Taiwan, in 2000, the M.S. degree in electronics engineering from the National Chiao Tung University, Hsinchu, in 2002, and the Ph.D. degree in electrical engineering from the University of Southern California (USC), Los Angeles, in 2008.

From 2002 to 2003, he was with the Industrial Technology Research Institute, Hsinchu. Since 2004, he has conducted research with USC as well as with the Mitsubishi Electric Research Laboratories, Cambridge, MA. Since 2010, he has been with the Research Center for Information Technology Innovation at Academia Sinica, Taipei, Taiwan. His research interests include resource allocation, interference management, detection and estimation, cognitive radio and cooperative communications for wireless communications and networking. He is a recipient of the Academia Sinica Postdoctoral Research Fellowship and the holder of five awarded or pending U.S. patents.

Wei-Ho Chung (M'11) was born in Kaohsiung, Taiwan, in 1978. He received the B.Sc. and M.Sc. degrees in Electrical Engineering from National Taiwan University, Taipei City, Taiwan, in 2000 and 2002, respectively, and the Ph.D. degree in Electrical Engineering from University of California, Los Angeles, USA, in 2009. From 2002 to 2005, he was a system engineer at ChungHwa TeleCommunications Company. In 2008, he was a research intern working on CDMA systems in Qualcomm Inc. Since January 2010, Dr. Chung has been a faculty member holding the position as an assistant research fellow in Research Center for Information Technology Innovation, Academia Sinica, Taiwan. His research interests include wireless communications, signal processing, statistical detection and estimation theory, and networks.