Mass Spectrum Labeling: Theory and Practice

Zheng Huang, Lei Chen

Jin-Yi Cai, Deborah Gross*, Raghu Ramakrishnan, James J. Schauer*, Stephen J. Wright

Computer Sciences Department, University of Wisconsin-Madison

ABSTRACT

In recent years, a number of instruments have been developed for continuous, real-time monitoring of the environment. Aerosol mass spectrometers can analyze several hundred atmospheric aerosol particles per minute and generate a plot of mass-to-charge versus intensity (a *mass spectrum*) for each particle. The mass spectrum could be used to identify the compounds present in the particle in real-time, in contrast to conventional filter-based approaches in which filters collect samples over a period of time and are then analyzed in a laboratory, but our ability to analyze the data is currently a bottle-neck. In this paper, we introduce the problem of *labeling a particle's mass spectrum* with the substances it contains, and develop several formal representations of the problem, taking into account practical complications such as unknowns and noise.

Our contributions include the introduction and formalization of a novel data mining problem, theoretical characterizations of the central difficulty underlying the problem, algorithms for solving the problem, metrics to measure the quality of labeling, experimental evaluation of the effectiveness of these algorithms, and comparisons with alternative machine learning techniques (showing that our algorithms, although slower, achieve uniformly superior accuracy *without the need for training datasets*!).

1. INTRODUCTION

Mass spectrometry techniques are widely used in many disciplines of science, engineering, and biology for the identification and quantification of elements, chemicals and biological materials. Historically, the specificity of mass spectrometry has been aided by upstream separation to remove mass spectral interference between different species. Examples include gas- (GCMS) and liquid-chromatography mass spectrometry (LCMS), and off-line wet chemistry clean-up techniques often employed upstream of inductively coupled plasma mass spectrometry

(ICPMS). Historically, these techniques have been employed in laboratory settings that did not require real-time data collection, and the time required for separation and clean-up has been acceptable.

In the past decade, a wide range of real-time mass spectrometry instruments have been employed, and the nature of these instruments often precludes separation and clean-up steps. The mass spectrum produced for a particle in real-time by one of these instruments is therefore comprised of overlaid mass spectra from several substances, and the overlap between these spectra makes it difficult to identify the underlying substances. The Aerosol Time-of-Flight Mass Spectrometer (ATOFMS) [11,9,13,15] is an example that is currently available commercially, and is used to monitor the size-resolved chemical composition of airborne particles. This instrument can obtain mass spectra for up to about 250 particles per minute, producing a time-series of unusual complexity. We have applied our results to ATOFMS data, although the data analysis challenges we describe are equally applicable to other real-time instruments that utilize mass spectrometry.

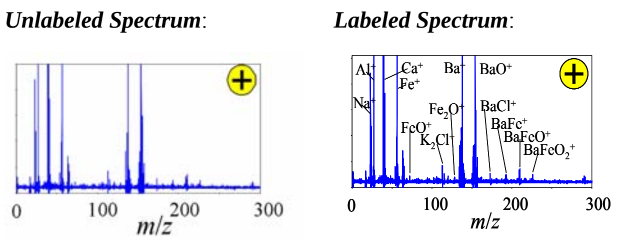


Figure 1: Mass Spectrum Labeling

Mass spectrum labeling consists of “translating” the raw plot of intensity versus mass-to-charge (m/z) value to a list of chemical substances or *ions* and their rough quantities (omitted in Figure 1) present in the particle. Labeling spectra allows us to think of a stream of mass spectra as a time-series of *observations*, one per collected particle, where each observation is a set of *labels* (or *ion-quantity* pairs). This is similar to a time-series of *transactions*, each recording the items purchased by a customer in a single visit to a store [1,4,13]. This analogy makes a wide range of association rule [3] and sequential pattern algorithms [2] applicable to the analysis of labeled mass spectrometry data.

The contributions of this paper include:

- In this and a companion paper [7], we introduce a new and important class of data mining problems involving analysis of mass spectra. The focus in this paper is on the labeling of individual spectra (Section 2), which is the foundation of a class of group-oriented labeling tasks discussed in [7]. Labeling also allows us to view spectra abstractly as “customer transactions” and borrow analysis techniques such as frequent itemsets and association rules. There is also a deeper connection to market-basket analysis in that our formulation of labeling allows us to search for buying patterns of interest (“phenomena” [10]) rather than just sets of items frequently purchased together [7].

- We present a theoretical characterization of ambiguity, which arises because of the presence of substances whose individual spectra overlap in the given input spectrum, and controls whether or not there is a unique way to label the input spectrum. (Section 3)
- We extend the labeling framework to account for practical complexities such as noise, errors, and the presence of unknown substances (Section 4). We then present two algorithms for labeling a spectrum, using this realistic problem formulation, together with several optimizations and results characterizing their behavior. (Section 5).
- A major difficulty for researchers is the effort involved in generating accurately labeled "ground truth" datasets. Such datasets are invaluable for training machine learning algorithms, tuning algorithms for a given application context using domain knowledge, and for comparing different algorithms. We present a detailed synthetic data generator that is based on real mass spectra, conforms to realistic problem scenarios, and allows us to produce labeled spectra while controlling several fundamental parameters such as ambiguity and noise. (Section 6).
- Finally, we discuss and rigorously define a metric for measuring the quality of labeling, and present a thorough evaluation of our labeling algorithm and a comparison with machine learning approaches, showing that our algorithms, although slower, achieve uniformly superior accuracy *without the need for training datasets*! (Section 7)

Related work and future directions are discussed in Section 8.

2. PROBLEM FORMULATION

A **mass spectrum** (or spectrum) is a vector $\vec{b} = [b_1, b_2 \cdots b_r]$, where $b_i \in R$ is the signal intensity at mass-to-charge (m/z) value *i*. For simplicity, we assume all spectra have the same 'range' and 'granularity' over the m/z axis; i.e., they have the same dimension r and the i^{th} element of a spectrum always corresponds to the same m/z value *i*. Intuitively, each m/z ratio corresponds to a particular isotope of some chemical element. The **signature** of an ion is a vector $\vec{s} = [I_1, I_2 \cdots I_r]$, $I_i \in R$ and $\sum_i I_i = 1$, representing a distribution of isotopes, i.e., I_i is the proportion of the isotope with m/z value *i*. A **signature library** is a set of known signatures $S = \{\vec{s}_1, \vec{s}_2 \cdots \vec{s}_n\}$, in which $\vec{s}_j$ is the signature of ion j. Additionally, there may be ions that appear on particles, and are therefore reflected in mass spectra, but that for which signatures are not included in the signature library.

The spectrum $\vec{b}$ of a particle is a linear combination of the signatures of ions that it contains, i.e., $\vec{b} = \sum_j w_j \vec{s}_j$, where w_j is the quantity of ion j in the particle. The task of **mass spectrum labeling** is to find the quantities w_j of all ions present in the particle, given an input spectrum. Formally, a **label** for an ion with respect to a given spectrum is an <*ion, quantity*> pair; a **label** for the spectrum is the collection of labels for all ions in the signature library. The task of labeling an input spectrum can be viewed as a search for a linear combination of ions that best approximates the spectrum, and the success that is achievable depends on the extent of unknown ions. When developing our algorithms, in Sections 3 to 5, for simplicity we assume that the signature library is complete, i.e., there are no unknown ions. We evaluate the impact of unknowns in Sections 6 and 7.

3. WHEN IS LABELING HARD?

In this section, we formulate the labeling task as solving a set of linear equations, and then discuss the fundamental challenge involved: interference between different combinations of signatures and the consequent ambiguity in labeling.

3.1 Linear System Abstraction

We can represent the signature library $S=\{\vec{s}_1,\vec{s}_2\cdots\vec{s}_n\}$ as a matrix $A=[\vec{s}_1,\vec{s}_2,...,\vec{s}_n]$, where $\vec{s}_k$, the k^{th} column of **A**, is the signature of ion *k*. A spectrum label is an *n*-dimensional vector $\vec{x}$ whose j^{th} component $\vec{x}[j]$ indicates the quantity of ion *j* in the particle. Labeling consists of solving the linear system $A\vec{x}=\vec{b}$, $\vec{x}\geq 0$. Noticing that $A\vec{x}=\vec{b}\ \Rightarrow A(\alpha\vec{x})=(\alpha\vec{b})$ for any constant α, we can assume without loss of generality that $\vec{b}$ is normalized (i.e., $\sum_i\vec{b}[i]=1$). By definition of signatures, each column of A also sums to 1. It follows immediately from this fact and $\sum_i\vec{b}[i]=1$ that $\sum_i\vec{x}[i]=1$. The exact quantity of all ions can be easily calculated by multiplying the quantity distribution vector $\vec{x}$ by the overall quantity of the particle, which is simply the sum of signal intensities over all m/z values in the original spectrum before normalization.

3.2 Uniqueness

In general, given the signature library A and an input spectrum $\vec{b}$, neither existence nor uniqueness of solutions for the system

$$A\vec{x}=\vec{b}\ ,\ \vec{x}\geq 0 \qquad (1)$$

is guaranteed. We define following important property.

Definition 1: An input spectrum $\vec{b}$ is said to have the **unique labeling** property with respect to signature library A if there exists a unique solution $\vec{x}_0$ to the system $A\vec{x}=\vec{b}$, $\vec{x}\geq 0$.

For an important class of libraries, we have the following result.

Theorem 1: Consider signature library $A=[\vec{s}_1,\vec{s}_2,...,\vec{s}_n]$ and a spectrum $\vec{b}$ where $\vec{s}_1,\vec{s}_2,...,\vec{s}_n$ are linearly independent (i.e., there is no vector $\vec{a}=[a_1,a_2,...,a_n]$ such that $\sum_{i=1}^{n}a_i\vec{s}_i$ and at least one $a_i\neq 0$). Then, either $\vec{b}$ has the unique labeling property w.r.t. *A*, or the system of equations (1) has no solution.

[Proof: See Appendix A]

Even if a signature library does not satisfy the conditions of Theorem 1, there may still be input spectra $\vec{b}$ for which the solution of (1) is unique. For example, when

$$A = \begin{pmatrix} 0 & 1 & 1/2 \\ 1 & 0 & 1/2 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

there is a unique solution $\vec{x}^T = [0,1,0]$.

Conversely, there will typically be infinitely many solutions for a given spectrum when the signature library does not satisfy the conditions of Theorem 1. For example, let there be a nonzero vector $\vec{a} = [a_1, a_2, ..., a_n]$ with $A\vec{a} = 0$, and equation (1) has a solution $\vec{x} = [x_1, x_2, ..., x_n]$ with $min_{i=1,2,...,n} x_i > 0$. Then $\beta\vec{a} + \vec{x}$ is a solution to (1) for all β with sufficiently small magnitude.

Theorem 2: Consider signature library $A = [\vec{s}_1, \vec{s}_2, ..., \vec{s}_n]$ and a spectrum $\vec{b}$ where $\vec{s}_1, \vec{s}_2, ..., \vec{s}_n$ are *not* linearly independent. If there is a solution $\vec{a} = [a_1, a_2, ..., a_n]$ to $A\vec{x} = \vec{b}$, $\vec{x} \geq 0$ such that $min_{i=1,2,...,n} a_i > 0$, then $\vec{b}$ has infinitely many labels.

[Proof: See Appendix B]

3.3 Spectra with Unique Labeling

In this section, we characterize the complete set of spectra that have the unique labeling property with respect to a given signature library. We explain the concept by means of a pictorial example and then state a theorem that describes this set.

Suppose the signature library has only four signatures $\vec{s}_1, \vec{s}_2, \vec{s}_3, \vec{s}_4$. Figure 2(a) shows the case in which $\vec{s}_1, \vec{s}_2, \vec{s}_4$ are linearly dependent. All normalized spectra that can be represented as a conic combination (that is, a linear combination of the vectors $\vec{s}_1, \vec{s}_2, \vec{s}_3, \vec{s}_4$ in which the coefficients are nonnegative) form the triangle $\Delta s_1 s_2 s_3$ in this example. The ambiguity of the labeling comes from the linear dependency among $\vec{s}_1, \vec{s}_2, \vec{s}_4$, since $\vec{s}_4$ is itself a conic combination of $\vec{s}_1$ and $\vec{s}_2$. However, any point lying on the line $\overline{s_1 s_3}$ can be uniquely represented as a conic combination of s_1 and s_3. The intuitive reason for this is clear: Any involvement of a positive fraction of $\vec{s}_2$ or $\vec{s}_4$ (or both) will lift the point out of the line $\overline{s_1 s_3}$. Similarly the points on the line $\overline{s_2 s_3}$ can be uniquely represented as a conic combination of $\vec{s}_2$ and $\vec{s}_4$. The case in which $\vec{s}_4$ combines all three vectors, $\vec{s}_1, \vec{s}_2, \vec{s}_3$ is shown in Figure 2(b). In this case, any point lying on the boundary of triangle $\Delta s_1 s_2 s_3$ can be uniquely represented as a conic combination of two signatures among $\vec{s}_1, \vec{s}_2, \vec{s}_3$.

We now characterize the tight boundary of spectra with unique labeling property formally. A polynomial time algorithm for testing the unique labeling property of a spectrum is also presented in Appendix C.

Definition 2: Given a signature library $S = \{\vec{s}_1, \vec{s}_2 \cdots \vec{s}_n\}$, the **convex hull** generated by S is defined as:

$$ch(S) = \{\textstyle\sum_{i=1}^{n} w_i \vec{s}_i \mid n \geq 1, \sum_{i=1}^{n} w_i = 1, w_i \geq 0, \vec{s}_i \in S, 1 \leq i \leq n\}$$

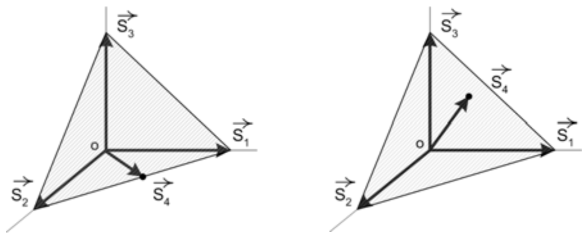
Theorem 3: The set of spectra with the unique labeling property w.r.t. library S is the set of points in *ch(S)* that do not lie in the interior of the convex span of some affine dependent subset of S.

[Proof: See Appendix D]

4. HANDLING AMBIGUITY AND ERRORS

In practice, signal intensities are not precisely calibrated, and background noise causes measurement errors and introduces uncertainty. We therefore introduce an error bound *E* and a distance function *D*, and recast the labeling problem in terms of mathematical programming, as an "exhaustive" feasibility task:

$$\text{Seek all } \vec{a} \text{ such that } D(A\vec{a}, \vec{b}) \leq E, \vec{a} \geq 0. \qquad (2)$$



(a) (b)

Figure 2: Vector Space Spanned by Signatures

Given a library *A* with *n* signatures and input spectrum $\vec{b}$, the search space for problem (2) is a *n*-dimensional[1] space. The **solution space** for input spectrum $\vec{b}$ is defined as follows.

Definition 3 **:** Given a signature library *A*, an input spectrum $\vec{b}$ and an error bound *E* with respect to distance function *D*, the **solution space** of spectrum $\vec{b}$ is $L_{\vec{b}} = \{\vec{a} \mid D(A\vec{a}, \vec{b}) \leq E, \vec{a} \geq 0\}$.

It is worth noting that the choice of the distance function *D* may affect the complexity of the problem significantly. We use *Manhattan Distance* (also known as ℓ_1 norm), which allows us to find a specific element of the solution set for (2) by using the following linear programming (LP) model:

$$min \textstyle\sum_i s_i \quad s.t.\ A\vec{a} - \vec{b} \leq s,\ A\vec{a} - \vec{b} \geq -s,\ \vec{a} \geq 0,\ \vec{s} \geq 0.$$

We observe that if the distance function is convex, the solution space of an input spectrum $\vec{b}$ is convex (see below). We will explore this property further in Section 5; for now, we note that *Manhattan Distance* is a convex distance function.

[1] Actually, an *n*-1 dimensional space if *b* is normalized.

Theorem 4: If the distance function D has the form $D(u,v) = d(u-v)$, where d is a convex function, then the solution space of the search described by Equation (2) is convex.

Proof: Let $\vec{a}_1$ and $\vec{a}_2$ be two elements of the solution space. We wish to show that $\vec{a} = \alpha\vec{a}_1 + (1-\alpha)\vec{a}_2$ is also a member of the solution space, for any $\alpha \in [0,1]$. Since $\vec{a}_1 \geq 0$ and $\vec{a}_2 \geq 0$, we have trivially that $\vec{a} \geq 0$ also. In addition, by definition of $\vec{a}$ and convexity of d, we have

$$\begin{aligned} D(A\vec{a},\vec{b}) &= d\left(A\vec{a}-\vec{b}\right) = d\left(\alpha(A\vec{a}_1-\vec{b}) + (1-\alpha)(A\vec{a}_2-\vec{b})\right) \\ &\leq \alpha d\left(A\vec{a}_1-\vec{b}\right) + (1-\alpha)d\left(A\vec{a}_2-\vec{b}\right) \leq \alpha E + (1-\alpha)E = E, \end{aligned}$$

so that $D(A\vec{a},\vec{b}) \leq E$ and $\vec{a}$ is a member of the solution space.

4.1 Discretization

Even if an input spectrum has an infinite number of labels (for a given signature library) because of ambiguity, in practice, we do not need to distinguish between solutions that are very similar. A natural approach to deal with the infinity in a continuous space is to discretize it into grids, so that the number of possible solutions becomes finite.

Formally, a **threshold vector** $\vec{t} = [t_0, t_1, ..., t_d]$ divides each dimension of the search space into *d* ranges, where t_i and t_{i+1} are the lower bound and upper bound of range *i*. Given a threshold vector, we introduce the notion of **index vector** to represent a continuous subspace.

Definition 4: Given a threshold vector $\vec{t} = [t_0, t_1, ..., t_d]$, an **index vector** $I = [(l_1,h_1),(l_2,h_2),...,(l_n,h_n)], l_i < h_i, l_i, h_i \in Z$ represents a continuous subspace,

$$S_I = \{\vec{a} \mid \forall i, \vec{t}[l_i] < \vec{a}[i] \leq \vec{t}[h_i], \vec{a}[i] \in R\}$$

Since an index vector represents a unique subspace, we will refer to a subspace simply by its corresponding index vector when the context is clear. Using the index vector representation, we in turn define the notion of **cell**.

Definition 5: A subspace $[(l_1,h_1),(l_2,h_2),...,(l_n,h_n)]$ is a **cell** if $\forall j, l_j + 1 = h_j$.

The cell is the finest granularity of the discretization, which reflects the degree of detail users care about. A threshold vector $\vec{t} = [t_0, t_1, ..., t_d]$ divides the whole search space into d^n *cells*, where *n* is the number of dimensions (which is equivalent to the total number of signatures in the signature library). Each cell also corresponds to a distinct *n*-dimensional integer vector

$$\vec{y} = [y_1, y_2, ..., y_n], 1 \leq y_i \leq d, \, y_i \in Z$$

which defines a subspace Y corresponding to the index vector $[(y_1, y_1+1), (y_2, y_2+1), ..., (y_n, y_n+1)]$.

4.2 Optimization Model

We now redefine the task of **spectrum labeling** as follows: *Find all the cells that intersect the solution space of the input spectrum.* A **label** of spectrum $\vec{b}$ is then simply an integer vector $\vec{x}$ whose cell intersects the solution space corresponding to $\vec{b}$. All integer vectors whose corresponding cells intersect $\vec{b}$'s solution space form the **label set** of spectrum $\vec{b}$. Formally,

Definition 6: $\vec{x}$ is a **label** of spectrum $\vec{b}$ if the subspace defined by the *index vector* $X = [(x_1, x_1+1), (x_2, x_2+1), \ldots, (x_n, x_n+1)]$ intersects the *solution space* of spectrum $\vec{b}$. Spectrum $\vec{b}$'s **label set** is $L = \{\vec{x} \mid \vec{x} \text{ is a label of } \vec{b}\}$

To simplify the discussions in the following sections, we also introduce the notion of **feasible space** to describe a subspace that intersects the solution space of the input spectrum. If the *feasible space* is a cell, it is also called *label*, because there is a one-one correspondence between a *label* and a *cell* that intersects the input spectrum's solution space.

Figure 3 illustrates the concepts discussed in this section. Suppose there are only two signatures in the signature library. The whole search space is a two dimensional space *ABCD* within which $S_1S_2S_3S_4$ forms the solution space of a input spectrum. It intersects the *cells LFGM* and *MGHA*, each of which corresponds to a *label*. Subspace *ALFH* intersects with the solution space, so it is a *feasible space*. *MBEG* is also a *feasible space.*

Table 1 summarizes our notation and describes an optimization model for the labeling task.

Notations:		
	$\vec{x}$	An n-dimensional integer vector, $1 \le \vec{x}[i] \le d$
	$\vec{b}$	Normalized input mass spectrum
	$\vec{t}$	Threshold vector for discretization
	d	Number of ranges per dimension under the discretization
	L	Label set of input spectrum
	A	Signature library with n signatures
	D	Distance function
	E	Error bound

```
$L = \varnothing$
for every possible $\vec{x}$
    Seek $\vec{a}$ s.t.
        $D(A\vec{a}, \vec{b}) \le E$
        $\vec{t}[j] < \vec{a}[i] \le \vec{t}[j+1], j = \vec{x}[i]$        (*)
    if (*) succeeds, $L = L \cup \{\vec{x}\}$
```

```
return L
```

Table 1: Operational Definition of Spectrum Labeling

5. LABELING ALGORITHMS

In Section 4, we showed that given *n* signatures and discretization granularity *d*, the search space contains d^n cells. A brute force approach that tests the feasibility of each cell is not practical, considering that there are hundreds of signatures. In this section, we propose two algorithms: DFS is general, and Crawling exploits convexity of distance functions.

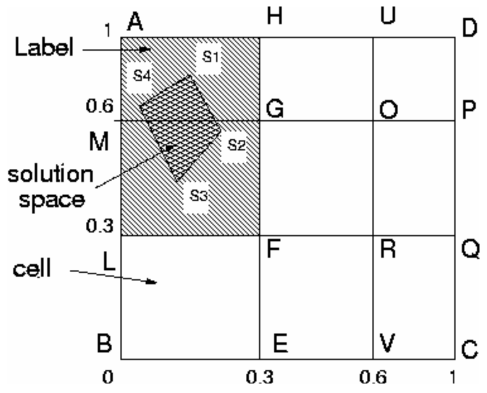


Figure 3: Illustration of Concepts

5.1 Feasibility Test

Given a subspace *S*, we use the algorithm shown in Table 2 to test the feasibility of the subspace; this algorithm is a building block. Notice that exactly one LP call is invoked for the test.

5.2 Depth-First Search (DFS) Algorithm

Before we specify the DFS algorithm, we first state an important property of subspace feasibility. The proof is straightforward and is omitted.

Theorem 5: Let a spectrum $\bar{b}$ and a signature library A with *n* signatures be given. If subspace *S* is feasible, then any subspace T, with $S \subset T$ is also feasible.

The DFS labeling algorithm explores a search tree in which each node is associated with a particular subspace, and the subtree rooted at that node corresponds to the subsets of that subspace. At each node, the algorithm first tests the feasibility of the subspace for that node. If not feasible, that node and its subtree are pruned. Otherwise, we select a dimension j that has not been subdivided to the finest possible granularity in the subspace of that node, and divide the subspace further along dimension j. Each smaller space created thus corresponds to a child of the current node.

In Table 3, the *pick_dimension* method chooses a dimension (which is not already at the finest granularity possible) to split the current subspace, and *split_subspace* divides the current subspace into smaller pieces along the chosen dimension. Variants of these two methods are discussed in the following subsections.

```
Input:     $\vec{b}$   Input mass spectrum
           $E$   Error bound
           $\vec{t}$   Threshold vector
Output:    TRUE if the subspace is feasible, FALSE otherwise

Is_feasible(subspace S)
 Seek $\vec{a}$ s.t.
     $D(A\vec{a},\vec{b}) \le E$          (*)
     $\vec{t}[l_j] < \vec{a}[j] \le \vec{t}[h_j]$, for $j = 1,2,...,n$
     $[(l_1,h_1),(l_2,h_2),...,(l_n,h_n)]$ is the index vector of subspace $S$
 if (*) succeeds, return TRUE, otherwise return FALSE
```

Table 2: Test The Feasibility of A Subspace

```
Input:     $\vec{b}$   Input mass spectrum
           $E$   Error bound
           $\vec{t}$   Threshold vector
Output:    Label set $L(\vec{b})$

Depth_First_Search(subspace S)
   $L \leftarrow \varnothing$
   if not Is_feasible(S)
      return L;
   else
      if S is a cell
         $L \leftarrow$ label corresponding to S;
         return L;
      else
         pick_dimension(j)
         $\{S_i\}$ = split_subspace(S, j)
```

```
        for each $S_i$
            $L \leftarrow L \cup Depth\_First\_Search(S_i)$
    return $L$;
Main : Depth_First_Search(whole search space W)
```

Table 3: Depth First Search Algorithm

5.2.1 Split Subspace

After the split dimension is chosen, the current subspace will be divided along the chosen dimension into smaller slices. Suppose the index vector corresponding to the current subspace is $I = [(l_1, h_1), (l_2, h_2), ..., (l_n, h_n)]$ and the split dimension is *j*, the *simple split* variant will divide the subspace into "slices", each of which has the finest granularity in dimension *j*. That is, the resulting "slices" to be explored individually at the next level of recursion are $\{ [(l_1, h_1), ..., (x, x+1), ..., (l_n, h_n)] \mid l_j \le x \le h_j - 1 \}$. The *binary split* variant divides the subspace into two "slices" of approximately equal width along dimension *j*. If a particular dimension, which corresponds to a signature in the library, has various values across different labels, the *simple split* scheme is better than the *binary split* scheme. If a convex distance function is used, we can even adopt a binary search strategy to determine the boundary of the spectrum's solution space and split the dimension in a way that only one of the resulting "slices" intersect with the spectrum's solution space. Our implementation of the DFS algorithm does not rely on the convexity of the distance function. The *simple split* scheme is used.

5.2.2 Pick Dimension

The choice of dimension along which to split affects the efficiency of the DFS algorithm in practice. Suppose we have two signatures in the library and the labels for the given input spectrum are [0,1] (cell *LFGM*) and [0,2] (cell *MGHA*), as illustrated in Figure 3. The *simple split* scheme is used. Figure 4(a) shows the search tree when we first choose dimension 1 (x-axis). It divides the whole search space into three slices *BEHA*, *EVUH* and *VCDU* and tests the feasibility of each. *BEHA* is a feasible subspace and is further split on dimension 2. Cells *BEFL*, *LFGM*, *MGHA* are tested. Two labels *LFGM* and *MGHA* are found and the algorithm terminates. Similarly, Figure 4(b) shows the case when dimension 2 is selected first. In this case, the search space is first split along the y-axis, resulting in two feasible subspaces *LQPM* and *MPDA* for further exploration. In the search trees shown in Figure 4, each node corresponds to a subspace upon which an LP call is invoked. A node is colored black if it corresponds to a feasible subspace. Leaf nodes correspond to cells in the discretized search space. A leaf node is black if it corresponds to a label. As we can see, the first search tree invokes 7 LP calls and the second one invokes 10 LP calls. When the tree is large, the performance difference between various *pick_dimension* schemes becomes even more significant. Generally speaking, during the search process, we would like to select those dimensions (signatures) which lead to fewer future splits. A "greedy" algorithm would select the split node to be the one that has the fewest feasible children. If all labels have the same value on a particular dimension (that is, just one feasible child), that dimension is 'good' and should be selected as early as possible.

Theorem 6 shows that if a convex distance function is used and the number of labels for the input spectrum is at most equivalent to the number of signatures in the signature library, the existence of such a “good” dimension is guaranteed. In our current implementation of the DFS algorithm, we simply choose a dimension that has not been selected before.

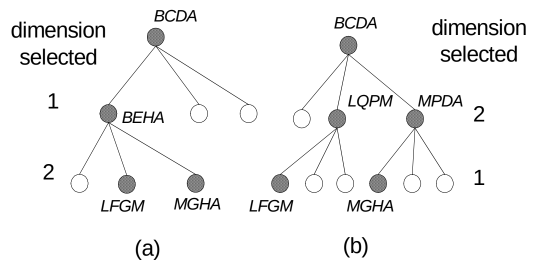


Figure 4. Search Trees for Different Split Dimension Choices

Theorem 6: Suppose the convex set K is contained in some “brick” $B=[L_1,U_1]\times\ldots\times[L_n,U_n]$ and that B is divided into r_i ranges along each dimension i, $L_i=a_{i,0}<a_{i,2}<a_{i,r_i}=U_i$, and B is the union of $\prod_{i=1}^{n} r_i$ many “sub-bricks” of the form $[a_{1,j_1},a_{1,j_1+1}]\times\ldots\times[a_{n,j_n},a_{n,j_n+1}]$. If K has nonempty intersection with at most *n* of these “subbricks”, then there exists a dimension *i*, and a “slice” $\{x\in B\,|\,x_i\in[a_{i,j_i},a_{i,j_i+1}]\}$ that contains K.

[Proof: See Appendix E]

5.2.3 Correctness and Complexity

Theorem 7: Given an input spectrum and a specified error bound, the DFS algorithm finds the complete label set of the spectrum.

Proof: The algorithm checks the feasibility of nodes from top to bottom of the search tree. Each child of a node will be visited if that node represents a feasible solution subspace. Each label corresponds to a feasible cell and, according to Theorem 5, all its supersets will also be feasible. Hence there is a path from the root search node to each label node, such that all the nodes along this path correspond to feasible subspaces and therefore will not be pruned. Because of this fact, the search node for each label will be visited eventually and the label will be output.

On the other hand, every output label corresponds to a cell. The *is_feasible* test on that cell guarantees that it is output as a label for the spectrum only if it is feasible. □

It can be shown that given a signature library with *n* signatures and a threshold vector that divides each dimension into *d* ranges, the number of LP calls invoked by DFS algorithm to label a spectrum with *k* labels is *O(knd)*. A detailed analysis based on a “node coloring” model is presented in [7].

5.3 The Crawling Algorithm

DFS is a general algorithm in which we can use any distance function *D*, even one that is non-convex. The Crawling algorithm requires the distance function to be convex and exploits the *connectivity property*[2] derived from the convexity of solution spaces, as described in Theorem 5.

Connectivity Property: Given two labels l_1 and l_n, there exists a path of connecting labels ($l_1, \dots, l_{i-1}, l_i, \dots l_n$) in which l_{i-1} , l_i are adjacent, i.e., differ only in one dimension by 1.

```
Input:    $\vec{b}$   Input mass spectrum
          $E$   The error bound
          $\vec{t}$   The threshold vector
Output:   Label set of $\vec{b}$

Crawling()
   Seek $\vec{a}$ s.t.
      $D(A\vec{a},\vec{b}) \le E, \vec{a}[i] \ge 0$   (*)
   if (*) fails
      return $\varnothing$
   else
      $L \leftarrow \varnothing$
      find $l_0 = [x_1, x_2, ..., x_n]$
         s.t. $t[x_i] < \vec{a}[i] \le t[x_i + 1]$, $i = 1,2,3,...,n$
      insert $l_0$ into the queue Q and mark l as generated
      while (Q is not empty) {
         pick the first index vector $l = [x_1, x_2, ..., x_n]$ from Q
         $L \leftarrow L \cup \{l\}$ and remove l from Q
         for $j$ = 1 to n do
            if ( $x_i \ne d - 1$ )
                  $l' = [x_1, x_2, ..., x_j + 1, ..., x_n]$
                  if $l'$ is a label of $\vec{b}$ and
                     $l'$ has not been generated before, then
                        insert $l'$ into Q, mark l' as generated
            if $x_i \ne 0$
                  $l'' = [x_1, x_2, ..., x_j - 1, ..., x_n]$
                  if $l''$ is a label of spectrum $\vec{b}$ and
                     $l''$ has not be generated before, then
                        insert $l''$ into Q, mark $l''$ as generated
      } // end of while loop
      return L
```

Table 4: Crawling Algorithm

The Crawling algorithm first finds a solution to the linear system $D(A\vec{a},\vec{b}) \le E, \vec{a}[i] \ge 0$ by invoking a LP call. The cell contains that solution is a label and is used as the start point to explore other connected cells in a breath-first fashion. If the cell discovered has not been visited before and is a label, its neighbors will be explored

[2] Convexity is actually stronger than the connectivity property.

subsequently. Otherwise, it is discarded and no further exploration will be incurred by it. The algorithm stops when all labels and their neighbors are visited. The connectivity property guarantees that all labels are connected to the first label we found and can be discovered by "crawling" from that start point. The algorithm is described in Table 4.

Let's take Figure 3 in Section 4 again as an example. The input spectrum's label set contains two labels which are the cells in shade. The crawling algorithm first finds a solution point. Suppose it falls in cell *LFGM*. It then starts from *LFGM* and explores its neighbors *LBEF*, *FROG* and *MGHA*. Among the three, only cell *MGHA* is a label and will incur further exploration. It has only two adjacent cells. One is already visited and the other is not a label. Thereby the algorithm terminates and outputs *LFGM* and *MGHA* as the input spectrum's label set.

Theorem 8 Given an input spectrum $\vec{b}$, a signature database $A=[\vec{S}_1,\vec{S}_2,...,\vec{S}_n]$ and a threshold vector $\vec{t}=[t_1,t_2,...,t_{d+1}]$, suppose the number of labels for $\vec{b}$ is k, the Crawling algorithm will find the complete set of the labels for input spectrum.

Proof:

Completeness: According to the connectivity property, any label l connects to the initial found label l_0, i.e., there is a label path $l_o,l_{x1},...,l_{xn},l$. Along this path, l_{x1} will be found when l_0 is removed from the queue *Q* and all the neighbors of l_0 are tested. l_{x2} will be found at least when l_{x1} is picked and its neighbors tested. (Note that l_{x2} does not necessarily to be generated from l_{x1} since there might be other path which lead from l_0 to l_{x2}). By using induction, it is easy to see that l will be found not later than the time when l_{xn} is picked and its neighbor tested. Hence, All the labels will finally be collected in queue *Q* and put in the result set L.

Correctness: All the labels found by Crawling algorithm are correct because one index vector has to go through the feasibility test before it is inserted into the queue *Q*. The result label set *L* only contains labels come from *Q*. Hence, all the labels in *L* should be correct. □

Theorem 9: Given an input spectrum $\vec{b}$, a signature library $A=[\vec{s}_1,\vec{s}_2,...,\vec{s}_n]$ and a threshold vector $\vec{t}=[t_0,t_1,...,t_d]$, suppose the number of labels for $\vec{b}$ is k, then the number of LP calls invoked by the Crawling algorithm is *O(kn)*. The number of index vectors stored in the queue is *O(kn)*.

Proof: Each cell in the grid has at most $2n$ neighbors, two in each dimension. We paint all the cells in the grid using the following rules: 1) If a cell is a label, we paint it black; 2) if a cell is not a label, but it has a label neighbor, we paint it grey; 3) if a cell neither is a label nor has a label neighbor, we paint it white. The number of black cells equals to the number of labels *k*. Since each cell has at most *2n* neighbors and there are *k* black cells, the number of grey cells is at most *2kn*. According to the *connectivity property*, all black cells are connected to each other by a path consisting of only black cells. The algorithm will only generate those cells in black or grey and invoke exactly one LP call for each of them. Since the number of black and grey cells together is no more than *k+2kn*, the number of LP calls invoked is *O(kn)*

The number of index vectors in the queue *Q* equals to the number of black cells and grey cells. As stated above, the number of black and grey cells is *O(kn)*.So the space complexity of the crawling algorithm, which is measured by the number of index vectors in the queue, is also *O(kn)*. □

5.4 Directional Bounding Algorithm

We now describe an algorithm that is similar to crawling in that it progressively identifies solutions by searching in different dimensions successively. The approach differs in that it uses a binary search to determine the extent of the solution space along a given dimension. Hence, it may not need to actually solve a linear program for each of the cells that actually solve (2). For solutions sets with certain shapes that extend over a moderate to large number of cells, and for moderate to large values of d, this approach may yield significant savings over the Crawling Algorithm and DFS.

In order to specify the directional bounding (DB) algorithm and the binary search at its core, we first define some notation:

$\text{label}(\vec{b}) = \text{Label set of } \vec{b}$

$D \subset \{1,2,...,n\} = \text{"explored" dimensions}$

$\vec{x}_i(\vec{a}) = \text{integer } x_i \text{ in } \{1,2,...,\text{d}\} \text{ such that } t[x_i] \le \vec{a}[i] < t[x_{i+1}]$

$\hat{Q} = \text{set of elements } (\vec{x}, D) \text{ where } \vec{x}_i \in \{1,2,...d\},\ i = 1,2,...n,$
  $\text{and } D \subset \{1,2,...,n\}.$

The binary search procedure in Table 5 picks a certain cell $\vec{x}$ belonging to $\text{label}(\vec{b})$ and explores a given dimension k. to see how far the label set extends along this dimension.

Input: Spectrum $\vec{b}$, threshold vector $\vec{t} = [t_1, t_2, ..., t_{d+1}]$, error bound E, label $\vec{x} \in \text{label}(\vec{b})$, search dimension k.

Output: labels (l,r) with $1 \le l \le r \le d$.

```
$(l,r) \leftarrow$ BinarySearch $(\vec{x}, k)$
    $r_U \leftarrow d+1, r \leftarrow \vec{x}_k$;
    while $r_U - r > 1$
        $p = \lfloor (r_U + r)/2 \rfloor$;
        if there exists $\vec{a}$ such that
            $D(A\vec{a},\vec{b}) \le E, \quad \vec{a} \ge 0,$
            $t[x_i] \le a[i] \le t[x_{i+1}], \quad i \ne k,$
            $t[p] \le a[k] \le t[p+1],$
        then $r \leftarrow p$
        else $r_U \leftarrow p;$
    end (while);

    $l_L \leftarrow 0, l \leftarrow \vec{x}_k;$
    while $l - l_L > 1$
        $p = \lceil (l_L + l)/2 \rceil;$
        if there exists $\vec{a}$ such that
            $D(A\vec{a},\vec{b}) \le E, \quad \vec{a} \ge 0,$
            $t[x_i] \le a[i] \le t[x_{i+1}], \quad i \ne k,$
            $t[p] \le a[k] \le t[p+1],$
        then $l \leftarrow p$
        else $l_L \leftarrow p;$
    end (while);
```

Table 5: Binary Search Procedure

We now describe the DB algorithm itself. The algorithm is initialized with a valid label $\vec{x}$. It searches outward from $\vec{x}$ along each dimension in turn, identifying valid labels in each direction and adding them to a set $\hat{Q}$ for further investigation.

```
Input: :    Spectrum $\vec{b}$, threshold vector $\vec{t} = [t_1, t_2, ..., t_{d+1}]$,
error bound $E$.
Output: $\text{label}(\vec{b})$

DB()
    Seek $\vec{a}$ such that $D(A\vec{a},\vec{b}) \le E, \quad \vec{a} \ge 0.$
    if no such $\vec{a}$
    then STOP with $\text{label}(\vec{b}) = \varnothing;$
    else $D \leftarrow \varnothing; \quad \hat{Q} \leftarrow \{(\vec{x}(\vec{a}), D)\};$
    while $\hat{Q} \ne \varnothing$
```

```
        remove an element $(\vec{x}, D)$ from $\hat{Q}$;
        **if** $D = \{1, 2, ..., n\}$
        **then** $\text{label}(\vec{b}) \leftarrow \text{label}(\vec{b}) \cup \{\vec{x}\}$
        **else**
          choose any $k \notin D$;
          $(l, r) \leftarrow \text{BinarySearch}(\vec{x}, k)$;
          **for** $p = l, l+1, ..., r$
            define $\vec{y}$ such that $\vec{y}_i = \vec{x}_i, i \neq k, \quad \vec{y}_k = p$;
            $\hat{Q} \leftarrow \hat{Q} \cup \{(\vec{y}, D \cup \{k\})\}$
          **end(for)**
        **end(if)**
      **end(while)**
```

Table 6: Directional Bounding Algorithm

6. DATA GENERATION

There is a fundamental Cdifficulty in evaluating algorithms for labeling mass spectra: Manual labeling of spectra (to create training sets) is laborious, and must additionally be cross-validated by other kinds of co-located measurements, such as traditional filter-based or "wet chemistry" techniques. For any given application, rigorously establishing appropriate "ground truth" datasets can take months of field-work. In this section, we describe a detailed approach to synthetic data generation that allows us to use domain knowledge to create signature libraries and input particle spectra that reflect specific application and instrument characteristics. This has two important benefits:

- It allows us to compare different labeling algorithms, and to quantify their effectiveness with respect to important parameters such as *noise, unknowns,* and *ambiguity*.
- It allows a domain scientist to generate "labeled" datasets that mimic their application, and to rapidly fine-tune algorithm settings.

Our generator has two parts: generation of the signature library, and generation of input spectra. We begin with a collection of real ion signatures, and select a set of *n* linearly independent signatures to serve as "seeds" for our generator. New signatures are generated using a non-negative weighted average of seed signatures. The set of all generated signatures is partitioned into two sets: the *signature library*, and the *unknowns*.

The generation of new signatures for the signature library is done in "groups" as follows, in order to control the degree of (non-)uniqueness, or ambiguity. Each group consists of two "base" signatures from the seeds (chosen such that no seed appears in multiple groups) plus several "pseudo-signatures" obtained using non-negative weighted averages of these two signatures. The generated signatures in each group are effectively treated as denoting new ions in the signature library. Of course, they do not correspond to real ions at all; rather, they represent ambiguity in that it is impossible to distinguish them from the weighted average of base signatures used to generate them when labeling an input spectrum that contains ions from this group. Intuitively, the larger the size of a group, the greater is the ambiguity in input spectra that contain ions from the group; observe that interference can only occur within groups. We create a total of *k* groups with *i+1* pseudo-signatures in group *i*.

The set of *n* original signatures plus the *(k+3)k/2* pseudo-signatures generated as above constitute our "universe" of all signatures. Next, we select some of these signatures to be unknowns, as follows: We randomly select one signature from each of the *k* groups; these *k* signatures are "interfering unknowns". We also randomly select *u-k* seed signatures that were not used in group generation; these *u-k* signatures are "non-interfering unknowns", giving us a total of *u* unknowns.

The second part of our generator is the generation of input spectra. An input spectrum is generated by selecting *m* signatures from the universe of signatures and adding them according to a weighting vector $\vec{w}$. Ambiguity and unknowns are controlled by the careful selection of signatures that contribute to the spectrum, and the input weighting vector controls the composition of the spectrum as well as the contribution of unknowns. We observe that the effect of many unknowns contributing to an input spectrum can be simulated by aggregating them into a single unknown signature with an appropriate weighting vector; accordingly, we use at most a single unknown signature. Table 11 summarizes the parameters for spectrum generation.

m	Number of signatures that contribute to the spectrum.
q	Degree of ambiguity, specified in terms of group number.
$\vec{w}$	Weighting vector of m elements sorted in descending order, except the right-most element, which controls the weight of unknown signature.
o	o = 1: unknown signatures is used in spectrum generation; o = 0: unknown signatures are not used
v	v = 1 if unknown signature selected interferes with known signatures selected. Otherwise v = 0.
g	Average amount of noise added to each m/z position.

Table 11: Parameters Used for Spectrum Generation

We begin by randomly selecting two signatures from group *q*. Then, if unknowns are desired in the generated spectrum (*o*=1), we choose either the q^{th} unknown signature, or a randomly selected non-interfering unknown signature, depending on whether or not the unknown is desired to interfere with known ions in the spectrum (v = 1 or 0). The contribution of unknowns is controlled by the last component of the weighting vector. Next, we randomly select signatures from the signature library that do not belong to any of the k "groups" to get a total of m signatures. These signatures are linearly independent seeds, and thus the ambiguity of the generated spectrum will depend solely on the first 2 (or 3, if an interfering unknown is chosen) signatures.

Finally, we select values for m random variables following a normal distribution whose means are given by the weighting vector of arity m. The values for these variables are used as the weights w_i to combine the m signatures: $\sum_{j=1}^{m} w_j S_j$. (We note that when an unknown signature is used in the generation, the last element of the weighting vector is reset to be the relative quantity of the unknown signature and the whole weighting vector is normalized to sum up to 1.)

We account for noise by simply adding a noise value (a random variable following a normal distribution) to each component (i.e., m/z position) of the generated spectrum.

7. EXPERIMENTAL RESULTS

We now describe experiments to evaluate our labeling algorithms with respect to both quality and labeling speed. To give the reader an idea of the speed, we observed an average processing rate of about 1 spectrum per second when we ran our algorithms on over 10,000 real mass spectra collected using an ATOFMS instrument in Colorado and Minnesota; this is adequate for some settings, but not all, and further work is required. Speed and scalability are not the focus of this paper, but are addressed in [7], and extensive experiments are reported. We also tested the accuracy of our labeling algorithm against a small set of manually labeled spectra; all were correctly labeled by the algorithm. Admittedly, this is not an extensive test, but we are limited by the fact that manual labeling is a tedious and costly process. (This underscores the importance of not requiring training datasets.)

In this section, we therefore evaluate our algorithms using the data generator from Section 6; this approach also allows us to study the effect of ambiguity, unknown signatures and noise levels in a controlled fashion. For comparison, we also evaluated machine learning (ML) algorithms based on WEKA [17]. However, the reader should note that our algorithms can label input spectra *given only the signature library,* whereas the ML approaches require extensive training datasets, which is unrealistic with manual labeling. In addition, the ML algorithm ignores equivalent alternatives and only generates one label. Nonetheless, we propose two different quality measures and include the comparison for completeness, and to motivate a promising direction for future work, namely the development of hybrid algorithms that combine the strengths of these approaches.

7.1 Machine Learning Approach

Our ML algorithm builds upon WEKA classifiers [17]. For each signature in the signature library, we train a classifier to take a given input spectrum and output a presence category (*absent, uncertain* or *present*), i.e., to detect whether or not the ion represented by the signature is present in the particle represented by the spectrum. The predictive attributes are the (fixed set of) m/z locations, taking on as values the signal intensities at these locations in the input spectrum. To label a spectrum, we simply classify it using the classifiers for all signatures in the library. When we are only interested in the presence of a subset of ions, of course, we need only train and run the classifiers for the corresponding signatures. We evaluated four types of classifiers: Decision Trees (J48), Naïve Bayes, Decision Stumps, and Neural Networks. Decision Trees consistently and clearly outperformed the other three, and we therefore only compare our algorithms against this approach in the rest of this section.

7.2 Datasets

The (common) dimension of all signatures and spectra is set to be 255, which is the value in our real dataset. We used n=78 base signatures of real ions, and generated k=5 groups containing 2,3,4,5, and 6 pseudo-signatures, respectively. Including the original 78, we thus obtained 98 signatures, 15 of which were withheld as unknown; the remaining 83 comprised the signature library. For generating input spectra, we set the number of signatures used for spectrum generation to be m=10. The relative proportion of these m signatures was controlled by the weighting vector [0.225, 0.2, 0.2, 0.1, 0.1, 0.06, 0.06, 0.03, 0.01, 0.01]. The weighting vector simulates the fact that there are usually around 3 majority ions in a particle and 4 to 5 ions of medium quantity.

We generated five testing datasets with controlled ambiguity, unknown signature and noise levels. Each dataset contains several files, each of which contains 1,000 spectra generated by using the same set of parameter values. Dataset 1 is designed to test the effect of noise. It consists of 10 files. Each file corresponds to a distinct noise level from 0% to 360% of a preset error bound[3]. No ambiguity or unknown signature is involved in this dataset. Dataset 2 tests the effect of ambiguity. It consists of 5 files corresponding to 5 ambiguity levels. No unknown signature or noise is introduced in this dataset. Dataset 3 has no noise or ambiguity, but contains some non-interfering unknown signatures. This dataset contains ten files, with the weight on the unknown signature varying from 0% to 180% of the preset error bound. Dataset 4 is identical to Dataset 3 except that the unknown signatures selected are interfering unknowns. Dataset 5 is designed to test the combined effect of noise and ambiguity. Five ambiguity degrees used in Dataset 2 and five noise levels selected from the 10 noise levels used in Dataset 1 result in 25 different combinations of noise and ambiguity, and 25 files are generated for each such combination. The discretization criteria used for all the dataset above is controlled by a threshold vector [0, 0.08, 0.18, 1] which divides the quantity of ion into three categories: *absent*, *uncertain* and *present*.

[3] The error bound is set to be 0.01. The noise level represents the overall average amount of noise added to 255 m/z positions. For instance, at noise level 0.01 (100% of error bound) the expected absolute noise at each m/z position is 0.01/255.

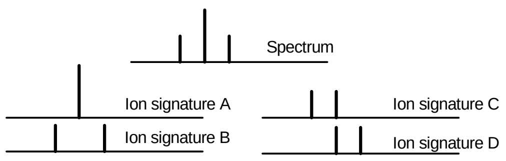


Figure 5: Indistinguishable Spectrum Labels

7.3 Labeling Quality

Given a particle, consider the "*ideal*" version of its spectrum obtained by eliminating noise and unknowns, and is therefore strictly the weighted sum of known ion signatures present in the particle. Even such an ideal mass spectrum might not have unique labels. The spectrum shown in Figure 5 might represent a particle that contains ions A and B, or a particle that contains C and D. Given only the input spectrum, the combinations AB and CD are mathematically indistinguishable, and should be presented to domain experts for further study. The complete set of such "*indistinguishable spectrum labels*" for the "*ideal*" version of an input spectrum is the best result we can expect from labeling; we call each label in this set a **correct** label. Intuitively, it is the set of all feasible combinations of ions in the particle. This is exactly the *label set* of the ideal spectrum defined in Section 4 (with the error bound set to 0). By Theorem 7, our algorithms generate this label set when no unknown or noise is present, i.e., the ideal version is the given input spectrum itself. However, as noise and unknowns are added, the labels found by our algorithm will no longer the same as the desired set of all correct labels.

Our first proposed metric compares the result of a labeling algorithm with the set of all correct labels. This metric consists of two ratios: the *hit ratio* and *false ratio*. The *hit ratio* is the percentage of correct labels in the result set of the labeling algorithm. The *false ratio* is the proportion of labels in the result set that are not correct labels. Formally, let the *label set* of a particle's *ideal spectrum* be L_T and let the set of labels found by a labeling algorithm for the particle's real spectrum under the presence of noise and unknowns be L_O:

$$\textit{Hit Ratio} = | L_T \cap L_O | / | L_T | \qquad \textit{False Ratio} = | L_O - L_T | / | L_O |$$

Experiments under this metric will be called *full labeling tests,* and are presented in Section 7.3.1.

Our second metric relaxes the requirement of finding the correct combinations of ions, and focuses on the proportion of individual ions whose presence or absence is correctly classified. Given a collection of *interesting ions*, we aggregate the set of correct spectrum labels to obtain a set of ion labels IL_T as follows: An ion of interest is marked *present* if all correct labels mark it as *present*, *absent* if all correct labels mark it as *absent*, and marked *uncertain* in all other cases. Similarly, we can obtain a set of ion labels IL_O from the result set of the labeling algorithm. Our second metric consists of two ratios based on these ion labels:

$$\textit{Partial Hit Ratio} = | IL_T \cap IL_O | / | IL_T | \qquad \textit{Partial False Ratio} = | IL_O - IL_T | / | IL_O |$$

Partial hit ratio is similar to *hit ratio*, and describes the percentage of ions that are correctly labeled, while *partial false ratio* is the proportion of ions that are incorrectly labeled. Experiments under this second metric will be called partial *labeling tests,* and are presented in Section 7.3.2.

It is worth noting that for any given spectrum, our algorithms will generate exactly the same result, namely, the label set of that spectrum. Therefore, for quality evaluation, we simply refer to them as "our algorithm" or the "LP" algorithm, since they both build upon linear programming.

7.3.1 Full Labeling Tests

In the following graphs, each data point for our algorithm is the average of results on 1000 spectra, while each data point for the ML algorithm is the result of 5-fold cross validation on the same 1000 spectra. Figure 6 shows the result on Dataset 1, which contains no ambiguity or unknowns. In this case, the labeling algorithm is not required to resolve any ambiguity. Even in this simple case, the ML algorithm performs poorly. Its hit ratio is close to zero while the false ratio is close to one. In contrast, our algorithm shows great strength in identifying a possible combination of ions to explain the spectrum. The hit ratio remains almost perfect when the noise is within 180% of error bound, but drops sharply when noise grows above that threshold. This shows a limitation of our algorithm: *the error bound is the only component that accounts for noise*, and our results are sensitive to the choice of the error bound relative to noise levels. While the error bound helps in accounting for noise, it also introduces a degree of freedom that allows incorrect labels to be included. Surprisingly, the *false ratio*, which measures the percentage of incorrect labels in the result, actually goes down as the noise level increases; the noise intuitively takes up the slack introduced by error bound. This observation suggests that we might be able to automatically tune the error bound by estimating the noise level. Figure 7 shows the results on Dataset 5, which contains both ambiguity and noise. As we can see, the already low hit ratio of the ML algorithm drops further, essentially to zero, and the false ratio goes over 95%. Our algorithm performs consistently well in Figure 7, demonstrating its ability to handle ambiguity even in the presence of noise. Figures 8 and 9 summarize the experimental results on Datasets 3 and 4, which show the effect of unknowns. Intuitively, if the unknown ion is non-interfering, it acts like additional noise at some m/z positions, which makes it harder to be compensate for. The hit ratio of our algorithm drops sharply when the non-interfering unknown proportion exceeds the error bound. The spike in the *false ratio* at the very end is an artifact caused by the fact that the number of labels found is reduced to one essentially, and that one is incorrect. The effect of interfering unknowns is more interesting. While it raises the false ratio as more and more unknowns are added, as expected, surprisingly, it also helps the hit ratio (because it can be interpreted as some linear combination of known signatures that effectively increases the quantity of the knowns).

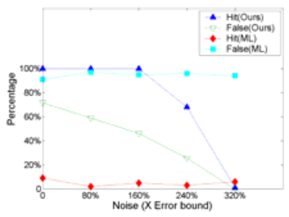


Figure 6: Effect of Noise without Ambiguity

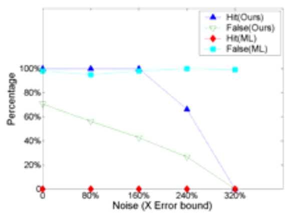


Figure 7: Effect of Noise with Ambiguity

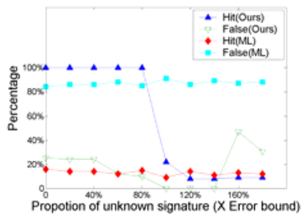


Figure 8: Effect of Non-interfering Unknown

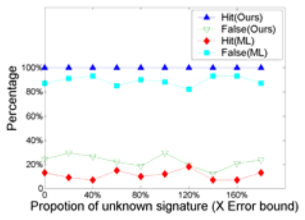


Figure 9: Effect of Interfering Unknown

7.3.2 Partial Labeling Tests

We run the exact same set of experiments as for the Full Labeling Test, but apply the second metric. The ten signatures of interest are set to be those used to generate the spectrum, so that ambiguity w.r.t. the signatures of interest is still under control. Figures 10 and 11 illustrate the effect of noise and unknowns combined with ambiguity. The figures only show *hit ratio*, since in our setting the *false ratio* is just *1- hit ratio*. In both graphs, the triangle series show the hit ratio of our algorithm and the square/diamond series represent the ML algorithm. Solid lines represent the results on datasets with no ambiguity while dotted lines represent a dataset with ambiguity. The first observation is that the ML algorithm achieves decent performance under this metric, although it is still uniformly dominated by the LP algorithm. The performance degradation of the ML algorithm from diamond curves to square curves in both graphs again shows the weakness of the ML approach, namely its *inability to handle ambiguity*. Both noise and unknowns have a similar effect on our algorithm as in the full labeling tests. On the other hand, the almost horizontal *hit ratio* curves for the ML algorithm illustrate an interesting point: *the ML algorithm tends to be less sensitive to unknowns than our algorithm*. This is because our algorithm assumes complete knowledge of ion signatures and tries to combine all signatures simultaneously, whereas the ML algorithm simply looks at one ion at a time.

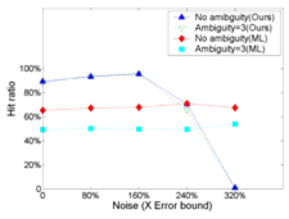


Figure 10: Effect of Noise

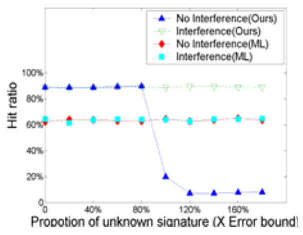


Figure 11: Effect of Unknowns

Overall, our algorithm clearly beats the ML algorithm in terms of labeling quality, even in partial labeling tests. In addition, the ML algorithm needs substantial training data. This is not realistic to get at all. However, the ML algorithm does show promise in partial labeling, which suggests a promising research direction, namely a hybrid algorithm that combines the speed of ML and the ambiguity-handling ability of our LP-based approach.

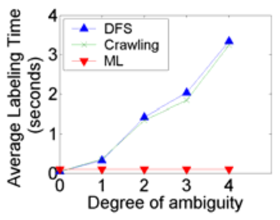


Figure 12: Effect of Ambiguity on Label Time

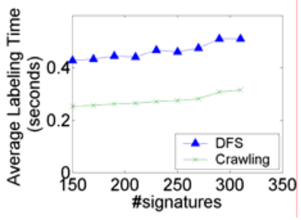


Figure 13: Scalability with Respect to # Signatures

7.4 Labeling Speed

We ran efficiency tests on the five datasets described in Section 7.2. Results show that the presence of noise and unknown signature does not affect the performance of our algorithms much, unless the noise or weight on the non-interfering unknown signature is significantly larger than the error bound. When no ambiguity is present, labeling takes about one second for both DFS and Crawling algorithms. However, as more ambiguity is included and the label set size increases sharply, the performance of our algorithms degrades significantly. Figure 12 shows the running time of our algorithms on Dataset 2, which contains five files of spectra with five different degrees of ambiguity. Series 1 and 2 show the performance of DFS and Crawling algorithms. The Crawling algorithm exploits the convexity of the distance function and runs slightly faster than DFS, but both become much slower as ambiguity is increased. This is mainly due to the dramatic increase in the number of correct labels. The ML algorithm is much faster than our algorithms, but it is worth noting that when no ambiguity is involved and the number of correct labels is small, the running time of our algorithm is almost the same as for ML. In addition, the training time of the ML approach is not reflected at all in these graphs. Further, when we are only interested in detecting a small number of signatures, we can revise our DFS algorithm to only pick the signatures of interest and do partial labeling. This optimization greatly speeds up DFS, to about 100 spectra per second. Figure 10 summarizes the results of algorithm scalability with respect to the number of signatures in the signature library.

We have not discussed efficiency issues in any depth, since this is not the focus of this paper. While some major performance results are included here for completeness, we address running time improvement and scalability issues in a separate paper [7].

8. CONCLUSION

Methods of categorizing aerosol particles using clustering and neural networks have been proposed [1,11,9,15], but none of them deals with the labeling problem directly. The linear programming method used in this paper is standard, see, e.g., [12]. Related nonlinear or integer programming tasks arising from the use of Euclidean distance or discretization are also well studied in the optimization community [6,12,18]. Recent work on knowledge-based optimization and machine learning [8,16] are promising extensions to the framework we propose. Machine learning methods such as clustering [5,19] can be applied to our basic linear programming approach by helping identify better initial points and optimization constraints.

Finally, while this paper addresses foundations and evaluates quality of labeling, in a related paper [7], we have studied how to scale our labeling algorithms to label a large group of spectra. There are several directions for future work:

- *Better labeling algorithms:* Can we use the algorithms presented here to label spectra, and use the results as training sets for machine learning algorithms? Can we adaptively "learn" important parameters such as the error bound? The overall objective is a fast hybrid algorithm that does not require manually labeled training data and that can adapt to changing input streams automatically.
- *Utilizing domain knowledge for labeling*: Scientists usually have a priori knowledge about spectra they collect, e.g., if ion A appears in one particle, then it is likely that ion B will also appear in the same particle. How can we let them specify this information and exploit it for better labeling? As another example, domain knowledge could be used to produce synthetic training data that closely mimics expected real-datasets, using the detailed generator that we described.
- *Discovering unknown signatures:* A limitation of our framework is that it does not distinguish between noise and unknown signatures, and the precision of labeling is influenced by the existence of unknowns. Can we discover unknowns?
- *Applying our techniques to diverse real datasets:* In this paper, we have abstracted many complications that arise in practice. This work is part of the EDAM project, an ongoing collaboration between chemists and computer scientists, and we plan to validate our techniques by applying them to real spectral datasets, with "ground truth" data collected simultaneously using filter-based techniques. This will also allow us to explore accurate calibration of quantity information.

9. REFERENCES

[1] Agrawal, R., Imielinski, T., Swami, A., Mining Associations between Sets of Items in Massive Databases, *Proc. ACM-SIGMOD*, 1993.

[2] Agrawal, R., Faloutsos, C. and Swami, A. Efficient Similarity Search in Sequence Databases. *Proc. Conf. on Foundations of Data Organization and Algorithms*, 1993.

[3] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. and Verkamo, A. I., Fast Discovery of Association Rules, *Advances in Knowledge Discovery and Data Mining*, 1995.

[4] Agrawal, R. and Srikant R.: Fast Algorithms for Mining Association Rules, *Proc. VLDB*, 1994

[5] Basu, Sugato, Banerjee, Arindam and Mooney, Raymond J., Semi-supervised Clustering by Seeding. *Proc. ICML,* 2002.

[6] Benson, Steven J., More, Jorge J, A Limited Memory Variable Metric Method, in *Subspaces and Bound Constrained Optimization Problems*, 2001.

[7] Chen, L., Huang, Z. and Ramakrishnan, R., Cost-Based Labeling of Groups of Spectra, *Proc. ACM-SIGMOD*, 2004.

[8] Fung, G., Mangasarian, O.L. and Shavlik, J., Knowledge-Based Support Vector Machine Classifiers. *Proc. NIPS 2002*.

[9] Gard, E., Mayer J.E., Morrical, B.D., Dienes, T., Fergenson, D.P. and Prather, K.A., Real-Time Analysis of Individual Atmospheric Aerosol Particles: Design and Performance of a Portable ATOFMS, *Anal. Chem.* 1997, 69, 4083-4091.

[10] McCarthy, J., Phenomenal data mining, *In Communications of the ACM 43(8)*, 2003

[11] Noble, C.A. and Prather K.A., Real-time Measurement of Correlated Size and Composition Profiles of Individual Atmospheric Aerosol Particles. *Environ. Sci. Technol,* 1996; 30, 2667-2680.

[12] Nocedal, J. and Wright, S.J., Numerical Optimization, Springer, 1st edition, 1999.

[13] Prather, K.A., Nordmeyer, T., and Salt, K. Real-time Characterization of Individual Aerosol Particles Using ATOFMS. *Anal. Chem.*, 1994; 66, 1403-1407.

[14] Srikant, R. and Agrawal, R., Mining Quantitative Association Rules in Large Relational Tables, *Proc ACM-SIGMOD*, 1996.

[15] Suess, D.T. and Prather K.A., Mass Spectrometry of Aerosols, *Chemical Reviews*, 1999, 99, 3007-3035.

[16] Towell, G.G. and Shavlik J., Knowledge-Based Artificial Neural Networks. *Artificial Intelligence*, 70, 1994.

[17] Witten, Ian H. and Frank, Eibe, Practical Machine Learning Tools and Techniques with Java Implementation, *Morgan Kaufmann*, 1999.

[18] Wolsey, L., Integer Programming, John Wiley, 1998.

[19] Zhang, T., Ramakrishnan, R. and Livny M., BIRCH: An Efficient Data Clustering Method for Very Large Databases, *Proc. ACM-SIGMOD*, 1996.

Appendix A

THEOREM Consider signature library $A=[\vec{s}_1,\vec{s}_2,...,\vec{s}_n]$ and a spectrum $\vec{b}$ where $\vec{s}_1,\vec{s}_2,...,\vec{s}_n$ are linearly independent (i.e., there is no vector $\vec{a}=[a_1,a_2,...,a_n]$ such that $\sum_{i=1}^{n} a_i\vec{s}_i$ and at least one $a_i \neq 0$). Then, either $\vec{b}$ has the unique labeling property w.r.t. A, or the system of equations

$$A\vec{x}=\vec{b}\ ,\ \vec{x}\geq 0 \qquad (1)$$

has no solution.

PROOF If $\vec{b}$ cannot be uniquely labeled and equation (1) has a solution, there must exist two vectors $\vec{x}_0$ and $\vec{x}_1$ such that

$$A\vec{x}_0 = \vec{b}\ , \vec{x}_0 \geq 0 \quad (2)$$

$$A\vec{x}_1 = \vec{b}\ , \vec{x}_1 \geq 0 \quad (3)$$

$$\vec{x}_1 \neq \vec{x}_0 \quad (4)$$

Subtract (3) from (2), we have $A(\vec{x}_0 - \vec{x}_1) = 0$. Since $\vec{x}_1 \neq \vec{x}_0$, vector $(\vec{x}_0 - \vec{x}_1) \neq 0$. It violates the precondition that $\vec{s}_1, \vec{s}_2, ..., \vec{s}_n$ are linearly independent. Thereby, $\vec{b}$ must have the unique labeling property or equation (1) does not have a solution.

Appendix B

THEOREM: Consider signature library $A = [\vec{s}_1, \vec{s}_2, ..., \vec{s}_n]$ and a spectrum $\vec{b}$ where $\vec{s}_1, \vec{s}_2, ..., \vec{s}_n$ are *not* linearly independent. If there is a solution $\vec{a} = [a_1, a_2, ..., a_n]$ to $A\vec{x} = \vec{b}$, $\vec{x} \geq 0$ such that $min_{i=1,2,...,n} a_i > 0$, then $\vec{b}$ has infinitely many labels.

PROOF Since $\vec{s}_1, \vec{s}_2, ..., \vec{s}_n$ are *not* linearly independent, by the definition, there exists a vector $\vec{a} = [a_1, ..., a_m]$, such that $A\vec{a} = 0$. Let constant h be $\dfrac{Min(x_i)}{Max(|a_i|)}$. Then for any constant f, 0<f<h, the vector $f\vec{a} + \vec{x} \geq 0$. In addition, $A(f\vec{a} + \vec{x}) = fA\vec{a} + A\vec{x} = 0 + \vec{b} = \vec{b}$. So, for any f, 0<f<h, $f\vec{a} + \vec{x}$ also satisfies equation $A\vec{x} = \vec{b}$, $\vec{x} \geq 0$ and thus is a possible labeling for the input spectrum too. Given a h>0, the number of f's is infinite, thereby the number of possible labels for spectrum $\vec{b}$ is also infinite.

Appendix C

A linear programming algorithm to test the unique labeling property of spectrum $\vec{b}$

```
Input:
    Signature database matrix A.  A : $n \times m$
    Spectrum $\vec{b}$ . $\vec{b}$ is a vector of length n
Output:
    True, if $\vec{b}$ has the unique labeling property with respect to A
    False, if $\vec{b}$ does not have the unique labeling property with respect to A

Procedure:
    Seek $\vec{a}$ s.t.  A $\vec{a}$=$\vec{b}$ , $\vec{a} \geq 0$          (1)
    $\vec{a}^*$ is the feasible solution obtained in (1)
    Define    $J = \{j \mid \vec{a}^*[j] > 0\}$
              $J^C = \{j \mid \vec{a}^*[j] = 0\}$
    For k = 1 to n
```

```
        Seek $\vec{d}$ s.t.   A $\vec{d}$ =0                    (2)
                   $\vec{d}[j] \geq 0 \ if \ \ j \in J^C$
                   $\vec{d}[k] = 1$
        If (2) succeeds
            Return False

    For each $k \in J$
        Seek $\vec{d}$ s.t.   A $\vec{d}$ =0                    (3)
                   $\vec{d}[j] \geq 0 \ if \ \ j \in J^C$
                   $\vec{d}[k] = -1$
        If (3) succeeds
            Return False

    Return True

End of Procedure
```

Note: (1), (2), (3) will call a standard linear programming procedure.

Proof:

Necessary condition

Suppose the algorithm returns True, but $\vec{b}$ does not have the unique labeling property with respect to A. There must exist a $\vec{e}$ s.t.

A $\vec{e} = \vec{b}$, $\vec{e} \geq 0$

$\vec{e} \neq \vec{a}^*$

Define $\vec{h} = \vec{e} - \vec{a}^*$

$$J = \{j \mid \vec{a}^*[j] > 0\}$$

$$J^C = \{j \mid \vec{a}^*[j] = 0\}$$

Since $\vec{e} \geq 0$, $\vec{h}[j] \geq 0 \ if \ \ j \in J^C$

Since $\vec{e} \neq \vec{a}^*$, there must be a k s.t. $\vec{h}[k] \neq 0$.

If $\vec{h}[k] > 0$, $\frac{1}{\vec{h}[k]}\vec{h}$ is a feasible solution for (2)

The program will return False.

Otherwise, $\vec{h}[k] < 0$, $\frac{1}{\vec{h}[k]}\vec{h}$ is a feasible solution for (3)

The program will return False too.

Therefore, $\vec{b}$ must have the unique labeling property.

Sufficient condition

Suppose the algorithm returns False, the linear programming in (2) or (3) must succeed.

If (2) succeeds, the feasible solution is $\vec{d}_1$.

Let $\alpha = \dfrac{Min(|\vec{a}^*[i]|)}{Max(|\vec{d}_1[i]|)}$

Both $\alpha\ \vec{d}_1 + \vec{a}^*$ and $\vec{a}^*$ are solutions to the equation $A\vec{x} = \vec{b}$, $\vec{x} \geq 0$ and $\alpha\ \vec{d}_1 \neq \vec{a}^*$.

Otherwise, (3) succeeds, the feasible solution is $\vec{d}_2$.

Let $\alpha = \dfrac{Min(|\vec{a}^*[i]|)}{Max(|\vec{d}_2[i]|)}$

Both $\alpha\ \vec{d}_2 + \vec{a}^*$ and $\vec{a}^*$ are solutions to the equation $A\vec{x} = \vec{b}$, $\vec{x} \geq 0$ and $\alpha\ \vec{d}_2 \neq \vec{a}^*$.

Therefore, $\vec{b}$ does not have the unique labeling property.

Appendix D

Given a set of points $S \subseteq \Re^r$, the affine subspace generated by *S* is defined as

$$as(S) = \{\sum_{i=1}^{m} w_i s_i \mid m \geq 1, \sum_{i=1}^{m} w_i = 1, w_i \in R, s_i \in S, 1 \leq i \leq m\}$$

The convex hull generated by *S* is defined as

$$ch(S) = \{\sum_{i=1}^{m} w_i s_i \mid m \geq 1, \sum_{i=1}^{m} w_i = 1, w_i \geq 0, s_i \in S, 1 \leq i \leq m\}$$

Such an expression $\sum_{i=1}^{n} w_i s_i$ with the constraints as in the definition of $ch(S)$ is called a convex combination. Clearly $ch(S) \subset as(S)$, but has the same dimension.

We will assume our set $S = \{s_0, s_1, \ldots, s_n\}$ is finite, and all s_i lie on the hyperplane $\prod : \eta^{\mathrm{T}} s = 1$, ,where $\eta^T = (1,\ldots,1) \in \Re^r$. Then $as(S)$, in particular, $ch(S)$, all lie on this hyperplane $\prod$. Hence $d = dim(as(S)) \leq r - 1$. We denote by *A* the $r \times (n + 1)$ matrix $[s_0, s_1, \ldots, s_n]$.

Define a set $\{s_0, \ldots, s_k\}$ to be affine independent, if whenever $\sum_{i=1}^{k} w_i s_i = 0$, with $\sum_{i=1}^{k} w_i = 0$ then all $wi = 0$.

CLAIM 1 *$\{s_0, s_1, . . . , s_l\}$ is affine independent iff $\{s_1 - s_0, . . . , s_k - s_0\}$ is linearly independent.*

CLAIM 2 *If $\{s0, s1, . . . , sk\} \subset \Pi$, then it is linearly independent iff it is affine independent.*

We show affine independence implies linear independence for $S \subset \Pi$. Suppose $\sum_{i=1}^{k} w_i s_i = 0$, then $\sum_{i=1}^{k} w_i = \eta^T(\sum_{i=0}^{k} w_i s_i) = 0$, and thus all $w_i = 0$.

$ch(S)$ is a convex polytope of dimension d. As such it is endowed with the Euclidean metric and topology of as(S). We denote by X^0 the interior points of X.

LEMMA 1 *If $x \in$ ch(S)o then x has a convex combination $\sum_{i=1}^{k} w_i s_i$ where all $w_i > 0$.*

Consider any s_i. We first obtain a convex combination for x in which the coefficient of s_i is positive. If $x = s_i$ then this is trivial. Suppose $x \neq s_i$. Let the 1-dimensional ray from s_i to x intersect the boundary of ch(S) at y. Then clearly x lies strictly between *si* and y. So, $x = \alpha s_i + (1-\alpha)y$, where $\alpha > 0$. Now substitute y by its convex combination from S, we get a convex combination of x where the coefficient of s_i is positive. To complete the proof of the lemma we simply take an average of such expressions, one for each s_i.

LEMMA 2 *If $x = \sum_{i=1}^{k} w_i s_i$, where $\{ s_i \mid w_i > 0 \}$ has rank $d + 1$, then $x \in ch(S)^o$*. Wolog, $\{s_i, s_1 . . . , s_d\}$ has rank d + 1, with $w_0, w_1, . . . , w_d > 0$. Then affine independent and $\{s_0, s_1 . . . , s_d\}$ spans a simplex Δ. Let $\alpha = \sum_{i=0}^{d} w_i$. If $\alpha = 1$ (i.e., there are no other positive coefficients in $x = \sum_{i=1}^{k} w_i s_i$) then clearly $x \in \Delta^o \subseteq ch(S)^o$. If $\alpha < 1$, then x can be expressed as a convex sum

$$x = \alpha x' + (1-\alpha)x''$$

where $x' = \sum_{i=0}^{d} (w_i/\alpha)s_i \in \Delta^o$, and $x'' = \sum_{i=d+1}^{n} (w_i/(1-\alpha))s_i \in ch(S)$. By varying x' within the open set Δ^o, we see clearly that $x \in ch(S)^o$

Corollary 1 *A point $x \in ch(S)$ is an interior point iff it has a convex combination where all coefficients are positive.*

It is perhaps tempting to think that every interior point x always has a "pure" convex combination where exactly d $+1$ coefficients are positive corresponding to some $d+1$ affine independent s_i. This is not true. An example is the d-dimensional octahedron, the convex hull spanned by the 2d unit vectors $\pm e_i$, $1 \leq i \leq d$. Consider the point 0. Clearly, by taking inner product with e_i, any convex combination of 0 must be of the form $\sum_i (w_i e_i + w_i(-e_i))$. Thus if 0 were to be in the interior of any simplex of dimension > 1, then at least two pair, $\pm e_i$ and $\pm e_j$, have $w_i, w_j > 0$. But these 4 vectors are already affine dependent. Another caution we observe from this example is that even though there are only *2d* vertices, the octahedron as its convex hull has *2d* facets. Thus it is in general infeasible to enumerate all the facets of $ch(S)$ computationally.

LEMMA 3 *If S is affine independent, then every point in ch(S) has a unique convex combination. If S is affine dependent, then every interior point of ch(S) has infinitely many convex combinations.*

PROOF The first claim is obvious: If S is affine independent, and $x=\sum_{i=1} w_i s_i = \sum_{i=1} w'_i s_i$, then $\sum_i (w_i - w'_i) s_i = 0$, with $\sum_i (w_i - w'_i) = 0$. Therefore by affine independence, all $w_i = w'_i$ Now suppose S is affine dependent. There exists some $\sum_i w_i s_i = 0$, with $\sum_i w_i = 0$, for w_i not all zero. We may assume all $|w_i| < \varepsilon$ for sufficiently small $\varepsilon > 0$. Now add this sum to a convex combination of x with all positive coefficients. The result follows. So assuming S is affine dependent, which necessarily is the case for $n \geq r$, the only possible case for unique convex combination is on the boundary of $ch(S)$.

Consider the following algorithm: Given any b. Solve for the feasibility of the linear programming: Find $w = (w_0, w_1, \ldots, w_n)^T$, such that $Aw = b, \sum_{i=0}^{n} w_i = 1, w_i \geq 0$. This is solvable for w iff $b \in ch(S)$. Suppose $b = \sum_{i=0}^{n} w_i s_i$ is one such solution. Let $B = \{ s_i \mid w_i > 0 \}$. If B is affine dependent, then return "non-unique". Suppose B is affine independent. Let P be the affine subspace generated by B. Let π be a linear projection $\Re^d$ along P, so that P is projected to a single point, and $\dim(\pi(\Re^d)) = d - \dim(P)$. We may assume $\pi(P) = 0$, since we are dealing with convex combinations in affine geometry. Let $C = \text{ch}(S - B)$, and $\underline{C} = \text{ch}(\pi(S - B))$. Clearly $\underline{C} = \pi(C)$. Now by solving a linear programming problem we can determine if 0 (= $\pi(b)$) is in C. If so, return "non-unique", otherwise return "unique".

THEOREM 1 *For any $b \in ch(S)$, the above algorithm solves the problem of whether b has a unique convex combination in S.*

PROOF Suppose $b \in ch(S)$. If B is affine dependent, then on P, b is an interior point of *ch(B)* by Corollary 1, and therefore by Lemma 3, b has non-unique convex combinations.

Now we assume B is affine independent, $\dim(P) = |B| - 1$, and b is an interior point of the simplex $ch(B)$.

We have a convex combination of b over $B: b = \sum_{s_i \in B} w_i s_i$. If b does not have a unique convex combination from S, then there is another expression $b = \sum_i w'_i s_i$. Since B is affine independent, the non-zero coefficients among w'_i can not be all from $\{i \mid s_i \in B\}$. Let some $w'_i > 0$ for $s_j \notin B$. Under the projection π, all terms corresponding to $s_i \in B$ disappeared, we get $0 = \pi(b) = \sum_\ell w'_\ell \pi(s_\ell)$, where the second sum involves only terms not in B, but does contain $w'_j \pi(s_j)$ with non-zero w'_j. By dividing out the sum of these coefficients $\sum_\ell w'_\ell > 0$, we get a convex combination of $0 = \pi(b)$, i.e., $0 \in \underline{C}$.

Conversely assume $0 = \pi(b) \in \underline{C}$. Let $0 = \sum_\ell w'_\ell \pi(s_\ell)$ be a convex combination, where all terms $s_\ell \notin B$. Now we can lift this back to $\Re^d$ to get some convex combination for a certain point $x \in P$,

$$x = \sum_\ell w_i s_i$$

where the sum is over all terms $s \notin B$. Now if $x = b$ then we are done, having obtained a different convex combination for b than $b = \sum_i w_i s_i$ among $s_i \in B$. (In fact a weighted sum gives infinitely many different convex combinations.) Suppose $x \neq b$. Draw a ray from x toward b, which intersects at y with the boundary of the simplex generated by B. This y has a convex combination among vectors from B, and b can be expressed as a weighted sum of x and y. Using the convex combination of x above with $s_\ell \neq B$, we obtain another convex combination for b where not all positive coefficients are among B. This is different than $b = \sum_i w_i s_i$ among $s_i \in B$ (and hence, by any weighted average we get infinitely many different convex combinations.)

The theorem is proved.

We note that if the points on the boundary of *ch(S)* are all on general position (i.e., no two coincide, no three colinear, . . ., in general any $k + 1$ points have affine rank k) then every point on the boundary has unique convex combination.

Appendix E

LEMMA 1 *Suppose we have a convex set K contained in the unit cube* $Q = [0,1]^n$. *If* Q *is divided along every dimension into half, obtaining* 2^n *(closed) subcubes. Suppose K intersects at most n of these subcubes. Then there exists a dimension i,* $1 \leq i \leq n$*, where K is either contained in the first half:* $0 \leq x_i \leq 1/2$*, or contained in the second half:* $1/2 \leq x_i \leq 1$*.*

This statement can be generalized. A straightforward induction proves the following.

THEOREM *Suppose we have a convex set K contained in some "brick"* $B = [L1, U1] \times [L2, U2] \times \cdots \times [Ln, Un]$*. Assume B is divided into* r_i *ranges along each dimension i,* $L_i = a_{i,0} < a_{i,1} < a_{i,2} < \cdots < a_{i,ri} = U_i$*, and B is the union of* $\prod_{i=1}^{n} r_i$ *many "subbricks" of the form* $[a_{1,j1}, a_{1,j1+1}] \times [a_{2,j2}, a_{2,j2+1}] \times \cdots \times [a_n,j_n, a_n,j_{n+1}]$*. Suppose K has only non-empty intersection with at most n of these subbricks, then there exists a dimension i,* $1 \leq i \leq n$*, and a "slice"* $\{ x \in B \mid xi \in [a_{i,ji}, a_{i,ji+1}] \}$ *which contains K.*

Thus, in our context, if the convex set of solutions K does not intersect many subranges, then the problem can be effectively reduced in dimension, until the number of relevant subranges which do intersect K.

Now we turn to the proof of the Lemma. First we observe that it is crucial that we speak of "closed" subcubes. If we were to consider "open" subcubes, then there is the counter example of the diagonal straight line segment as our set *K*. Also the number *n* is optimal, as demonstrated by the example of a simplex with vertices at *0* and at v_i, $1 \leq i \leq n$, where v_i has coordinates 3/4 in dimension *i* and *0* elsewhere.

PROOF We prove the Lemma by induction on *n*. The base case $n = 1$ is trivial.

For any binary string $b \in \{0,1\}^n$, we denote by Q_b^n the subcube where $0 \leq x_i \leq 1/2$ if $b_i = 0$ and $1/2 \leq xi \leq 1$ if $b_i = 1$. By assumption $\{b \mid K \cap Q_b^n \neq \varnothing\}$ has cardinality at most *n*.

Consider the orthogonal projection $\pi : \Re^n \to \Re^{n-1}$, where $\pi(x_1,...,x_{n-1},x_n) = (x_1,...,x_{n-1})$. Let $K' = \pi(K)$. Among the $2^n - 1$ subcubes of $[0, 1]^{n-1}$ of the form $Q_{b'}^{n-1}$, where $b' \in \{0,1\}^{n-1}$, if K' only intersects at most $n-1$ of them, then by induction, there exists a dimension $1 \leq i \leq n-1$, such that K' is contained in either the first half or the second half along that dimension. Then clearly, in $\Re^n$, *K* is also contained in either the first or the second half along that dimension.

Therefore we may assume that there are n distinct subcubes $Q_{b_1'}^{n-1}, Q_{b_2'}^{n-1}, ..., Q_{b_n'}^{n-1}$, with which K' has a non-empty intersection. Let $p'_j \in K' \cap Q_{b_j'}^{n-1}$, for $1 \leq j \leq n$. (Note that p'_j's need not be distinct.) Let $p_j \in K$ such that $p'_j = \pi(p_j)$. We write $p_j = (p'_j, t_j)$, where $p'_j \in \{0,1\}^{n-1}$ and $0 \leq t_j \leq 1$.

If $t_j \leq 1/2$, then *K* has a non-empty intersection with the subcube $Q_{b_j}^n$, where $b_j = b'_j 0$. Similarly if $t_j \geq 1/2$ then K has a non-empty intersection with the subcube $Q_{b_j}^n$, where $b_j = b'_j 1$. We have obtained at least n subcubes $Q_{b_j}^n$, each has a non-empty intersection with *K*. Note that these *n* binary strings $b_1, . . . , b_n$ are all distinct (because $b'_1,...,b'_n$ are all distinct, even though the points $p'_1,...,p'_n$ need not be distinct.)

(If some $t_j = 1/2$, then we have actually obtained $n+1$ subcubes, and the proof is finished. So we may assume none of $t_j = 1/2$ and each b_j is unambiguously defined.)

Assume K is not contained in the first half or the second half along dimension n, then it can ot be the case that all $t_j \le 1/2$ or all $t_j \ge 1/2$. It follows that there are some $1 \le j \ne k \le n$, such that $t_j > 1/2$ and $t_k < 1/2$. In particular, $p_j \ne p_k$.

Consider the line segment from p_j to p_k. There exists a unique point p along this line segment where the n-th coordinate is exactly 1/2. By convexity of $K, p \in K$. Write $p = (x_1, x_2, \ldots, x_{n-1}, 1/2)$.

Define $b' = b[1 \ldots n-1] \in [0,1]^{n-1}$ as follows: For $1 \le j \le n - 1$, if $x_j \le 1/2$ then $b[j] = 0$, and otherwise $b[j] = 1$. Then let $b = b'0$ and $b^* = b'1$. Clearly $K \cap Q_b^n \ne \varnothing$ and $K \cap Q_{b^*}^n \ne \varnothing$, because both contain p.

We claim at most one of Q_b^n or $Q_{b^*}^n$ can be among the previous n subcubes $Q_{b_j}^n$, $1 \le j \le n$. This is because both Q_b^n and $Q_{b^*}^n$ project under π to the same $Q_{b'}^{n-1}$ and yet all n subcubes $Q_{b_j}^n$ project to distinct $n - 1$ dimensional subcubes $Q_{b'_j}^{n-1}$.