Amino acid sequence assignment from single molecule peptide sequencing data using a two-stage classifier

Matthew Beauregard Smith[1], Zack Booth Simpson[2], Edward M. Marcotte[3]

[1]Oden Institute, The University of Texas at Austin, Austin, TX 78712

[2]Erisyon Inc., Austin, TX

[3]Department of Molecular Biosciences, The University of Texas at Austin, Austin, TX 78712

Correspondence: mbsmith93@utexas.edu, zack@erisyon.com, marcotte@utexas.edu

Abstract

We present a machine learning-based interpretive framework (*whatprot*) for analyzing single molecule protein sequencing data produced by fluorosequencing, a recently developed proteomics technology that determines sparse amino acid sequences for many individual peptide molecules in a highly parallelized fashion [1] [2]. Whatprot uses Hidden Markov Models (HMMs) to represent the states of each peptide undergoing the various chemical processes during fluorosequencing, and applies these in a Bayesian classifier, in combination with pre-filtering by a k-Nearest Neighbors (kNN) classifier trained on large volumes of simulated fluorosequencing data. We have found that by combining the HMM based Bayesian classifier with the kNN pre-filter, we are able to retain the benefits of both, achieving both tractable runtimes and acceptable precision and recall for identifying peptides and their parent proteins from complex mixtures, outperforming the capabilities of either classifier on its own. Whatprot's hybrid kNN-HMM approach enables the efficient interpretation of fluorosequencing data using a full proteome reference database and should now also enable improved sequencing error rate estimates.

Introduction

Proteins are key components of living organisms, but their heterogenous chemical natures often complicate their biochemical analyses, and consequently, the state of protein identification and quantification methods (e. g., mass spectrometry, antibodies, affinity assays) has generally tended to lag the remarkable progress exhibited by DNA and RNA sequencing technologies. However, improvements to protein analyses could potentially directly inform better biological understanding and better translate into biomedicine and clinical studies. Thus, the field of single molecule protein sequencing attempts to apply concepts from DNA and RNA sequencing to protein analyses in order to take advantage of the high parallelism, sensitivity, and throughput potentially offered by these approaches [3] [4] [5] [6].

Fluorosequencing is one such single-molecule protein sequencing technique inspired by methods used for DNA and RNA [1] [2]. In fluorosequencing, proteins in a biological sample are denatured and cleaved enzymatically into peptides. The researcher then chemically labels specific amino acid types, or alternatively, specific post-translational modifications (PTMs), within each peptide with different fluorescent dyes, then covalently attaches the peptides by their C-termini to the surface of a single-molecule microscope imaging flow-cell (**Figure 1A**). Sequencing proceeds by alternating between acquiring fluorescence microscopy images of the immobilized peptides and performing chemical removal of the N-terminal-most amino acid from each peptide, using the classic Edman degradation chemistry [7] [8] (**Figure 1B**). In this manner, the sequencing cycle (corresponding to amino acid position) at which different fluorescent dyes are removed is measured on a molecule-by-molecule basis, with these data collected in parallel for all the peptide molecules observed in the experiment (**Figure 1C**).

In theory, this process gives a direct readout of each peptide's amino acid sequence, at least for the subset of labeled amino acids (**Figure 1D**), but in practice there are several complications because of the

single-molecule nature of this sequencing method. Single molecule fluorescence intensities are intrinsically noisy, arising from the repeated stochastic transitions of each individual dye molecule between ground state and excited state, making stoichiometric data inexact, particularly when there are large fluorophore counts. Typically, no more than 5-6 copies of the same amino acid, hence dye, are expected for average proteolytic peptide lengths, with the number of distinct colors (*i.e.,* fluorescent channels) set by the microscopy optics and available dyes, here assumed to be 5 or fewer. However, inevitably with any chemical process, some fluorophore labeling reactions fail to occur, and

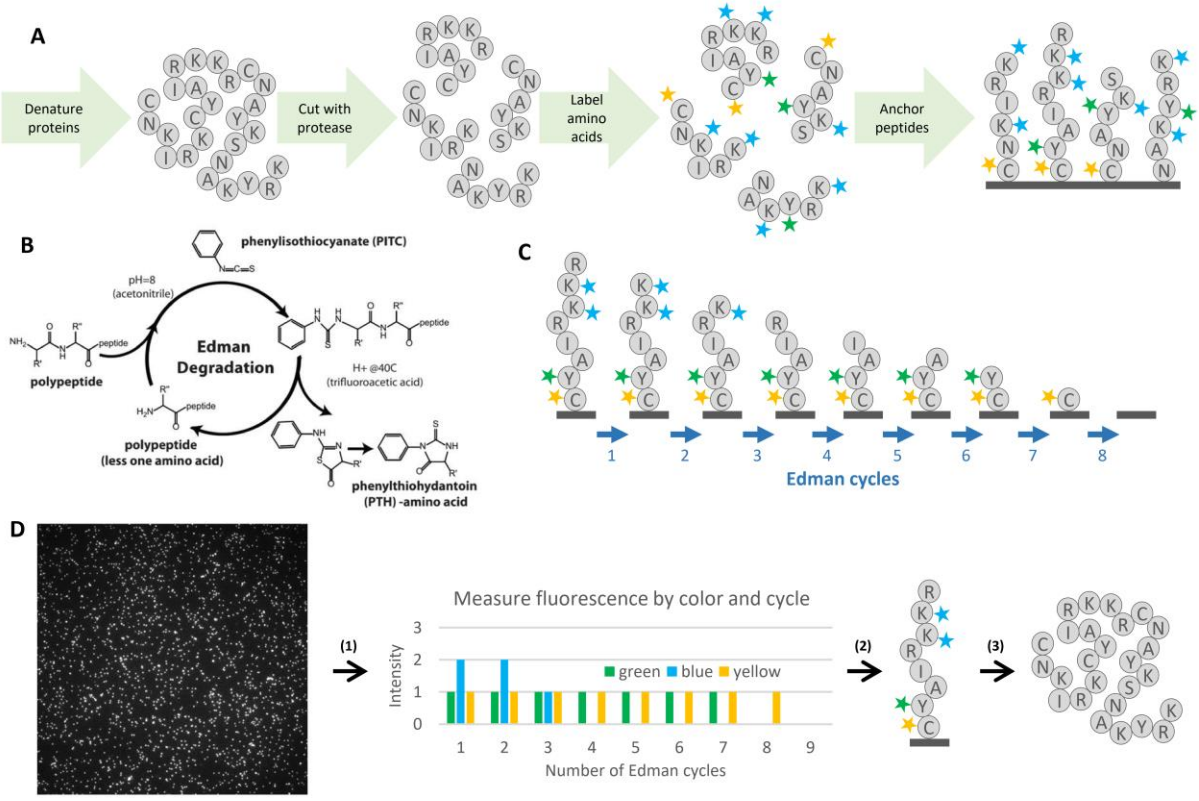

Figure 1. Overview of protein fluorosequencing. (**A)** illustration of the sample preparation process. Each grey circle represents an amino acid, and the letter in the circle corresponds to the standardized single letter amino acid codes. In the diagram, proteins are denatured, cleaved with protease, labeled with fluorescent dyes, and then labeled peptides are immobilized by their C-termini on the surface of a flow-cell. (**B**) The Edman degradation chemical reaction cycle, used to predictably remove one amino acid per cycle from each peptide. (**C)** For a given peptide, the sequencing process removes amino acids one at a time from the N-terminus, taking with them any attached fluorescent dyes. **D**: Major steps in computational data analysis include: **(1)** For each field of view, performing image analysis to extract fluorescence intensities for each spot (peptide) in each fluorescent channel across time steps (cycles), collating the fluorescence intensity data per spot across timesteps and colors. A vector of fluorescence intensities is produced, giving a floating-point value for every timestep and fluorophore color combination. **(2)** These raw sequencing intensity vectors (raw reads) must then be classified as particular peptides from a reference database. This step is the primary concern of this paper. **(3)**

64 photobleaching or chemical destruction can destroy fluorophores in the middle of a fluorosequencing
65 run. At some low rate, peptides may detach from the flow-cell during sequencing, and Edman
66 degradation can skip a cycle. These error rates, while individually small (approximately 5% each in
67 published analyses [2]), collectively add difficulty to peptide identification, necessitating computational
68 methods to process these data.

69 Currently, there are no published algorithms for mapping fluorosequencing reads to a reference
70 proteome to identify the proteins in a sample. The first analyses of fluorosequencing data used Monte
71 Carlo simulations to generate realistic simulated data as a guide for data interpretation and fitting of
72 experimental error rates [1] [2]. While this strategy did not scale well computationally to full proteomes,
73 it suggested that probabilistic modeling of the fluorosequencing process could provide a powerful
74 strategy for interpreting these data. In this paper, we explore the application of machine learning to
75 develop a classifier that correctly accounts for the characteristic fluorosequencing errors but is
76 computationally efficient enough to scale to the full human proteome.

77 Viewing this as a machine learning problem is challenging due to the large numbers of possible peptides
78 in many biological experiments. For example, in the human proteome, there are about 20,000 proteins,
79 which when processed with an amino-acid specific protease such as trypsin can correspond to hundreds
80 of thousands or even millions of distinct peptides, each of which can potentially vary due to post-
81 translational modifications or experiment-specific processing. This puts fluorosequencing data analysis
82 squarely in the realm of Extreme Classification problems, which are known to be challenging to handle in
83 practice [9].

84 To analyze these data, we took advantage of the ability to generate simulated fluorosequencing data
85 using Monte Carlo simulations [1] [2] to test k-Nearest Neighbors (kNN) classification and found it gave
86 results of poor quality but is able to scale efficiently to the full human proteome while maintaining

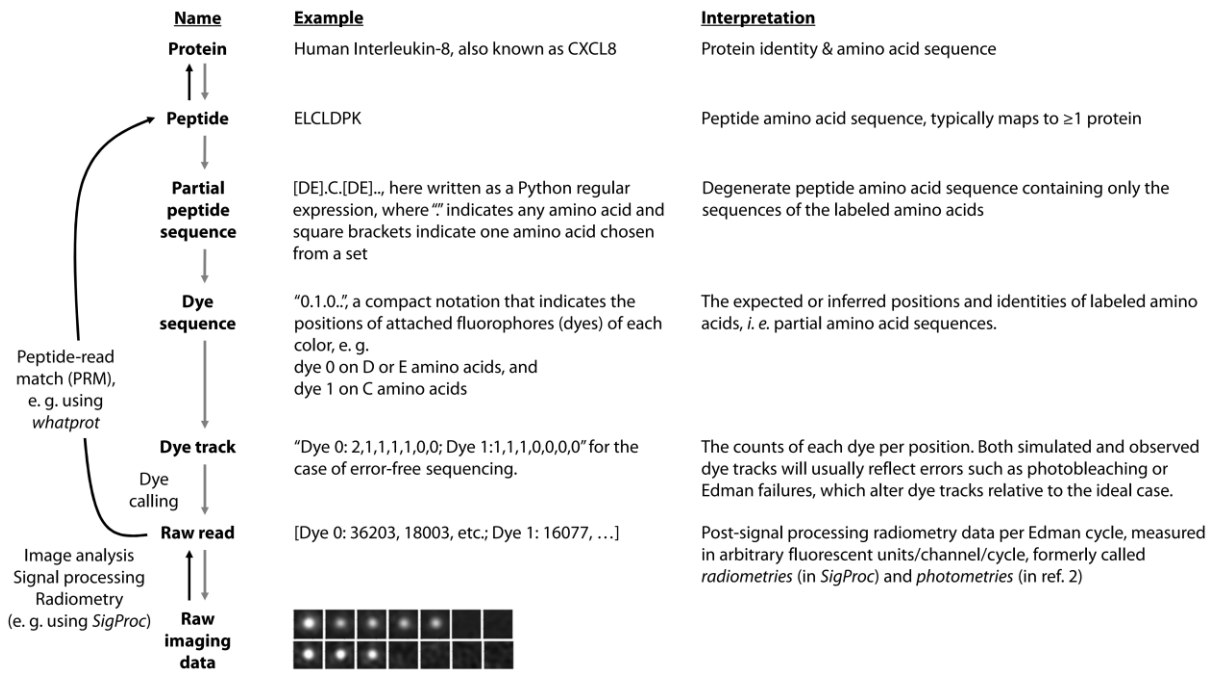

Figure 2. Nomenclature for different stages of fluorosequencing data analysis. The *whatprot* algorithm maps raw single-molecule protein sequencing reads to peptides and their parent proteins in the reference proteome (black arrows) by comparing experimental data (at bottom) to synthetic data generated using a Monte Carlo simulation (gray arrows).

reasonable runtimes. These initial explorations motivated the developments presented in this manuscript, which focuses on the specific challenge of matching fluorosequencing reads to peptides from a reference proteome (*peptide-read matching*).

Here, we propose a specialized classifier which combines heavily optimized Hidden Markov Models (HMMs) to model the peptide chemical transformations during fluorosequencing, in combination with kNN pre-classification to reduce runtime. We call this tool *whatprot*, compare it with kNN and a classifier which uses HMMs without the kNN based runtime reduction, and demonstrate that the hybrid HMM-kNN approach offers a powerful and scalable approach for interpreting protein fluorosequencing data with the use of a reference proteome.

Methods

Monte Carlo simulation

To generate training and testing data typical of fluorosequencing experiments, we performed Monte

Carlo simulations based on the model and parameters described in [1] [2]. These parameters are the dye loss rates p_c, which differ for each color c, the missing fluorophore rates m_c, the Edman cycle failure rate e, the peptide detachment rate d, the average fluorophore intensity μ_c, and the standard deviation of fluorophore intensity σ_c. We additionally model a background standard deviation σ_c'. Based on prior estimates for the dye Atto647N ([1] [2]), we used the following values unless otherwise noted: $p_c = .05$, $m_c = .07$, $e = .06$, $d = .05$, $\mu_c = 1.0$ (arbitrary rescaling of intensity values), $\sigma_c = 0.16$, $\sigma_c' = .00667$. Although the code permits different values for different colors c, for our simulations, we modeled each color of fluorophore with identical error values for simplicity.

An overview of the process with definitions for key terms is provided in **Figure 2**. We generate simulated data in two formats. The first of these formats we refer to as a *dye track,* and it indicates the number of remaining fluorophores of each color at each time step after considering sequencing errors. Thus, each copy of one particular peptide sequence may give rise to a different specific dye track in a sequencing experiment depending on the details of the labeling schemes and sequencing efficiencies. To simulate a dye track, we randomly alter (with a pseudo random number generator) a representation of a dye sequence in a series of timesteps, writing to memory the count of each color of fluorophore as we progress until we reach a pre-set number of timesteps. In this simulation, we initially remove fluorophores with a probability of m_c before beginning sequencing. We then additionally perform a series of random events after logging fluorophore counts for each timestep: we remove the entire peptide and all fluorophores with a probability of d to simulate peptide detachment from the flow cell, we remove the last amino acid (and any attached fluorophore) with a probability of $(1 - e)$, and we remove each fluorophore with a probability of p_c, where c is the color of the fluorophore, to simulate fluorophore destruction. Each fluorophore count is stored as a two-byte numeric value.

The other format of data we consider is a *raw read*, which consists of radiometry data for each fluorescent color and Edman cycle. Raw reads result experimentally from signal processing and

radiometry of the microscope imaging data from a fluorosequencing experiment. To simulate a raw read, we first simulate a dye track, and then we convert each fluorophore count into a double-precision floating point value indicating the fluorescent intensity. When we have a dye track entry indicating Λ_c fluorophores for a given fluorophore color c, we sample a normal distribution with a mean of $\Lambda_c \mu_c$ and a variance of $\sigma_c'^2 + \Lambda_c \sigma^2$. We perform this calculation for each channel at each time-step to simulate a raw read.

These radiometry *raw reads* simulate the fluorescent intensity data we would expect to collect from processing raw single molecule microscope images, a process currently performed for experimental data using the algorithm *SigProc* (Part of Erisyon's tool *Plaster*, https://github.com/erisyon/plaster_v1), as in [10] [11].

Bayesian classification with HMMs

Whatprot builds an independent HMM for each peptide in a provided reference proteome dataset. Each state in this HMM represents a potential condition of the peptide, including the number of successfully removed amino acids, and the combination of fluorophores which have not yet photobleached or been destroyed by the chemical processing (**Figure 3**). Transition probabilities between these states can be approximated using previously estimated success and failure rates of each step of protein fluorosequencing.

We can use the HMM forward algorithm to associate a specific peptide to each *raw read* (a series of observed fluorescence intensities over time and across different fluorescence channels). We obtain the probability of the peptide given the raw read in two steps. First, we compute the HMM forward algorithm using each possible peptide in the dataset to obtain the probability of the raw read given each peptide. This uses the forward algorithm formula

$$\boldsymbol{f}^{(t+1)} = \boldsymbol{O}^{(t+1)} \boldsymbol{T} \boldsymbol{f}^{(t)} \quad (1)$$

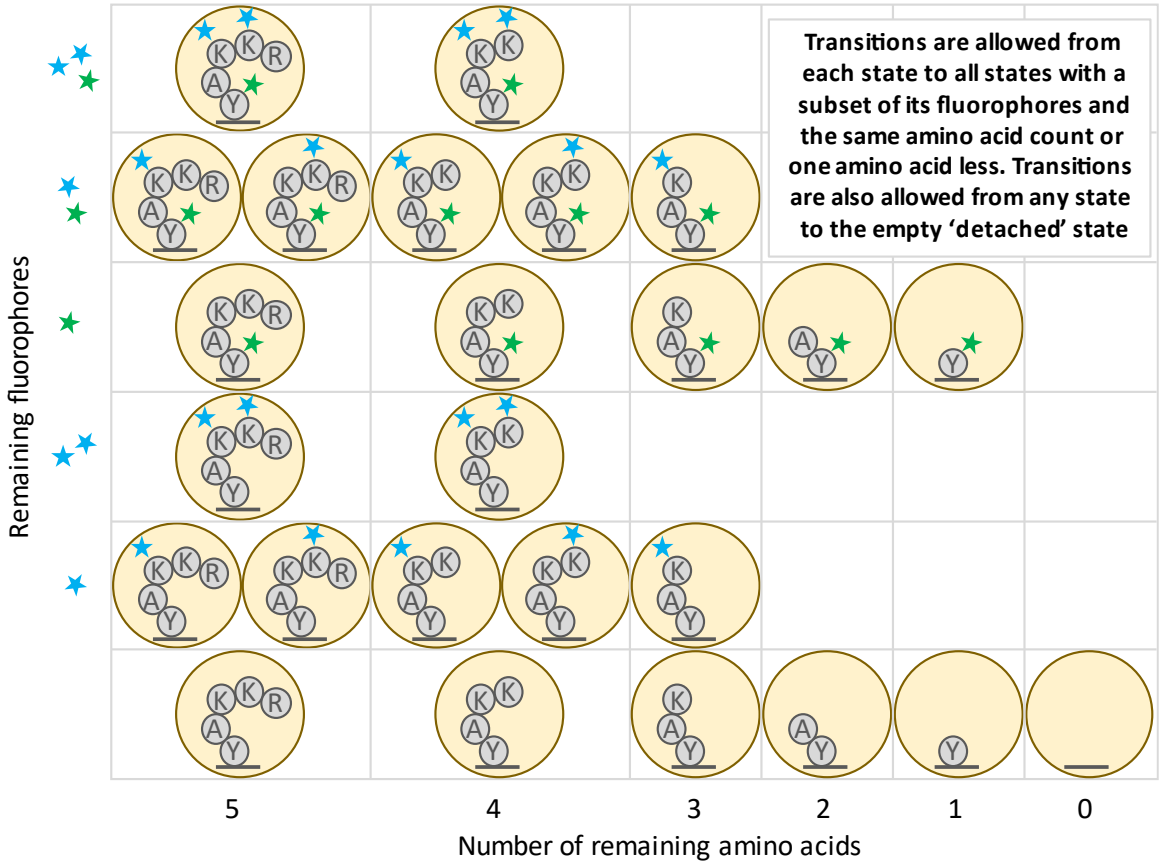

Figure 3. Illustration of the states and transitions of the HMM for an example peptide. For the amino acid sequence RKKAY, we illustrate the case where the lysine (K) residues are labeled with fluorescent dyes of one color (blue stars) and the tyrosine (Y) residue is labeled by a second color (green star).

Where $\boldsymbol{f}^{(t)}$ represents the cumulative probabilities for each state at timestep t, where $0 \leq t \leq T$, $\boldsymbol{O}^{(t)}$ represents the diagonal emission matrix for the observation seen at timestep t, and $\boldsymbol{T}$ represents the transition matrix which is the same at every timestep. The entries in each $\boldsymbol{f}^{(t)}$, $\boldsymbol{O}^{(t)}$, and in $\boldsymbol{T}$, represent the following probabilities:

$$\boldsymbol{f}_i^{(t)} = p(Y_{1:t} = y_{1:t}, X_t = i | Z = z) \tag{2}$$

Where $Y_{1:T}$ are the random variables for the observations, $y_{1:T}$ are their true values, $X_{1:T}$ are the random variables for the state in the HMM and Z is the random variable representing the peptide, and z is a value it can take. We also have diagonal matrices $\boldsymbol{O}^{(t)}$ defined as:

$$\boldsymbol{O}_{ii}^{(t)} = p(Y_t = y_t | X_t = i, Z = z) \tag{3}$$

And :

$$\boldsymbol{T}_{ij} = p(X_{t+1} = i | X_t = j, Z = z) \tag{4}$$

We start from an initial state $\boldsymbol{f}^{(0)}$ which we compute by taking into account the missing fluorophore rate m_c. Applying (1) repeatedly starting with the initial state $\boldsymbol{f}^{(0)}$ yields a value for $\boldsymbol{f}^{(T)}$, and we can sum the entries to compute:

$$p(Y_{1:T} = y_{1:T} | Z = z) = \sum_i p(Y_{1:T} = y_{1:T}, X_T = i | Z = z) = \sum_i \boldsymbol{f}_i^{(T)} \tag{5}$$

Then, by using Bayesian inversion to normalize the data, we compute the probability of the peptide given the raw read, as given by:

$$p(Z = z | Y_{1:T} = y_{1:T}) = \frac{p(Y_{1:T} | Z = z) p(Z = z)}{\sum_{\tilde{z}} p(Y_{1:T} = y_{1:T} | Z = \tilde{z}) p(Z = \tilde{z})} \tag{6}$$

We implemented several algorithmic optimizations to this approach to reduce runtime. These included reducing the number of states in the HMMs, factoring the HMMs' transition matrices into a product of matrices with higher sparsity, pruning the HMM forward algorithm to consider only reasonably likely states at each timestep, and combining the HMM classifier with a kNN pre-filter that can rapidly select a short-list of candidate peptides for re-scoring by the HMM. We implemented the linear algebra and tensor operations being performed in a manner that makes productive use of spatial and temporal locality of reference. We describe these optimizations in more detail in the following sections and in the supplemental **Appendices**.

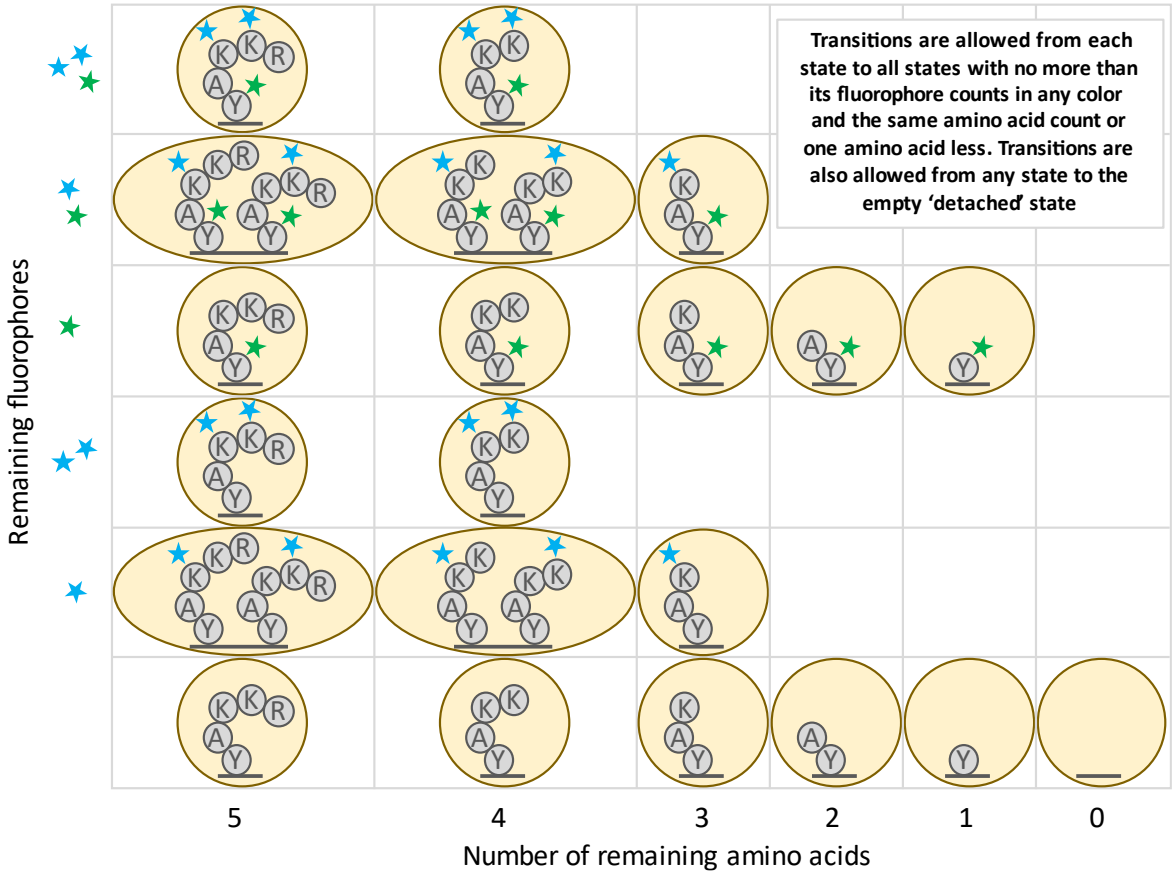

Figure 4. Illustration of HMM state space reduction for the peptide of Figure 3. States are combined that have both the same number of amino acids remaining and the same fluorophore counts for each color of fluorophore.

HMM state space reduction

We combine states of certain peptide conditions into more inclusive states in our model. For peptide conditions to be combined into these more inclusive states, they must have experienced the same number of *successful* Edman degradation events (so that they will have the same number of amino acids remaining), and they must have the same numbers of fluorophores of all colors. An example of the resulting HMM for a sample peptide is shown in **Figure 4**.

A similar state reduction to ours was previously described by Messina and colleagues in [12]. The reduction requires fluorophores to behave independently of each other, so that the status of one fluorophore is uncorrelated with the status of any other. While this is not true in practice due to FRET

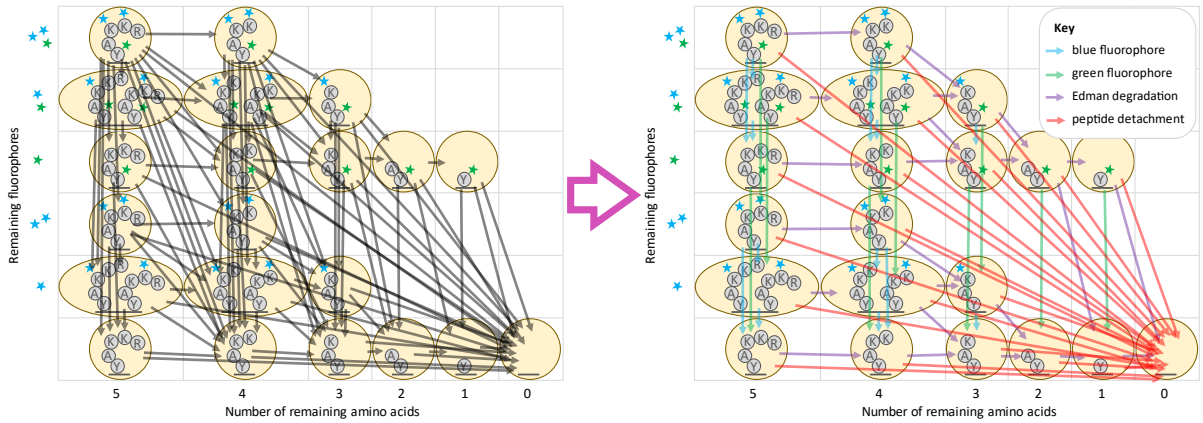

Figure 5. An illustration of the factoring of the transition matrix for the peptide from Figure 4. Note especially the reduction in the total number of transitions (arrows) when the transition matrix is factored. At left, black arrows represent non-zero entries in the unfactored transition matrix. At right, colored arrows (see key) represent non-zero entries in each of the matrices in the factored product. In both diagrams, arrows from a state to itself are omitted for visual clarity.

(Förster resonance energy transfer) and other dye-dye interactions, quantification of this effect in the imaging conditions used for fluorosequencing suggests that these effects are negligible enough to ignore [2]. The authors of [12] also require the fluorophores to be indistinguishable to reduce the numbers of states. This is not true in our case because we use Edman degradation and because we use multiple colors of fluorophores.

Nonetheless, we demonstrate in **Appendix A1** that despite these complications, this state space reduction incurs no loss in the theoretical accuracy of the model. Further, we demonstrate that this reduces the algorithmic complexity from what would otherwise be exponential with respect to the number of fluorophores, to instead be tied to the product of the counts of fluorophores of each color.

Transition matrix factoring

In the HMM forward algorithm, a vector of probabilities with one value for each state in the HMM's state space is repeatedly multiplied by a square transition matrix. This operation is the dominant contribution to the algorithmic complexity of the HMM forward algorithm. Therefore, by making multiplication by the transition matrix more algorithmically efficient, we can improve the theoretical complexity of our computational pipeline.

We factor this transition matrix into a product of highly sparse matrices. This factorization is done by creating a separate matrix for each independent effect under consideration, including loss of each color of dye (where each color is factored separately), Edman degradation, and finally, peptide detachment (**Figure 5**). As with the state space reduction, this optimization incurs no loss in the accuracy of the model, and furthermore, these matrix factors, even in combination, are far sparser than the original transition matrix when computed for larger peptides. This greater sparsity can be leveraged to achieve superior algorithmic complexity results (see **Appendix A2**).

HMM pruning

Despite significant improvements in the algorithmic complexity of an HMM for one peptide given so far from state space reduction and matrix factorization, performance can be improved if we consider

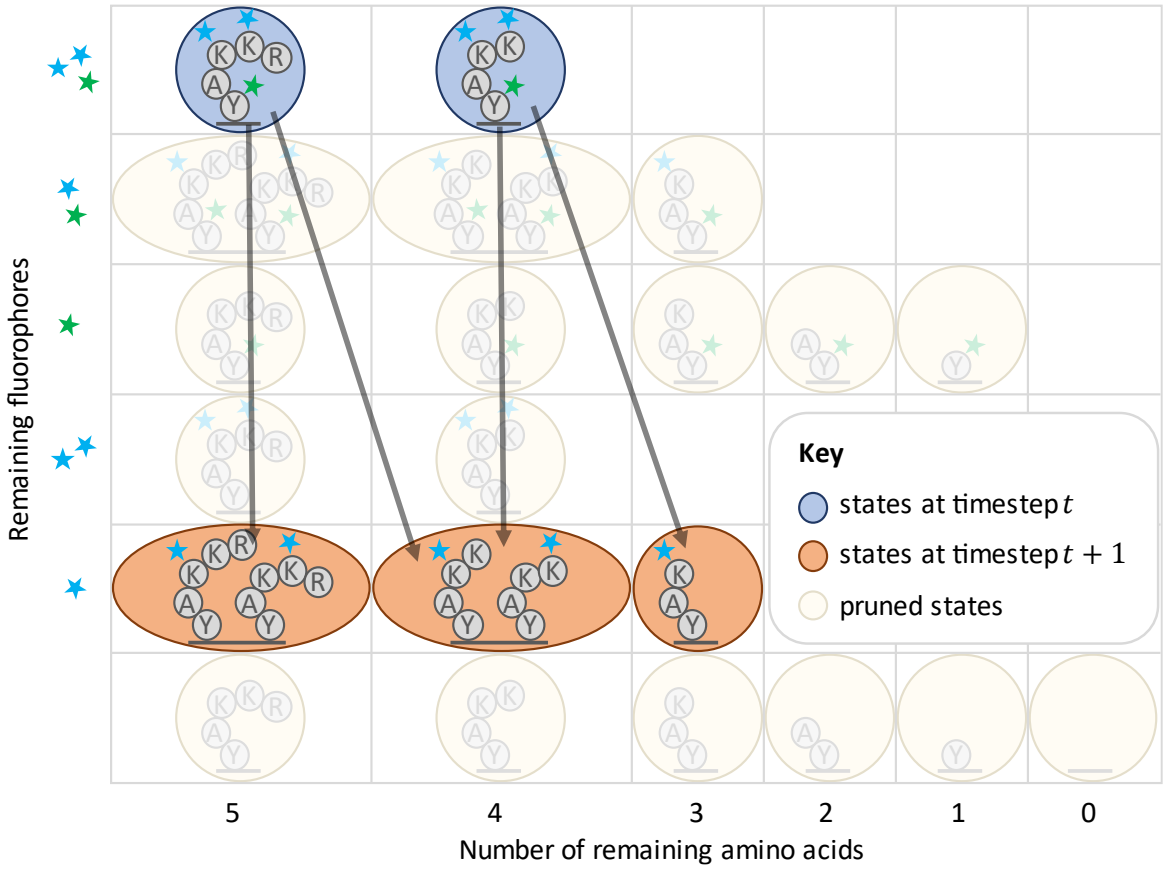

Figure 6. Illustration of the effects of HMM pruning for the peptide of Figure 5.

206 approximations. Intermediate computations contain mostly values close to zero, which will have
207 inconsequential impact on the result of the HMM forward algorithm. The most significant contribution
208 to this occurs for the HMM emission calculations. While there may be many states of a peptide which
209 have a significant probability of producing a particular observation value, in most states (particularly for
210 larger peptides) the observed value is extremely unlikely.

211 Emission computations can be viewed as multiplication by a diagonal matrix, different for each emission
212 in a raw read. The entries represent the probability of the indexed state producing the known emission
213 value for that timestep. We prune this matrix by setting anything below a threshold to zero, which
214 increases the sparsity of the matrix. Although use of a *naïve*, but standard, sparse matrix computational
215 scheme would reduce runtime, we show that better algorithmic complexity can be achieved with a more
216 complicated bi-directional approach in **Figure 6** and **Appendix A3**. While we did not implement this
217 approach precisely, a consideration of this effect served as inspiration for a technique combining
218 pruning with matrix factoring, as described next.

219 ## Combining transition matrix factoring with HMM pruning

220 Both transition matrix factoring and HMM pruning appear, at first glance, to be incompatible
221 improvements. These approaches can be combined, but the bi-directional sparse matrix computational
222 scheme introduces significant additional difficulties.

223 We view the various factored matrices as tensors and propagate contiguous blocks of indices forwards
224 and backwards before running the actual tensor operations to avoid unnecessary calculations (**Figure 7**).
225 Contiguous blocks of indices are needed because propagating lists of indices across the various factors
226 of the matrices has the same computational complexity as multiplying a vector by these matrices. This
227 may make the pruning operation seem less optimal in a sense, as some values that get pruned may be

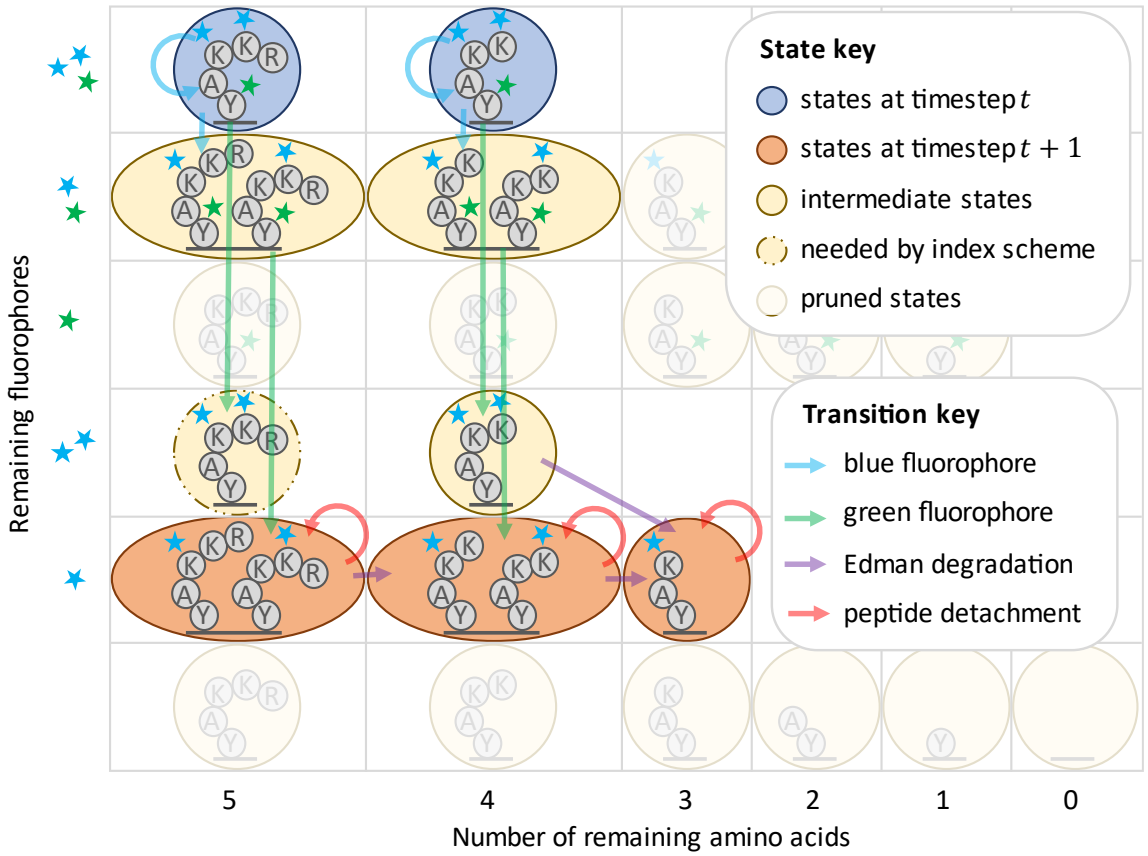

Figure 7. Illustration of HMM pruning combined with transition matrix factoring for the peptide of Figure 5. We emphasize that this is an anecdotal example; while there are more arrows here than in **Figure 6**, this strategy provides an improvement in asymptotic complexity, as described in **Appendix A4** and shown in experiments with simulated data.

bigger than some that are kept due to this form of indexing. Nevertheless, we found the tradeoff to be favorable in practice (**Appendix A4**).

Of note, instead of pruning by the raw values, we prune all states such that the known emission value is outside of their pre-configured confidence interval. In this way we provide some confidence that the fraction of true data inadvertently zeroed out is negligible.

k-Nearest Neighbors classification

Most traditional machine learning classifiers have an algorithmic complexity which scales proportionally or worse to the number of classification categories. The Bayesian classifier we have so far described is no exception; each raw read must be compared against every peptide in the reference dataset to be

classified. There are many problems in biology which require large reference datasets, human proteomic analysis being one example. The human proteome has 20,000 proteins, which when trypsinized generate hundreds of thousands of peptides. Classification against these many categories is computationally intractable with a fully Bayesian approach.

In contrast, the algorithmic complexity of kNN scales logarithmically with the number of training points used. For this reason, tree-based methods are common in other Extreme Classification applications [9], where similarly massive numbers of categories are under consideration. Unfortunately, the resulting faster runtimes come at a significant cost; kNN often gives far worse results in practice than a more rigorous Bayesian approach.

For purely kNN based classification, we simulate 1000 raw reads per peptide in the reference to create a training dataset and put these into a custom KD-Tree implementation for fast and easily parallelizable nearest neighbor lookups. We do not allow edits in our KD-Tree after it is built so as to allow parallelized lookups to occur without any concern for locks or other common issues in parallel data structures. We also reduce the memory footprint of the KD-Tree through an unusual compression scheme. For our training data, we use dye tracks instead of raw read radiometry data; this alone reduces the memory footprint of the KD-Tree by a factor of four (dye tracks have a two-byte numeric value for every timestep/color combination, while radiometry data have an eight-byte double-precision floating point number). But this allows another further compression technique; we find all identical dye tracks and merge them into one entry. With these dye tracks entries in the KD-Tree we store lists of peptides that produced the dye track when we simulated our training data, along with how many times each peptide produced that dye track.

To classify an unknown raw read, the k nearest dye track neighbors to a raw read query are retrieved. These neighbors then vote on a classification, with votes weighted using a Gaussian kernel function,

$\exp\left(-\frac{\delta^2}{2\sigma_{kNN}^2}\right)$, where δ is the Euclidean distance between the query raw read and the neighbor, and σ_{kNN} is a parameter of the algorithm. A neighbor is also weighted proportionally to the number of times it occurred as a simulation result and will split its voting weight among all of the peptides that produced that dye track proportionally to the numbers of times each peptide produced the dye track during simulation of training data.

Once voting is complete, the highest weighted peptide is then selected as the classification, with its classification score given as a fraction of its raw score over the total of all the raw scores. We have explored multiple choices of k and σ values to optimize the performance.

Hybridizing kNN with Bayesian HMM classification

To combine the computational efficiency of kNN with the accuracy of the HMM model, we defined a classifier which hybridizes these two disparate methods. We use a kNN classifier to reduce the reference dataset, for each raw read, down to a smaller shortlist of candidate peptides. These candidates can then be used in the Bayesian classifier by building HMMs to compare them against the specific raw read. While this can result in the true most likely peptide not being in the shortlist and therefore not being selected by this hybrid classifier, with a sufficiently long shortlist this is highly unlikely. A larger problem is in performing Bayes' rule, as in (6). An exact formula for Bayes' rule requires an exhaustive set of probability values for every potential outcome, which are summed in the denominator. Avoiding determining every probability makes this impossible. Instead, we can estimate Bayes' rule as follows:

$$p(Z = z|Y_{1:T} = y_{1:T}) = \frac{p(Y_{1:T}|Z = z)p(Z = z)}{\sum_{\tilde{z}\in\zeta_h} p(Y_{1:T} = y_{1:T}|Z = \tilde{z})p(Z = \tilde{z})}$$

Where ζ_h is the set of up to h peptides selected by the kNN method; we require $z \in \zeta_h$. Although we lose theoretical guarantees of optimal accuracy given the model, this change provides a considerable improvement to the algorithmic complexity. The algorithmic complexity to classify one raw read using a

fully Bayesian approach is $O(RW)$, where R is the number of peptides in the reference dataset, and W is the average amount of work needed to run an HMM for one peptide fluorophore combination. In comparison, with the hybridized classifier, the algorithmic complexity is $O(\log(RQ) + hW)$ where Q is the number of raw reads in the training dataset simulated for each possible peptide.

We chose specific values for h, σ, and k, by comparing the runtime and PR curves on simulated datasets.

Maintaining spatial locality of reference

Spatial and temporal locality of reference is the tendency of some computer programs to access nearby data points at similar times. Modern CPUs are designed to make this extremely efficient through multi-level batch caching schemes which cache data from RAM that is nearby a memory address being accessed, so that nearby data can be read more quickly. Programs which exploit this in read-write intensive pieces of code can often achieve significant runtime acceleration compared to programs which do not.

We wrote highly optimized kernel functions to perform our structured matrix/tensor operations which exploited the sparse nature of the problem while also iterating over elements in what we believe to be an optimal or near optimal fashion for most computer architectures. We believe this provided considerable improvements in performance, though this has not been rigorously tested.

Results

We simulated the fluorosequencing of peptides to obtain labeled training and testing data (**Figure 2**). We generated several datasets, each with a randomized subset of the proteins in the human proteome. We selected 20 proteins (.1% of the human proteome), 206 proteins (1% of human proteome), 2,065 proteins (10%), and 20,659 proteins (the full human proteome). We repeated this randomized selection scheme to examine several protease and labeling schemes. These were (1) trypsin (which cleaves after

lysine (K) and arginine (R) amino acids) with fluorescent labels for aspartate (D) and glutamate (E) (these share a fluorophore color due to their equivalent reactivities), cysteine (C), and tyrosine (Y), (2) cyanogen bromide (which cleaves after methionine (M) amino acids) with D/E, C, Y, and K, (3) EndoPRO protease (which cleaves after alanine (A) and proline (P) amino acids) with D/E, C, and Y, (4) EndoPRO with D/E, C, Y, and K, (5) EndoPRO with D/E, C, Y, K, and histidine (H). Thus, in the schemes examined, of the 20 canonical amino acid types found in most proteins, either one or two were recognized by the protease and up to 6 additional amino acids were labeled by fluorescent dyes.

These databases of peptides were used to generate databases of idealized *dye sequences*, *dye tracks*, and *raw reads*, used for training and testing purposes for the various models. For our test data for each dataset, we generated 10,000 raw sequencing reads by randomly selecting peptides with replacement from the dataset and simulating sequencing on them using the methods described in the *Monte Carlo simulation* section of this paper. For both dye tracks and simulated fluorescent intensity measurements, results where there were zero fluorophores throughout sequencing were discarded, as these would fail to be observed in an actual sequencing experiment.

We collected and compared runtime data and precision-recall curves for several different purposes. With the trypsinized 3-color dataset, we performed a parameter sweep of the pruning cut-off for the HMM Bayesian classifier (**Figure B1**). Losses in precision and recall performance were negligible for cut-off values of 5 and greater, though the precision recall curves grew worse at smaller values. Runtimes shrank rapidly as the cut-offs were decreased. The pruning cut-off parameter sweep was also performed on the 20-protein cyanogen bromide dataset (**Figure B2**). We saw that in this second dataset, runtime improvements for lower cut-offs were even more extreme; a speed-up factor of about 1000 could be achieved with minimal effects on the precision recall plots. From these two simulations, we chose a cutoff value of 5 as providing the optimal trade-off between runtime and precision recall performance.

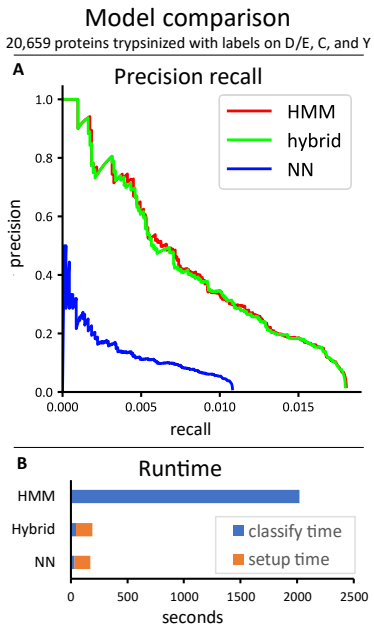

Figure 8. Comparison of the HMM (Bayesian), hybrid, and NN classifiers on a dataset of 10K reads from peptides chosen randomly from all 20,659 human proteins trypsinized and labeled on D/E, C, and Y. (**A)** The precision recall curves. (**B)** Runtimes.

On the trypsinized dataset (full human proteome), we also swept the k and σ_{kNN} parameters of the NN classifier (**Figures B3, B4**). Here large values of k introduce modest reductions in precision recall performance, while the model is extremely sensitive to the selection of σ_{kNN}. Based on this analysis, we suggest that good choices of these parameters are $k = 10$ and $\sigma_{kNN} = 0.5$.

We swept all parameters of the hybrid classifier for the trypsinized dataset as well (hybrid h parameter, k, σ_{kNN}, and cut-off) (**Figures B5-8**). Here, the HMM cut-off parameter had less impact on runtime than for the pure HMM Bayesian classifier, but we still found a cut-off of 5 to be optimal. Higher values of k improved precision recall performance for the hybrid model, contrary to the results of the NN classifier on its own, and we therefore suggest setting k to 10000. σ_{kNN} had minimal impact of any kind, in contrast to its significant impact on the precision recall of the NN classifier; we nevertheless chose to set it to 0.5 in light of the data from parameter tuning for the NN classifier on its own. We also found that higher values of h improved performance, though the impact plateaus after h of about 1000, and we used that value for later experiments.

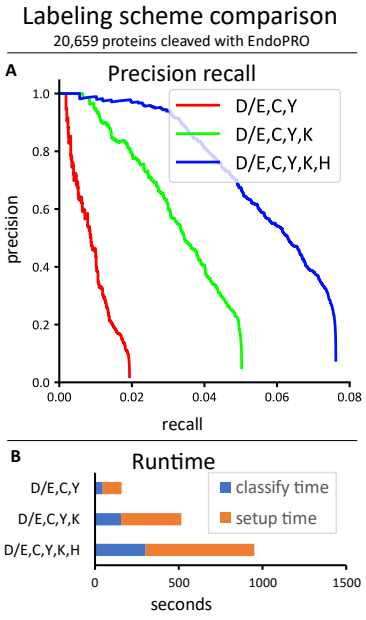

Figure 9. Comparison of the hybrid classifier on a dataset of 10K reads from peptides chosen randomly from all 20,659 human proteins cleaved with EndoPRO and labeled with three different labeling strategies. (**A)** The precision recall curves. (**B)** Runtimes.

After tuning parameters, we compared the performance of the different classifiers when applied to 10,000 simulated fluorosequencing reads of peptides drawn randomly from all tryptic

peptides in the human proteome (**Figure 8**). The hybrid classifier achieved similar precision recall curves to the Bayesian HMM classifier, which was much better than the precision recall curve of the NN classifier. The hybrid classifier also achieved runtimes of the same order of magnitude as the NN classifier, which was significantly faster than the runtime of the Bayesian HMM classifier.

We also studied how the number of fluorophore colors affected the runtime and precision/recall of the hybrid classifier (**Figure 9**). We found that improvements in precision recall were possible with each additional color of fluorophore, but this did come at the cost of longer runtimes.

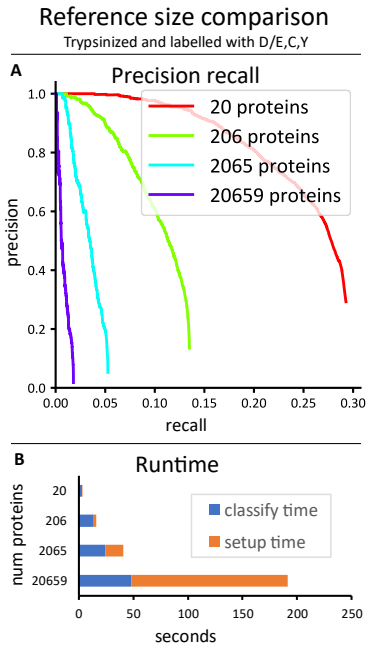

Figure 10. Comparison of the hybrid classifier across datasets of 10K reads each from peptides chosen randomly from different numbers of proteins treated with the same protease and labeling scheme. (**A)** The precision recall curves. (**B)** Runtimes.

We also investigated the effect of varying sizes of reference proteomes on the hybrid classifier's performance, using the three color trypsinized dataset (**Figure 10**). We found that significantly better performance was possible when the reference database was smaller.

The precision/recall curves plotted above (**Figures 8-10**) show the actual precision/recall scores based on data with known peptide classifications. When working with real data this will typically not be possible, because the real classifications will not be known. It is therefore important that the assignment probabilities produced by the classifier be well-calibrated, so that an estimate of the precision/recall (or as is more often the case in protein mass spectrometry, the false discovery rate (FDR)) can be computed in the absence of known labels. We verified that the probabilities output by the hybrid HMM classifier were indeed well-calibrated relative to the true assignment probabilities (**Figure 11A**). This in turn allowed us to compute a predicted precision/recall curve assuming that each classification is fractionally

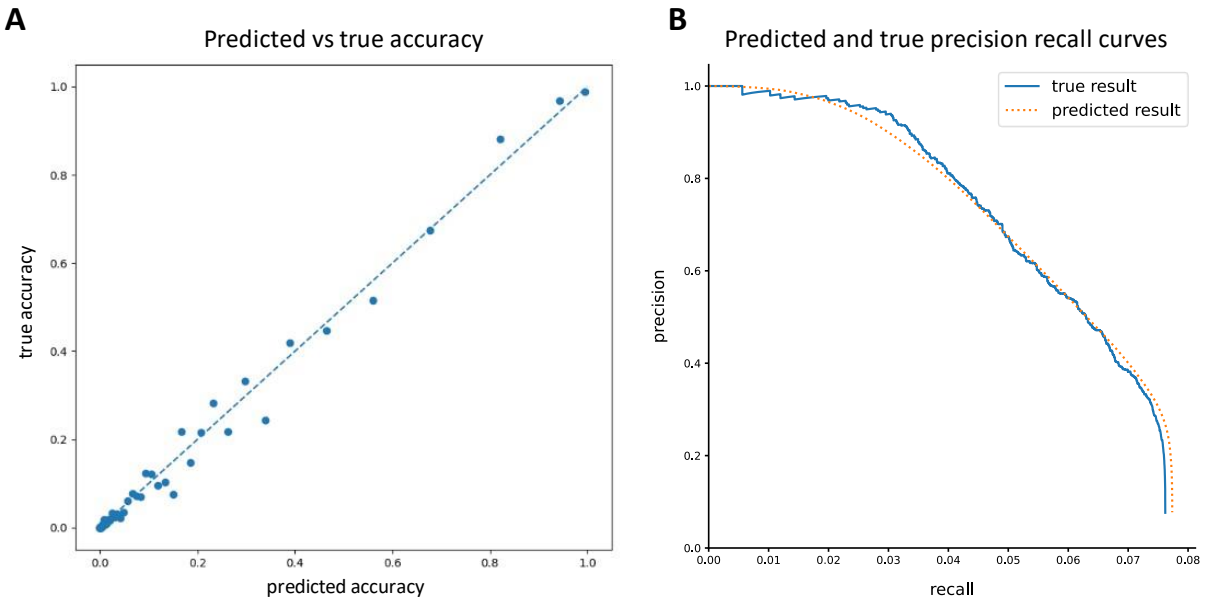

Figure 11. Analysis of the accuracy of probability estimates given as scores by the classifier. Based on 10K reads from peptides chosen randomly from all 20,659 human proteins cleaved with EndoPRO and labeled on D/E,C,Y,K,H. **(A)** Classification results were sorted by their predicted accuracy scores, and then equally distributed between 100 buckets. The average predicted and true accuracy scores were then computed for each bucket and plotted. **(B)** The true result precision/recall curve was computed as normal, while the predicted result precision/recall curve was plotted assuming each classification was fractionally correct according to its predicted accuracy score.

correct with a probability given by its classification score. A comparison of this predicted P/R with the actual precision/recall curve for the same set of reads shows excellent agreement (**Figure 11B**).

Whatprot specifically attempts to assign each raw fluorosequencing read to one or more peptides from the reference database, *i.e.*, to identify and score *peptide-read matches* (PRMs), a process highly analogous to analytical interpretation of shotgun mass spectrometry (MS) proteomics data in which a key step is comparing experimental peptide mass spectra to a reference proteome (finding *peptide-spectral matches*, or PSMs [13]). However, observing multiple reads mapping to the same peptide will tend to increase the confidence that peptide is present in the sample, just as observing multiple peptides from the same protein will similarly increase confidence in that protein being present. Thus, we asked if considering the PRMs collectively led to performance increases for identifying peptides and proteins.

As shown in (**Figure 12**), proteins can be identified correctly at much higher rates than peptides, which are similarly identified at higher rates than individual reads. In fact, provided that a protein possesses some well-identified peptides that are unique, it can typically be identified with very high accuracy. For this test, we used a very simple protein inference scheme. First each peptide was scored to the maximum score of all reads identifying it, while penalizing reads which identified more than one peptide (dividing by n if n peptides were identified). Second each protein was scored as the maximum score of all peptides it contains, penalizing peptides which are associated with more than one protein (again dividing by n if n proteins were associated). However, the problem of integrating peptide level observations to protein observations has been studied extensively for MS [14] [15], and it is likely that these techniques will offer similarly strong interpretive power to the case of single molecule protein sequencing.

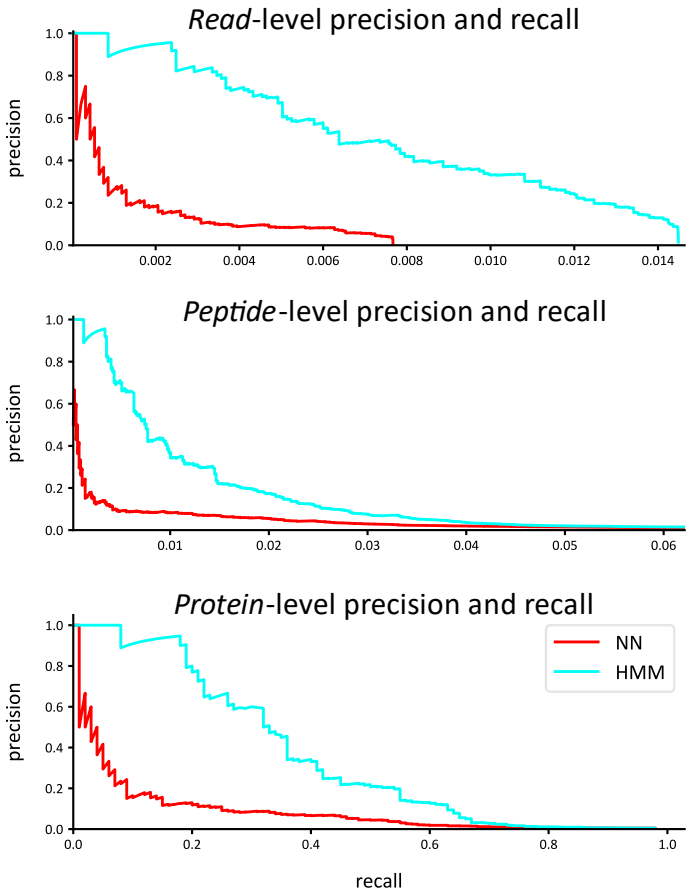

Figure 12. Precision and recall are improved for proteins by integrating identifications across peptides. The example shows 10K reads from peptides derived from 100 proteins randomly selected from the human proteome, considering trypsin digestion and labels on D/E, C, and Y.

Discussion

We developed an HMM for interpreting single molecule protein fluorosequencing data and showed that a hybrid HMM/kNN model can achieve a high precision and recall comparable the HMM alone while maintaining a runtime comparable to the much faster kNN.

It is worth emphasizing that these analyses were performed on datasets of 10,000 raw fluorosequencing reads. In practice, users will likely want to analyze millions to billions of reads, so runs that completed in a seemingly reasonable amount of time might still be intractable in these scenarios with larger datasets, or at a minimum require computing clusters with high parallelization. For analyzing the current datasets in the runtime charts (**Figures 8-10**), note that the *blue* part of the bar graphs indicates the classify time, which will scale with the number of reads being classified (if all else remains equal), and the *orange* part of the graph indicates setup time, which should remain constant regardless of the number of reads (though it changes depending on the model and the size of the reference set).

It is also interesting that the HMM pruning operation is more necessary with longer peptides and more colors of fluorophores; with the trypsinized dataset labeling D/E, C, and Y, omitting pruning had little consequence, but in moving to cyanogen bromide with D/E, C, Y, and K, we observed a runtime speedup of about 1000-fold.

Finally, our data demonstrates that with a proper selection of parameter values, the hybrid model can achieve precision and recall performance virtually identical to the HMM Bayesian approach alone, while providing those results in a fraction of the time. Similarly, the pruning operation employed in the HMMs has no noticeable positive or negative effect on the precision recall curves while providing a considerable improvement in runtime performance.

A number of analytical techniques common in related fields were not explored. In tandem mass spectrometry (MS/MS), peptide spectral mapping is typically done either through database lookups and/or the use of simulated outcomes. Simulated mass spectra, consisting of ion pairs for the C- and N-terminal fragments for each potential breakage point, can be compared to the real data collected from the instrument [13]. Recent advances have been achieved by using deep learning to predict fragmentation behavior with higher quality than is possible with more traditional methods [16][24]. While we use the notion of matching fluorosequencing reads to a reference database, the specific algorithms are distinct.

Nevertheless, of possible relevance from the field of MS/MS is the analysis of the false discovery rate (FDR) [25][26]. The FDR is affected by two distinct sources: a peptide may be misattributed to the wrong peptide, even when the true peptide is present in the reference dataset, and MS/MS datasets contain significant amounts of modified peptides or contaminants, whose spectra may be mistakenly assigned to peptides in the reference set [27]. FDR is typically evaluated using a decoy database, such as is generated using reversed proteins from the target database. The FDR can then be set by referring to the number of hits in the decoy database given a particular score, as the decoy database is designed such that it should in theory never have hits for the biological sample being analyzed [17]. While an estimate of FDR based in theoretical analysis of the problem could find the misattribution rate of true peptides, even this estimate would be incomplete, because there are errors in mass spectra of peptides that cannot be accounted for by existing theory; furthermore, any effect of modifications or contaminants would likely be omitted.

The utility of a similar decoy database strategy for estimating FDR for fluorosequencing is unknown and remains to be established. We note however that, due to the rigorous probabilistic nature of our analysis, a reasonable estimate of FDR can be performed by subtracting the sum of PRM scores from the number of PRMs. This is the same as one minus the precision in a predicted precision/recall curve, and

451 the proximity of our predicted precision/recall curve to the real curve for a known dataset demonstrates
452 the feasibility of this approach (**Figure 11**). This analysis likely fails to account for the contributions of
453 modifications and contaminants. We therefore plan to explore this problem more extensively in future
454 work.

455 We also considered techniques for DNA sequence reconstruction. In general, DNA sequencing provides
456 *de novo* sequence reconstructions and does not use reference database matching, and therefore is not a
457 good model for fluorosequencing. Nevertheless, base calling strategies may have some relevance. For
458 example, methods for base-calling from conventional (e.g. Illumina style) DNA sequencing are
459 straightforward [18] [19], and although errors occur, they are rare [20]. Analysis of errors in DNA
460 sequencing is typically performed using multiple sequence alignment or k-mer based methods [21].
461 Because the error rates are typically much lower in DNA sequencing than in fluorosequencing, we
462 believe existing software is unlikely to be effective in this new domain.

463 Nanopore DNA sequence analysis methods could also be considered. Nanopores, similar to
464 fluorosequencing, deal with single molecule data and the concomitant statistical noise that process
465 involves. However, nanopore data is on a real time continuum, with a DNA fragment which may move
466 through the nanopore at variable rates during sequencing. Base-calling, the assignment of nucleic acid
467 bases to chunks of sequencing information, is again the step most analogous to fluorosequencing. State
468 of the art base-calling methods for nanopore sequencing typically use either HMMs or recurrent neural
469 networks (RNNs). Comparisons of existing approaches suggest that RNNs slightly outperform HMMs in
470 this domain [22]. While this suggests that RNNs are worth exploring for fluorosequencing data, we have
471 avoided this approach for two reasons. First, RNNs are a deep learning technique, which invariably
472 requires access to massive amounts of data; this is not currently feasible with fluorosequencing unless
473 that data is simulated. Second, our approach suggests the possibility of direct estimation of parameters

using some variant of the Baum-Welch algorithm adapted to our use case, which we believe would be significantly more difficult in an RNN based approach [23].

Conclusions

We have developed a powerful computational tool for the analysis of protein fluorosequencing data, which significantly increases the complexity of applications available to this new technology. This tool includes critically important optimizations which make our approach feasible in practice.

In future work we plan to implement a variation of the Baum-Welch algorithm to fit the parameters to data from a known peptide. We also wish to explore peptide and protein inference methods using peptide data classified using these methods. We may also explore *de novo* recognition of labels without use of a reference database.

Acknowledgements

Code is available at https://github.com/marcottelab/whatprot. The authors gratefully acknowledge Jagannath Swaminathan, Angela Bardo, Brendan Floyd, Daniel Weaver, and Eric Anslyn for helpful guidance and discussion throughout the course of this project. M.B.S. acknowledges support from a Computational Sciences, Engineering, and Mathematics graduate program fellowship. E.M.M. acknowledges support from Erisyon, Inc., the National Institute of General Medical Sciences (R35GM122480), the National Institute of Child Health and Human Development (HD085901), and the Welch Foundation (F-1515).

Disclosures

E.M.M. and Z.B.S. are co-founders and shareholders of Erisyon, Inc., and are co-inventors on granted patents or pending patent applications related to single-molecule protein sequencing. E.M.M. serves on the scientific advisory board.

References

1. Swaminathan J, Boulgakov AA, Marcotte EM. A theoretical justification for single molecule protein sequencing. *PLoS Computational Biology*. 2015;11(2):1-17.
2. Swaminathan J, Boulgakov A, Hernandez ET, Bardo AM, Bachman JL, Marotta J, *et al.* Highly parallel single-molecule identification of proteins in zeptomole-scale mixtures. *Nature Biotechnology*. 2018;36(11):1076-1082.
3. Callahan N, Tullman J, Kelman Z, Marino J. Strategies for development of a next-generation protein sequencing platform. *Trends in Biochemical Sciences*. 2020;45(1):76-89.
4. Floyd BM, Marcotte EM. Protein sequencing, one molecule at a time. *Annual Review of Biophysics*. 2022;51:181-200.
5. Restrepo-Pérez L, Joo C, Dekker C. Paving the way to single-molecule protein sequencing. *Nature Nanotechnology*. 2018;13:786-796.
6. Vistain LF, Tay S. Single-cell proteomics. *Trends in Biochemical Sciences*. 2021;46(8):661-672.
7. Edman P, Method for determination of the amino acid sequence in peptides. *Acta Chemica Scandinavica*. 1950;4:283-293.
8. Edman P, Begg G. A protein sequenator. *European Journal of Biochemistry*. 1967;1(1):80-91.
9. Prabhu Y, Kag A, Harsola S, Agrawal R, Varma M. Parabel: Partitioned label trees for extreme classification with applications to dynamic search advertising. *Proceedings of the international world wide web conference*. 2018 Apr 23-27; Lyon, France.
10. Zhang L, Floyd BM, Chilamari M, Mapes J, Swaminathan J, Bloom S, *et al.* Photoredox-catalyzed decarboxylative C-terminal differentiation for bulk- and single- molecule proteomics. *ACS Chemical Biology*. 2021;16(11):2595-2603.
11. Hinson CM, Bardo AM, Shannon CE, Rivera S, Swaminathan J, Marcotte EM, *et al.* Studies of surface preparation for the fluorosequencing of peptides. *Langmuir*. 2021;37(51):14856-14865.

12. Messina TC, Kim H, Giurleo JT, Talaga DS. Hidden Markov Model analysis of multichromophore photobleaching. *The Journal of Physical Chemistry B*. 2006;110(33):16366-16376.

13. Eng JK, McCormack AL, Yates JR. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society of Mass Spectrometry*. 1994;5(11):976-989.

14. Keller A, Nesvizhskii AI, Kolker E, Aebersold R. Empirical statistical model to estimate the accuracy of peptide identifications made by MS/MS and database search. *Analytical Chemistry*. 2002;74(20):5383-5392.

15. Käll L, Canterbury JD, Weston J, Noble WS, MacCoss MJ. Semi-supervised learning for peptide identification from shotgun proteomics datasets. *Nature Methods*. 2007;4:923-925.

16. Dincer AB, Lu Y, Schweppe DK, Oh S, Noble WS. Reducing peptide sequence bias in quantitative mass spectrometry data with machine learning. *Journal of Proteome Research*. 2022;21(7):1771-1782.

17. Elias JE, Gygi SP. Target-decoy search strategy for mas spectrometry-based proteomics. *Methods in Molecular Biology*. 2010;604:55-71.

18. Saiki RK, Gelfand DH, Stoffel S, Scharf SJ, Higuchi R, Horn GT, *et al*. Primer-directed enzymatic amplification of DNA with a thermostable DNA polymerase. *Science*. 1988;239(4839):487-491.

19. Fedurco M, Romieu A, Williams S, Lawrence I, Turcatti G. BTA, a novel reagent for DNA attachment on glass and efficient generation of solid-phase amplified DNA colonies. *Nucleic Acids Research*. 2006;34(3):e22.

20. Stoler N, Nekrutenko A. Sequencing error profiles of Illumina sequencing instruments. *NAR Genomics and Bioinformatics*. 2021;3(1):1-9.

21. Mitchell K, Brito JJ, Mandric I, Wu Q, Knyazev S, Chang S, *et al*. Benchmarking of computational error-correction methods for next-generation sequencing data. *Genome Biology*. 2020;21(71):1-13.

22. Cali DS, Kim JS, Ghose S, Alkan C, Mutlu O. Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Briefings in Bioinformatics*. 2018;20(4):1542-1559.

23. Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, *et al*. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*. 2021;8(53):1-74.

24. Gessulat S, Schmidt T, Zolg DP, Samaras P, Schnatbaum K, Zerweck J, *et al*. Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. *Nature Methods*. 2019;16(6):509-518.

25. Käll L, Storey JD, MacCoss MJ, Noble WS. Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. *Journal of Proteome Research*. 2008;7(1):29-34

26. Keich U, Kertesz-Farkas A, Noble WS. Improved False Discovery Rate Estimation Procedure for Shotgun Proteomics. *Journal of Proteome Research*. 2015;14(8):3148-61

27. Kong AT, Leprevost FV, Avtonomov DM, Mellacheruvu D, Nesvizhskii AI. MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry-based proteomics. *Nature Methods*. 2017;14(5):513-520

Appendix A – detailed descriptions and proofs of algorithms

A0 – crib sheet for variables used throughout appendix A

Variable crib sheet

$\boldsymbol{f}^{(t)}$	Cumulative probabilities for each state at timestep t
$\boldsymbol{\mathcal{f}}^{(t)}$	Tensor form of $\boldsymbol{f}^{(t)}$
$\boldsymbol{O}^{(t)}$	HMM Emission matrix for timestep t
$\boldsymbol{\mathcal{O}}^{(t)}$	Tensor form of $\boldsymbol{O}^{(t)}$
$\widehat{\boldsymbol{\mathcal{O}}}^{(t)}$	A pruned approximation of $\boldsymbol{\mathcal{O}}^{(t)}$
$\boldsymbol{T}$	The HMM transition matrix
$\boldsymbol{\mathcal{T}}$	Tensor form of $\boldsymbol{T}$
$\widehat{\boldsymbol{\mathcal{T}}}^{(t)}$	A pruned approximation of $\boldsymbol{\mathcal{T}}$ at timestep t
T	The number of timesteps
$Y_{1:T}$	Random variables representing the series of observations
$y_{1:T}$	The true values of $Y_{1:T}$
$X_{1:T}$	Random variables representing the state of the HMM across time
Z	Random variable representing the peptide
z	True value of Z
s	The number of states of a fluorophore
Λ	The number of fluorophores
C	The number of colors of fluorophore
Λ_c	The number of fluorophores of color c
ρ	Number of amino acids successfully removed by Edman degradation
$\bar{c}$	Color of N-terminal amino acid
$\lambda_{\rho,c}$	Number of amino acids which can accept a label of color c when ρ amino acids have been removed from the peptide
ϕ_c	Number of functioning fluorophores of color c remaining for the peptide
α	Number of amino acids in the peptide (before sequencing)
p_c	Fluorophore loss rate for color c
$\boldsymbol{\mathcal{B}}^{(c)}$	A factored component of $\boldsymbol{\mathcal{T}}$ representing loss of fluorophores of color c
$\widehat{\boldsymbol{\mathcal{B}}}^{(t,c)}$	A pruned approximation of $\widehat{\boldsymbol{\mathcal{B}}}^{(c)}$ at timestep t
e	Edman cycle failure rate
$\boldsymbol{\mathcal{E}}$	A factored component of $\boldsymbol{\mathcal{T}}$ representing Edman degradation success and failure
$\widehat{\boldsymbol{\mathcal{E}}}^{(t)}$	A pruned approximation of $\boldsymbol{\mathcal{E}}$ at timestep t
d	Peptide detachment rate
$\boldsymbol{\mathcal{D}}$	A factored component of $\boldsymbol{\mathcal{T}}$ representing peptide detachment
$\widehat{\boldsymbol{\mathcal{D}}}^{(t)}$	A pruned approximation of $\boldsymbol{\mathcal{D}}$ at timestep t
r	Number of values of $\boldsymbol{\mathcal{O}}$ kept during pruning
r_c	Number of fluorophore counts of color c kept during pruning
$r_c^{(t)}$	r_c at time t
$\bar{r}$	Number of amino acid counts kept during pruning
$\bar{r}^{(t)}$	$\bar{r}$ at time t
h	The number of peptides selected by the kNN in the hybrid model
ζ_h	The set of peptides selected by the kNN in the hybrid model

A1 – HMM state space reduction

The basics

For fluorophores that are distinguishable and dependent on each other, where s is the number of states of one fluorophore, and Λ is the number of fluorophores, the number of states of the whole system is given as in [12] by:

$$s^{\Lambda} \tag{7}$$

In contrast, if the fluorophores are indistinguishable and independent of each other, the number of states in the system is instead given by the combinatoric equation from [12]:

$$\binom{\Lambda + s - 1}{s - 1} \tag{8}$$

Imaging is typically performed with high concentrations of the antioxidant Trolox [2] and for relatively short time intervals (100 msec); reversibly photobleached fluorophores do not occur frequently in our data and we ignore them as a first approximation. Therefore, we take s to be 2, with one state for a functioning fluorophore, and another for a missing, photobleached, or chemically destroyed fluorophore. This reduces (8) to $\Lambda + 1$ states given Λ indistinguishable and independent fluorophores.

More colors

Obviously, a red fluorophore is distinguishable from a blue one. But we would still like to benefit from the indistinguishability of red fluorophores from red fluorophores, and of blue fluorophores from blue fluorophores. This is modeled by first considering the states for each color of fluorophore independently, and then taking the cartesian product of these state spaces. For C colors of fluorophore, where Λ_c is the number of fluorophores of color c, this results in the number of states given by:

$$\prod_{c=1}^{C} (\Lambda_c + 1) \tag{9}$$

Each state then represents the number of remaining active fluorophores for each of our C colors of fluorophore.

Edman degradation

Our second challenge is the inclusion of Edman degradation in the HMM. Sequential removal of the N-terminal amino acid from each peptide breaks the assumption of indistinguishable fluorophores, which is the basis for the state reduction performed in [12]. However, through inductive reasoning we show that our model meets a weaker criterion, which can be used to merge states together as desired:

Any two states with the same numbers of fluorophores of each color and the same number of amino acids are equally likely. (10)

If we ignore Edman degradation, this follows directly from the assumed indistinguishability property of fluorophores of the same color; if two fluorophores behave identically, they are equally likely to be missing, photobleached, or chemically destroyed, thus it follows by symmetry that any two states with the same numbers of indistinguishable fluorophores of each color are equally likely. If we consider Edman degradation, then (10) is true for all states where no amino acids have yet been successfully removed. Let ρ indicate the number of amino acids removed from the original peptide. We have then shown that (10) is true when $\rho = 0$.

If states with identical fluorophore counts are equally probable for all states with ρ amino acids removed, it can be shown that all states with equal fluorophore counts are equally probable for all states with $\rho + 1$ amino acids removed. For removal of an amino acid that can't accept fluorophores under the experimental setup this is trivial, so consider a peptide with ρ removed amino acids, an N-terminal amino acid which accepts fluorophores of color $\bar{c}$, and $\lambda_{\rho,\bar{c}}$ amino acids total which can accept a

label of color $\bar{c}$. Then let $\phi_{\bar{c}}$ represent the number of remaining functional fluorophores for the peptide, satisfying $0 \leq \phi_{\bar{c}} \leq \lambda_{\rho,\bar{c}}$.

There are several conditions of the peptide with $\phi_{\bar{c}}$ functioning fluorophores scattered among the $\lambda_{\rho,\bar{c}}$ amino acids that can accept a label. When we remove the N-terminal amino acid, we may or may not remove with it a functioning fluorophore. The states which do have a functioning fluorophore in the N-terminal position (only possible when $\phi_{\bar{c}} > 0$) will have their other $\phi_{\bar{c}} - 1$ fluorophores distributed between the $\lambda_{\rho,\bar{c}} - 1$ remaining amino acids which can be labeled. Furthermore, these states are equally likely, as they are a subset of the equally likely states with $\phi_{\bar{c}}$ fluorophores. Since trivially $\lambda_{\rho,\bar{c}} - 1 = \lambda_{\rho-1,\bar{c}}$, these states map one-to-one with the states for the peptide with one less amino acid remaining, when it has $\phi_{\bar{c}} - 1$ dyes.

When $\phi_{\bar{c}} < \lambda_{\rho,\bar{c}}$, there are states with no fluorophore in the N-terminal position, even though the N-terminal amino acid can accept one. Then the $\phi_{\bar{c}}$ fluorophores will be distributed with equal probabilities among the $\lambda_{\rho,\bar{c}} - 1 = \lambda_{\rho-1,\bar{c}}$ remaining amino acids which can be labeled. Similarly to the other case, these states map one-to-one with the states for the peptide less one amino acid when it has $\phi_{\bar{c}}$ dyes.

The equally distributed probabilities and one-to-one correspondence between states across this amino acid removal ensures that these transformations do not break our guarantees of equal probabilities for $\rho + 1$ amino acids removed. Iteratively applying this reasoning, starting with $\rho = 0$, until we prove that states where $\rho = \alpha$ are equally likely if they have the same fluorophore counts, demonstrates that (5) is true under the assumptions we have taken.

This proves (10), which allowed us to safely merge states that share both the same fluorophore counts by color and the same numbers of amino acids.

Transition probabilities

We also need to know the transition probabilities for our new reduced state space. To deal with peptide detachment is trivial. Dye-loss, either for dyes missing before sequencing begins, or from chemical destruction during sequencing, can be modeled with a binomial distribution. This follows from the assumption that the fluorophores behave independently of each other.

For Edman degradation, there is of course a probability of success or failure of the degradation step, which we model as a Bernoulli random variable. In the case of success, we employ an additional Bernoulli random variable to model the probability of losing or not losing a functioning fluorophore. Because the true states within a merged state are equally likely, we can use combinatorics to count the number of states which will lose a dye, and the number that won't. Together these values can be used to find the probability of losing a fluorophore given a successful Edman degradation, as shown in the following formula, which conveniently reduces to a simple fraction:

$$\frac{\binom{\lambda_{\rho,\bar{c}}-1}{\phi_{\bar{c}}-1}}{\binom{\lambda_{\rho,\bar{c}}}{\phi_{\bar{c}}}} = \frac{\phi_{\bar{c}}}{\lambda_{\rho,\bar{c}}} \tag{11}$$

State reduction conclusions

This state reduction provides a considerable algorithmic complexity improvement to the HMM forward algorithm. The complexity of the forward algorithm is $O(S^2T)$, where S is the number of states, and T is the number of timesteps. Then, if implemented with the true state space of a labeled peptide, the number of states S is $O\left(\alpha 2^{\Lambda}\right)$, and we get a complexity of $O(\alpha^2 4^{\Lambda} T)$ for the HMM forward algorithm, where α is the number of amino acids and Λ is the total number of fluorophores (of any color). However, if we use the reduced state space, then S is $O\left(\alpha \prod_{c=1}^{C} \Lambda_c\right)$, giving an algorithmic complexity of $O(\alpha^2 \left(\prod_{c=1}^{C} \Lambda_c^2\right) T)$ for the forward algorithm, where C is the number of fluorophore colors being used and Λ_c is the number of fluorophores of color c. The scaling in either case is dominated by values of Λ or

Λ_c, which ranges from 1 to about 25 for human tryptic peptides, though in rare cases Λ_c can exceed 100.

A2 – Transition matrix factoring

The concept

Multiplication by sparse matrices is far more efficient than with dense matrices. Matrix vector multiplication with a dense matrix is $O(S^2)$ where S is the size of the vector; for this application vectors with thousands of entries are not uncommon, and even larger vectors are possible, although this depends on the protease and labeling scheme used. For a sparse matrix, matrix vector multiplication can be made to be $O(V)$, where V is the number of non-zero entries in the matrix. For highly sparse matrices this can be a significant improvement.

Since peptides cannot gain amino acids or functioning fluorophores during sequencing, a basic transition matrix for fluorosequencing has zeros except for entries for transitions in which the numbers of fluorophores of each color is decreasing or staying the same. While this does reduce the number of necessary operations, it only does this by a constant factor, with no effect on the asymptotic behavior in the limit. Additionally, the number of amino acids either stays the same, decreases by one (from a successful Edman degradation), or decreases to zero (from a peptide detachment event). This did improve the asymptotic behavior in the number of non-zero entries of the transition matrix, reducing this from $O\left(\alpha^2 \prod_{c=1}^{C} \Lambda_c^2\right)$ to $O\left(\alpha \prod_{c=1}^{C} \Lambda_c^2\right)$.

However, we did better by factoring this matrix (**Figure 4**). We used the independence of our different forms of error, with one matrix in the factored product for each type of error. To demonstrate this factorization, we reformulated our problem in tensor notation. The vector for the state space of a peptide with C colors not undergoing Edman degradation or peptide detachment can be viewed as a tensor of order C. Each index of the tensor maps to the fluorophore counts of a different color, and the

value of an index i_c indicates the number of functioning fluorophores of color c, and satisfies $0 \leq i_c \leq \Lambda_c$. We also have indices j_c which are similarly defined. Since the transition matrix is a linear mapping from and to this tensor of order C, it is necessarily of order $2C$. We use the Einstein summation convention, and three multi-indices $\boldsymbol{i} = i_1 i_2 \ldots i_C$ and $\boldsymbol{j} = j_1 j_2 \ldots j_C$ and $\boldsymbol{k} = k_1 k_2 \ldots k_C$ for convenience. the matrix vector multiplication operation for one step of the HMM forward algorithm is then given by:

$$\boldsymbol{\mathcal{f}}_{\boldsymbol{k}}^{(t+1)} = \boldsymbol{\mathcal{O}}_{\boldsymbol{kj}}^{(t+1)} \boldsymbol{\mathcal{T}}_{\boldsymbol{ji}} \boldsymbol{\mathcal{f}}_{\boldsymbol{i}}^{(t)} \tag{12}$$

Where (t) and $(t+1)$ indicate the timestamp of the values in the order C tensor $\boldsymbol{\mathcal{f}}^{(t)}$, which is indexed by the numbers of working fluorophores for each color and is the tensor form of $\boldsymbol{f}$ from (1), $\boldsymbol{\mathcal{T}}$ is the transition matrix $\boldsymbol{T}$ converted into tensor form, $\boldsymbol{\mathcal{O}}$ is the emission matrix $\boldsymbol{O}$ converted into tensor form.

Considering fluorophore loss only

Assuming no interactions between different fluorophores and ignoring Edman degradation and peptide detachment, $\boldsymbol{\mathcal{T}}$ satisfies the following equation:

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{ji}} = \begin{cases} \prod_{c=1}^{C} \binom{i_c}{j_c} p_c^{i_c - j_c} (1 - p_c)^{j_c}, & \text{if } \boldsymbol{j} \leq \boldsymbol{i} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

Where p_c is the per cycle dye loss rate of the fluorophores for color c. This is simply the product of the binomial distributions for each indexed color of fluorophore. To improve the sparsity of this representation, we can factor $\boldsymbol{\mathcal{T}}$ into second order tensors $\boldsymbol{\mathcal{B}}^{(1)} \boldsymbol{\mathcal{B}}^{(2)} \ldots \boldsymbol{\mathcal{B}}^{(C)}$ such that:

$$\boldsymbol{\mathcal{B}}_{ji}^{(c)} = \begin{cases} \binom{i}{j} p_c^{i-j} (1 - p_c)^{j}, & \text{if } j \leq i \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

This produces a factorization of $\boldsymbol{\mathcal{T}}$:

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{ji}} = \boldsymbol{\mathcal{B}}_{j_1 i_1}^{(1)} \boldsymbol{\mathcal{B}}_{j_2 i_2}^{(2)} \ldots \boldsymbol{\mathcal{B}}_{j_C i_C}^{(C)} \tag{15}$$

We can plug this into (12) and find:

$$\boldsymbol{\mathcal{f}}_{\boldsymbol{j}}^{(t+1)} = \boldsymbol{\mathcal{B}}_{j_1 i_1}^{(1)} \boldsymbol{\mathcal{B}}_{j_2 i_2}^{(2)} \dots \boldsymbol{\mathcal{B}}_{j_C i_C}^{(C)} \boldsymbol{\mathcal{f}}_{\boldsymbol{i}}^{(t)} \tag{16}$$

This reduces the algorithmic complexity in this simple case from $O\left(\prod_{c=1}^{C} \Lambda_c^2\right)$ to $O\left(\left(\prod_{c=1}^{C} \Lambda_c\right)\left(\sum_{c=1}^{C} \Lambda_c\right)\right)$.

Fluorophore loss and Edman degradation

We can expand on this to consider the Edman degradation: In that case we need more indices for the number of remaining amino acids. We modify (12) with additional indices u and v which satisfy $0 \leq u \leq \alpha$ and $0 \leq v \leq \alpha$, indicating the number of successful amino acid removals, or alternatively the position of an amino acid in the peptide (i. e., the amino acid at the N-terminus of the peptide when u amino acids have been removed). This gives:

$$\boldsymbol{\mathcal{f}}_{\boldsymbol{vk}}^{(t+1)} = \boldsymbol{\mathcal{O}}_{\boldsymbol{kj}}^{(t+1)} \boldsymbol{\mathcal{T}}_{\boldsymbol{vjui}} \boldsymbol{\mathcal{f}}_{\boldsymbol{ui}}^{(t)} \tag{17}$$

Note that the emission tensor $\boldsymbol{\mathcal{O}}$ is unaffected by the amino acid count, and depends only on the fluorophore counts, so it does not need to be modified.

We modify $\boldsymbol{\mathcal{T}}$ from (13) to model Edman degradation, and the exact form of $\boldsymbol{\mathcal{T}}$ will depend on the peptide under consideration. Let $\bar{c}_u$ be a number indicating the color of fluorophore at position u in the peptide, with a value of 0 indicating no fluorophore, and let $\lambda_{u,\bar{c}_u}$ indicate the number of fluorophores of color $\bar{c}_u$ remaining when $u-1$ amino acids have been removed from the peptide. Then $\boldsymbol{\mathcal{T}}$ is defined by:

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{vjui}} = \begin{cases} e\beta(\boldsymbol{i},\boldsymbol{j}), & \text{if } \boldsymbol{j} \leq \boldsymbol{i} \text{ and } v = u \\ (1-e)\beta(\boldsymbol{i},\boldsymbol{j}), & \text{if } \boldsymbol{j} \leq \boldsymbol{i} \text{ and } v = u+1 \text{ and } \bar{c}_u = 0 \\ (1-e)\left(\left(1 - \dfrac{i_{\bar{c}_u}}{\lambda_{u,\bar{c}_u}}\right)\beta(\boldsymbol{i},\boldsymbol{j}) + \left(\dfrac{i_{\bar{c}_u}}{\lambda_{u,\bar{c}_u}}\right)\bar{\beta}(\boldsymbol{i},\boldsymbol{j},u)\right), & \text{if } \boldsymbol{j} \leq \boldsymbol{i} \text{ and } v = u+1 \text{ and } \bar{c}_u > 0 \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

Where:

$$\beta(\boldsymbol{i},\boldsymbol{j}) = \prod_{c=1}^{C} \binom{i_c}{j_c} p_c^{i_c - j_c} (1 - p_c)^{j_c} \tag{19}$$

And:

$$\bar{\beta}(\boldsymbol{i},\boldsymbol{j},u) = \binom{i_{\bar{c}_u} - 1}{j_{\bar{c}_u}} p_{\bar{c}_u}^{i_{\bar{c}_u} - 1 - j_{\bar{c}_u}} \left(1 - p_{\bar{c}_u}\right)^{j_{\bar{c}_u}} \prod_{\substack{1 \le c \le C \\ c \ne \bar{c}_u}} \binom{i_c}{j_c} p_c^{i_c - j_c} (1 - p_c)^{j_c} \tag{20}$$

The probability of an Edman degradation failure is essentially the same as in (13), but multiplied by e to account for the probability of failure. The probability for a transition involving a successful Edman degradation event which removes an unlabelable amino acid is similarly just like in (13) but multiplied by $(1 - e)$, the probability of success. If the amino acid in question is labelable by a color $\bar{c}_u$, then we may or may not remove a fluorophore of that color in the transition, so we need to take the sum of both possibilities. β in (19) gives the standard product of binomials formula from (13), but needs to be multiplied by the probability of no dye loss, which in (18) is $\left(1 - \frac{i_{\bar{c}_u}}{\lambda_{u,\bar{c}_u}}\right)$. This is then summed with $\bar{\beta}$ from (20) which gives the product of binomial probabilities starting with one less fluorophore of the color $\bar{c}_u$, which in (18) is multiplied with the probability of losing a fluorophore with the Edman degradation, $\frac{i_{\bar{c}_u}}{\lambda_{u,\bar{c}_u}}$. The sum of these two possibilities is then multiplied by the probability of an Edman degradation success, given by $(1 - e)$.

To make this more efficient, we introduce a new tensor $\boldsymbol{\mathcal{E}}$ which represents a transformation for Edman degradation. We define tensor $\boldsymbol{\mathcal{E}}$ as:

$$\boldsymbol{\mathcal{E}}_{\boldsymbol{vkuj}} = \begin{cases} e, & \text{if } v = u \text{ and } \boldsymbol{k} = \boldsymbol{j} \\ 1 - e, & \text{if } v = u + 1 \text{ and } \boldsymbol{k} = \boldsymbol{j} \text{ and } \bar{c}_u = 0 \\ (1-e)\left(1 - \frac{j_{\bar{c}_u}}{\lambda_{u,\bar{c}_u}}\right), & \text{if } v = u + 1 \text{ and } \boldsymbol{k} = \boldsymbol{j} \text{ and } \bar{c}_u > 0 \\ (1-e)\left(\frac{j_{\bar{c}_u}}{\lambda_{u,\bar{c}_u}}\right), & \text{if } v = u + 1 \text{ and } k_{\bar{c}_u} = j_{\bar{c}_u} - 1 \text{ and } k_c = j_c \forall c \neq \bar{c}_u \text{ and } \bar{c}_u > 0 \\ 0, & \text{otherwise} \end{cases} \quad (21)$$

This provides the following factorization of $\boldsymbol{\mathcal{T}}$:

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{vkui}} = \boldsymbol{\mathcal{E}}_{\boldsymbol{vkuj}} \boldsymbol{\mathcal{B}}^{(1)}_{j_1 i_1} \boldsymbol{\mathcal{B}}^{(2)}_{j_2 i_2} \dots \boldsymbol{\mathcal{B}}^{(C)}_{j_C i_C} \quad (22)$$

By substituting into (17) and adding an additional multi-index $\boldsymbol{l} = l_1 l_2 \dots l_C$ we get:

$$\boldsymbol{\mathcal{f}}^{(t+1)}_{\boldsymbol{vl}} = \boldsymbol{\mathcal{O}}^{(t+1)}_{\boldsymbol{lk}} \boldsymbol{\mathcal{E}}_{\boldsymbol{vkuj}} \boldsymbol{\mathcal{B}}^{(1)}_{j_1 i_1} \boldsymbol{\mathcal{B}}^{(2)}_{j_2 i_2} \dots \boldsymbol{\mathcal{B}}^{(C)}_{j_C i_C} \boldsymbol{\mathcal{f}}^{(t)}_{\boldsymbol{ui}} \quad (23)$$

Despite its high dimensionality, $\boldsymbol{\mathcal{E}}$ is highly sparse, with no more than three non-zero entries per column (here, meaning column in the original non-tensor form matrix). This reduces the algorithmic complexity from $O(\alpha \prod_{c=1}^{C} \Lambda_c^2)$ to $O\left(\alpha\left(\prod_{c=1}^{C} \Lambda_c\right)\left(\sum_{c=1}^{C} \Lambda_c\right)\right)$. We note that while the extraction of the Edman degradation tensor appears to have little direct effect on the algorithmic complexity reduction, which is because it has a sparsity effect on the original transition tensor, properly handling Edman degradation is critical to this decomposition. We feel this is the easiest way to do this while also factoring the fluorophore loss effects into separate tensors.

Everything all together

Handling peptide detachment is simpler. We modify $\boldsymbol{\mathcal{T}}$ to be:

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{vjui}} = \begin{cases} (1-d)e\beta(\boldsymbol{i},\boldsymbol{j},p), & \text{if } \boldsymbol{j} \leq \boldsymbol{i} \text{ and } v = u \\ (1-d)(1-e)\beta(\boldsymbol{i},\boldsymbol{j}), & \text{if } \boldsymbol{j} \leq \boldsymbol{i} \text{ and } v = u + 1 \text{ and } \bar{c}_u = 0 \\ (1-d)(1-e)\left(\left(1 - \frac{i_{\bar{c}_u}}{\lambda_u}\right)\beta(\boldsymbol{i},\boldsymbol{j}) + \left(\frac{i_{\bar{c}_u}}{\lambda_u}\right)\bar{\beta}(\boldsymbol{i},\boldsymbol{j},u)\right), & \text{if } \boldsymbol{j} \leq \boldsymbol{i} \text{ and } v = u + 1 \text{ and } \bar{c}_u > 0 \\ d, & \text{if } j_c = 0 \forall c \text{ and } v = \alpha \\ 0, & \text{otherwise} \end{cases} \quad (24)$$

This creates a new "empty" state which can always be transitioned to with probability d of detachment. The probability of avoiding this state is $(1-d)$. The functions β and $\bar{\beta}$ are the same as before in (19) and (20). The matrix vector multiplication step of the HMM forward algorithm has not changed from (17). We can then construct a new tensor $\boldsymbol{\mathcal{D}}$ for peptide detachment which satisfies:

$$\boldsymbol{\mathcal{D}}_{\boldsymbol{whvk}} = \begin{cases} 1-d, & \text{if } \boldsymbol{h} = \boldsymbol{k} \text{ and } w = v \leq \alpha \\ d & \text{if } h_c = 0 \forall c \text{ and } w = \alpha + 1 \end{cases} \tag{25}$$

Then we find that:

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{wlui}} = \boldsymbol{\mathcal{D}}_{\boldsymbol{wlvk}} \boldsymbol{\mathcal{E}}_{\boldsymbol{vkuj}} \boldsymbol{\mathcal{B}}^{(1)}_{j_1 i_1} \boldsymbol{\mathcal{B}}^{(2)}_{j_2 i_2} \dots \boldsymbol{\mathcal{B}}^{(C)}_{j_C i_C} \tag{26}$$

Substituting into (17) with another multi-index $\boldsymbol{m} = m_1 m_2 \dots m_C$ provides:

$$\boldsymbol{\mathcal{f}}^{(t+1)}_{\boldsymbol{wm}} = \boldsymbol{\mathcal{O}}^{(t+1)}_{\boldsymbol{ml}} \boldsymbol{\mathcal{D}}_{\boldsymbol{wlvk}} \boldsymbol{\mathcal{E}}_{\boldsymbol{vkuj}} \boldsymbol{\mathcal{B}}^{(1)}_{j_1 i_1} \boldsymbol{\mathcal{B}}^{(2)}_{j_2 i_2} \dots \boldsymbol{\mathcal{B}}^{(C)}_{j_C i_C} \boldsymbol{\mathcal{f}}^{(t)}_{\boldsymbol{ui}} \tag{27}$$

$\boldsymbol{\mathcal{D}}$ is clearly highly sparse, with two entries in each column of the original matrix in non-tensor form. Thus, $\boldsymbol{\mathcal{D}}$ has no impact on the algorithmic complexity of this operation. Although $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{E}}$ could be combined to achieve this same algorithmic improvement, we found that this separation made our model easier to reason about and work with.

Transition matrix factoring conclusions

One of the benefits of this approach to algorithmic complexity reduction is that this factorization provides no loss to the theoretical accuracy of the forward algorithm. No theoretical approximations were necessary, aside from the unavoidable differences in floating-point round-off errors. This allows for highly accurate results with much more efficient runtime characteristics than a naïve implementation.

A3 – HMM pruning

Because the emission matrix is diagonal, it is equivalent to the diagonal part of its Singular Value Decomposition (SVD), but with a reordering of its indices. This makes sparsification of this matrix equivalent to the Eckart-Young-Mirsky theorem; we can keep the largest r values for some chosen value of r, and replace the rest of the matrix entries with zeros, having the minimum possible impact on the spectral and Frobenius norms for the chosen value of r.

Furthermore, we can propagate this sparsification to the transition matrix. Consider the forward algorithm, with $\boldsymbol{T}$ representing the transition matrix, and $\boldsymbol{O}^{(t)}$ representing the diagonal emission matrix for time t. Then if $\boldsymbol{f}^{(t)}$ represents the vector of intermediate probabilities at time t, we have:

$$\boldsymbol{f}^{(t+1)} = \boldsymbol{O}^{(t+1)}\boldsymbol{T}\boldsymbol{f}^{(t)} \tag{28}$$

Now we sparsify each $\boldsymbol{O}^{(t)}$ as discussed above, to get a series of $\widehat{\boldsymbol{O}}^{(t)}$. This gives:

$$\boldsymbol{f}^{(t+1)} = \widehat{\boldsymbol{O}}^{(t+1)}\boldsymbol{T}\boldsymbol{f}^{(t)} \tag{29}$$

Note that we have many copies of $\boldsymbol{T}$, which are equal. For our next improvements we need these to be different for each timestep, so we can rewrite (29) with $\boldsymbol{T}^{(t)}$ for each timestep t, giving

$$\boldsymbol{f}^{(t+1)} = \widehat{\boldsymbol{O}}^{(t+1)}\boldsymbol{T}^{(t)}\boldsymbol{f}^{(t)} \tag{30}$$

Here the values of many rows and columns of each $\boldsymbol{T}^{(t)}$ have been made unnecessary by the sparsification of its neighboring $\widehat{\boldsymbol{O}}^{(t+1)}$ and $\widehat{\boldsymbol{O}}^{(t)}$, as any vector product with $\widehat{\boldsymbol{O}}^{(t)}$ will necessarily have zeros except for the r entries retained, such that we need only keep the corresponding r columns of $\boldsymbol{T}^{(t)}$. Similarly, any entry in the vector product with $\boldsymbol{T}^{(t)}$ which is not multiplied by one of the r entries retained in $\widehat{\boldsymbol{O}}^{(t+1)}$ is multiplied by zeros, and is thus unnecessary, so we need only keep the corresponding r rows of $\boldsymbol{T}^{(t)}$. Calling these approximations $\widehat{\boldsymbol{T}}^{(t)}$, we get

$$\boldsymbol{f}^{(t+1)} = \widehat{\boldsymbol{O}}^{(t+1)}\widehat{\boldsymbol{T}}^{(t)}\boldsymbol{f}^{(t)} \tag{31}$$

This allows significant sparsity to be used (**Figure 5**). Previously this formula would have been $O(\alpha^2 T \prod_{c=1}^{C} \Lambda_c^2)$ to compute, while this reduces the algorithmic complexity to $O(r^2 T)$. This improvement is beyond what is possible in a more traditional usage of sparse matrix multiplication. For sparse matrix multiplication, we would need to first multiply $\widehat{\boldsymbol{T}}^{(t)}$ by $\widehat{\boldsymbol{O}}^{(t)}$ or multiply $\widehat{\boldsymbol{O}}^{(t+1)}$ by $\widehat{\boldsymbol{T}}^{(t)}$. This will only permit you to sparsify your operations on the rows or the columns of $\boldsymbol{T}$ but not both, giving a complexity of $O\left(r\alpha T \prod_{c=1}^{C} \Lambda_c\right)$. While this is better than not using this inherent sparsity at all, preprocessing the transition matrix in consideration of the emission matrices on either side gives better results in algorithmic complexity.

In practice, we use a more complicated pruning scheme, as detailed next.

A4 – Combining transition matrix factoring with HMM pruning

By making r suitably small, HMM pruning can exhibit better algorithmic complexity than if we factor the transition matrix. However, we believe it is much better to combine these algorithmic enhancements (**Figure 6**). To do this, we need to switch into tensor notation, replacing our matrices and vectors with the tensor equivalents we constructed previously. This yields:

$$\boldsymbol{\mathcal{f}}_{\boldsymbol{vk}}^{(t+1)} = \widehat{\boldsymbol{\mathcal{O}}}_{\boldsymbol{kj}}^{(t+1)}\widehat{\boldsymbol{\mathcal{T}}}_{\boldsymbol{vjui}}^{(t)}\boldsymbol{\mathcal{f}}_{\boldsymbol{ui}}^{(t)} \tag{32}$$

We also want to use the factorization from (26), using timestamp specific sub-tensors of each of the factored pieces. The factorization of (26) becomes:

$$\widehat{\boldsymbol{\mathcal{T}}}_{\boldsymbol{wlui}}^{(t)} = \widehat{\boldsymbol{\mathcal{D}}}_{\boldsymbol{wlvk}}^{(t)}\widehat{\boldsymbol{\mathcal{E}}}_{\boldsymbol{vkuj}}^{(t)}\widehat{\boldsymbol{\mathcal{B}}}_{j_1 i_1}^{(t,1)}\widehat{\boldsymbol{\mathcal{B}}}_{j_2 i_2}^{(t,2)} \ldots \widehat{\boldsymbol{\mathcal{B}}}_{j_C i_C}^{(t,C)} \tag{33}$$

Substituting into (32) gives:

$$\boldsymbol{\mathcal{f}}_{\boldsymbol{wm}}^{(t+1)} = \widehat{\boldsymbol{\mathcal{O}}}_{\boldsymbol{ml}}^{(t+1)}\widehat{\boldsymbol{\mathcal{D}}}_{\boldsymbol{wlvk}}^{(t)}\widehat{\boldsymbol{\mathcal{E}}}_{\boldsymbol{vkuj}}^{(t)}\widehat{\boldsymbol{\mathcal{B}}}_{j_1 i_1}^{(t,1)}\widehat{\boldsymbol{\mathcal{B}}}_{j_2 i_2}^{(t,2)} \ldots \widehat{\boldsymbol{\mathcal{B}}}_{j_C i_C}^{(t,C)}\boldsymbol{\mathcal{f}}_{\boldsymbol{ui}}^{(t)} \tag{34}$$

Suppose we were to use standard sparse tensor multiplication techniques and carry this operation out from right to left. Each tensor $\widehat{\boldsymbol{\mathcal{B}}}^{(t,c)}_{j_c i_c}$ can introduce any entry of input (index i_c) into as many as Λ_c indices of output (index j_c). The resulting computational complexity of the forward algorithm, even with the given sparsity, is then $O\left(r\alpha T \prod_{c=1}^{C} \Lambda_c\right)$.

If we preprocess the computation, pruning each operation now from both directions, the algorithmic complexity does not improve the way it does in the matrix case, although likely this would behave faster in practice. The problem is that the pruning operation itself needs to determine which rows to propagate forwards, which requires accessing every non-zero entry reachable in the forward direction. Many of these values are later pruned in the backwards direction, so the computation itself has much better sparsity, but the time to prune then dominates the algorithmic complexity result.

To improve this further, we add structure to the pruning of $\widehat{\boldsymbol{\mathcal{O}}}^{(t)}$. Instead of keeping the r largest values in $\widehat{\boldsymbol{\mathcal{O}}}^{(t)}$, we prune each index of $\widehat{\boldsymbol{\mathcal{O}}}^{(t)}$ independently. For additional convenience, we limit each index to a contiguous range of values. Then we let each index for any fluorophore color c have r_c values and allow $\bar{r}$ values to index the number of amino acids. These simplifications may cause the pruning to be non-optimal, but we accept this trade-off.

We can then prune our tensors using their known structures (for example, the tensors $\widehat{\boldsymbol{\mathcal{B}}}^{(t,c)}$ correspond to an upper triangular matrix). This time when we propagate the pruning results in both directions, the time required is only $O(C^2)$ (the number of minimum and maximum indices to be propagated through each tensor scales with C, as does the number of tensors to be pruned). For the runtime of the tensor operations, consider each tensor individually. $\widehat{\boldsymbol{\mathcal{D}}}^{(t)}$ and $\widehat{\boldsymbol{\mathcal{E}}}^{(t)}$ are both highly sparse, so they contribute a constant modification to the number of rows or columns when propagating in either direction. $\widehat{\boldsymbol{\mathcal{D}}}^{(t)}$ requires special handling. We track the detached state separately from the ordinary range, to avoid unnecessarily including a large range of states which don't need to be.

Then each $\widehat{\boldsymbol{B}}^{(t,c)}$ operates on an independent index, and therefore can be considered on its own. This tensor after pruning will have dimensions that are $O(r_c^2)$, and should have a constant effect on the number of elements input vs output. Therefore, each of these tensors will require an algorithmic complexity of $O\left(r_c \bar{r} \prod_{\tilde{c}=1}^{C} r_{\tilde{c}}\right)$. Bringing this all together we get an algorithmic complexity of $O\left(C^2 + \bar{r}\left(\sum_{c=1}^{C} r_c\right)\left(\prod_{c=1}^{C} r_c\right)\right)$ for processing one timestep. The full forward algorithm then has a complexity of $O\left(T\left(C^2 + \bar{r}\left(\sum_{c=1}^{C} r_c\right)\left(\prod_{c=1}^{C} r_c\right)\right)\right)$.

One remaining clarification is the manner of choosing r_c and $\bar{r}$. In fact, these values should not be kept constant; let us refer to the values for time t as $r_c^{(t)}$ and $\bar{r}^{(t)}$. $\bar{r}^{(0)} = 1$ and $\bar{r}^{(t+1)} = \bar{r}^{(t)} + 1$, due to the possibility of amino acid removal. These on average are proportional to α. To get $r_c^{(t)}$, we keep all index values where a fluorophore count of that value has the observed fluorescence intensity for color c at time t within a specified confidence interval – perhaps within 3σ of the mean, where σ is the standard deviation of the distribution. These will necessarily be contiguous. The number of indices kept is then $r_c^{(t)}$.

The standard deviation of a normal distribution scales with the square root of the intensity, and the number of possible index values is limited by the total possible number of fluorophores of color c. It follows that any removal of index values proportional to the standard deviation will satisfy $r_c^{(t)} < \gamma\sqrt{\Lambda_c}$ for some constant γ dependent on the cutoff. Then the algorithmic complexity is given by $O\left(T\left(C^2 + \alpha\left(\sum_{c=1}^{C} \sqrt{\Lambda_c}\right)\left(\prod_{c=1}^{C} \sqrt{\Lambda_c}\right)\right)\right)$.

We chose a specific pruning cut-off by sweeping this parameter and balancing the experimental runtime effects and the precision-recall curves which result from simulated data.

Appendix B – Supporting figures

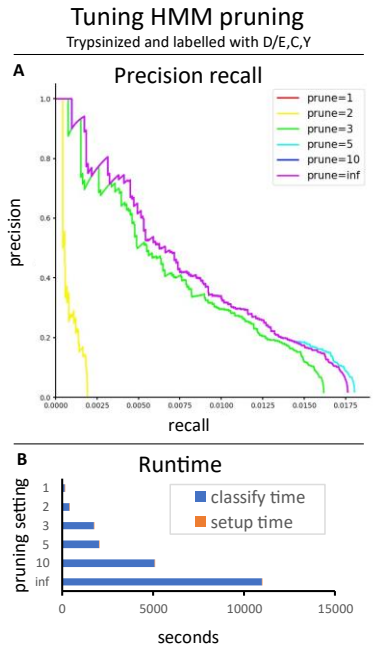

Figure B1. Tuning the pruning parameter of the Bayesian HMM classifier. Setting this parameter to 5 (*i. e.*, 5 σ) appears to provide the best trade-off. **A:** Precision/recall curves.

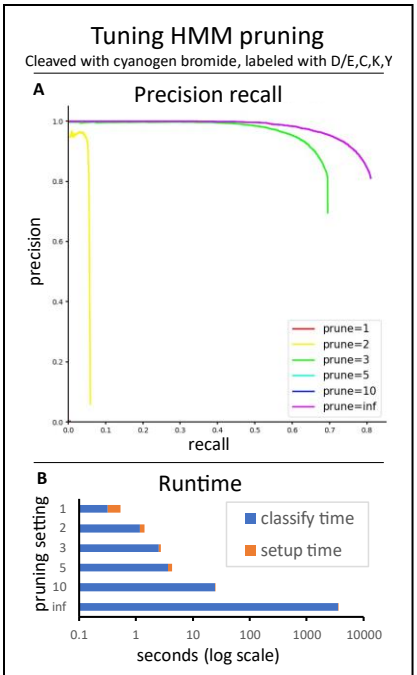

Figure B2. Tuning the pruning parameter of the Bayesian HMM classifier. Here we show a more extraordinary case than **Figure B2**. **A:** Precision/recall curves. **B:** Runtimes.

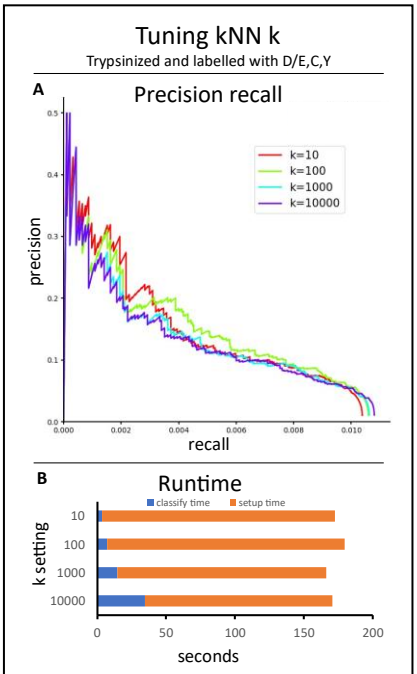

Figure B3. Tuning the k parameter of the kNN classifier. Setting k to 10 seems to provide the best trade-off **A:** Precision/recall curves. **B:**

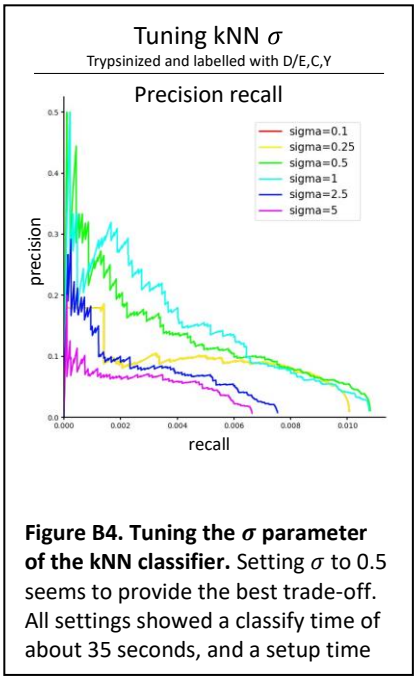

Figure B4. Tuning the σ parameter of the kNN classifier. Setting σ to 0.5 seems to provide the best trade-off. All settings showed a classify time of about 35 seconds, and a setup time

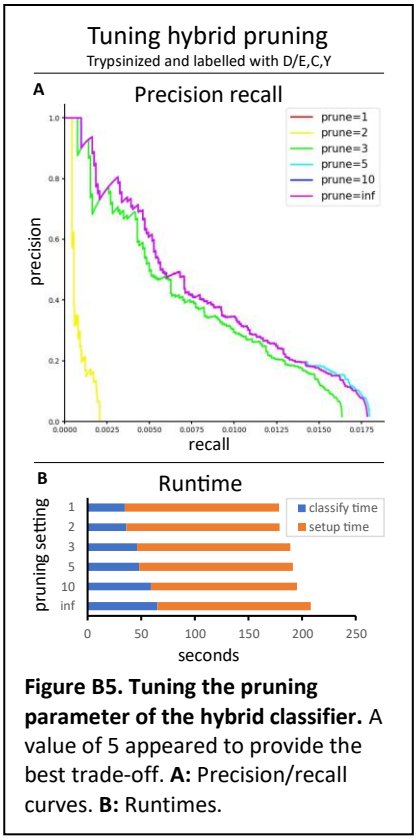

Figure B5. Tuning the pruning parameter of the hybrid classifier. A value of 5 appeared to provide the best trade-off. **A:** Precision/recall curves. **B:** Runtimes.

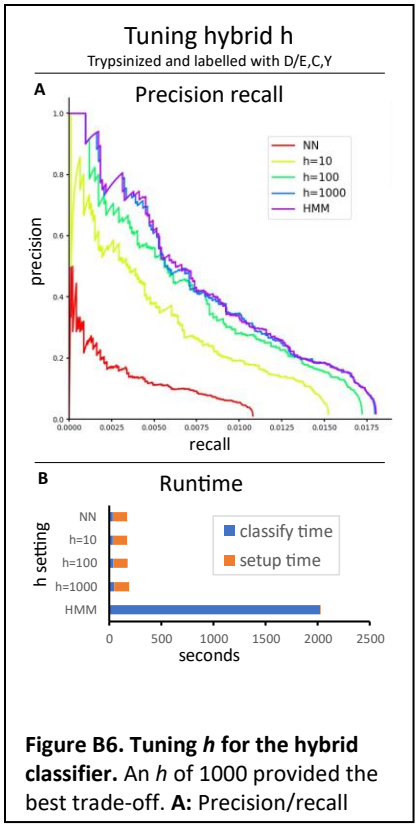

Figure B6. Tuning *h* for the hybrid classifier. An *h* of 1000 provided the best trade-off. **A:** Precision/recall

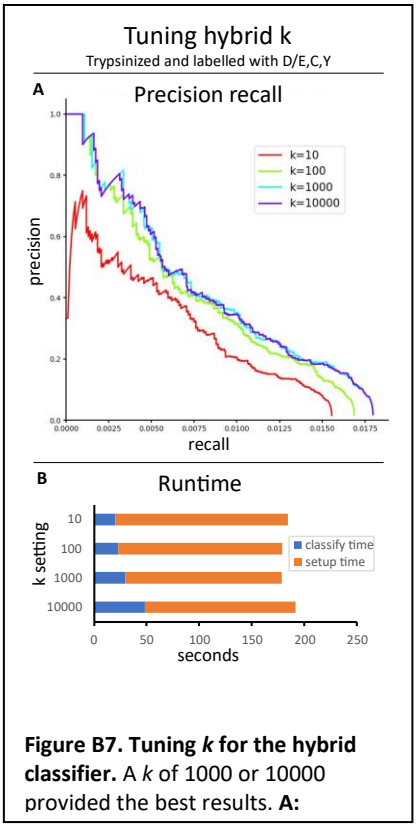

Figure B7. Tuning *k* for the hybrid classifier. A *k* of 1000 or 10000 provided the best results. **A:**

854

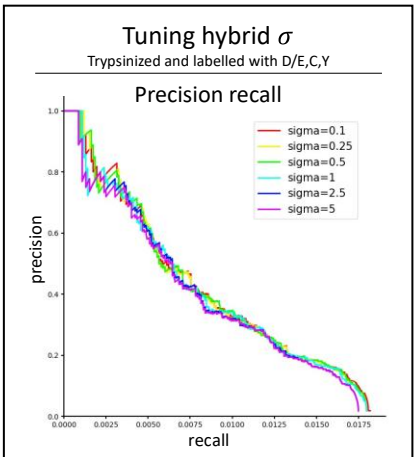

Figure B8. Tuning the σ parameter of the hybrid classifier. Setting σ to 0.5 seemed to provide the best trade-off. All settings had a classify time of about 50 seconds, and a