Journal of proteome research

pubs.acs.org/jpr

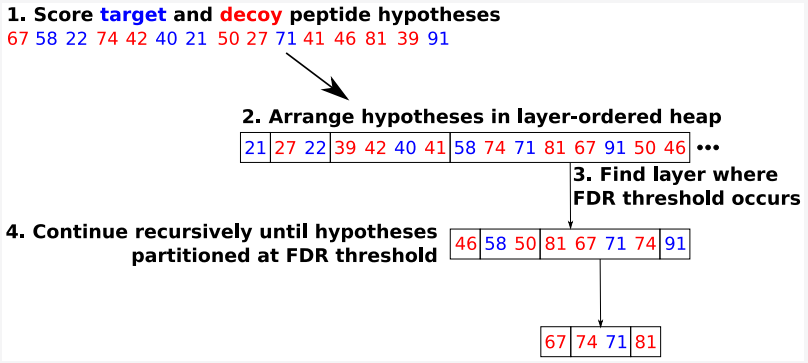
Article

Performing Selection on a Monotonic Function in Lieu of Sorting Using Layer-Ordered Heaps

Kyle Lucke, Jake Pennington, Patrick Kreitzberg, Lukas Käll, and Oliver Serang*

Cite This: *J. Proteome Res.* 2021, 20, 1849–1854

Read Online



ABSTRACT: Nonparametric statistical tests are an integral part of scientific experiments in a diverse range of fields. When performing such tests, it is standard to sort values; however, this requires $\Omega(n\ log(n))$ time to sort n values. Thus given enough data, sorting becomes the computational bottleneck, even with very optimized implementations such as the C++ standard library routine, std::sort. Frequently, a nonparametric statistical test is only used to partition values above and below a threshold in the sorted ordering, where the threshold corresponds to a significant statistical result. Linear-time selection and partitioning algorithms cannot be directly used because the selection and partitioning are performed on the transformed statistical significance values rather than on the sorted statistics. Usually, those transformed statistical significance values (e.g., the p value when investigating the family-wise error rate and q values when investigating the false discovery rate (FDR)) can only be computed at a threshold. Because this threshold is unknown, this leads to sorting the data. Layer-ordered heaps, which can be constructed in $O(n)$, only partially sort values and thus can be used to get around the slow runtime required to fully sort. Here we introduce a layer-ordering-based method for selection and partitioning on the transformed values (e.g., p values or q values). We demonstrate the use of this method to partition peptides using an FDR threshold. This approach is applied to speed up Percolator, a postprocessing algorithm used in mass-spectrometry-based proteomics to evaluate the quality of peptide-spectrum matches (PSMs), by >70% on data sets with 100 million PSMs.

KEYWORDS: *sorting, false discovery rate, Percolator, peptide search, layer-ordered heap, nonparametric statistical test, partition, algorithms, performance, tandem mass spectrometry*

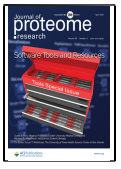
INTRODUCTION

Rank-based nonparametric statistical tests, such as the Wilcoxon signed-rank test,[1] are common operations used across scientific disciplines.[2,3] Their widespread use is primarily due to a lack of parametrization, which is beneficial because no underlying distribution is assumed. Hence such tests can be applied more broadly. This is in contrast with parametric statistical tests such as the t test,[4] which assumes that the data are normal.

Given x_1, x_2, ..., x_n and the identity of the test statistic's hypothesis, the test statistics of hypotheses are ordered such that $x_1 \leq x_2 \ldots \leq x_n$. The statistical test computes a significance value (e.g., p value or q value) of the form $y_i = f(x_i) = g(x_1, x_2, \ldots, x_i)$. Because hypotheses are added best first, the significance score, y_i, is monotonic with i. Sorting n hypotheses' test statistics x_1, x_2, ..., x_n in this manner costs $\Omega(n \log(n))$, which means the amount of work done is always going to be greater than or equal to $n \log(n)$. Once sorted, it is possible to go through every threshold and compute g efficiently. Because sorting is super linear and other routines, such as building the

Special Issue: Software Tools and Resources 2021

Received: September 14, 2020
Published: February 2, 2021



https://dx.doi.org/10.1021/acs.jproteome.0c00711
J. Proteome Res. 2021, 20, 1849–1854

empirical cumulative density function, have become so optimized, sorting may become a computational bottleneck for performing nonparametric statistical tests, especially for problems with large amounts of data.[5]

Linear-time selection[6] can be used to generate the k best x_1, x_2, ..., x_k in any order $\in O(n)$ time, which means the amount of work done is always going to be less than or equal to n, within a constant factor; however, the typical use case for testing is to find the largest rank k such that y_1, y_2, ..., $y_k \leq \tau$ for some threshold τ. Often there is no closed form for f, so it is not possible to simply take the inverse of f and solve directly.

Percolator

Mass-spectrometry-based proteomics is currently the most comprehensive way to analyze the proteome; however, the technology is verbose in that it generates large amounts of data. Consequently, data processing represents one of the more important and time-consuming steps of a mass spectrometry experiment.

A common way to interpret the spectra generated in an experiment is to use search engines that match spectra predicted from a target database of the amino acid sequences of the analyzed proteins to the observed spectra.[7−16] The results of this operation are referred to as peptide-spectrum matches (PSMs), which have an associated score based on how well the theoretical peptide spectra match the observed spectra. Because the approach will match every spectrum, regardless of whether the spectrum was the result of proteins present in the searched sequence database, we are subsequently left with the problem of determining which of the PSMs were results of correctly and incorrectly formed PSMs. This is normally done based on a score threshold for the PSM scores, where all PSMs scoring better than the threshold are considered correct, whereas the ones below the threshold are considered incorrect.

Score thresholds are often selected based on target-decoy analysis.[17] We can assess different statistics of the PSMs above (and below) some threshold by analyzing the score distribution of deliberately incorrectly formed PSMs, referred to as decoy PSMs, stemming from matches against nonpresent protein sequences. Most commonly, one assesses the false discovery rate (FDR).[18] Calculation of the FDR corresponds to the g function. Given a set of sorted hypotheses and their labels, (x_1, T), (x_2, D), ..., (x_n, D), find the largest k such that $g((x_1, T), (x_2, D), ..., (x_k, T)) \leq \tau$, where $g = \frac{\text{no. of decoy arguments}}{\text{no. of target arguments}}$, and the "decoy" and "target" terms correspond to the label of the PSM's hypothesis (e.g., whether the PSM is a target or decoy). Hence g estimates the FDR by looking at the identities of the hypotheses and determining the ratio of decoys to targets. Thus the FDR is essentially a surrogate for the percentage of incorrect discoveries found above a particular score threshold. Note also that in this case, g only cares about the identity of x_i's hypothesis.

Normally, the FDR calculation is not seen as very time-consuming compared with matching spectra to peptides; however, whereas peptide-spectrum matching scales linearly with the number of PSMs, nominal FDR calculations are $\Omega(n \log(n))$ because they sort scores to find the threshold at which to partition scores. Hence given a large enough set of spectra, the FDR calculation will become the most time-consuming step of the entire program. This is true even for the best comparison sorting algorithms, which have been highly optimized.

PSM scoring has been improved by post-processing with machine-learning algorithms. Percolator[19] is one of the leading postprocessing algorithms that integrates more features than just the search engine score into an aggregate score. The PSMs are then sorted by score, and all target and decoy PSMs above a fixed FDR threshold are used as positive and negative examples (respectively) for training a support vector machine (SVM). The trained SVM is then used to rescore the PSMs. This process is iteratively repeated for a user-specified number of times; then, the final scores are produced. Hence, Percolator is dependent on FDR calculations not just for creating a final list of PSMs for the user but also for creating training examples for the SVM. So the FDR calculation, and hence sorting, is also an intrinsic part of the training procedure.

Mass spectrometry experiments produce increasingly larger amounts of data as technology continues to advance. Much effort has been devoted to improving the efficiency of Percolator to ensure that it can meet these demands.[20−22] One of the optimizations relied on improving the SVM training via CGLS, a special conjugate-gradient solver,[23] whereas other optimizations rely on multithreading, which may limit their effectiveness in environments with limited resources.

With these sophisticated optimizations to Percolator's machine-learning algorithms in place, the more classic problem of sorting limits the runtime performance of Percolator, at least for large sets of PSMs. (See Table 1.)

Table 1. Percent of Percolator's Overall Runtime Spent Calculating the FDR Using Sorting[a]

	percent of Percolator's runtime spent calculating the FDR	
number of PSMs	threads = 1	threads = 64
21 028	8%	4%
406 216	38%	38%
601 211	39%	41%
789 071	39%	44%
935 536	40%	48%
1 119 346	40%	48%
100 741 051	N/A	68%

[a]Machine-learning algorithms used by Percolator have become so optimized that a significant bottleneck is sorting the PSM scores to find the score corresponding to the desired q-value threshold. For 100 741 051 PSMs, the single-threaded Percolator did not finish in the 12 h allowed.

Layer-Ordered Heaps

A natural approach would be to employ a faster sorting algorithm; however, comparison sorting $\in \Omega(\log(n!)) = \Omega(n \log(n))$. Instead, a sorting algorithm with a faster runtime constant could also be used; however, std::sort is already well optimized and supports multithreaded CPUs. But simply because Percolator's q-value threshold routine reduces to sorting does not imply that sorting is necessary.

Note again that g does not care about order. Hence only a partial ordering may be needed. For example, the partitioning $x_1, x_2, x_3 \leq x_4, x_5, x_6$... is equal in quality to the partition $x_2, x_1, x_3 \leq x_6, x_4, x_5$... because g cardinally operates by counting arguments, and so the order of arguments is unimportant. Layer-ordered heaps[24] (LOHs) can be used to circumvent the slow runtime of sorting or the repeated use of linear-time selection. Because f is known to be monotonic with x, simply

https://dx.doi.org/10.1021/acs.jproteome.0c00711
J. Proteome Res. 2021, 20, 1849−1854

find the first x_i that crosses a given threshold and keep all y_i less than that. This is similar to divide and conquer on median but will perform much better in practice due to the fact that LOHs are contiguous in memory.

An LOH is a list partitioned into layers, where a layer's elements are guaranteed to be ordered with respect to layers that follow. (Here we will use a strict ordering using the > operator, meaning every value in a layer is strictly greater than the values of subsequent layers.) Note that elements within layers are unordered. An LOH also has the property that asymptotically, the ratio of the sizes of layers is α,[25] meaning that for $\alpha > 1$, the layers grow exponentially.

The parameter α can be chosen arbitrarily but should be chosen to be $\gg 1$. If α is small, then we are closer to sorting the list. (In fact, if $\alpha = 1$, then it is equivalent to sorting.) Similarly, if α is too large, then the layer that contains the desired $f(x_i)$ may be unnecessarily large, which will require many recursions to precisely locate the desired $f(x_i)$. Other than avoiding sorting, the choice of α is otherwise a practical matter that is essentially balancing the runtime constant for data movement against the runtime constant for performing comparisons.

Because ordering between layers is strict, it is possible that a layer cannot be subdivided because the layer contains too many duplicate values, unless this limits the fragmentation of a layer. For a constant α, a list can be lohified (i.e., partitioned into an LOH) in linear time. An example of LOHs with varying α values can be seen in Figure 1. LOHs have been used

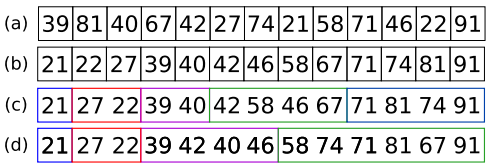


Figure 1. Illustration showing the difference between sorting and layer ordering. The same numbers are shown in (a) random order, (b) sorted order (which is also an LOH with $\alpha = 1$), (c) an LOH with $\alpha = 1.5$, and (d) an LOH with $\alpha = 2$.

in optimal selection on the Cartesian product of two lists[26] and in the fastest known selection on the Cartesian-product of many lists.[24] This approach was recently used to produce NeutronStar,[27] the world's fastest publicly available exact isotope calculator upon release (April 2020).

Quick Lohify

In addition to the traditional LOH, there is also the Quick-LOH. These are constructed by partitioning the list on random elements. In this way, Quick-LOH does not explicitly require an α parameter. Whatever random pivot value is chosen, the elements greater than or equal to the pivot form a contiguous layer to the right of it, whereas the method recurses on the values less than the pivot. This continues until the problem size $\in O(1)$, at which point the list is sorted in constant time. Whereas the worst possible case for constructing a Quick-LOH $\in O(n^2)$, the expected construction time is $\in O(n)$.[25] Thus Quick-LOH behaves similarly to a standard LOH and runs in the same linear time. Unlike a standard LOH, Quick-LOH does not produce an ordering with a constant expected α. $\mathbb{E}[\alpha]$ of a Quick-LOH $\in \Theta(\log(n))$,[25] which means the amount of work done is always going to be roughly equal to $\log(n)$, within a constant factor. Whereas the theoretical behavior of the Quick-LOH is not guaranteed to be as good as a traditional LOH, it is simpler to implement (as it does not require linear-time selection[6]) and often faster in practice. For these reasons, Quick-LOH is the data structure that we chose to use in Percolator.

While not completely sorted, a layer ordering gives us enough information to efficiently calculate the minimum or maximum score threshold at which a particular FDR occurs. Here we describe how to significantly decrease the runtime of Percolator's training procedure by using LOHs to partition the PSMs about an FDR threshold.

METHODS

Percolator calculates the FDR by first sorting the scores and then counting the total number of scores seen before the last time the FDR raises above a given threshold, τ. The ordering between the scores that are below the threshold does not matter; it is only the number of scores that are below the threshold that matters. This means the scores do not have to be fully sorted; however, it is not known in advance where to partition the list. Here we describe how LOHs may be used to partition the scores about τ without having to resort to sorting (Figure 2).

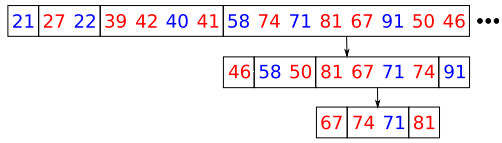


Figure 2. Illustration showing how to partition a list of decoys and targets about an FDR threshold using LOHs. Scores associated with a decoy PSM are shown in red, whereas scores associated with a target PSM are shown in blue. If the given FDR threshold falls between the pessimistic and optimistic FDR values for a layer, then the layer is recursed on. This continues until the score that partitions the PSMs about the FDR threshold is found. Notice that the scores become asymptotically more sorted about the FDR threshold while circumventing the need to sort any other part of the list.

Calculating the FDR Using Layer-Ordered Heaps

To calculate the FDR, the list of scores must first be lohified. If the list was sorted, then one could proceed score-by-score to calculate the FDR, but for a lohified list one, must proceed layer-by-layer. Percolator uses three different procedures to estimate the FDR: target-decoy competition,[17,28] separate target-decoy searches,[19] and the mix-max procedure.[29] Here we will show how to calculate the FDR for a target-decoy competition procedure; however, the same idea can be applied to the other two schemes in a similar manner.

FDR Calculation

$$\mathrm{FDR}(\text{layer } i) = \frac{\sum_{j=0}^{i} D_j}{\sum_{j=0}^{i} T_j} \tag{1}$$

The FDR calculation used in the lohified version can be seen in eq 1 where D_j and T_j are the total number of decoy and target PSMs in layer j, respectively.

First, a linear pass must be made to count the total number of decoys and targets, so that the exact FDR at the beginning of a layer may be calculated. Then, the layers are traversed, starting with the low scoring layers (which will also be the largest layers) and moving toward the high scoring layers.

https://dx.doi.org/10.1021/acs.jproteome.0c00711
J. Proteome Res. 2021, 20, 1849−1854

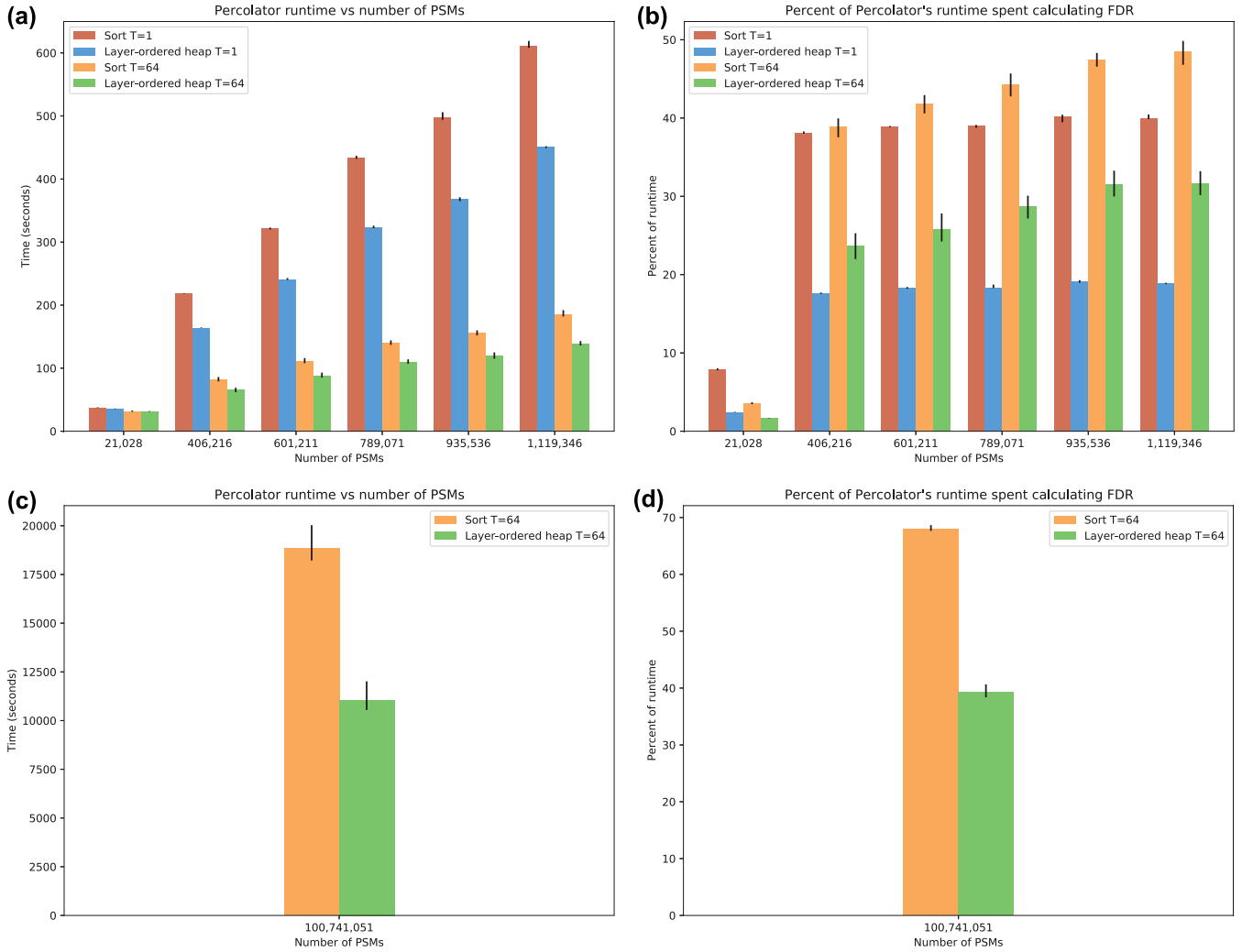


Figure 3. Runtimes for Percolator when calculating the FDR using sorting versus using layer-ordered heaps. (a,c) Overall runtimes of Percolator. (b,d) Percentage of the overall runtime spent calculating the FDR. The smallest data set was created by selecting PSMs from mass spectrometry samples on a standard 18 protein mixture. There were six different larger data sets created by selecting PSMs from mass spectrometry samples on 31 human tissue types. The samples range in size from 21 028 to 100 741 051 PSMs. On all data sets but the largest (a,b), reported times and percentages were averaged over 10 iterations, and each method was implemented using 1 thread and using 64 threads. On the largest data set (c,d), reported times and percentages were averaged over three iterations due to the decreased variability in results and a roughly 80-fold increase in the runtime. The single threaded implementations did not finish in the 12 h allowed. Error bars are included to indicate the minimum and maximum run times observed.

For each layer, the most pessimistic and the most optimistic FDRs are calculated. The pessimistic FDR assumes that all decoys in the layer are seen before any targets, and the optimistic FDR assumes the opposite. Because the FDR is known at the start of the layer, these bounds are exact. If τ falls between the bounds of the layer, then the layer is recursed upon. Ties are handled during the lohification of the list by being put into the same layer.

The recursion on a layer applies the same algorithm to the layer as it did the entire list with one exception: It must pass the cumulative number of decoys and targets in all layers whose scores are higher than the current one so that the FDR calculations in the recursions may also be exact. The algorithm stops recursing on a layer if τ no longer lies between the pessimistic and the optimistic FDRs or the layer size is small enough to be sorted in constant time. Because the layer is now sorted and the FDR at the beginning of the layer is known, the result is exact.

There are three special cases when investigating a layer that allow the layer to be immediately discarded or the correct score to be returned without any recursions. If the layer is all targets, then the FDR can only decrease over the layer, so it may be skipped. If the layer is all decoys, then the FDR at the end of the layer can be calculated directly from eq 1. The third special case is the entire layer has the same score, in which case the exact FDR at the end of the layer is able to be calculated without any recursions.

RESULTS

All experiments were run on a workstation equipped with 256 GB of RAM and two AMD EPYC 7351 processors with a total of 64 threads running Ubuntu 18.04.4 LTS.

Runtime

Substituting the LOH method for calculating the FDR has resulted in a significant speed-up over the conventional sorting method in both single-threaded and multithreaded environ-

https://dx.doi.org/10.1021/acs.jproteome.0c00711
J. Proteome Res. 2021, 20, 1849−1854

ments. The smaller data set was obtained by taking 21 028 PSMs from the ISB standard 18 protein mixture, a mixture of 18 proteins from various organisms.[30] The six larger data sets were obtained by taking between 406 211 and 100 741 051 PSMs from a data set that is made from 19 433 acquisitions on 31 human tissue types.[31] The runtimes for Percolator with the two separate methods of calculating the FDR are shown in the left column of Figure 3, where we see a speed-up of up to 36 and 70% for the single- and multithreaded versions, respectively. The time spent in the FDR calculating routine as a percentage of the overall runtime of Percolator is shown in the right column of Figure 3. The average percent of time spent calculating the FDR decreases from 40 to 19% for the single-threaded and from 48 to 30% for the multithreaded. Note that the times for the largest data set were averaged over three iterations due to the decreased variability in results. The single-threaded implementation did not finish.

DISCUSSION

In this Article, we describe how to use an LOH to partition a list of scores about a given FDR threshold. We also demonstrate the practical performance of this routine over sorting by modifying the shotgun proteomics postprocessing software package Percolator to use our new routine.

Percolator is an important and commonly used postprocessing step for shotgun proteomics. The machine-learning algorithms used in Percolator have been optimized for speed, but this makes sorting the scores a performance bottleneck, which we have shown to be alleviated by the use of LOHs. It is also important to note that there are two different processing steps within Percolator that require an FDR calculation. The first step uses the FDR to partition the PSMs about that threshold for subsequent SVM training. The second step uses the FDR to calculate q values for the final list of PSMs. In this Article, we demonstrated how Percolator's processing step can be sped up by modifying the first subroutine; however, we have not been able to speed up the second step. One potential future improvement to Percolator is to optimize the q-value calculation of the second step by using, for example, approximate sorting.

The speedup to the FDR calculation is very robust; however, is it important to note that the overall speedup will depend on what portion of the total runtime is spent in the FDR calculation. With very old hardware, it is conceivable that the SVM training in Percolator may put more stress on the hardware (e.g., due to a significant number of nonsequential random access operations while training the SVM, which could overwhelm a very small CPU cache).

The significant decrease in runtime may be attributed to three main benefits of using LOHs. First, a list may be lohified in linear time as opposed to the $\Omega(n \log(n))$ time required with sorting. Second, if we find the FDR cutoff in layer L_i, then we will never look at layers $L_{0,\ldots,i-1}$ beyond the initial lohification of the list. Third, it is unlikely that we will recurse on layers $L_{i+1,\ldots}$ to find the exact cutoff, meaning that we will likely only need a single linear pass to count the decoys and targets in the layer.

Because FDR calculations and the subsequent subset selection based on FDR thresholds are not domain-specific problems unique to Percolator, this algorithm could be used in many different parts of the data-processing pipeline for mass-spectrometry-based proteomics as well as any other types of expression analysis of biopolymers or small molecules. Hence, our algorithm is widely applicable.

AUTHOR INFORMATION

Corresponding Author

Oliver Serang − *Department of Computer Science, University of Montana, Missoula, Montana 59812, United States;* orcid.org/0000-0003-1245-7051; Email: Oliver.Serang@umontana.edu

Authors

Kyle Lucke − *Department of Computer Science, University of Montana, Missoula, Montana 59812, United States*

Jake Pennington − *Department of Mathematics, University of Montana, Missoula, Montana 59812, United States*

Patrick Kreitzberg − *Department of Mathematics, University of Montana, Missoula, Montana 59812, United States*

Lukas Käll − *Science for Life Laboratory, KTH Royal Institute of Technology, Solna 171 65, Sweden;* orcid.org/0000-0001-5689-9797

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jproteome.0c00711

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by grant number 1845465 from the National Science Foundation. It was also supported by a grant to L.K. from the Chan Zuckerberg Initiative (EOSS2-0000000115).

REFERENCES

(1) Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biom. Bull.* **1945**, *1*, 80−83.

(2) Wieand, S.; Gail, M. H.; James, B. R.; James, K. L. A Family of Nonparametric Statistics for Comparing Diagnostic Markers with Paired or Unpaired Data. *Biometrika* **1989**, *76*, 585−592.

(3) Siddiqui, S. H.; Parizek, R. R. Application of Nonparametric Statistical Tests in Hydrogeology. *Groundwater* **1972**, *10*, 26−30.

(4) Student. The Probable Error of a Mean. *Biometrika* **1908**, *6*, 1−25.

(5) Ekvall, M.; Höhle, M.; Käll, L. Parallelized calculation of permutation tests. *Bioinformatics* **2020**, btaa1007.

(6) Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. Time bounds for selection. *Journal of Computer and System Sciences* **1973**, *7*, 448−461.

(7) Eng, J. K.; McCormack, A. L.; Yates, J. R., III An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *J. Am. Soc. Mass Spectrom.* **1994**, *5*, 976−989.

(8) Perkins, D. N.; Pappin, D. J. C.; Creasy, D. M.; Cottrell, J. S. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis* **1999**, *20*, 3551−3567.

(9) Eng, J. K.; Jahan, T. A.; Hoopmann, M. R. Comet: An open-source MS/MS sequence database search tool. *Proteomics* **2013**, *13*, 22−24.

(10) Fenyo, D.; Beavis, R. C. A method for assessing the statistical significance of mass spectrometry-based protein identification using general scoring schemes. *Anal. Chem.* **2003**, *75*, 768−774.

(11) Dorfer, V.; Pichler, P.; Stranzl, T.; Stadlmann, J.; Taus, T.; Winkler, S.; Mechtler, K. MS Amanda, a universal identification algorithm optimized for high accuracy tandem mass spectra. *J. Proteome Res.* **2014**, *13*, 3679−3684.

(12) Kim, S.; Pevzner, P. A. MS-GF+ makes progress towards a universal database search tool for proteomics. *Nat. Commun.* **2014**, *5*, 5277.
(13) Cox, J.; Neuhauser, N.; Michalski, A.; Scheltema, R. A.; Olsen, J. V.; Mann, M. Andromeda: A Peptide Search Engine Integrated into the MaxQuant Environment. *J. Proteome Res.* **2011**, *10*, 1794−1805.
(14) Clauser, K. R.; Baker, P. R.; Burlingame, A. L. Role of accurate mass measurement (± 10 ppm) in protein identification strategies employing MS or MS/MS and database searching. *Anal. Chem.* **1999**, *71*, 2871.
(15) Geer, L. Y.; Markey, S. P.; Kowalak, J. A.; Wagner, L.; Xu, M.; Maynard, D. M.; Yang, X.; Shi, W.; Bryant, S. H. Open mass spectrometry search algorithm. *J. Proteome Res.* **2004**, *3*, 958−964.
(16) Tabb, D. L.; Fernando, C. G.; Chambers, M. C. MyriMatch: highly accurate tandem mass spectral peptide identification by multivariate hypergeometric analysis. *J. Proteome Res.* **2007**, *6*, 654−661.
(17) Elias, J. E.; Gygi, S. P. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. *Nat. Methods* **2007**, *4*, 207−214.
(18) Benjamini, Y.; Hochberg, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society B* **1995**, *57*, 289−300.
(19) Käll, L.; Canterbury, J.; Weston, J.; Noble, W. S.; MacCoss, M. J. A semi-supervised machine learning technique for peptide identification from shotgun proteomics datasets. *Nat. Methods* **2007**, *4*, 923−25.
(20) Spivak, M.; Weston, J.; Bottou, L.; Käll, L.; Noble, W. S. Improvements to the Percolator algorithm for peptide identification from shotgun proteomics data sets. *J. Proteome Res.* **2009**, *8*, 3737−3745.
(21) The, M.; MacCoss, M. J.; Noble, W. S.; Käll, L. Fast and accurate protein false discovery rates on large-scale proteomics data sets with percolator 3.0. *J. Am. Soc. Mass Spectrom.* **2016**, *27*, 1719−1727.
(22) Halloran, J. T.; Zhang, H.; Kara, K.; Renggli, C.; The, M.; Zhang, C.; Rocke, D. M.; Käll, L.; Noble, W. S. Speeding Up Percolator. *J. Proteome Res.* **2019**, *18*, 3353−3359.
(23) Sindhwani, V.; Keerthi, S. S. Large Scale Semi-Supervised Linear SVMs. *SIGIR '06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* **2006**, 477−484.
(24) Kreitzberg, P.; Lucke, K.; Serang, O. Selection on $X_1 + X_2 + \cdots + X_m$ with Layer-Ordered Heaps. 2020, arXiv:1910.11993. arXiv.org e-Print archive. https://arxiv.org/abs/1910.11993.
(25) Pennington, J.; Kreitzberg, P.; Lucke, K.; Serang, O. Optimal Construction of a Layer-Ordered Heap. 2020, arXiv:2007.13356. arXiv.org e-Print archive. https://arxiv.org/abs/2007.13356.
(26) Serang, O. Optimally Selecting the Top k Values from $X + Y$ with Layer-Ordered Heaps. 2020, arXiv:2001.11607. arXiv.org e-Print archive. https://arxiv.org/abs/2001.11607.
(27) Kreitzberg, P.; Pennington, J.; Lucke, K.; Serang, O. Fast exact computation of the k most abundant isotope peaks with layer-ordered heaps. *Anal. Chem.* **2020**, *92*, 10613−10619.
(28) Elias, J. E.; Gygi, S. P. Target-decoy search strategy for mass spectrometry-based proteomics. *Methods Mol. Biol.* **2010**, *604*, 55−71.
(29) Keich, U.; Kertesz-Farkas, A.; Noble, W. Improved False Discovery Rate Estimation Procedure for Shotgun Proteomics. *J. Proteome Res.* **2015**, *14*, 3148−3161.
(30) Klimek, J.; Eddes, J. S.; Hohmann, L.; Jackson, J.; Peterson, A.; Letarte, S.; Gafken, P. R.; Katz, J. E.; Mallick, P.; Lee, H.; Schmidt, A.; Ossola, R.; Eng, J. K.; Aebersold, R.; Martin, D. B. The standard protein mix database: a diverse data set to assist in the production of improved peptide and protein identification software tools. *J. Proteome Res.* **2008**, *7*, 96−103.
(31) Wilhelm, M.; Schlegl, J.; Hahne, H.; Gholami, A. M.; Lieberenz, M.; Savitski, M. M.; Ziegler, E.; Butzmann, L.; Gessulat, S.; Marx, H.; et al. Mass-spectrometry-based draft of the human proteome. *Nature* **2014**, *509*, 582−587.

https://dx.doi.org/10.1021/acs.jproteome.0c00711
J. Proteome Res. 2021, 20, 1849−1854