GPU-Accelerated SVM Learning for Extremely Fast Massive-Scale Proteomics Classification

Anonymous Authors[1]

Abstract

In proteomic analysis pipelines, semi-supervised support vector machine (SVM) learning (Kall et al., 2007) is a critical step towards accurately identifying the generating peptides of tandem mass spectra. Called Percolator, this algorithm iteratively learns the linear decision boundary between correct and incorrect peptide-spectrum matches (PSMs) and uses the converged decision boundary to rerank the input PSMs. While this reranking greatly improves PSM identification accuracy, Percolator requires substantial analysis time in practice. Recent work (Halloran & Rocke, 2018b) has reduced such lengthy runtimes by updating Percolator's SVM solver to state-of-the-art, multithreaded solvers. In this work, we present linear SVM primal solvers novelly designed to take advantage of graphical processing units (GPUs) which significantly outperform previous multithreaded speedups. Most importantly, we show how GPU optimizations may be mixed with multithreading to enable such speedups for commonly produced large-scale proteomics datasets which do not fit in GPU memory alone. On a massive proteomics dataset of nearly a quarter-billion data instances, we show that such mixed-architecture speedups reduce SVM analysis time from over half a week down to less than a single day while efficiently using limited GPU memory.

1. Introduction

Introduced slightly over a decade ago, semi-supervised SVM learning using the Percolator algorithm (Kall et al., 2007) has become vital to accurately analyze proteomics data collected via tandem mass spectrometry (MS/MS). Given a collection of MS/MS spectra representing the proteins present in a complex biological sample, the first stage of proteomics analysis typically consists of identifying the input spectra by searching a database of peptides. Database-search thus results in a list of *peptide-spectrum matches* (PSMs). In practice, however, database-search scoring functions are often poorly calibrated, making PSMs from different spectra difficult to compare and diminishing overall identification accuracy. To correct for this, PSM scores are often post-processed using Percolator, which first estimates PSM labels then learns the linear decision boundary between labeled PSMs, repeating these two steps for a user-specified number of iterations. Input PSM scores are subsequently *recalibrated* using the final learned decision boundary.

The accuracy improvements of Percolator recalibration have been well demonstrated for a wide variety of PSM scoring functions (e.g., linear (Kall et al., 2007; Brosch et al., 2009; Xu et al., 2013), p-value based (Granholm et al., 2013; Howbert & Noble, 2014; Lin et al., 2018), and dynamic Bayesian networks (Halloran et al., 2016)), complex PSM feature sets (e.g., Fisher kernels (Halloran & Rocke, 2017; 2018a), subscores of linear functions (Spivak & Noble, 2012), ensembles of scoring functions (Wen et al., 2015), and features derived using deep models (Gessulat et al., 2019)), and relative to other popular post-processors (Tu et al., 2015). Indeed, many complex, state-of-the-art proteomics workflows have adapted Percolator as a critical component of their analysis pipelines. However, while Percolator offers significant accuracy gains, they come at lengthy runtimes as the size of commonly produced proteomic datasets has dramatically increased (by several orders of magnitude) since Percolator's debut. For instance, modest datasets comprised of only several million PSMs require several hours of Percolator analysis (Halloran & Rocke, 2018b), while more common proteomics datasets–whose PSMs regularly number in the tens-of-millions–may require up to a day (or more) of analysis time (Matthew et al., 2016).

To speed up the lengthy analysis times required of large-scale studies, recent work (Halloran & Rocke, 2018b) has updated Percolator's original SVM solver to the state-of-the-art Trust Region Newton (*TRON*) algorithm (Lin et al., 2008; Hsia et al., 2017) and utilized large numbers of compute cores. Optimized for use within Percolator, this multithreaded version of *TRON* was shown to drastically reduce large-scale analysis time. As the critical bottleneck computations in *TRON* are linear algebra operations (Lee et al., 2015), GPUs (which greatly outperform CPUs for large-scale linear algebra calculations) are a ideal computational tools to further speedup Percolator analysis time. How-

ever, while *TRON* easily lends itself to multithreaded (CPU) optimizations on shared memory systems, effective GPU optimizations are far more difficult to achieve; *TRON* heavily relies on random access of feature instances throughout each iteration of the algorithm. While this is naturally supported in multicore environments, such random access prevents memory coalescing (and is thus deleterious) for GPU computation. Furthermore, large memory transfers between the CPU and GPU are expensive, so that the complex, sequential dependency of variables in *TRON* further makes optimal GPU use difficult.

Herein, we present two *TRON* solvers which overcome the inherent multicore design of the algorithm and extensively use GPUs to accelerate overall computation. We assume a single GPU (referred to as the *device*) and refer to the multicore CPU as the *host*. The first presented GPU-optimized solver decouples the original sequential dependence of variables, allowing the computation of large algorithm blocks to saturate the device while using as few transactions between device and host as possible. This solver (called *TRON GPU*) drastically reduces overall SVM learning time, resulting in a **7.4 fold speedup** over Percolator's current SVM learning engine on a dataset of over 23 million PSMs, thus reducing Percolator learning time from 14.4 hours to just 1.9 hours.

For even larger datasets, the device-memory requirements of *TRON GPU* become restrictive as the memory bandwidths of state-of-the-art GPUs are far more limited than those found in shared host memory systems. Thus, we next present a mixed-architecture solver, called *TRON CPU+GPU*, which combines the strengths of both architectures; CPU multithreading for the essential random access components of *TRON* and a GPU for fast computation in contiguous memory. On a massive proteomics dataset of over 215 million PSMs (too large to be analyzed using *TRON GPU*), *TRON CPU+GPU* dominates all previously proposed multithreaded speedups and achieves a **5.2 fold speedup** averaged across all computational threads, reducing Percolator learning time from 4.4 days down to just 19.7 hours.

2. Semi-supervised SVM Learning for MS/MS Data using Percolator

In practical MS/MS experiments, ground truth labels (i.e., the true peptides responsible for generating each MS/MS spectrum) are not known a priori; indeed, it is the role of the database-search scoring algorithm to identify these generating peptides. In order to assess the confidence of peptide identifications, two peptide databases are typically searched–a *target* database of real peptides and a *decoy* database of permuted target sequences, which we know do not occur in nature–and used to compute the false discovery rate (FDR).

Percolator receives as input both decoy and target PSMs, along with features derived for each PSM. This data is semi-supervised, as we know that decoy PSMs are incorrect identifications (i.e., belong to the negative class), but we are not certain which target PSMs are correct. Each iteration of Percolator thus begins by calculating the target PSMs which achieve a stringent FDR of 0.01% (i.e., are highly confident identifications) and assigning these targets positive training labels. In order to prevent overfitting and improve generalizability, three-fold cross-validation is carried out over three disjoint partitions of the original dataset, followed by further nested cross-validation within each fold (Granholm et al., 2012). This results in a total of nine unique train and test sets. For each of these training sets, a linear SVM is trained to discriminate between decoys and positive-labeled targets. At the end of each iteration, the separately learned SVM parameters are then merged and used to recalibrate all PSM scores. This process is repeated for a user-specified number of iterations (ten by default).

While this semi-supervised algorithm is robust in practice and widely used throughout the proteomics community, it is also computationally intensive as analysis time is dominated by the iterative training of many SVMs. To combat the extensive Percolator analysis times required of regularly conducted large-scale protein studies, recent work (Halloran & Rocke, 2018b) updated Percolator's original SVM solver (called *L2-SVM-MFN* (Keerthi & DeCoste, 2005)) to the Trust Region Newton (*TRON*) algorithm (Lin et al., 2008), the state-of-the-art solver used in the popular machine learning packages LIBLINEAR (Fan et al., 2008) and scikit-learn (Pedregosa et al., 2011). *TRON* was optimized within Percolator to utilize multithreading on shared-memory systems and demonstrated to decrease total SVM training time on datasets of several million PSMs. Most importantly, *TRON* was shown to speedup Percolator without affecting learned SVM parameters, unlike recently proposed random-subsampling approaches (Matthew et al., 2016).

3. Trust Region Newton for Primal SVM Learning

Consider feature vectors $\boldsymbol{x}_i \in \mathbb{R}^n, i = 1, \ldots, l$ and label vector $\boldsymbol{y} \in \{-1, 1\}^l$. Let $X = [\boldsymbol{x}_1 \ldots \boldsymbol{x}_l]^T$, $\mathbf{1}$ denote the indicator function, and $*$ denote element-by-element vector multiplication. For vectors, index-set subscripts denote subvectors and for matrices, pairs of index-set subscripts denote submatrices.

The L2-regularized, L2-SVM primal objective function, which we wish to minimize w.r.t. $\boldsymbol{w}$, is

$$f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i))^2, \quad (1)$$

the gradient of which is $\nabla f(\boldsymbol{w}) = \boldsymbol{w} + 2CX_{I,:}^T(X_{I,:}\boldsymbol{w} - \boldsymbol{y}_I)$, where $I \equiv \{i|1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i > 0\}$ is an index set and the operator : denotes all elements along the corresponding dimension (i.e., all columns in this case). The generalized Hessian of $f(\boldsymbol{w})$ (Keerthi & DeCoste, 2005) is $\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + 2CX^T DX$, where $\mathcal{I}$ is the identity matrix and D is a diagonal matrix with elements $D_{ii} = \mathbf{1}_{i\in I}$.

Algorithm 1 *TRON* for L2-SVMs

1: Given w, Δ, and σ_0
2: **while** Not converged **do**
3: Find $\boldsymbol{d}$ in Equation 2 using a conjugate gradient procedure
4: Calculate $\sigma = \frac{f(\boldsymbol{w}+\boldsymbol{d})-f(\boldsymbol{w})}{q(\boldsymbol{d})}$
5: **if** $\sigma > \sigma_0$ **then** $\boldsymbol{w} \leftarrow \boldsymbol{w}+\boldsymbol{d}$, increase trust region Δ.
6: **else** Shrink Δ.
7: **end if**
8: **end while**

The *TRON* algorithm is detailed in Algorithm 1. At each iteration, given the current parameters $\boldsymbol{w}$ and trust region interval Δ, *TRON* considers the following quadratic approximation to $f(\boldsymbol{w} + \boldsymbol{d}) - f(\boldsymbol{w})$, $q(\boldsymbol{d}) = \nabla f(\boldsymbol{w})^T\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{d}$, to find a truncated Newton step confined in the trust region by solving

$$\min_{\boldsymbol{d}} q(\boldsymbol{d}) \quad \text{s.t. } \|\boldsymbol{d}\|_2 \leq \Delta. \tag{2}$$

If $q(\boldsymbol{d})$ is close to $f(\boldsymbol{w} + \boldsymbol{d}) - f(\boldsymbol{w})$, $\boldsymbol{w}$ is updated to $\boldsymbol{w} + \boldsymbol{d}$ and the trust region interval is increased for the subsequent iteration. Otherwise, $\boldsymbol{w}$ remains unchanged and the trust region interval is shrunk. The conjugate gradient method used to solve Equation 2 involves only a single Hessian-vector product, the structure of which is exploited to avoid loading the entire Hessian into memory; owing to the diagonal form of D, we have $\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + 2CX_{I,:}^T D_{I,I} X_{I,:}$. Thus, for a vector $\boldsymbol{v}$, the Hessian-vector product computed during conjugate gradient descent is $\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2CX_{I,:}^T(D_{I,I}(X_{I,:}\boldsymbol{v}))$, and the algorithm is very efficient overall.

3.1. TRON Implementations

Due to the special forms of the gradient and generalized Hessian (in particular, the derivation and use of I throughout the algorithm), computation in *TRON* heavily relies on random access. This naturally allows efficient design of the algorithm on a shared-memory, multicore system. As detailed in (Lee et al., 2015) (and utilized within Percolator in (Halloran & Rocke, 2018b)) the computation of $f(\boldsymbol{w})$, $\nabla f(\boldsymbol{w})$, and $\nabla^2 f(\boldsymbol{w})\boldsymbol{v}$ may be efficiently computed across multiple parallel threads using OpenMP. While efficient for multicore architectures, the non-contiguous nature of the algorithm (i.e., I is recomputed every iteration) make designing an efficient GPU implementation far less straightforward; for GPU computing, device-side computation performs best over contiguous memory. Furthermore, large memory transfers between host (i.e., CPU) and device (i.e., GPU) are expensive, hindering approaches where I is first computed then a randomly accessed subset of the data is formed on the host and transferred to the device.

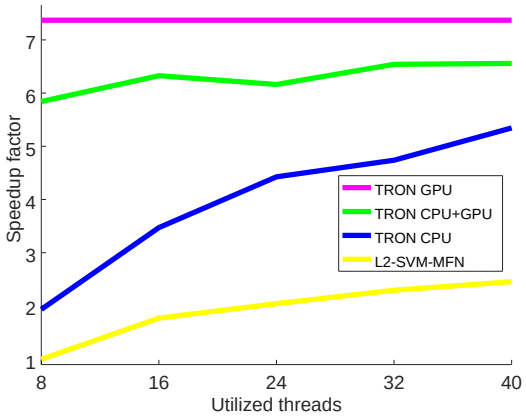

Figure 1: Factor of speedup for SVM learning in Percolator for a large-scale dataset containing 23,330,311 PSMs. Speedup factor is calculated as the original Percolator SVM learning time divided by the sped up learning time. The x-axis displays the number of threads utilized by multithreaded methods "L2-SVM-MFN," "TRON CPU," and "TRON CPU+GPU."

We present two efficient GPU implementations of *TRON* with complimentary strengths and weaknesses. Written in CUDA, both implementations first load X and $\boldsymbol{y}$ onto the device and, at the start of each iteration, transfer $\boldsymbol{w}$ from host to device. Both solvers also make distinct use of the insight that, on the device-side, prior to computing $f(\boldsymbol{w})$ in each iteration, I may be computed and stored in device memory for future compute. The major operations of each solver are listed in Table 1. The GPU-optimized solver, *TRON GPU*, performs all intensive computing on the GPU with very few transactions between host and device (only two small transfers from device to host), at the cost of higher device-side memory to compute and store $X_{I,:}$ every iteration. The mixed architecture solver, *TRON CPU+GPU*, utilizes the GPU for heavy lifting before using multithreading for efficient random access after I is computed, utilizes less device-side memory but requires several large data transfers between host and device.

4. Results and Discussion

All experiments were run on a dual Intel Xeon Gold 5118 compute node with 40 computational threads, an NVIDIA Tesla V100 GPU, and 768 GB of memory. All methods are

TRON GPU	*TRON CPU+GPU*
$\boldsymbol{z} = \boldsymbol{y} * (X\boldsymbol{w})$ is calculated on the device, where $*$ is vector element-wise product.	$\boldsymbol{z} = \boldsymbol{y} * (X\boldsymbol{w})$ is calculated on the device, then transferred to the host.
$I = \{i : \boldsymbol{z}_i < 1\}$ is calculated on the device, then $f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(1 - z_i > 0)^2$ is computed on the device while the host runs independent, sequential operations.	I is calculated on the device and transferred to the host. The transfer is interleaved with the device-side computation of $f(\boldsymbol{w})$.
On the device, $\hat{\boldsymbol{z}} = \boldsymbol{y}_I * (\boldsymbol{z}_I - 1)$ and $\hat{X} = X_{I,:}$ are computed and stored in device memory. The gradient $g(\boldsymbol{w}) = \nabla f(\boldsymbol{w})$ is then computed as $g(\boldsymbol{w}) = \boldsymbol{w} + 2C\hat{X}^T\hat{\boldsymbol{z}}$ and transferred to the host.	With I and z on the host, $g(\boldsymbol{w}) = \nabla f(\boldsymbol{w})$ is computed using multithreading.
The Hessian-product is computed on the device as $\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2C\hat{X}^T(\hat{X}\boldsymbol{v})$ and transferred to the host.	Using multithreading, the Hessian-product is calculated on the host as $\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2CX_{I,:}^T(D_{I,I}(X_{I,:}\boldsymbol{v})) = \boldsymbol{v} + 2C\sum_{i\in I}(\boldsymbol{x}_i^T\boldsymbol{v})\boldsymbol{x}_i$.

Table 1: Major operations of the *TRON* solvers designed for GPU compute.

tested using two extremely large datasets, the first of which (referred to as the Kim dataset) is a larger version of the Kim dataset used in (Halloran & Rocke, 2018b) consisting of 23,330,311 PSMs. The second dataset, referred to as the Wilhelm dataset, was collected from a map of the human proteome (Wilhelm et al., 2014) and contains 215,282,771 PSMs. The GPU optimized *TRON* solvers are compared against the multithread-optimized versions of *TRON* (referred to as *TRON CPU*) and *L2-SVM-MFN* from (Halloran & Rocke, 2018b). All multithreaded solvers were tested using 8, 16, 24, 32, and 40 threads. As in (Halloran & Rocke, 2018b), to effectively measure the runtime of multithreaded methods without any excess thread-scheduling overhead, parallelization of Percolator's outermost cross-validation was disabled.

Reported runtimes are the minimum wall-clock times measured over five runs for the Kim dataset and three runs for the Wilhelm dataset. The original Percolator SVM learning runtimes (collected using Percolator v3.02.0) were 14.4 hours and 4.4. days for the Kim and Wilhelm datasets, respectively. For the Kim dataset, speedup results for all discussed methods are illustrated in Figure 1. For the Wilhelm dataset, the Tesla V100 memory bandwidth (16 GB total) is exceeded for *TRON GPU*. However, the reduced memory requirements of *TRON CPU+GPU* allow GPU speedups to substantially decrease Percolator analysis time for this massive dataset (illustrated in Figure 2).

Both GPU solvers greatly accelerate Percolator SVM learning time while dominating previously proposed multithreaded speedups. For the Kim dataset, *TRON CPU+GPU* and *TRON GPU* complete 6.6 (for 40 threads) and 7.4 times faster than Percolator's current SVM learning engine, while *TRON CPU+GPU* completes 5.3 (for 40 threads) times faster for the Wilhelm dataset. Together, these two solvers present versatile trade-offs for different compute environments; when the dataset does not exceed the GPU memory, *TRON GPU* offers superior performance. However, when onboard GPU memory is limited, a small portion of speed may be traded for much less memory consumption by using *TRON CPU+GPU*. Furthermore, when the number of computational threads is also limited, *TRON CPU+GPU* offers significantly better (and more stable) performance at low numbers of utilized threads compared to the purely multithreaded solvers *TRON CPU* and *L2-SVM-MFN*. In future work, the runtime benefits of the presented GPU-optimized solvers will be further evaluated over other large/massive-scale datasets and the memory footprints (on both host and device) will be carefully analyzed.

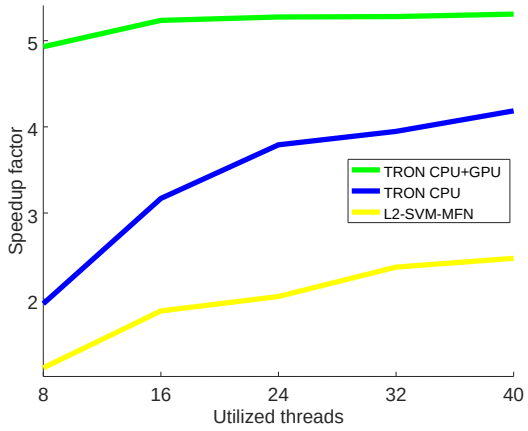

Figure 2: Factor of speedup for SVM learning in Percolator for a massive dataset containing 215,282,771 PSMs, too large to be analyzed using "TRON GPU." Speedup factor is calculated as the original Percolator SVM learning time divided by the sped up learning time. The x-axis displays the number of threads utilized by multithreaded methods "L2-SVM-MFN," "TRON CPU," and "TRON CPU+GPU."

References

Brosch, M., Yu, L., Hubbard, T., and Choudhary, J. Accurate and sensitive peptide identification with Mascot Percolator. *J. Proteome Res.*, 8(6):3176–3181, 2009.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug): 1871–1874, 2008.

Gessulat, S., Schmidt, T., Zolg, D. P., Samaras, P., Schnatbaum, K., Zerweck, J., Knaute, T., Rechenberger, J., Delanghe, B., Huhmer, A., et al. Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. *Nature methods*, 16(6):509, 2019.

Granholm, V., Noble, W. S., and Kall, L. A cross-validation scheme for machine learning algorithms in shotgun proteomics. *BMC bioinformatics*, 13(16):S3, 2012.

Granholm, V., Kim, S., Navarro, J. C., Sjolund, E., Smith, R. D., and Kall, L. Fast and accurate database searches with ms-gf+ percolator. *J. Proteome Res.*, pp. 890–897, 2013.

Halloran, J. T. and Rocke, D. M. Gradients of generative models for improved discriminative analysis of tandem mass spectra. In *Advances in Neural Information Processing Systems*, pp. 5728–5737, 2017.

Halloran, J. T. and Rocke, D. M. Learning concave conditional likelihood models for improved analysis of tandem mass spectra. In *Advances in Neural Information Processing Systems*, pp. 5420–5430, 2018a.

Halloran, J. T. and Rocke, D. M. A matter of time: Faster percolator analysis via efficient svm learning for large-scale proteomics. *Journal of proteome research*, 17(5): 1978–1982, 2018b.

Halloran, J. T., Bilmes, J. A., and Noble, W. S. Dynamic bayesian network for accurate detection of peptides from tandem mass spectra. *Journal of Proteome Research*, 15 (8):2749–2759, 2016.

Howbert, J. J. and Noble, W. S. Computing exact p-values for a cross-correlation shotgun proteomics score function. *Molecular & Cellular Proteomics*, pp. mcp–O113, 2014.

Hsia, C.-Y., Zhu, Y., and Lin, C.-J. A study on trust region update rules in newton methods for large-scale linear classification. In *Asian Conference on Machine Learning*, pp. 33–48, 2017.

Kall, L., Canterbury, J., Weston, J., Noble, W. S., and MacCoss, M. J. A semi-supervised machine learning technique for peptide identification from shotgun proteomics datasets. *Nat. Methods*, 4:923–25, 2007.

Keerthi, S. S. and DeCoste, D. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(Mar):341–361, 2005.

Lee, M.-C., Chiang, W.-L., and Lin, C.-J. Fast matrix-vector multiplications for large-scale logistic regression on shared-memory systems. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pp. 835–840. IEEE, 2015.

Lin, A., Howbert, J. J., and Noble, W. S. Combining high-resolution and exact calibration to boost statistical power: A well-calibrated score function for high-resolution ms2 data. *Journal of proteome research*, 17(11):3644–3656, 2018.

Lin, C.-J., Weng, R. C., and Keerthi, S. S. Trust region newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9(Apr):627–650, 2008.

Matthew, T., MacCoss, M. J., Noble, W. S., and Käll, L. Fast and accurate protein false discovery rates on large-scale proteomics data sets with percolator 3.0. *Journal of The American Society for Mass Spectrometry*, 27(11): 1719–1727, 2016.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Spivak, M. and Noble, W. S. Learning score function parameters for improved spectrum identification in tandem mass spectrometry experiments. *J. Proteome Res.*, 11(9): 4499–4508, 2012. PMC3436966.

Tu, C., Sheng, Q., Li, J., Ma, D., Shen, X., Wang, X., Shyr, Y., Yi, Z., and Qu, J. Optimization of search engines and postprocessing approaches to maximize peptide and protein identification for high-resolution mass data. *Journal of proteome research*, 14(11):4662–4673, 2015.

Wen, B., Du, C., Li, G., Ghali, F., Jones, A. R., Käll, L., Xu, S., Zhou, R., Ren, Z., Feng, Q., et al. Ipeak: An open source tool to combine results from multiple ms/ms search engines. *Proteomics*, 15(17):2916–2920, 2015.

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*, 509(7502): 582–587, 2014.

Xu, M., Li, Z., and Li, L. Combining percolator with x! tandem for accurate and sensitive peptide identification. *Journal of proteome research*, 12(6):3026–3033, 2013.