Lee *et al. BMC Bioinformatics* (2018) 19:221
https://doi.org/10.1186/s12859-018-2210-6

BMC Bioinformatics

SOFTWARE Open Access

Blazing Signature Filter: a library for fast pairwise similarity comparisons

Joon-Yong Lee[1], Grant M. Fujimoto[1], Ryan Wilson[1], H. Steven Wiley[2] and Samuel H. Payne[1*]

Abstract

Background: Identifying similarities between datasets is a fundamental task in data mining and has become an integral part of modern scientific investigation. Whether the task is to identify co-expressed genes in large-scale expression surveys or to predict combinations of gene knockouts which would elicit a similar phenotype, the underlying computational task is often a multi-dimensional similarity test. As datasets continue to grow, improvements to the efficiency, sensitivity or specificity of such computation will have broad impacts as it allows scientists to more completely explore the wealth of scientific data.

Results: The Blazing Signature Filter (BSF) is a highly efficient pairwise similarity algorithm which enables extensive data mining within a reasonable amount of time. The algorithm transforms datasets into binary metrics, allowing it to utilize the computationally efficient bit operators and provide a coarse measure of similarity. We demonstrate the utility of our algorithm using two common bioinformatics tasks: identifying data sets with similar gene expression profiles, and comparing annotated genomes.

Conclusions: The BSF is a highly efficient pairwise similarity algorithm that can scale to billions of comparisons without the need for specialized hardware.

Keywords: Pairwise similarity comparison, Filtering, Large-scale data mining

Background

Data mining is frequently used in scientific research for hypothesis generation, mechanistic insight, or validation. Similarity metrics are an essential component of data mining, and are used to identify relevant data. In computational biology, a wide variety of similarity metrics have been devised to maximize specificity and sensitivity in sequence alignment [1], proteomic mass spectrometry [2], evolutionary tree building [3], co-expression network creation [4], etc. These algorithms are typically used to facilitate comparing a data point against a curated library of experiments to derive insight [5]. As modern data generation capabilities have created a deluge of potential data to compare against, exhaustive similarity search may become computationally prohibitive for inefficient algorithms. Therefore, efficient and accurate algorithms for computing similarity are necessary. For instance, the library of integrated network-based cellular signatures (LINCS) program has generated over one million gene expression experiments [6]. To compute the pairwise similarity between all experiments therefore requires 0.5 trillion similarity calculations.

When doing similarity-based computations on very large data, a significant drawback is that most of the pairwise comparisons yield a negative result, i.e. the two data points are not similar. An example of this is sequence alignment of genomic data. The current NCBI nr database contains >78 million proteins (as of January 2017, release 80), the vast majority of which are not related to an input query. It would be a significant waste of time to perform the Smith-Waterman local alignment search against all 78 million sequences [7]. To overcome this limitation and enable large-scale data mining, sequence comparison algorithms commonly filter the set of sequences in the library prior to a more sensitive search. The BLAST algorithm requires a shared k-mer between the query sequence and candidate sequences from the library [8]. Only those proteins which contain a shared k-mer

*Correspondence: Samuel.Payne@pnnl.gov
[1]Integrative Omics, Pacific Northwest National Laboratory, 99352 Richland, WA USA
Full list of author information is available at the end of the article

progress to a full alignment. This style of filtering candidates before a more computationally expensive scoring scheme is a common strategy which allows data mining to scale to ever-larger data volumes [9–11].

A second method to improve the speed of an algorithm is to adopt a faster core calculation. Most scientific data is stored as floating-point numbers; multiplication or division of floats is relatively slow. Therefore, optimizing an approach to minimize these will improve the computational speed. Bit operations (e.g. AND, OR, XOR) are dramatically faster than multiplication, yet require a restructuring of the basic approach or a data transform. The FastBit algorithm transforms data into bitmaps, then performs a hybrid compression to enable several common algorithmic operations (e.g. less than operator, histograms, exact pattern matching). This method is specifically designed to facilitate querying very large libraries with scientific data of high cardinalities [12]. Similarly, bit-vectors have been used to improve the speed of string matching [13].

We combine these two ideas in the Blazing Signature Filter (BSF), a new approach to prune unproductive pairwise similarity calculations and enable large-scale data mining. The BSF identifies signatures in digital data through bit representation (non-full precision) and bitwise operators. These binary operands are dramatically more efficient than floating-point multiplication and division in terms of CPU cycles per comparison. This simple heuristic allows us to remove $> 98\%$ of pairwise comparisons rapidly and therefore concentrate computational effort on pairs that are more likely to be meaningfully similar, enabling data mining tasks which previously appeared infeasible. We demonstrate the power of the BSF by computing the pairwise similarity of all publicly available LINCS datasets and identifying similarity of all genomes annotated by the Kyoto Encyclopedia of Genes and Genomes (KEGG).

Implementation

A simplified example of the BSF is illustrated in Fig 1, where a 64-element signature is compared to a pool of candidates in a library. This 64-element signature is entirely binary, meaning that we only keep track of whether the element is part of the signature or not. Bit operations on two 64-bit binary signatures happen in a single instruction as two registers are compared with operators like AND, or XOR. Counting the number of successes in the comparisons (1s in the resulting array) is rapidly done using the hardware instruction 'POPCNT' [14]. In comparison, identifying the cosine similarity or Euclidean distance between two 64-element floating point vectors requires over 100 additions, multiplications, divisions and square root operations. For modern processors, cosine distance and Euclidean distance have an average throughput of $398\sim1579$ and $199\sim724$ clock cycles, respectively, whereas BSF uses $2\sim4$ clock cycles. Although this simplified example shows a 64-element signature, the software implementation of the BSF has been engineered to allow an arbitrary signature length. This is essential for comparing gene expression signatures which may scale to tens of thousands of elements, e.g. 20,000 human proteins. As this is larger than the size of a single CPU register, the data is chunked into appropriate sizes and comparisons are flowed through the registers.

The BSF code, written in C++, is an open source software project licensed under BSD. Source can be found at https://github.com/PNNL-Comp-Mass-Spec/bsf-core. For ease of access, we have written a python wrapper to interface with the BSF C++ library, https://github.com/PNNL-Comp-Mass-Spec/bsf-py. Python extensions and numpy C-API are employed to implement the python wrapper.

The BSF is accessed through an API which ensures that input data is appropriate and meaningful, and interprets the output tables which are returned. Input to the BSF consists of two binary tables of size K-by-N and K-by-M. K is the vector length of a signature. N is the number of signatures in the query. M is the number of signatures in the library. For the input tables, all the bits of each column are stored into an array of 64-bit unsigned integers. It enables bitwise operators using the 64-bit registers. The user also specifies which binary operator to use. The return from BSF is an N-by-M table with each cell representing the value of comparing element i in the query table with element j in the library table. The pseudocode is described in Algorithm 1.

Algorithm 1 BSF algorithm to analyze the similarity between columns of a library matrix and a query matrix

Input: $lib[K][M] \leftarrow$ a library matrix, $q[K][N] \leftarrow$ a query matrix
Output: $out[N][M] \leftarrow$ a 2D array of unsigned integers

```
function BSFCore1(lib, q)
    for all i in [0, N] do
        for all j in [0, M] do
            pop_count ← 0
            for all k in [0, K] do
                sim ← lib[k][j] & q[k][i]
                pop_count ← pop_count + __builtin_popcountll(sim)
            out[i][j] ← count
    return out
```

When computing all the pairwise comparisons between two signatures within a single K-by-M matrix, the BSF outputs a M-by-M strictly upper triangular matrix with

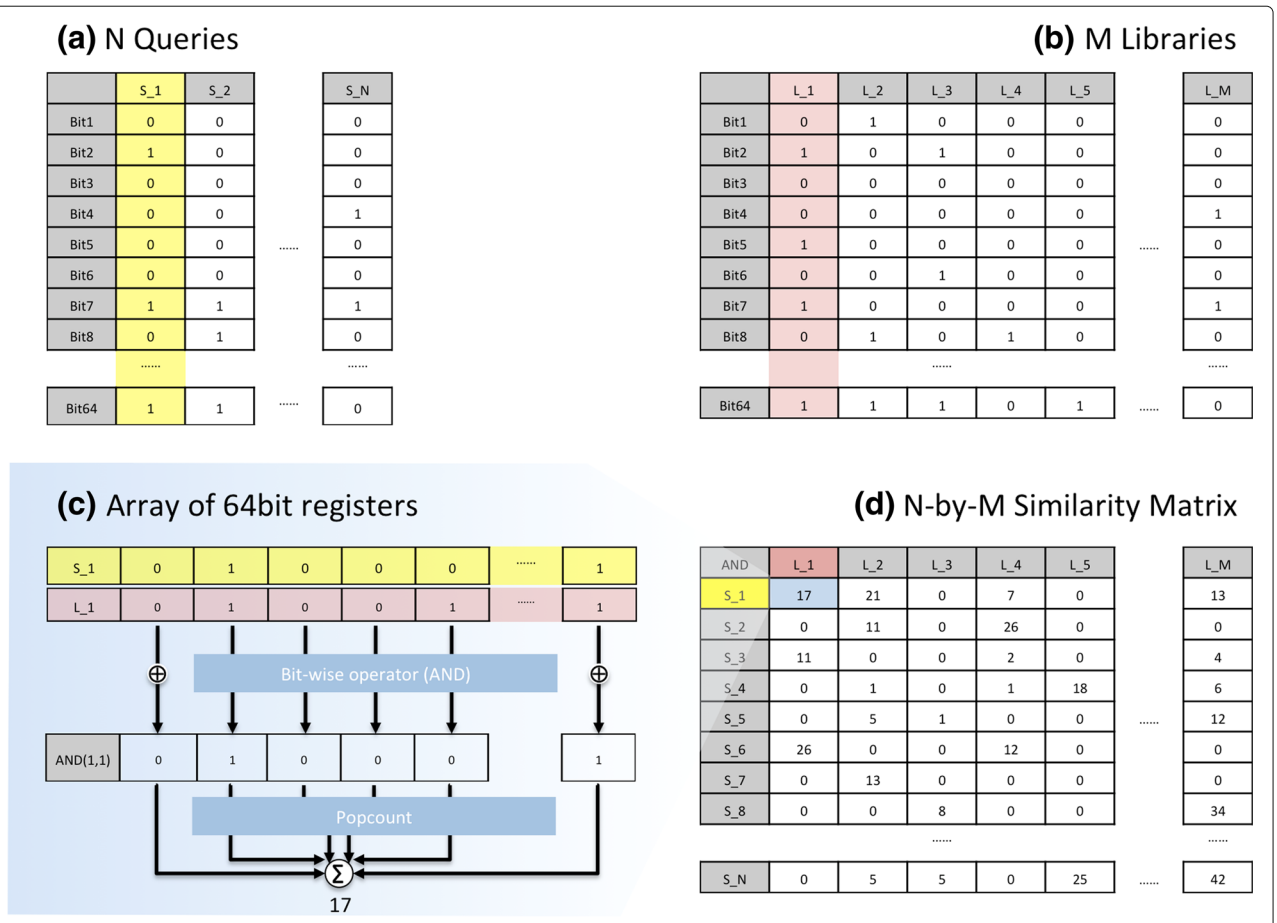

Fig. 1 Illustration for the core of BSF. This simple example shows how the BSF deals with the binary data to identify the similarity between query and library signatures. Each signature has 64 elements (rows in the matrix). The binary data represents whether an element is part of the signature, i.e. '1' means that the element is part of the signature, '0' means that it is not part of the signature. **a** A set of query signatures represented as a binary matrix. **b** A set of library signatures to which the query signatures are compared. **c** The binary comparison for a single column in the query and library matrices. **d** The results matrix containing the similarity for each pairwise comparison. In the 64-bits example, clock cycle needs for the BSF are 1 for 'AND' and 3 for 'POPCNT', while cosine and Euclidean metric use > 500 and > 700 clock cycles, respectively. (See Methods)

zero diagonal entries so that we avoid the redundant computation of a_{ij} for $i > j$ which must be equal to a_{ji}. The pseudo code for this is shown in Algorithm 2.

Algorithm 2 BSF algorithm to analyze the similarity between columns of a single library matrix

Input: $lib[K][M] \leftarrow$ a 2D array of unsigned 64bit integers

Output: $out[M]\,[M] \leftarrow$ a 2D array of unsigned integers

```
function BSFCore2(lib)
    for all i in [0, M − 1] do
        for all j in [i + 1, M] do
            pop_count ← 0
            for all k in [0, K] do
                sim ← lib[k] [i] & lib[k] [j]
                pop_count ← pop_count + __builtin_popcountll(sim)
            out[i] [j] ← count
    return out
```

For very large matrices, the result file size may become too large to fit in physical memory, which would lead to an out-of-memory exception. To safely avoid this memory issue, we split the output matrix into multiple chunks of a manageable size. Given the file size, the size of a chunk matrix is automatically decided. Details are described in Algorithm 3.

Results

Mining large data repositories to identify similar datasets is a common technical task. Depending on the number of comparisons to be done, the time involved in this simple task may be prohibitive. In most large-scale pairwise similarity searches (e.g., identifying similarity between all public transcriptomics datasets), the vast majority of pairs will be dissimilar. Thus, the most efficient way to speed such data mining explorations is to rapidly identify dissimilar pairs and remove them from

Algorithm 3 Chunk split for avoiding out-of-memory in handling a big library matrix

Input:

$lib[K]\,[M] \leftarrow$ a library matrix,

$size \leftarrow$ the chunk size (GB, e.g., 4 for $4GB$)

Output: multiple binary files of which size is less than s_{chunk}

$s_{chunk} \leftarrow int(\sqrt{size \times 10^9/4})$

(e.g., $\sqrt{4 \times 10^9 \text{ bytes}/4 \text{ bytes}} \approx 31,623$ for splitting into 4GB chunks)

tail $\leftarrow M \pmod{s_{chunk}}$

$N_{chunk} \leftarrow \text{tail} = 0?(M/s_{chunk}) : (M/s_{chunk}) + 1$

for all i in $[0, N_{chunk}]$ **do**

  for all j in $[i, N_{chunk}]$ **do**

    if $i < N_{chunk} - 1$ **then**

      $rows \leftarrow s_{chunk}$

    else

      $rows \leftarrow tail = 0?s_{chunk} : tail$

    if $j < N_{chunk} - 1$ **then**

      $cols \leftarrow s_{chunk}$

    else

      $cols \leftarrow tail = 0?s_{chunk} : tail$

    $x1 \leftarrow i * s_{chunk}$, $x2 \leftarrow i * s_{chunk} + rows$

    $y1 \leftarrow j * s_{chunk}$, $y2 \leftarrow j * s_{chunk} + cols$

    $chunk[\,rows]\,[\,cols] \leftarrow$ An empty matrix

    if $i = j$ **then**

      $chunk \leftarrow \text{BSFCore2}(lib[K]\,[x1 : x2])$

    else

      $chunk \leftarrow \text{BSFCore1}(lib[K]\,[x1 : x2]\,, lib[K]\,[y1 : y2])$

    Write $chunk[rows]\,[\,cols]$ into a binary file

the analysis pipeline. The purpose of the BSF is to identify pairwise similarity comparisons which are unlikely to be statistically meaningful. Our heuristic is to binarize the data and calculate a similarity metric on the binary data using bit operators, as bitwise computation is dramatically faster than floating point operations. In this way, the BSF can work as a front-end filter to computational analysis tools and dramatically speed up their pipeline.

Performance benchmarking

To demonstrate the speed of the similarity metrics, we performed a benchmark test of BSF, cosine similarity and Euclidean distance using a synthetic dataset mimicking gene expression measurements. In gene expression experiments, the goal is often to identify the up/down regulated genes relative to a reference condition. For example, in a gene knockout experiment, the desire is to understand and investigate which proteins are altered in their regulation relative to wild-type. The synthetic data was generated as measurements of 20,000 genes for 15,000 experiments (See Materials and Methods). Full precision floating point data was used by cosine and Euclidean distance metrics, whereas the BSF used binarized data showing up or down regulated genes.

We performed the full pairwise comparison of all 15K experiments for both the up and down matrix ($\sim$225 million comparisons). To characterize the time-dependence of each algorithm on the length of the signature, we tested each algorithm with a different number of genes ranging from 1000 to 20,000. This is essential to understanding the utility of each algorithm, as different applications may contain highly variable signature lengths. As shown in Fig 2, the time taken by each algorithm grows with the length of the signature. However, we note that the time dependence of the BSF grows dramatically more slowly than other methods. For the full 20,000 length signature (approximately what would be used for human gene expression data), the BSF algorithm ran in 45 s, while Euclidean and cosine methods took $\sim$2000 or $\sim$6000 s respectively. Both the Euclidean and cosine method show a linear time dependence on the signature length, $O(n)$, while the BSF shows a log-linear dependence, $O(\log n)$, consistent with algorithmic expectations.

LINCS network analysis

The LINCS L1000 project is a large-scale gene expression analysis, where numerous perturbations are applied to a variety of human cell lines and the response measured at multiple time points (https://clue.io/). The L1000 assay acquires transcript measurements on $\sim$ 1000 carefully chosen landmark genes followed by imputing the expression values for the remaining $\sim$ 21,000 human genes. Perturbations used in the LINCS L1000 project include small molecule inhibitors, gene knockdowns and gene over-expression. Identifying similarities between perturbations is a primary focus of the project, which could enable the characterization of drug compounds having unknown targets or identifying signaling cross-talk.

As a real-world test for the BSF, we computed the pairwise similarity for the publicly available subset of the LINCS L1000 datasets [6]. We downloaded the December 2016 snapshot which contains the differentially expressed genes from $117K$ experiments. We convert the up/down regulated genes into a 22,688-by-117,373 binary matrix and computed the 6.89 billion pairwise comparisons for the up-regulated genes and another 6.89 billion comparisons for the down-regulated genes. Results from these were merged to show the number of up/down regulated genes shared between two experiments. Additional file 1: Figure S1 shows that 98.8% of all pairs of experiments shared less than 10 up/down regulated genes. By spending about 2 h determining this lack of pattern similarity using the BSF, a more sensitive similarity metric did not need to be calculated for these unproductive pairs, thus saving 9.6 days of computation.

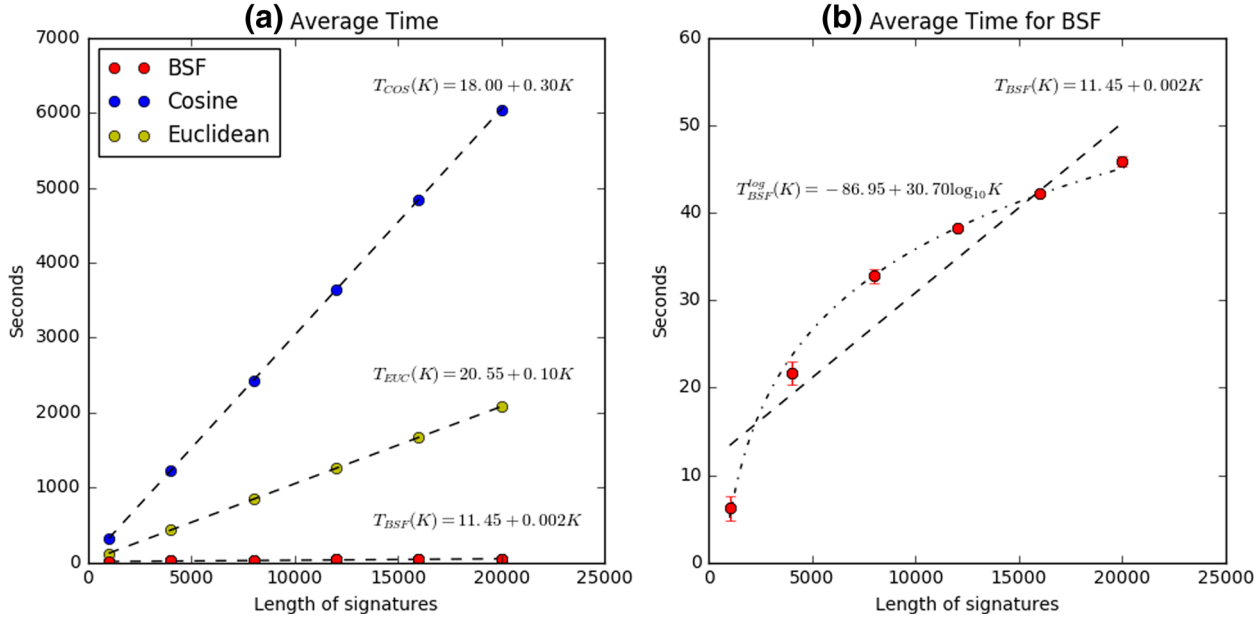

Fig. 2 Benchmark result of BSF. **a** shows the average running time of computing each metric. $T_{COS}(K)$, $T_{EUC}(K)$, $T_{BSF}(K)$ indicate the linear functions to fit the time points in terms of K of cosine similarity, Euclidean distance, and BSF, respectively, where K means the length of each signature. **b** is a zoomed portion of (**a**) to focus on the BSF. The BSF has a logarithmic time complexity while others have a linear time complexity. Refer to Methods and https://github.com/PNNL-Comp-Mass-Spec/BSF_publication

After computing all pairwise comparisons within the LINCS dataset, we built a network connectivity graph to identify similar signatures of gene expression. In exploring this graph, we first examined perturbations using small molecule histone deacetylase (HDAC) inhibitors. We queried the network using nine well-known HDAC inhibitors (belinostat, entinostat, mocetinostat, pracinostat, trichostatin A, vorinostat, rocilinostat, HDAC6 inhibitor ISOX, and valproic acid), which generated a subnetwork of 1066 nodes and 6.3 million edges. Figure 3 shows the network of the top 500 connections. Each node is a perturbation dataset and its size indicates the number of up/down-regulated genes by each perturbation. The weight of an edge shows the similarity score between two nodes. This analysis revealed that the nodes clustered by cell line rather than drug, indicating the response to various HDAC inhibitors is more cell line-specific than drug-specific. In addition to the query perturbations, six additional drug treatments were also found to show a similar signature and thus form part of the sub-network. Among these six are known or putative HDAC inhibitors such as HC toxin [15], panobinostat [16] and scriptaid, one of the first HDAC inhibitors discovered via high-throughput screening [17]. THM-I-94 had previously been hypothesized to act as an HDAC inhibitor based on structural similarity [18], and its clustering here supports that assertion. Other small molecules which cluster with the HDAC inhibitors include unnamed or poorly characterized pharmacological agents. With respect to the differential gene expression pattern shared by these drug treatments, we found enrichment in pathways associated with the cell cycle, MAPK signaling and KEGG's Pathways in Cancer network based on the Fisher's exact test.

A second data mining example from LINCS investigates the effect on human cell lines of non-human medication. Niclosamide is used to treat tapeworm infestations, but has recently been explored as a treatment for MRSA and Zika virus [19, 20]. With the capability of BSF, we identified the LINCS datasets which were most similar to niclosamide (Additional file 2: Figure S2). Even though it wasn't designed to target human cells, niclosamide has strong connectivity with IMD-0354, which is an IKKβ inhibitor and blocks I$\kappa B\alpha$ phosphorylation. In addition to their high concordance in affecting the NF-κB pathway (Additional file 2: Figure S2 b), the two signatures have very high overlap in KEGG's cell cycle pathway, with both showing strong down regulation of cyclins, cyclin dependent kinases, checkpoint kinases and the MCM complex (Additional file 2: Figure S2 c).

Whole genome similarity

A second application of the BSF is to compare gene content across a large number of genomes. Sequenced genomes are functionally annotated both by sequence repositories for inclusion in RefSeq [21] or Uniprot [22],

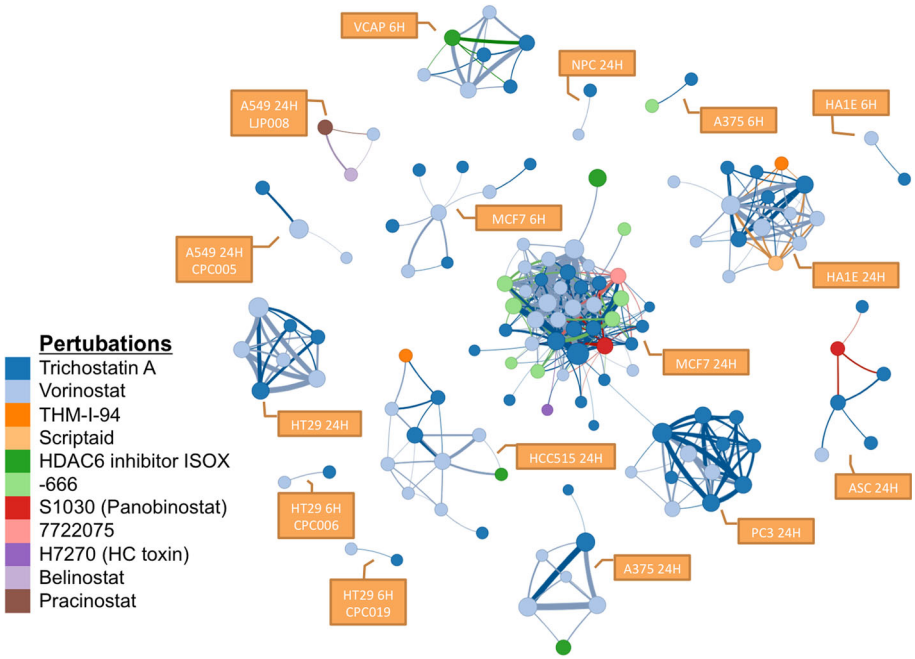

Fig. 3 A sub-network associated to known HDAC inhibitors. The top 500 edges (among 6.3 million) are shown which includes the perturbations from the query (belinostat, pracinostat, trichostatin A, vorinostat, and HDAC6 inhibitor ISOX) and other compounds, some of which are known (scriptaid) and putative (THM-I-94) HDAC inhibitors. H7270 and S1030 are catalog numbers for HC toxin and panobinostat, both recognized HDAC inhibitors. Other perturbations are unnamed drugs (See Methods). The networks naturally form tight clusters, mostly distinguished by cell type and time point. The line width represents its similarity score between two nodes

or they can be annotated by a variety of systems biology style knowledgebases like KEGG [23] or RAST [24]. At the advent of genome sequencing, large scale comparisons of all genomes was used to understand protein function and evolution [25]. As genome sequencing technology has improved, the number of publicly available genomes grows dramatically and an all-versus-all comparison is much less frequently done due to computational costs.

KEGG is a functional annotation system which organizes whole genomes into pathways and molecular interactions for 20,624 protein orthologs in 4648 organisms (356 eukaryotes, 4049 bacteria, and 243 archaea). Annotated genes are identified by their KEGG ortholog number, which are used to define metabolic, signaling and information pathways. Using the BSF, we computed the functional similarity between all genomes annotated by KEGG. Because gene presence/absence is already a binary measure, genome similarity comparisons are a simple and natural use for the BSF. We prepare the binary matrix consisting of 20,624 rows (orthologs) by 4648 columns (genomes) where each cell represents where whether an ortholog is present in a genome. Computation for the full pairwise comparison took 5.2 s.

To show the diversity of genomic content within a taxonomic grouping, we plotted the average number of shared genes between genomes within a taxonomic group, e.g. Homo sapiens compared to all vertebrates (Fig 4). Eukaryotic genomes are generally larger than genomes of bacteria and archaea, and therefore it is not surprising to find a higher number of shared genes among eukaryotes. Additionally, KEGG contains a significant number of orthologs annotated in human disease pathways, and therefore the number of shared genes among animals is notably higher than among plants. We note that there is a broad range of similarity within a taxonomic group, most of which appears to be driven by genome size. For example, within alphaproteobacteria, most organisms share between 500-1100 orthologs. There are, however, a few which share < 140 genes. These are 4 different strains of Candidatus Hodgkinia cicadicola (See Materials and methods), an endosymbiont of the cicada, which has a tiny 144 kb genome [26]. To look at the comparisons within a taxonomic division, we plotted the average similarity between all genera within the class Bacilli (Fig 5). Many genera within Lactobacillales have very low similarity to all other genera of Bacilli. Some of this can be due to low gene counts (e.g. Weissella), however, several have high self-similarity but very low average overlap with any other taxa (e.g. Streptococcus, Leuconostoc, Melissococcus). Thus they likely represent an adaptive genomic response to unique environmental niche.

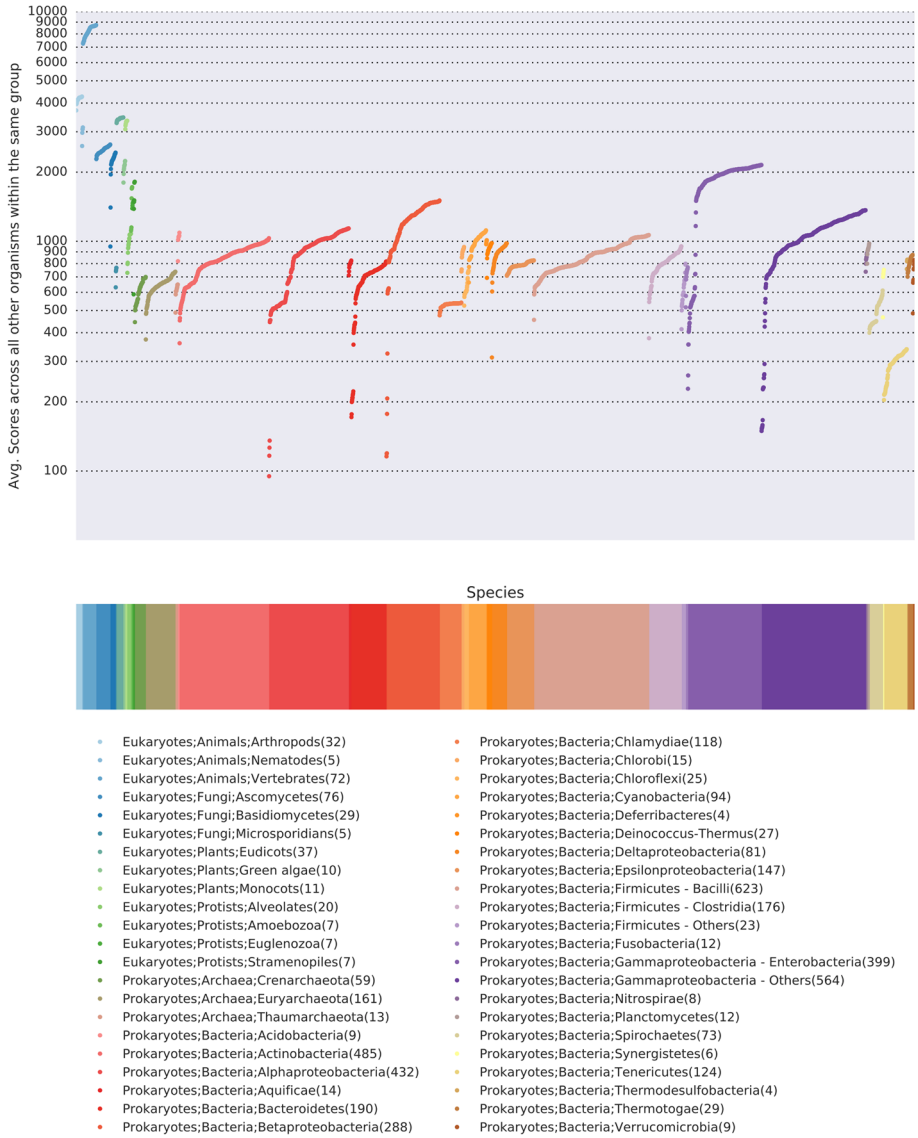

Fig. 4 The average number of shared genes across all the genomes. Using protein orthologs from KEGG, we compare the gene presence/absence across all annotated genomes. Data points represent the average number of shared genes of a genome with the other genomes in the same taxonomic family. The graph is plotted on a log scale. Data points are colored according to taxonomic family membership. For clarity, taxonomic groups with fewer than four organisms were not plotted

Most organisms that share a small number of genes with other organisms are genome reduced and live as obligate symbionts. To compare the functions retained by genome reduced organisms, we plotted the similarity between organisms which had fewer than 600 genes annotated by KEGG (Fig 6). The lack of similarity between these minimalist genomes points to the wide variety of possible adaptations to environmental conditions. This is even true for parasites/pathogens which have nominally similar environments: e.g., Coxiella and Borrelia are both tick borne pathogens infecting humans.

Adapting BSF to other contexts

In the previous two sections, we have shown how the Blazing Signature Filter can be used in common bioinformatics applications to improve the speed of computation. The step of taking data and creating a binary representation is a critical data transformation prior to using the BSF. Therefore, we will describe here a few additional contexts where the BSF can be used to identify similarity between datasets. In some applications, the data is inherently binary. This is like the genome content example for KEGG orthologs. Many other datasets are simple lists and could convert trivially to binary matrices. This

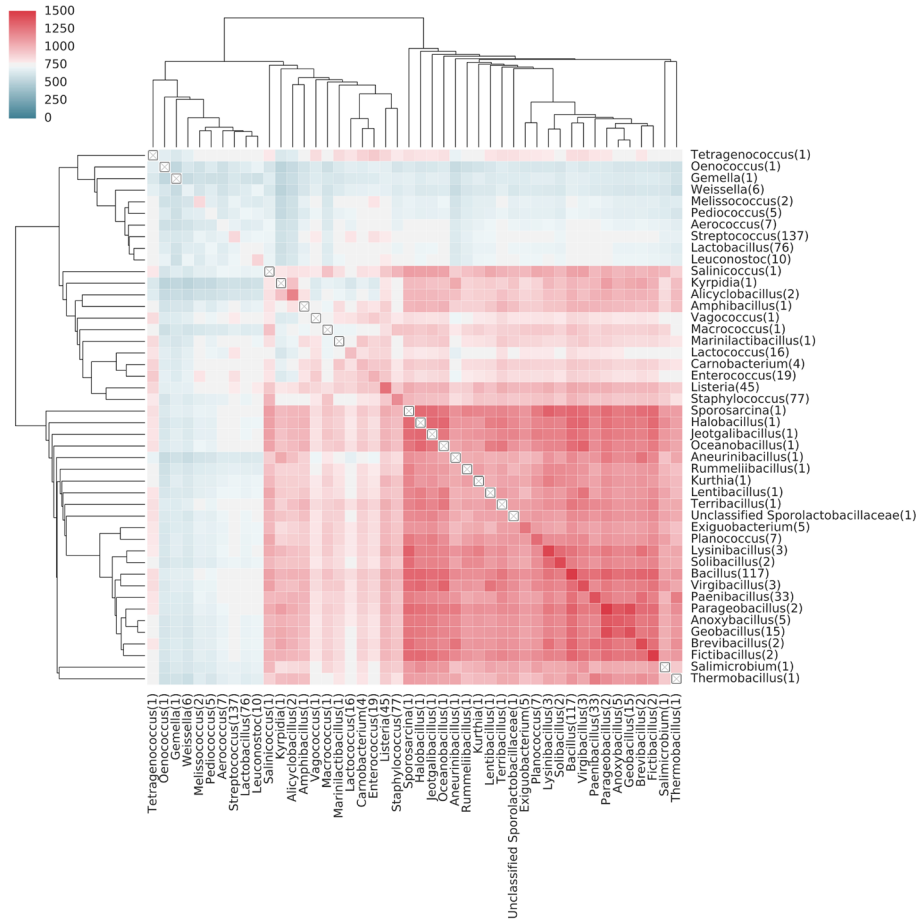

Fig. 5 The average number of shared genes in genera within the class Bacilli. Rows and columns are ordered via hierarchical clustering. When only one sequenced member is present, the diagonal is marked with a boxed ×. A number in the bracket of each label shows the number of species of each genus

is the case for gene sets, such as pathways. By creating a large table enumerating all genes in rows, the columns can represent the genesets as part of a pathway or MSigDB entry. Finally, many datasets are natively continuous, and can be productively analyzed in the BSF by converting the data to binary via a threshold. It is important to remember that converting continuous data to binary will result in some loss of fidelity/information. In this setting it is strongly advised to use the BSF as a filter prior to a more mathematically precise similarity metric using the full-precision data. When using the BSF as a filter, its primary value is to remove the vast majority of computations that are unproductive.

In the LINCS example of gene co-expression, a preliminary step was required to transform the continuous data of gene expression into binary by choosing differentially expressed genes (up or down) according to a statistically derived threshold. The resulting matrix described the set of dis-regulated genes in an experiment. By passing this matrix through the BSF, we identified pairs of experiments that share dis-regulated genes; the similarity of all experiment pairs was saved to a matrix which can be thought of as a graph adjacency matrix. To show an additional example of how to use thresholding to create a binary data matrix out of a continuous data matrix, we will describe how to continue processing the adjacency matrix to identify clusters within the graph of nodes that have a similar set of edges. First, we would convert the adjacency matrix to binary by dropping edges which are deemed to be non-significant. Based on the data in Additional file 1: Figure S1, we might set this threshold at 20. By thresholding the adjacency matrix, we simplify it to note only that there is an edge between node i and node j, and do not retain the weight of that edge. Now in the binary adjacency matrix, the vector for column i states all the other nodes for which there is an edge connecting them to node i. This binary

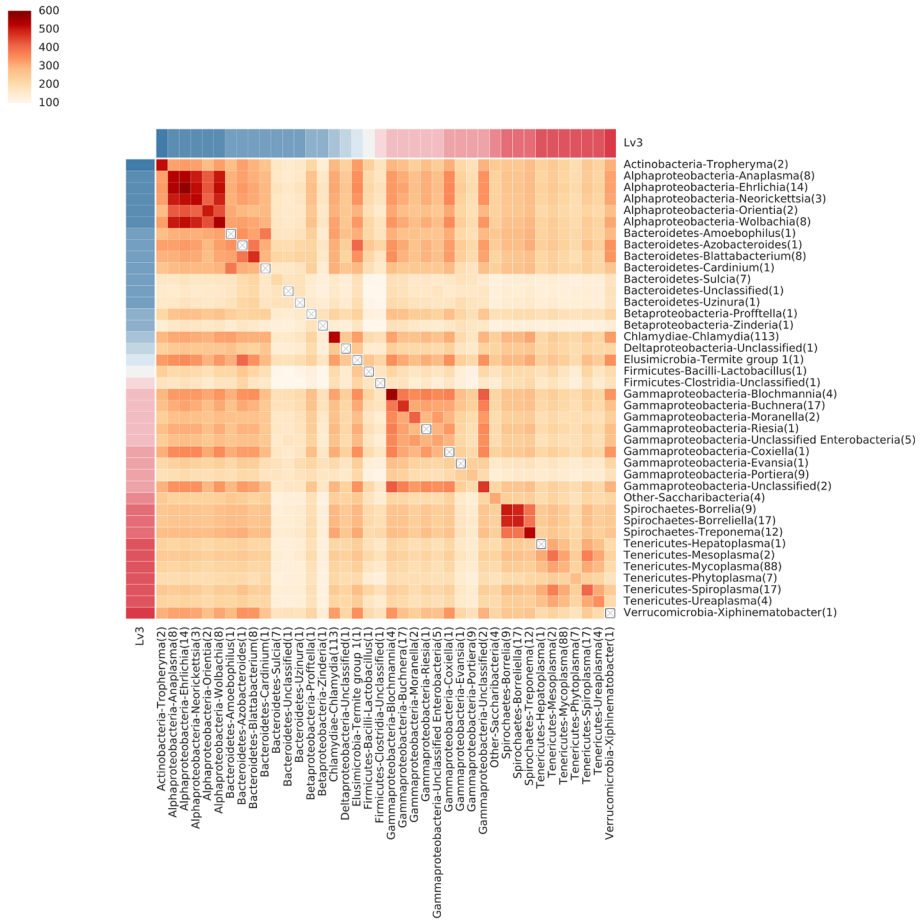

Fig. 6 The average number of shared genes between minimalist bacteria. These organisms have 200-600 genes annotated by KEGG. Organisms are grouped by taxonomy

adjacency matrix can be sent through the BSF. The semantic meaning of this computation will be: How many nodes are incident on both node i and j? Therefore, this shows the similarity of edges between two nodes and is an effective heuristic for clustering a graph. Given the speed and efficiency of the BSF computation, this is an effective filter for graphs that contains hundreds of thousands of nodes and millions/billions of edges.

Conclusions

As technologies for scientific data generation continue to dramatically improve and facilitate an ever greater characterization and description of the natural world, data mining for hypothesis generation and validation becomes both more important and more technically challenging. With the BSF, we introduce a simple and efficient method for identifying patterns, or signatures, in massive amounts of data. This is enabled by the rapid pairwise comparison of data as binary vectors. We show two example applications where pairwise comparisons are a common bioinformatics task: comparing genomes for similar gene content and identifying experiments with similar gene expression patterns. In both applications, the sheer number of comparisons would be time prohibitive without optimized computational methods such as the BSF.

New experimental technologies will improve the ability to make comprehensive datasets. For example, the task of identifying genetic interactions between pairs of genes was previously difficult to scale to whole genomes [27]. However, the CRISPR technology now makes is dramatically simpler to explore the effects of multiple knockouts [28], and we anticipate that whole genome double knockouts will be common in the near future. Even for genomically compact bacteria with ~ 2000 genes, the number of double knockouts exceeds millions of strains. The subsequent task of identifying similarity (or differences) between the millions of strains will

then require trillions of calculations. In these scenarios, efficient similarity metrics like the BSF will be essential to enable scientific discovery.

For datasets that are natively binary (e.g. gene content), the BSF works trivially. Another computation that is inherently binary is the set overlap calculation that is part of a Fisher's exact test, commonly used for gene set enrichment. For datasets which are numeric or categorical, use of the BSF requires a meaningful transformation into binary space such as was done in the LINCS gene expression compendium. A wide variety of bioinformatics needs, e.g. proteomics library searches and FBA modeling, could benefit from using the BSF to quickly filter out unproductive data point prior to a more sensitive computation on the native (i.e. non-binary) data.

Materials and methods

The data and analysis methods for all figures are available at https://github.com/PNNL-Comp-Mass-Spec/BSF_publication.

LINCS application

The LINCS L1000 project measures gene expression (transcriptomics) over different cell lines with a broad range of small molecule perturbations and genetic manipulations (knockout, knockdown and over-expression) [29, 30]. In this manuscript, we use the L1000 mRNA gene-expression signatures computed using the characteristic direction signatures method [6, 29], giving binary up and down regulated genes for each of the $\sim 117,000$ datasets. It is publicly available at http://amp.pharm.mssm.edu/public/L1000CDS_download/. This site contains a single MongoDB bson file for the expression values for each of the perturbations and associated meta-data describing experimental parameters. After downloading, we converted the expression values into two binary matrices. If the characteristic direction algorithm determined that a gene was up-regulated in an experiment, the corresponding bit in the binary matrix file for up-regulated genes was set to 1. We created the binary matrix for down-regulated genes in the same manner. When trying to interpret the function of various perturbations, we used the pert_desc field in the MongoDB to get the name of the compounds and combined that with the meta-data available at NCBI-GEO for GSE70138 which contains a more complete description of perturbations in the pert_info.txt file.

KEGG application

All KEGG annotations were taken from in Release 81.0 downloaded on January 1, 2017. A binary matrix representing the table of orthologs was created where rows represent KEGG Ortholog accessions and the columns represent genomes. If KEGG has annotated an ortholog as being present in a genome, the corresponding matrix coordinate was set to 1. This matrix was fed into the BSF using the python interface to compare all genomes against each other. The output of this shows the number of shared orthologs between genomes. The Figs. 4, 5, and 6 were all derived from this output file using iPython notebooks which are publicly available in the kegg_data section at https://github.com/PNNL-Comp-Mass-Spec/BSF_publication.

Benchmarking

The synthetic benchmarking data was created as a table ($15K$ columns $\times$ $20K$ rows) of floating point numbers drawn randomly from the Gaussian distribution of $N(0, 0.5)$. Rows can be thought of as different gene measurements, and columns as distinct datasets. This continuous data was binarized into two tables to represent the extremes of the distribution, i.e. values < -0.6 were written as 1 in a binary table representing the 'low' values and values > 0.6 were written as 1 to a binary table representing high values.

We use the cosine distance and Euclidean distance on the original floating point data to compare the performance with BSF. In the manuscript, we discussed clock cycles required for various operations. For a description of core clock cycles per instruction set, refer to the latency and reciprocal throughput in http://www.agner.org/optimize/instruction_tables.pdf. Supposing the M-by-N signature matrix $[S_1, S_2, S_3, \cdots, S_N]$, the formulae for cosine and Euclidean similarity are:

$$euclidean(S_i, S_j) = \sqrt{\sum_{k=1}^{M} (a_k - b_k)^2} \quad (1)$$

$$cosine(S_i, S_j) = \frac{s_i \cdot s_j}{|s_i||s_j|} = \frac{\sum_{k=1}^{M} a_k b_k}{\sqrt{\sum_{k=1}^{M} a_k^2}\sqrt{\sum_{k=1}^{M} b_k^2}} \quad (2)$$

where $S_i = [a_1, a_2, a_3, \cdots, a_M]^T$ and $S_j = [b_1, b_2, b_3, \cdots, b_M]^T$ $(i, j = 1, 2, 3, \cdots, N)$.

Availability and requirements

Project name: Python wrapper for Blazing Signature Filter

Project home page: https://github.com/PNNL-Comp-Mass-Spec/bsf-py

Operating system(s): Linux/OSX/Windows 8+

Programming language: C++ and Python

Other requirements: e.g. GCC 4.9 or higher, Python 3.4 or higher

License: BSD-2 License

Additional files

Additional file 1: Figure S1. Score distribution of the 6.89 billion pairwise comparisons in the LINCS L1000 dataset. The color of each point describes the number of pairs which have shared genes. X and Y axes indicate the number of shared up-regulated genes and down-regulated genes, respectively. For example, a point of (50, 50) has 147, which means 147 pairs of two signatures share 50 up-regulated genes and 50 down-regulated genes. The overwhelming majority of pairwise comparisons, ∼6.80 billions or 98.8%, are located in a small box of up-regulated genes < 10 and down-regulated genes < 10. These represent pairs of experiments, which do not share a discernable signature of regulated gene expression and are unproductive data mining events. (PNG112 kb)

Additional file 2: Figure S2. A sub-network of LINCS L1000 experiments most similar to niclosamide. (a) We extracted the network for 88 datasets associated with non-human medications such as niclosamide (tapeworm infestations) and daminozide (plant growth regulator). It shows 257 experiments of 20 drugs highly connected to these 88 signatures. Refer to Materials and Methods for details. (b) Differentially expressed genes shared between niclosamide and IMD 0354, an IKKβ inhibitor. Most of all common genes are down-regulated and cell cycle looks slow down. (c) Shared differential genes shown for the NF-κB signaling pathway; most of the genes are up-regulated. (PNG 690 kb)

Abbreviations

BLAST: Basic local alignment search tool; BSF: Blazing signature filter; CRISPR: Clustered regularly interspaced short palindromic repeats; FBA: Flux balance analysis; HDAC: Histone deacetylases; KEGG: Kyoto encyclopedia of genes and genomes; LINCS: Library of integrated network-based cellular signatures; MSigDB: Molecular signatures database; RAST: Rapid annotation using subsystem technology; RefSeq: Reference sequence database; UniProt: Universal protein knowledgebase

Funding

This work was supported by the National Cancer Institute (NCI) CPTAC award U24 CA210972 (SHP), and the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Research, Pan-omics Program. Battelle operates the Pacific Northwest National Laboratory for the DOE under contract DE-AC05-76RLO01830. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Availability of data and materials

The BSF library is available under the BSD license from https://github.com/PNNL-Comp-Mass-Spec/bsf-py and https://github.com/PNNL-Comp-Mass-Spec/bsf-core. The repository provides tutorials and installation guides for easy testing and performing simple examples. The library can be installed through the Docker by typing *docker pull coldfire79/bsf-py*.

Authors' contributions

J-YL and SHP designed the method. J-YL, GMF and RW wrote software. J-YL, HSW and SHP interpreted data. J-YL and SHP wrote the manuscript with input from all authors. All authors read and approved the final manuscript.

Ethics approval and consent to participate

Not applicable

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details

[1]Integrative Omics, Pacific Northwest National Laboratory, 99352 Richland, WA USA. [2]Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory, 99352 Richland, WA USA.

Received: 25 July 2017 Accepted: 17 May 2018
Published online: 11 June 2018

References

1. Henikoff S, Henikoff JG. Amino acid substitution matrices from protein blocks. Proc Natl Acad Sci U S A. 1992;89(22):10915–9.
2. Nesvizhskii AI, Keller A, Kolker E, Aebersold R. A statistical model for identifying proteins by tandem mass spectrometry. Anal Chem. 2003;75(17):4646–58.
3. Saitou N, Nei M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol. 1987;4(4):406–25.
4. Stuart JM, Segal E, Koller D, Kim SK. A gene-coexpression network for global discovery of conserved genetic modules. Science. 2003;302(5643):249–55. https://doi.org/10.1126/science.1087447.
5. Lamb J, Crawford ED, Peck D, Modell JW, Blat IC, Wrobel MJ, Lerner J, Brunet JP, Subramanian A, Ross KN, Reich M, Hieronymus H, Wei G, Armstrong SA, Haggarty SJ, Clemons PA, Wei R, Carr SA, Lander ES, Golub TR. The connectivity map: using gene-expression signatures to connect small molecules, genes, and disease. Science. 2006;313(5795):1929–35. https://doi.org/10.1126/science.1132939.
6. Duan Q, Reid SP, Clark NR, Wang Z, Fernandez NF, Rouillard AD, Readhead B, Tritsch SR, Hodos R, Hafner M, Niepel M, Sorger PK, Dudley JT, Bavari S, Panchal RG, Ma'ayan A. L1000cds2: Lincs l1000 characteristic direction signatures search engine. NPJ Syst Biol Appl. 2016;2:16015. https://doi.org/10.1038/npjsba.2016.15.
7. Smith TF, Waterman MS. Identification of common molecular subsequences. J Mol Biol. 1981;147(1):195–7.
8. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990;215(3):403–10. https://doi.org/10.1016/S0022-2836(05)80360-2.
9. Kent WJ. Blat–the blast-like alignment tool. Genome Res. 2002;12(4):656–64. https://doi.org/10.1101/gr.229202.ArticlepublishedonlinebeforeMarch2002.
10. Frank A, Tanner S, Bafna V, Pevzner P. Peptide sequence tags for fast database search in mass-spectrometry. J Proteome Res. 2005;4(4):1287–95. https://doi.org/10.1021/pr050011x.
11. Tabb DL, Saraf A, Yates JR. Gutentag: high-throughput sequence tagging via an empirically derived fragmentation model. Anal Chem. 2003;75(23):6415–21. https://doi.org/10.1021/ac0347462.
12. Wu K, Ahern S, Bethel EW, Chen J, Childs H, Cormier-Michel E, Geddes C, Gu J, Hagen H, Hamann B, Koegler W, Lauret J, Meredith J, Messmer P, Otoo E, Perevoztchikov V, Poskanzer A, Prabhat, Rübel O, Shoshani A, Sim A, Stockinger K, Weber G, Zhang WM. Fastbit: interactively searching massive data. J Phys Conf Ser. 2009;180(1):012053.
13. Myers G. A fast bit-vector algorithm for approximate string matching based on dynamic programming. J ACM. 1999;46(3):395–415. https://doi.org/10.1145/316542.316550.
14. Haque IS, Pande VS, Walters WP. Anatomy of high-performance 2d similarity calculations. J Chem Inf Model. 2011;51(9):2345–51. https://doi.org/10.1021/ci200235e.
15. Walton JD. Hc-toxin. Phytochemistry. 2006;67(14):1406–13. https://doi.org/10.1016/j.phytochem.2006.05.033.
16. Ellis L, Pan Y, Smyth GK, George DJ, McCormack C, Williams-Truax R, Mita M, Beck J, Burris H, Ryan G, Atadja P, Butterfoss D, Dugan M, Culver K, Johnstone RW, Prince HM. Histone deacetylase inhibitor panobinostat induces clinical responses with associated alterations in gene expression profiles in cutaneous t-cell lymphoma. Clin Cancer Res. 2008;14(14):4500–10. https://doi.org/10.1158/1078-0432.CCR-07-4262.
17. Su GH, Sohn TA, Ryu B, Kern SE. A novel histone deacetylase inhibitor identified by high-throughput transcriptional screening of a compound library. Cancer Res. 2000;60(12):3137–42.
18. Siavelis JC, Bourdakou MM, Athanasiadis EI, Spyrou GM, Nikita KS. Bioinformatics methods in drug repurposing for alzheimer's disease. Brief Bioinform. 2016;17(2):322–35. https://doi.org/10.1093/bib/bbv048.
19. Xu M, Lee EM, Wen Z, Cheng Y, Huang WK, Qian X, Tcw J, Kouznetsova J, Ogden SC, Hammack C, Jacob F, Nguyen HN, Itkin M, Hanna C, Shinn P, Allen C, Michael SG, Simeonov A, Huang W, Christian KM, Goate A, Brennand KJ, Huang R, Xia M, Ming GL, Zheng W, Song H, Tang H. Identification of small-molecule inhibitors of zika virus infection and induced neural cell death via a drug repurposing screen. Nat Med. 2016;22(10):1101–1107. https://doi.org/10.1038/nm.4184.
20. Rajamuthiah R, Fuchs BB, Conery AL, Kim W, Jayamani E, Kwon B, Ausubel FM, Mylonakis E. Repurposing salicylanilide anthelmintic drugs to combat drug resistant staphylococcus aureus. PLoS One. 2015;10(4):0124595. https://doi.org/10.1371/journal.pone.0124595.

21. Pruitt KD, Tatusova T, Brown GR, Maglott DR. Ncbi reference sequences (refseq): current status, new features and genome annotation policy. Nucleic Acids Res. 2012;40(Database issue):130–5. https://doi.org/10.1093/nar/gkr1079.
22. UniProt C. Uniprot: a hub for protein information. Nucleic Acids Res. 2015;43(Database issue):204–12. https://doi.org/10.1093/nar/gku989.
23. Kanehisa M, Goto S. Kegg: kyoto encyclopedia of genes and genomes. Nucleic Acids Res. 2000;28(1):27–30.
24. Aziz RK, Bartels D, Best AA, DeJongh M, Disz T, Edwards RA, Formsma K, Gerdes S, Glass EM, Kubal M, Meyer F, Olsen GJ, Olson R, Osterman AL, Overbeek RA, McNeil LK, Paarmann D, Paczian T, Parrello B, Pusch GD, Reich C, Stevens R, Vassieva O, Vonstein V, Wilke A, Zagnitko O. The rast server: rapid annotations using subsystems technology. BMC Genomics. 2008;9:75. https://doi.org/10.1186/1471-2164-9-75.
25. Tatusov RL, Galperin MY, Natale DA, Koonin EV. The cog database: a tool for genome-scale analysis of protein functions and evolution. Nucleic Acids Res. 2000;28(1):33–6.
26. McCutcheon JP, McDonald BR, Moran NA. Origin of an alternative genetic code in the extremely small and gc-rich genome of a bacterial symbiont. PLoS Genet. 2009;5(7):1000565. https://doi.org/10.1371/journal.pgen.1000565.
27. Tong AH, Evangelista M, Parsons AB, Xu H, Bader GD, Page N, Robinson M, Raghibizadeh S, Hogue CW, Bussey H, Andrews B, Tyers M, Boone C. Systematic genetic analysis with ordered arrays of yeast deletion mutants. Science. 2001;294(5550):2364–8. https://doi.org/10.1126/science.1065810.
28. Boettcher M, Tian R, Blau J, Markegard E, Wu D, Biton A, Zaitlen N, McCormick F, Kampmann M, McManus MT. Decoding directional genetic dependencies through orthogonal crispr/cas screens. bioRxiv. 2017. https://doi.org/10.1101/120170.
29. Clark NR, Hu KS, Feldmann AS, Kou Y, Chen EY, Duan Q, Ma'ayan A. The characteristic direction: a geometrical approach to identify differentially expressed genes. BMC Bioinformatics. 2014;15:79. https://doi.org/10.1186/1471-2105-15-79.
30. Liu C, Su J, Yang F, Wei K, Ma J, Zhou X. Compound signature detection on lincs l1000 big data. Mol Biosyst. 2015;11(3):714–22. https://doi.org/10.1039/c4mb00677a.