RESEARCH

Sort-Seq Tools: software for analyzing massively parallel experiments on mutagenized sequences

William Ireland[1], Rob Phillips[2] and Justin B Kinney[3*]

*Correspondence: jkinney@cshl.edu
[3]Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory, 11375, Cold Spring Harbor, NY
Full list of author information is available at the end of the article

Abstract

We introduce Sort-Seq Tools, a software package for analyzing a variety of massively parallel experiments on mutagenized sequences. Such experiments include Sort-Seq studies of bacterial promoters, massively parallel reporter assays of mammalian enhancers, and deep mutational scanning experiments of proteins. Methods for data simulation, data preprocessing, sequence analysis, quantitative modeling, and visualization are provided. These functionalities are available through a command-line interface, as well as through a Python module, `sortseq`, that is available on PyPI. Example applications of Sort-Seq Tools in the context of real and simulated data sets are described.

Keywords: sequence analysis; transcriptional regulation; deep mutational scanning; mutual information

Introduction

Ultra-high-throughput DNA sequencing is being used to do far more than just sequence genomes [1]. One rapidly growing area of application is the high-resolution measurement of the quantitative sequence-function relationships that govern how specific sequences of interest work. A variety of "massively parallel" experimental methods have already been described for dissecting the functional architecture of transcriptional regulatory sequences [2, 3, 4] and for mapping functional domains and residues within proteins [5].

Figure 1 Various massively parallel experiments. (A) The "Sort-Seq" assay of [6]. Mutagenized promoters are used to drive the expression of a fluorescent protein (green). Cells carrying expression constructs are sorted in "bins" according to measured fluorescence using FACS, after which the mutant promoters found in each bin of sorted cells are sequenced. (B) The "massively parallel reporter assay" of [8]. Variant enhancers are used to drive the transcription of RNA that contains enhancer-specific tags. Expression constructs are transfected into cell culture, after which tag-containing RNA is isolated and sequenced. Output sequences consist of the variant enhancers that correspond to expressed tag. (C) The "deep mutational scanning" assay of [15]. Randomly mutagenized proteins (colored bells) that bind a ligand (brown circles) are expressed on the surface of phage (gray rectangle). Panning is used to enrich for phage that bind this ligand of interest. The variant proteins enriched after multiple rounds of panning are sequenced.

Sort-Seq (Fig. 1A) was the first massively parallel assay described for dissecting transcriptional regulatory sequences in living cells [6]. Starting with a library consisting of a randomly mutagenized version of a specific bacterial promoter, Sort-Seq is able to measure the transcriptional activity of thousands to millions of these variant promoters. This is accomplished by coupling fluorescence-activated cell sorting (FACS) to ultra-high-throughput DNA sequencing. From these data one can identify functional binding sites, quantitatively model the sequence specificities of the

proteins that bind these sites, and infer quantitative biophysical models for how the wild type promoter functions [6]. Similar experiments have been performed in yeast [7].

Multiple "massively parallel reporter assays" (MPRAs) [8, 9, 10] have been described for dissecting the functional architecture of enhancers in mammalian cells (Fig. 1B). These approaches have their origin in [11], which used a similar method to dissect promoter function *in vitro*. Mutant versions of a specific enhancer of interest are used to drive the expression of RNA transcripts that include enhancer-specific barcodes, or "tags." Expression constructs are introduced into cells, after which RNA is extracted and expressed tags are sequenced. This provides a measurement of the transcriptional activity of each variant enhancer. Such experiments can be performed in cell culture [8], in dissected tissue [10], or in the organs of living animals [9].

A variety of "deep mutational scanning" (DMS) experiments have been described for dissecting the functional features of proteins [5]. Fig. 1C illustrates the earliest such study [12]. To identify which regions of a specific protein were responsible for binding a target ligand, phage expressing mutant versions of this protein on their surface were grown. Phage that bound this ligand were enriched using multiple rounds of "panning" and then sequenced. DMS experiments using a wide range of selection procedures tailored to different protein activities of interest have since been described [5].

The software available for analyzing data from massively parallel experiments such as these is limited. Indeed virtually all of these experiments, including those of the authors [6, 13], have thus far been analyzed using custom scripts. To our knowledge, two packages have been described for analyzing DMS experiments: Enrich [12] and DMS Tools [14]. Both of these packages provide routines for estimating and visualizing per-nucleotide enrichment of pre- and post-selected sequences. However, there is far more that one can (and should) do when analyzing data from massively parallel experiments on mutagenized sequences.

Figure 2 A common form for massively parallel experiments on mutagenized sequences. (A) In all these experiments a "library" is generated by introducing random mutations into a specific "wild type" sequence of interest (blue). Library sequences are used as input to an experiment (black box) that outputs these sequences into one or more "bins;" library sequences are assigned to bin 0 by convention. A subset of sequences in each bin (opaque sequences) are then sequenced. (B) The resulting data consist of two or more bins of variant sequences. Sort-Seq Tools provides routines for dealing with such data, including routines for preprocessing, simulation, quality control, sequence analysis, and quantitative modeling. Convenient visualization methods are also provided.

Here we describe a software package, called Sort-Seq Tools, for analyzing data from massively parallel experiments performed on libraries of mutagenized sequences. Specifically, Sort-Seq Tools is designed as a fast and easy-to-use suite of functions for analyzing any massively parallel experiment that has the general form depicted in Fig. 2A. The functionality of Sort-Seq Tools is summarized in Fig. 2B: methods are provided for analyzing data that consists of multiple lists (one for each bin) of sequences that differ from a specific sequence of interest by randomly scattered substitution mutations.[1] Sort-Seq Tools includes methods for performing quality

[1]In particular, all sequences are assumed to have the same length.

control analysis, basic sequence analysis, and modeling of quantitative sequence-function relationships. Data preprocessing routines are also provided. Methods for simulating a variety of massively parallel experiments are also included; these simulation tools can be used to aid in experimental design, or to test the capabilities of analysis pipelines.

We now provide a step-by-step elaboration of the functionality of Sort-Seq Tools. In particular, we demonstrate the capabilities of this software on the Sort-Seq data of [6], the MPRA data of [8], and the deep mutational scanning data of [15]. The simulation capabilities of Sort-Seq are also demonstrated and used to evaluate the multiple methods of quantitative model inference. Figs. 3-10 illustrate these analyses, as well as the specific commands used to perform them. Sort-Seq Tools is available as open source software at `https://github.com/jbkinney/15_sortseqtools`; the data sets used in these analyses and the scripts used to produce the figures herein are included in this repository. Sort-Seq Tools is also available as the python module `sortseq`, available from PyPI.

Methods and Results

Overview

Figure 3 Example analysis using Sort-Seq Tools. (A) The Sort-Seq Tools function `profile_freqs` transforms `data.txt`, a single column list of transcription factor binding sites, into `freqs.txt`, a list of nucleotide occurrence frequencies at each position in the alignment. The Sort-Seq Tools function `draw --logo` provides a sequence logo visualization of `freqs.txt`. (B) Command line implementation of the analysis in (A). Every Sort-Seq Tools method is a subcommand of the function `sortseq`. (C) Sort-Seq Tools commands can be piped in series. (D) Analysis using Sort-Seq Tools can also be performed in Python via the `sortseq` package.

Sort-Seq Tools provides set of transform tabular text files of one type into tabular text files of another. An example of this is given in Fig. 3A. Here, a table `data.txt` contains a list of aligned DNA binding sites for the *Drosophila melanogaster* transcription factor hunchback. The Sort-Seq Tools function `profile_freqs` transforms this table into a second table, `freqs.txt`, that lists the occurrence frequencies of each nucleotide at each position within the alignment.

Sort-Seq Tools also provides a single function, `draw`, for visualizing the contents of tabular text files. The specific columns in the text file, together with optional flag aguments, determine how the tabular data is interpreted graphically. For instance, in Fig. 3A, the function `draw --logo` then transforms `freqs.txt` into a sequence logo [16] using the WebLogo package [17]. The resulting graphics are stored as a .PDF file.

The functionality of Sort-Seq is primarily designed to be used at the command-line, and every method is implemented as a subcommand of the function `sortseq`. For example, Fig. 3B shows the commands used to perform the analysis illustrated in Fig. 3A. For many such analysis, commands can be piped together in series (Fig. 3C). This reduces the need for temporary files and can speed up exploratory data analysis.

Alternatively, Sort-Seq Tools can be used from within Python via the `sortseq` package, which is available on PyPI. Indeed, the command line function `sortseq` is a wrapper for functions in the `sortseq` Python package. Fig. 3D illustrates Python

code equivalent to the command line analysis of Fig. 3B and C. When used from inside Python, the methods of Sort-Seq Tools use `pandas` data frames instead of tabular text files.

A key aspect of Sort-Seq Tools is its open format. Aside from `draw`, each function both inputs and outputs a human-readable table. It is therefore possible to grab any intermediate file in a Sort-Seq Tools analysis and analyze it using custom software. It is also possible to use isolated functions in Sort-Seq Tools within other pipelines.

The tables inputted and outputted by Sort-Seq Tools must have appropriate column labels. For instance, the labels "`pos`," "`freq_A`," etc. alert `draw` that the table provided is a list of DNA nucleotide frequencies and may therefore be visualized as a sequence logo. Table 1 lists the labels recognized in the current Sort-Seq Tools. Label modifiers, such as "`_A`" in "`freq_A`," are preceded by an underscore.

Table 1 Table column labels recognized by Sort-Seq Tools.

Header	Description
`bin`	number of sequence bin (0, 1, 2, ...)
`file`	name of file containing sequence or tag data
`ct`	count of sequence or tag occurrences
`seq`	sequence of DNA (default), RNA (`_rna`), or protein (`_pro`)
`tag`	expression tag
`pos`	nuclotide/residue position within a set of aligned sequences
`wt`	wild-type nucleotide or residue
`obs`	observed nucleotide or residue
`mut`	mutation rate
`le`	log enrichment
`freq`	occurrence frequency of nucleotide/residue
`val`	model parameter
`lr`	log ratio
`info`	mutual information in bits
`_0, _1 ...`	associated bin number
`_A, _C ...`	associated nucleotide or residue
`_err`	estimated uncertainty

Quality Control

Figure 4 Library quality control. (A) `library.txt` contains library sequence data from [6]; listed are sequences and corresponding occurrence counts. (B) Pipeline for computing the per-base mutation rate: `profile_counts` computes base occurrences at each nucleotide position; `profile_mutrate` then computes position-specific mutation rates; `draw` then plots these mutation rates. (C) `draw` applied directly to `library.txt` yields a Zipf plot illustrating library diversity. (D) The commands used to perform these analyses.

The sequence data files used as input to Sort-Seq Tools functions are tabular text files containing one "`seq`" column that lists the sequences observed and one (optional) "`ct`" column, which lists the number of times each sequence was observed. Fig. 4A shows a file, `library.txt`, that contains library sequences from the Sort-Seq experiments of [6]. We now illustrate how Sort-Seq Tools can be used to perform various quality controls test on such data. To make the pipelines discussed from here on more compact, we will often illustrate tables using only their column labels.

This sequence library from [6] was designed to have a uniform 12% mutation rate within the region of interest. Fig. 4B illustrates a pipeline for verifying this library composition. First, the Sort-Seq Tools command `profile_counts` is used to count the number of occurrences of each base at each position within the

aligned sequences of `library.txt`. These counts are saved to `counts.txt`. Next, `profile_mutate` is used to determine the mutation rate at each position, which is saved to `mutate.txt`.[2] Both the estimated mutation rate (`mut`) and the uncertainty in this estimate (`mut_err`) are computed. Finally, the mutation profile in `mutate.txt` is illustrated with `draw`. From this we see that the measured mutation rates are indeed approximately 12%.

The position-specific mutation rate does not, however, reveal how diverse the library is. Sort-Seq Tools makes it easy to check this, however, using the `draw` command. Applied directly `library.txt`, this command returns a Zipf plot of sequence counts versus rank order. In this way we are able to verify that the library indeed contains [SHOULD SEE TOTAL NUMBER OF SEQUENCES] and is not dominated by a large number of counts for very few sequences. We emphasize that the libraries used in massively parallel experiments can easily loose diversity due to unforeseen experimental problems, and it is important to check whether this has occurred.

Sequence analysis

Figure 5 Information footprints. Shown is the pipeline for performing the information footprint computations in [6]. (A) Sort-Seq data from [6] is distributed among ten files: `library.txt`, `bin_1.txt`,...,`bin_9.txt`. (B) To collate these files, we prepare the list of files shown in `files.txt`. (C) This file list (`files.txt`) is transformed by `gatherseqs` into a single data file (`data.txt`), which lists the counts in each bin for each unique sequence. The data file (`data.txt`) is then transformed by `profile_info` into an information footprint (`info.txt`). The `draw` then renders a graphical information footprint from `info.txt`. Note that information is quantified in the familiar units of "bits." (D) The known binding sites of CRP and of RNAP are evident in the information footprint computed in (C). (E) The Sort-Seq Tools commands used to perform this analysis.

The first things one typically wishes to learn from massively parallel experiments like those in Fig. 1 is the location of functional nucleotides or residues within the wild type sequence. A convenient way to identify these locations is to compute "information footprints," which quantify how much information the identity of the nucleotide or residue at each position carries about the bin in which the sequence is found [6].

Here we show how Sort-Seq Tools can be used to compute information footprints. This is demonstrated on Sort-Seq data from [6]. This experiment assayed randomly mutated versions of a 75 bp region of the *lac* promoter of *E. coli*. This experiment yielded ten bins of sequences with bin 0 containing sequences from the unsorted library. In analyzing this data set, we begin with the ten files illustrated in Fig. 5A. Each of these files is a list of sequences with corresponding counts.

To process these data we need one additional file, `files.txt`, which lists each bin number along with the name of the file that contains the associated data (Fig. 5B). Next, we pass `files.txt` as input to the Sort-Seq Tools function `gatherseqs`, which returns a dataset file ,`data.txt` listing each unique sequence along with the number of times it was observed in each bin.

[2]For convenience, the wild type base at each position is assumed to be the most frequent base; this behavior, however, can be overridden by manually specifying the wild type sequence via the `--wt` flag.

Next, `data.txt` is used as input to the Sort-Seq Tools function `profile_info`, which outputs an `info.txt` file that lists the mutual information in bits (`info` column) at each position (`pos` column), along with the estimated uncertainty in this information value (`info_err` column). Passing `info.txt` to the `draw` command then produces the desired information footprint.

The *lac* promoter of *E. coli* has two functional binding sites within the 75bp region probed by the Sort-Seq experiments of [6]: one binding site for the σ^{70} RNA polymerase holoenzyme (RNAP), and one binding site for the transcription factor CRP. Both proteins read out their binding sites in a bipartite way, illustrated in Fig. 5D. This sequence readout is well reflected in the information footprint produced by the Sort-Seq Tools pipeline in Fig. 5C.

Figure 6 Enrichment analysis. Shown is the pipeline for computing enrichment profiles from DMS data as in [15]. (A) First, the collated set of DNA sequence data (`dnadata.txt`) is transformed into protein sequence data (`proteindata.txt`) using the `convertseqs` command with the `--protein` flag. Next, the command `profile_enrichment` computes log enrichment ratios for each possible amino acid / stop codon at each position (`enrichment.txt`). This enrichment profile is visualized with `draw`. (B) The Sort-Seq Tools commands used to perform this analysis.

The computation of "enrichment ratios" is a common type of analysis in the context of deep mutational scanning experiments (see [5]). Here we illustrate how to compute enrichment profiles using the data from [15]. This experiment probed how the binding affinity of WW domains for the peptide ligand (GTPPPPYTVG) depends on WW domain sequence. To do this, phage-displayed WW domain proteins were selected using serial rounds of panning. Protein coding sequences recovered after either three rounds (bin 1) or six rounds (bin 2) of panning were sequenced. The enrichment among selected sequences versus library sequences of each possible amino acid / stop codon at each position was then computed. Here we use Sort-Seq Tools to compute the enrichment profile of bin 2 relative to the library (bin 0).

Fig. 6A shows the pipeline used to compute this enrichment profile. We start from a collated data file `dnadata.txt`, which can be constructed from individual data files using `gatherseqs` as in Fig. 5. This collated data file lists how many time each DNA coding sequence was observed in each bin. We convert these DNA sequences to peptide sequences using the command `convertseqs`, which requires an additional flag (`--protein` or `--rna`) indicating the type of target sequence. Next, we profile the enrichment in bin 2 relative to bin 0 by using the `profile_enrichment` command. The enrichment profile lists the wild type amino acid / stop codon at each position (`wt` column) as well as the log enrichment of each of the twenty one amino acid / stop codon possibilities (`le_A`, ..., `le_Y`, `le_*` columns). The command `draw` provides a quick illustration of this enrichment profile.

Figure 7 Tag expression analysis. Shown is the pipeline for computing log expression values from MPRA data as in [8]. Input to the pipeline consists of (A) tag sequencing data in `library.txt` and `expression.txt`, (B) A list of data files, `files.txt`, and (C) a file `key.txt` associating each tag with a unique sequence. (D) The file list and tag key are transformed by `gatherseqs` into a collated data set, `data.txt`. `logratios` then computes log count ratio between bin 1 and bin 0 that is observed each tag (`lr.txt`). Finally, `errfromtags` computes the standard deviation in the log ratio value for each unique sequence. (E) The Sort-Seq Tools commands used to perform this analysis.

The quantification of enrichment ratios for entire sequences is also common. This is especially true in the analysis of massively parallel reporter assays (MPRAs), in which the primary data contains lists of enhancer-specific tags, not the enhancers themselves. Sort-Seq Tools provides simple methods for analyzing such data. Fig. 7 illustrates these methods on the MPRA data of [8]. We note that Sort-Seq Tools does not require, for this analysis, that all assayed sequences be identical in length; identical length sequences are required *only* when computing profiles (e.g. information footprints) or when fitting quantitative models (discussed below).

The input to this analysis pipeline is as follows. The raw sequence data is in two files, `library.txt` and `expression.txt`, which contain a list of sequence tags and associated counts for both the library and expression data sets (Fig. 7A). A separate file, `files.txt`, lists these files and their associated bin numbers (Fig. 7B). Finally a tag key file, `key.txt`, associates each tag to a single enhancer sequence (Fig. 7C). Note that each enhancer sequence will typically be associated with multiple tags; this allows for independent measurements of each enhancer within the same experiment, and also ameliorates possible tag-specific artifacts.

The Sort-Seq Tools function `gatherseqs` is used to collate counts for each tag across the different bins. Passing the file `key.txt` to this function via the `--tagkey` flag allows `gatherseqs` to include in this collated counts file the specific enhancer sequences associated with each tag. Next, the `logratios` command transforms this list of counts into a list of the log of the ratio of counts in bin 1 versus bin 0. Finally, the command `errfromtags` computes, for each unique enhancer sequence, the standard deviation in these log ratio values (specified by the `--var` flag) across all of the associated tags. The result is a list of the log ratio enrichment value and associated error for each assayed enhancer sequence.

Quantitative modeling

Figure 8 Quantitative modeling. Shown is a pipeline for learning matrix models of transcription factor specificity from the Sort-Seq of [6]. (A) Starting from the collated data set `data.txt` from Fig. 5C, the command `learn_matrix` uses least squares regression to fit a matrix model to data. Models that span subsequences are specified by the `--start` and `--end` flags; the interval specified here encompasses the CRP binding site. By default `draw` yields a heat map representation of the resulting model; a sequence logo representation is generated using the `--logo` flag. (B) The Sort-Seq Tools commands used to perform this analysis.

Moving beyond basic sequence analysis, data from massively parallel experiment presents exciting opportunities for learning precise quantitative models of sequence-function relationships. One particularly pressing problem in molecular biology is understanding the sequence specificity of transcription factors for their DNA binding sites. Sort-Seq Tools provides streamlined methods for learning such models from massively parallel experiments. Fig. 8 illustrates the inference of such "matrix" models of transcription factor specificity using the Sort-Seq data of [6].

Using the collated data set `data.txt` from Fig. 5, we need only run the command `learn_matrix` to infer a matrix model. The flags `--start` and `--end` specify which (inclusive) positions this sequence the matrix will span. The output is a matrix model, `crp_model.txt`, which contains a list of values contributing to this score,

one value for each possible base at each position.[3] By default, the `draw` command displays such matrix models in the form of a heat map. As in Fig. 3, adding the `--logo` tag will cause Sort-Seq Tools to instead render a sequence logo interpretation of this model. The model shown in Fig. 8A illustrates the sequence specificity of the CRP transcription factor.

It should be emphasized that there are multiple methods for learning matrix models from massively parallel data. If data consists of only two sequence bins, one bin containing library sequences and the other containing selected sequences, then the simplest method for learning matrix models is to use the approach of Berg and von Hippel [18, 19] (see also [20]). This inference approach is triggered by use of the `--bvh` flag.

Alternatively, models can be fit to data using least squares regression, as was done in [8]. This requires more computation, but can still be performed rapidly. Unlike the Berg and von Hippel approach, linear regression inference naturally accommodates the use of more than two bins. This approach is triggered by use of the `--leastsq` flag (it is also the default behavior).

The least biased method of model inference is mutual information maximization [21, 22]. This approach, which was used to fit models to Sort-Seq data in [6], can be invoked by using the `--mi` flag. Unlike least squares regression, mutual information maximization does not implicitly assume a specific model for experimental noise. This causes the resulting models to be less biased. However, maximizing mutual information requires a Monte Carlo optimization procedure that takes substantially longer to compute than does least squares regression.

Figure 9 Assessing model performance. (A) `predictiveinfo` computes the mutual information (in bits) between the sequence-dependent predictions of a model (`model.txt`) and the bin that each sequence in a specified data set (`data.txt`) is assigned to. (B) Command line instantiation of the pipeline in (A). To perform a cross-comparison of multiple models on multiple datasets, one must specify (C) a list of collated data set files (`datasets.txt`) and (D) a list of models (`models.txt`). (E) The command `compare_predictiveinfo` computes the predictive information of each specified model on each specified dataset. (F) The `draw` command plots, for each data set, these predictive information of each model as a fraction of the predictive information of the best performing model. The cross-comparison shown reproduces results from Fig. S7 of [6].

The ultimate test of model performance, of course, is the ability of a model to predict data that it was not trained on. Sort-Seq Tools provides commands for rapidly assessing model performance in this way. Fig. 9A illustrates the command `predictive info`, which computes the mutual information between the predictions of a model (supplied as standard input) on any dataset of choice (supplied through the flag `--dataset`). The output is a two-line text file reporting the mutual information between model predictions and and measurements in the dataset provided.

To cross-compare multiple models on multiple datasets, Sort-Seq Tools provides the `compare_predictive_information` function Fig. 9E. To use this, one specifies

[3] The score assigned to each binding site is computed as a sum of the individual contributions from each base at each position. Higher scores are interpreted as indicating higher affinity sites, although it should be emphasized that the exact interpretation will depend on what experiment was performed. These scores are defined only up to an arbitrary multiplicative scale.

a list of data files (each file containing a collated data set; Fig. 9C) and a separate list of model files (Fig. 9D).

The output from this cross-comparison, as illustrated by the `draw` command, is shown in Fig. 9F. In particular, this figure illustrates that the model of CRP binding specificity learned from the "full-0" dataset of [6] performs much worse on other data sets from this reference than do the CRP models learned from the other data sets. Indeed, this is exactly what is expected based on the biology of the system: the "full-0" experiment of [6] was performed in the absence of cAMP, a small molecule required by CRP for DNA binding. In the "full-0" experiment there was no active CRP in the cell, and thus the "full-0" dataset provides no real information about CRP specificity.

Simulation

Figure 10 Data simulation. (A) Illustration of and (B) pipeline for a simulated Sort-Seq experiment. Given a wild type sequence, `simulate_library` simulates a sequence library (`library.txt`). `simulate_sort` then partitions sequences from this library into bins (using three bins by default). Specifically, a matrix model `matrix.txt` is used to assign scores to each sequence, Gaussian random noise is added to this score, and thresholding is used to assign each sequence to a bin. (C) The Sort-Seq Tools commands used to perform this analysis.

Sort-Seq Tools also includes methods for simulating Sort-Seq, MPRA, and DMS experiments. Such simulations are useful for testing the capabilities of analysis pipelines. For instance, This information, in turn, can be used to inform experimental design.

This simulation capability is illustrated in Fig. 10A,B. Starting from a wild type sequence, the `simulate_library` command is used to create a library of sequences (`library.txt`). The method `simulate_sort` is then used to simulate a Sort-Seq experiment using an exiting model stored in `model.txt` and illustrated in Fig. 10A. The `simulate_sort` command has a variety of options and flags that allow the user to adjust the number of bins, the noise in the simulated fluorescence measurements, etc.. The output of this command is a collated data set, `simdata.txt`, that can be used for the same types of downstream analyses as experimental data. Other methods for simulating MPRA experiments and DMS experiments are also provided.

Discussion

Here we have described Sort-Seq Tools, a software package for the analysis of massively parallel experiments on mutagenized sequences. This package is designed for the analysis of data from a variety of experiments, including Sort-Seq experiments on promoters, massively parallel reporter assays on enhancers, and deep mutational scanning experiments on proteins. Routines are provided for data simulation, data preprocessing, library quality control, sequence analysis, quantitative model inference, and visualization. This functionality can be accessed either at the commandline, or within the Python language via the `sortseq` module.

Sort-Seq Tools is constructed from a set of individual commands that transform tables stored as columnar text files. This open, human-readable format allows intermediate results from any stage of an analysis pipeline to be used as input to

custom analysis routines. It also provides simple visualization commands which can be used to quickly illustrate the contents of many of these tables. The modeling capabilities of Sort-Seq Tools have been designed to be simple and light-weight.

The array of massively parallel assays is rapidly expanding, and the data that these experiments produce suggests a variety of analysis tasks beyond those included in the current version package. We plan to continue improving this package. In particular, we expect to and invite the quantitative biology community to join in this effort.

Competing interests
The authors declare that they have no competing interests.

Author's contributions
WI, RP, and JBK designed the research. WI and JBK wrote the software. WI and JBK wrote the paper. . . .

Acknowledgements
We thank Douglas Fowler for providing preprocessed DMS data from [15]. We also thank Tarjei Mikkelsen for posting the preprocessed MPRA data from [8] on NCBI GEO. The work of JBK was supported by the Simons Center for Quantitative Biology at Cold Spring Harbor Laboratory. WI was supported by XXX. RP is supported by XXX.

Author details
[1]Department of Physics, California Institute of Technology, 91125, Pasadena, CA. [2]Department of Applied Physics, California Institute of Technology, 91125, Pasadena, CA. [3]Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory, 11375, Cold Spring Harbor, NY.

References

1. Shendure, J., Lieberman Aiden, E.: The expanding scope of DNA sequencing. Nature Biotechnology **30**(11), 1084–1094 (2012)
2. Haberle, V., Lenhard, B.: Dissecting genomic regulatory elements in vivo. Nature Biotechnology **30**(6), 504–506 (2012)
3. White, M.A.: Understanding how cis-regulatory function is encoded in DNA sequence using massively parallel reporter assays and designed sequences. Genomics **106**(3), 165–170 (2015)
4. Inoue, F., Ahituv, N.: Decoding enhancers using massively parallel reporter assays. Genomics **106**(3), 159–164 (2015)
5. Fowler, D.M., Fields, S.: Deep mutational scanning: a new style of protein science. Nature Methods **11**(8), 801–807 (2014)
6. Kinney, J.B., Murugan, A., Callan, C.G., Cox, E.C.: Using deep sequencing to characterize the biophysical mechanism of a transcriptional regulatory sequence. Proc. Natl. Acad. Sci. USA **107**(20), 9158–9163 (2010)
7. Sharon, E., Kalma, Y., Sharp, A., Raveh-Sadka, T., Levo, M., Zeevi, D., Keren, L., Yakhini, Z., Weinberger, A., Segal, E.: Inferring gene regulatory logic from high-throughput measurements of thousands of systematically designed promoters. Nature Biotechnology **30**(6), 521–530 (2012)
8. Melnikov, A., Murugan, A., Zhang, X., Tesileanu, T., Wang, L., Rogov, P., Feizi, S., Gnirke, A., Callan, C.G., Kinney, J.B., Kellis, M., Lander, E.S., Mikkelsen, T.S.: Systematic dissection and optimization of inducible enhancers in human cells using a massively parallel reporter assay. Nature Biotechnology **30**(3), 271–277 (2012)
9. Patwardhan, R.P., Hiatt, J.B., Witten, D.M., Kim, M.J., Smith, R.P., May, D., Lee, C., Andrie, J.M., Lee, S.-I., Cooper, G.M., Ahituv, N., Pennacchio, L.A., Shendure, J.: Massively parallel functional dissection of mammalian enhancers in vivo. Nature Biotechnology **30**(3), 265–270 (2012)
10. Kwasnieski, J.C., Mogno, I., Myers, C.A., Corbo, J.C., Cohen, B.A.: Complex effects of nucleotide variants in a mammalian cis-regulatory element. Proc Natl Acad Sci U S A **109**(47), 19498–19503 (2012)
11. Patwardhan, R.P., Lee, C., Litvin, O., Young, D.L., Pe'er, D., Shendure, J.: High-resolution analysis of DNA regulatory elements by synthetic saturation mutagenesis. Nature Biotechnology **27**(12), 1173–1175 (2009)
12. Fowler, D.M., Araya, C.L., Gerard, W., Fields, S.: Enrich: software for analysis of protein function by enrichment and depletion of variants. Bioinformatics (Oxford, England) **27**(24), 3430–3431 (2011)
13. Razo-Mejia, M., Boedicker, J.Q., Jones, D., DeLuna, A., Kinney, J.B., Phillips, R.: Comparison of the theoretical and real-world evolutionary potential of a genetic circuit. Physical biology **11**(2), 026005 (2014)
14. Bloom, J.D.: Software for the analysis and visualization of deep mutational scanning data. BMC Bioinformatics **16**, 168 (2015)
15. Fowler, D.M., Araya, C.L., Fleishman, S.J., Kellogg, E.H., Stephany, J.J., Baker, D., Fields, S.: High-resolution mapping of protein sequence-function relationships. Nature Methods **7**(9), 741–746 (2010)
16. Schneider, T.D., Stephens, R.M.: Sequence logos: a new way to display consensus sequences. Nucleic acids research **18**(20), 6097–6100 (1990)
17. Crooks, G.E., Hon, G., Chandonia, J.-M., Brenner, S.E.: WebLogo: a sequence logo generator. Genome Research **14**(6), 1188–1190 (2004)
18. Berg, O., von Hippel, P.: Selection of DNA binding sites by regulatory proteins. Statistical-mechanical theory and application to operators and promoters. Journal of Molecular Biology **193**(4), 723–750 (1987)
19. Berg, O., von Hippel, P.: Selection of DNA binding sites by regulatory proteins. II. The binding specificity of cyclic AMP receptor protein to recognition sites. Journal of Molecular Biology **200**(4), 709–723 (1988)

20. Stormo, G.D.: DNA binding sites: representation and discovery. Bioinformatics (Oxford, England) **16**(1), 16–23 (2000)
21. Kinney, J.B., Atwal, G.S.: Parametric inference in the large data limit using maximally informative models. Neural Comput. **26**(4), 637–653 (2014)
22. Atwal, G.S., Kinney, J.B.: Learning quantitative sequence-function relationships from massively parallel experiments. arXiv:1212.3647 [q-bio.QM] (2015). 1506.00054v2

Additional Files

Additional file 1 — Sample additional file title
Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure.

Additional file 2 — Sample additional file title
Additional file descriptions text.

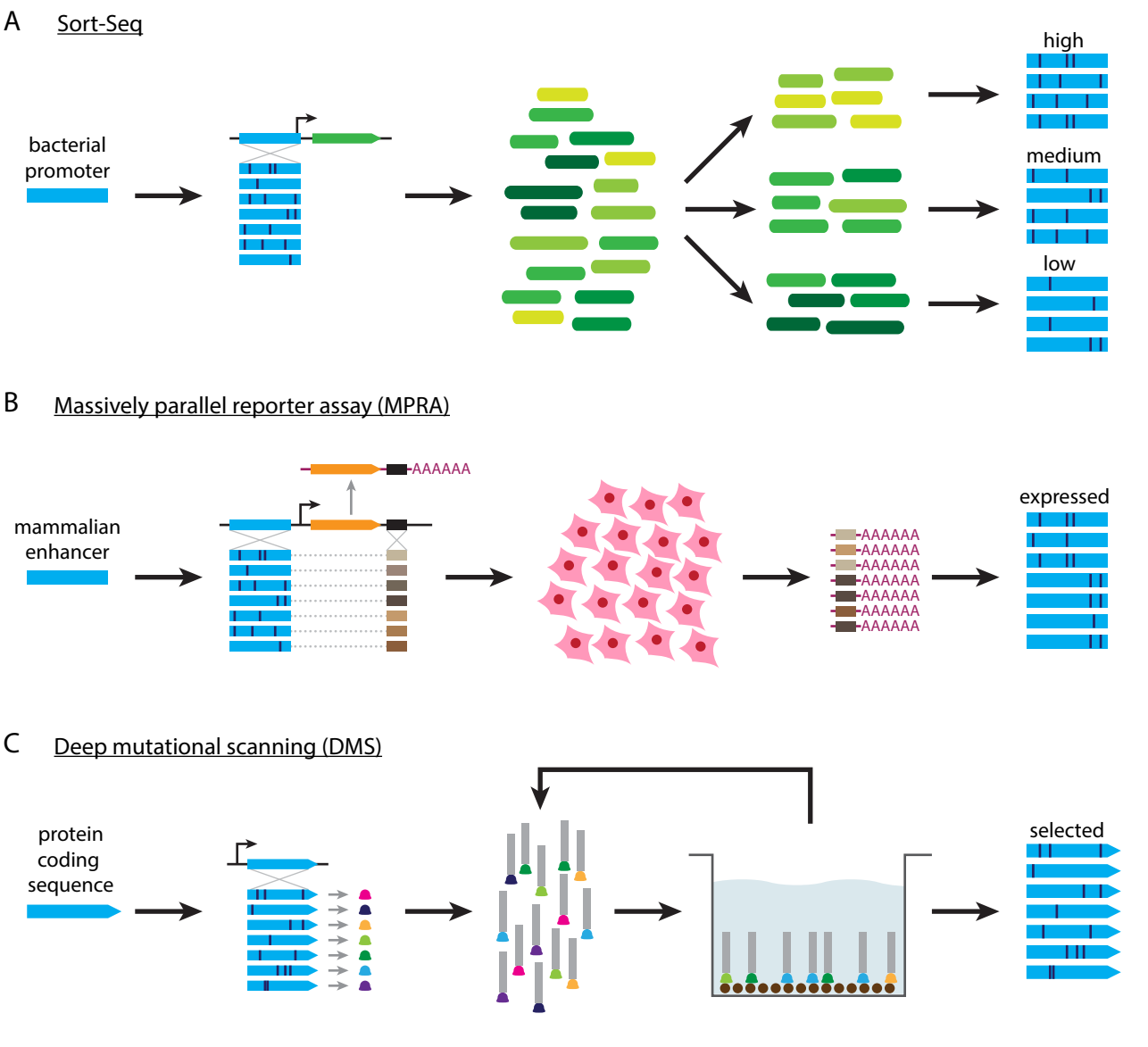
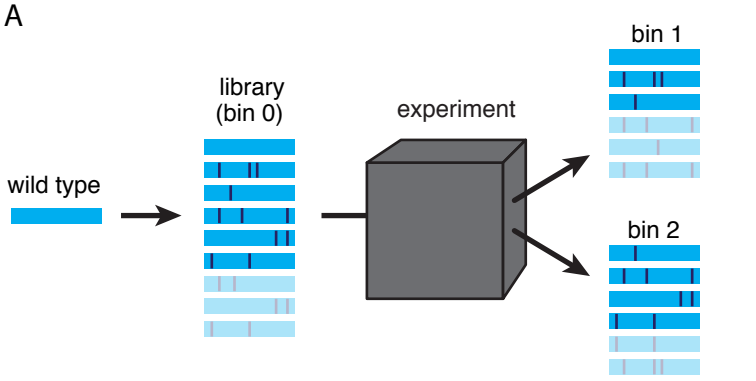

A
Sort-Seq
bacterial
promoter
high
medium
low
B
Massively parallel reporter assay (MPRA)
mammalian
enhancer
AAAAAA
expressed
C
Deep mutational scanning (DMS)
protein
coding
sequence
selected

Figure 1


A
wild type
library
(bin 0)
experiment
bin 1
bin 2

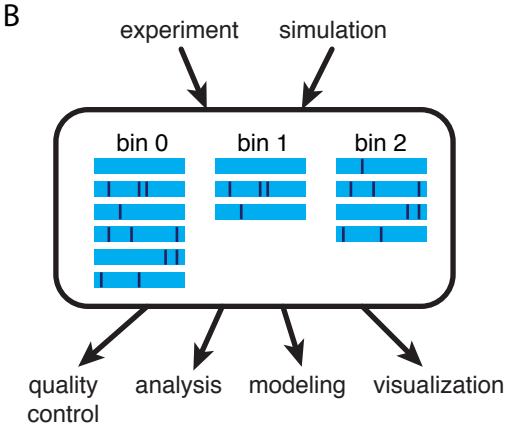

B
experiment
simulation
bin 0
bin 1
bin 2
quality
control
analysis
modeling
visualization

Figure 2

A

sites.txt

seq
GCATTAAAAA
CCGTAAAGAG
TCGTAAAAAC
GCACAAAAAT
TCATAAAAAC
CACTAAAAAT
GAATTAAAAA
GCCAAAAAAA
CAATAAAAAG

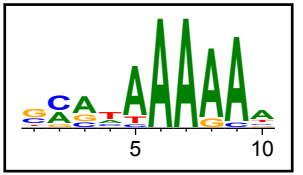
profile_freqs

freqs.txt

pos	freq_A	freq_C	freq_G	freq_T
0	0.0625	0.3125	0.5	0.125
1	0.375	0.5	0.125	0.0
2	0.5625	0.1875	0.25	0.0
3	0.25	0.1875	0.0625	0.5
4	0.8125	0.0625	0.0	0.125
5	1.0	0.0	0.0	0.0
6	1.0	0.0	0.0	0.0
7	0.875	0.0	0.125	0.0
8	0.9375	0.0625	0.0	0.0
9	0.5625	0.125	0.125	0.1875

draw --logo

logo.pdf



B

```
$ sortseq profile_freqs -i sites.txt -o freqs.txt
$ sortseq draw -i freqs.txt -o logo.pdf --logo
```

C

```
$ echo sites.txt | sortseq profile_freqs| sortseq draw --logo > logo.pdf
```

D

```
import sortseq
data = sortseq.load("data.txt")
freqs = sortseq.profile_freqs(data)
fig = sortseq.draw(freqs,"logo")
fig.save("logo.pdf")
```

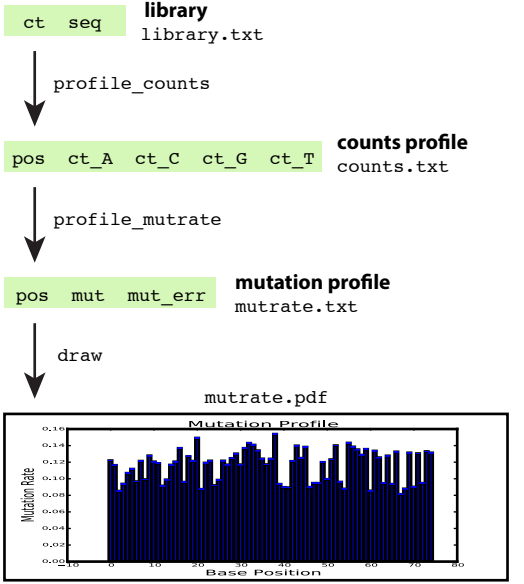
Figure 3

A

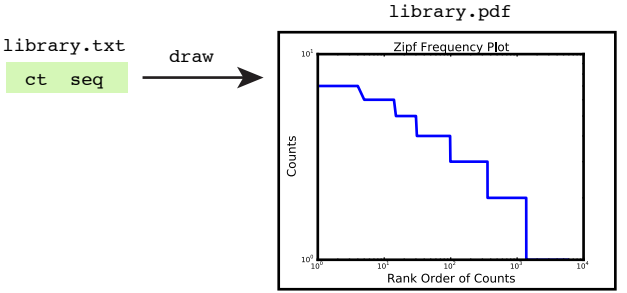
library.txt

ct	seq
8	AATTACAGTGAGTTAGCTTACTCATTCGGCACC...
7	CATTACGGTGACATAGCTCACTCATTATGCACC...
7	AATTAATGTGAGTTGGTTCACTCATTAGGGACC...
7	GATTAATGTGAGTTAACTCACTCATTAGGCACC...
7	AATTACTGTCTGTTAGCTCACTCATTACTCAGC...
6	AATTAATGTGAGTTAGTTCACTCATGAGTCACT...
6	CATTAATGTGAGTTAGCTCACTCATTAGGCACC...
6	TATTAATGTAAGTTAGCTCACACATAACGCACC...
6	AATTAATGTGAATGAGCTCACTCATAAGGCACA...

B



C



D

```
$ sortseq profile_counts -i library.txt -o counts.txt
$ sortseq profile_mutrate -i counts.txt -o mutrate.txt
$ sortseq draw -i library.txt -o library.pdf
$ sortseq draw -i mutrate.txt -o mutrate.pdf
```

Figure 4

A

ct	seq	library.txt
ct	seq	bin_1.txt
ct	seq	bin_2.txt
⋮		⋮
ct	seq	bin_9.txt

B

files.txt

bin	file
0	library.txt
1	bin_1.txt
2	bin_2.txt
3	bin_3.txt
4	bin_4.txt
5	bin_5.txt
6	bin_6.txt
7	bin_7.txt
8	bin_8.txt
9	bin_9.txt

C

bin file — **file list** files.txt

↓ gatherseqs

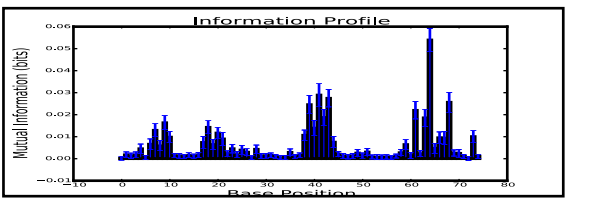
ct_0 ct_1 ... ct_9 seq — **dataset** data.txt

↓ profile_info

pos info info_err — **information profile** info.txt

↓ draw

info.pdf



D

E

```
$ sortseq gatherseqs -i files.txt -o data.txt
$ sortseq profile_info -i data.txt -o info.txt
$ sortseq draw -i info.txt -o info.pdf
```

Figure 5

A

```
ct_0  ct_1  ct_2  seq
```

DNA sequence data
`dnadata.txt`

```
convertseqs --protein
```

```
ct_0  ct_1  ct_2  seq_pro
```

protein sequence data
`proteindata.txt`

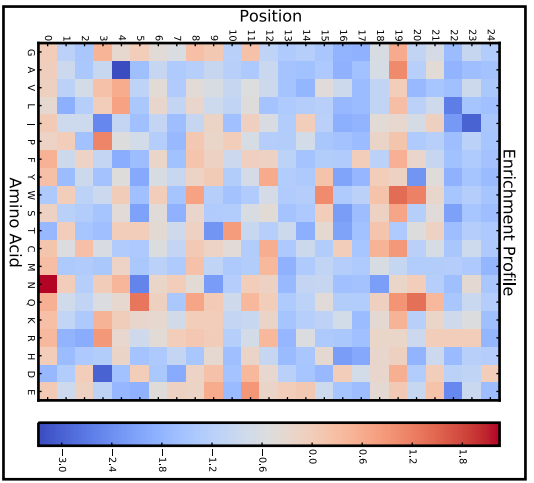
```
profile_enrichment --bin 2 --lib 0
```

```
pos  wt  le_A  le_C  le_D  ...  le_Y
```

enrichment profile
`enrichment.txt`

```
draw
```

`enrichment.pdf`



B

```
$ sortseq convertseqs --protein \
    -i dnadata.txt -o proteindata.txt
$ sortseq profile_enrichment --bin 2 --lib 0  \
    -i proteindata.txt -o enrichment.txt
$ sortseq draw -i enrichment.txt -o enrichment.pdf
```

Figure 6

A

ct	tag	library.txt
ct	tag	expression.txt

B

files.txt

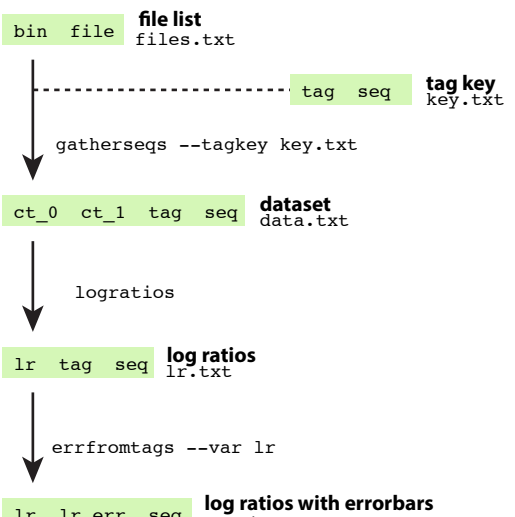
bin	file
0	library.txt
1	expression.txt

C

key.txt

tag	seq
ATCTGCGGCC	GCACCAGACAGTGACGTCAGCTGC...
TACTGCGAAC	GCACCAGACAGTGACGTCAGCTGC...
GGCTTGCACA	GCACCAGACAGTGACGTCAGCTGC...
TAAAACGATA	GCACCAGACAGTGACGTCAGCTGC...
ATGGCTAAAA	GCACCAGACAGTGACGTCAGCTGC...
GACTAAACGC	GCACCAGACAGTGACGTCAGCTGC...
AAGATGGCTG	GCACCAGACAGTGACGTCAGCTGC...
CACTATAGTA	GCACCAGACAGTGACGTCAGCTGC...
ACCGATCGCG	GCACCAGACAGTGACGTCAGCTGC...

D



E

```
$ sortseq gatherseqs -i files -o dataset.txt \
    --tagkey key.txt
$ sortseq logratios -i dataset.txt -o lr.txt
$ sortseq errfromtags -i lr.txt -o lr_with_errs.txt
```

Figure 7

A

```
ct_0  ct_1  ...  ct_9  seq
```

dataset
data.txt

```
learn_matrix --start 3 --end 25
```

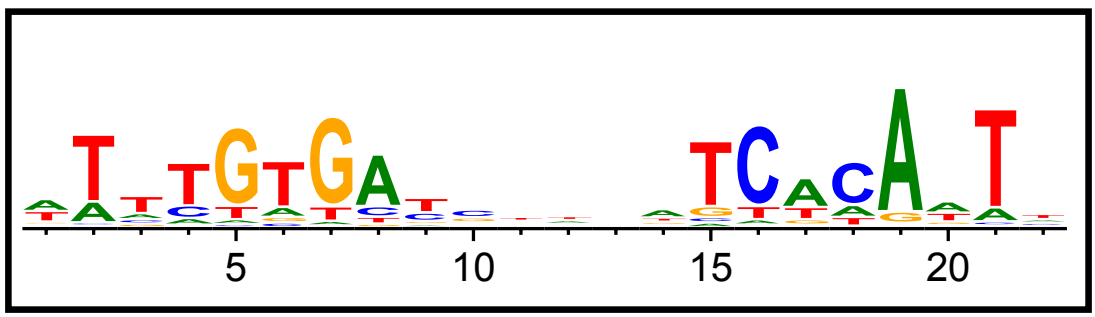
```
pos  val_A  val_C  val_G  val_T
```

matrix model
crp_model.txt

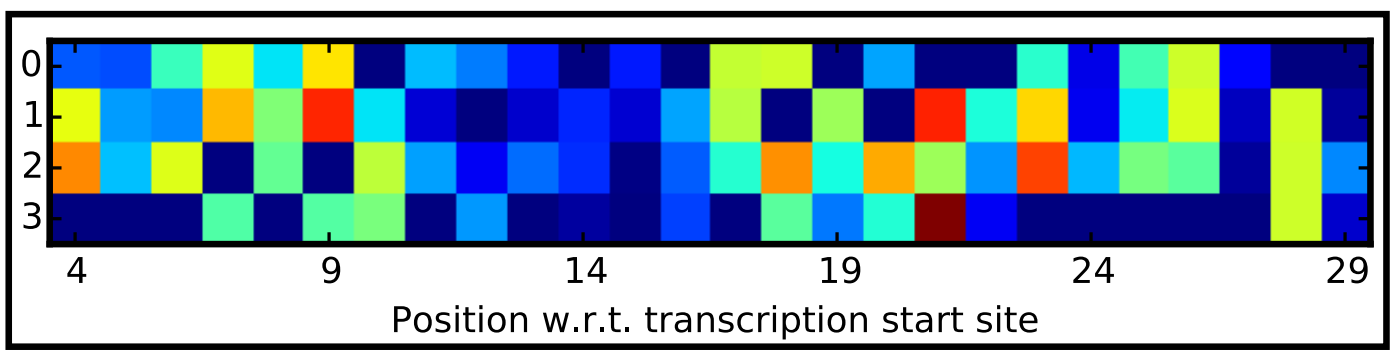
draw

draw --logo

crp_logo.pdf



crp_model.pdf



B

```
$ sortseq learn_model --start 3 --end 25 \
    -i data.txt -o crp_model.txt
$ sortseq draw -i crp_model.txt -o crp_matrix.pdf
$ sortseq draw --logo -i crp_model.txt -o crp_logo.pdf
```

Figure 8

A

pos	val_A	val_C	val_G	val_T

model
model.txt

ct_0	ct_1	...	ct_9	seq

dataset
data.txt

```
predictiveinfo --dataset data.txt
```

info
1.352

predictive information
info.txt

B

```
$ sortseq predictiveinfo --dataset data.txt \
    -i model.txt -o info.txt
```

C

datasets.txt

dataset	file
full-wt	full-wt_data.txt
crp-wt	crp-wt_data.txt
full-0	full-0_data.txt
full-150	full-150_data.txt
full-500	full-500_data.txt

D

models.txt

model	file
full-wt	full-wt_crp_model.txt
crp-wt	crp-wt_crp_model.txt
full-0	full-0_crp_model.txt
full-150	full-150_crp_model.txt
full-500	full-500_crp_model.txt

E

```
$ sortseq compare_predictive_information \
    --datasets datasets.txt \
    --models models.txt -s 3 -e 25 \
    -o comparison.txt
$ sortseq draw -i comparison.txt -o comparison.pdf
```

F

comparison.pdf

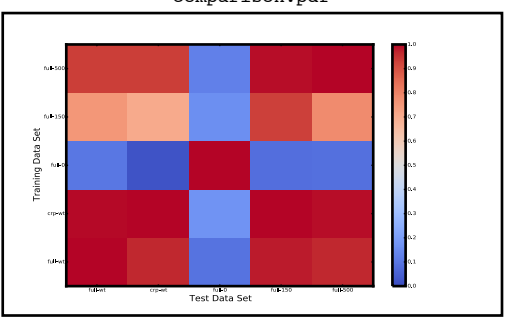
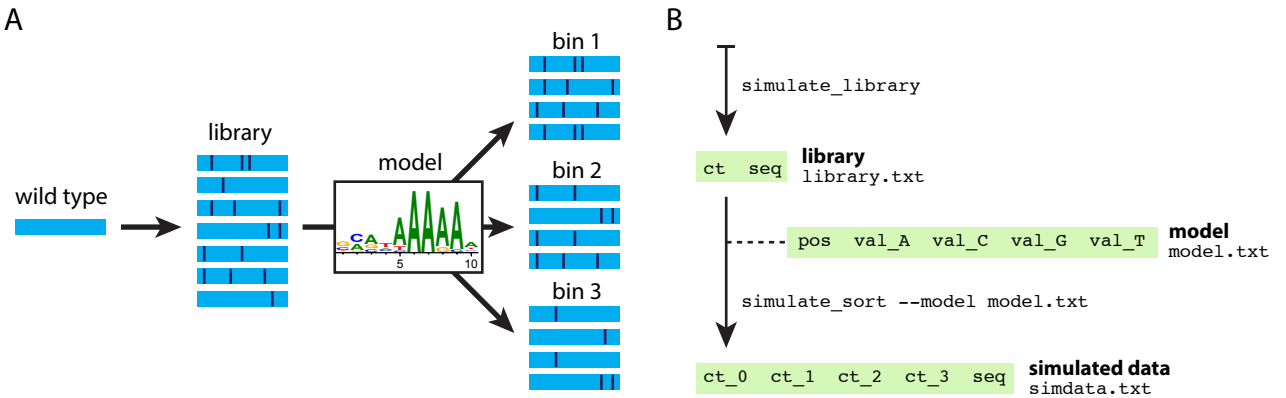


Figure 9



C

```
$ sortseq simulate_library --wtseq ACCACATTGAGAGATTTAGGGATA --mutrate 0.10 -o library.txt
$ sortseq simulate_sort -i library.txt -o simdata.txt --model true.mat
```

Figure 10