Chapter 14

Operating on Genomic Ranges Using BEDOPS

Shane Neph, Alex P. Reynolds, M. Scott Kuehn, and John A. Stamatoyannopoulos

Abstract

The bulk of modern genomics research includes, in part, analyses of large data sets, such as those derived from high resolution, high-throughput experiments, that make computations challenging. The BEDOPS toolkit offers a broad spectrum of fundamental analysis capabilities to query, operate on, and compare quantitatively genomic data sets of any size and number. The toolkit facilitates the construction of complex analysis pipelines that remain efficient in both memory and time by chaining together combinations of its complementary components. The principal utilities accept raw or compressed data in a flexible format, and they provide built-in features to expedite parallel computations.

Key words Bioinformatics, Sequencing, Algorithm, Overlap, Genomics, Compression, Parallelization

1 Introduction

Investigators seek to answer a range of genomics questions by analyzing experimental data. With modern, highly multiplexed sequencing technologies, the volume of data produced through a single experiment is unprecedented and continues to grow quickly [1]. Massive data sets generated from sequencers are commonplace, with publicly available, whole-genome experiments in human alone numbering in the thousands [2]. Analyzing these and similarly large data sets can stress computational resources to the point of failure. By imposing structure on input data, the BEDOPS analysis toolkit addresses this concern directly for many common analysis tasks [3].

One fundamental analysis is drawing associations between feature sets. As a simple example, a researcher might work with high-throughput sequencing reads following a ChIP-seq experiment for the CTCF transcription factor (TF). Significantly populated regions make up an important set of features for the experiment, as it shows where CTCF is mostly likely bound in that cell or tissue type under the experimental conditions. Annotating overlapping or

Ewy Mathé and Sean Davis (eds.), *Statistical Genomics: Methods and Protocols*, Methods in Molecular Biology, vol. 1418, DOI 10.1007/978-1-4939-3578-9_14,

nearby genomic features, such as genes or the binding sites for a different TF, can provide insight into CTCF activity.

Among other capabilities, BEDOPS offers a variety of methods to draw associations between data sets. The *bedops* tool applies overlap and difference operations to include or exclude genomic elements from one set, based on their full or partial overlaps with regions in another set. Other associations can be drawn with *closest-features*, which finds those features from a secondary source that fall nearest to the regions of interest. Statistical summaries with *bedmap* can involve various calculations on associated features, such as the mean signal value from a second data source over the regions of interest.

The toolkit includes utilities to convert efficiently common genomic data formats, such as BAM, GFF, and WIG, to the BED format required by BEDOPS. These tools make it easy to integrate data from a variety of sources into one framework for analysis. A data management utility archives BED in a deeply compressed form that offers descriptive metadata and random access to data by chromosome, which reduces disk space overhead and makes it simple to parallelize analysis pipelines.

2 Methods

2.1 Genomic Data Formats

The BEDOPS toolkit works with data formatted as Browser Extensible Data (BED), which is a text-based, tab-delimited format used for storing position and annotation information along the genome. The accepted format relaxes some of the constraints found in the original BED specification, which was developed for use with the UCSC Genome Browser [4]. In contrast to the original UCSC BED, the extended format accepts general measurements and not only integer values, allows 0- or 1-based coordinate indexing, and places no constraint on the type or number of columnar data that may exist beyond a small default minimum of 3–5 columns. These modest extensions to BED offer broad versatility for analyses and data format conversions.

A bewildering array of open and commercial representations of genomic data exist, yet data found in many other popular formats can be recast as extended BED without loss of information. In order to enable BEDOPS to work with data stored in other formats, we include the *convert2bed* tool, along with a convenience wrapper script per input format, to transform the data into extended BED.

The binary BAM format and its text-based analogue, SAM, are often used to store low-level sequence alignment information [5]. The GTF and GFF file types store gene annotations. The open Variant Call Format (VCF) handles variant, deletion, and insertion data with associated genotype information. The PSL format from

UCSC stores positional alignment results. The UCSC WIG file format, which itself includes more than one variation, contains genomic positions with continuous signal data. The information in these and other data formats can be converted in their entirety into extended BED for use with BEDOPS.

Various data formats call for different interpretations of genomic coordinates, which often leads to confusion when integrating data from a variety of sources. Some formats use 0-based indexing while others use 1-based coordinate indexing, where the 5′-most base (forward strand) of a chromosome is labeled, respectively, as position 0 or 1. Positions along a chromosome are labeled with increasing integer values starting from the label of that first base. Even closely related formats, such as BAM (0-based) and SAM (1-based), often differ in their coordinate indexing specifications. The UCSC Genome Browser tool requires 1-based BED inputs for proper data visualization, while the same suite explicitly defines BED as a 0-based coordinate format. Adding further confusion, popular data conversion utilities that are not part of the BEDOPS toolkit inconsistently handle indexing shifts between various formats.

As a small extension to the original BED format definition, BEDOPS does not check or specify coordinate indexing. In short, BEDOPS fully supports 0-based and 1-based coordinate indexing simply through non-interpretation, as all underlying algorithms are unaffected by that choice. If inputs have a 0-based index, for example, then any output results will also have a 0-based index. However, coordinate indexing can be changed explicitly with the *bedops* utility through its *--range* option. The next program call example shows how to use *bedops* to produce a 0-based output from a 1-based input. The subsequent call shows how to achieve a coordinate shift in the opposite direction. In each case, the two signed integers, separated by a colon and part of the *--range* argument, are added to the start and end coordinate positions, respectively, of each row of input.

```
$ bedops --everything --range -1:-1 one-based-input.bed \
  > zero-based-output.bed
$ bedops --everything --range 1:1 zero-based-input.bed \
  > one-based-output.bed
```

There is yet another interpretation related to genomic coordinates that also differs between various genomic formats. The location of a genomic feature typically includes chromosome information, as well as the start and end coordinate positions of that feature on the chromosome. Are the start and end positions inclusive to this interval? For example, given a genomic feature on chromosome 2 with start position 101 and end position 120, should this be interpreted as an interval that includes base positions 101 and 120, or perhaps only the base positions in between these

two end points? The UCSC BED format specifies a half-open interval interpretation, where the start coordinate is part of the interval and the end coordinate is 1 nucleotide beyond the interval range, concisely denoted as *[start, end)*. In this example, the BED specification dictates that base positions 101 through 119 are part of the interval, inclusively, while base position 120 is not. The BEDOPS toolkit honors this interpretation. This meaning simplifies interval length calculations, which can be derived by subtracting the half-open coordinates. In the example, the number of base positions in the interval is 19, which is also the difference of 120 and 101.

Different genomic formats use different indexing schemes and interval interpretations. One must ensure that all data sources used together as inputs to BEDOPS are created using consistent coordinate interpretations. To assist, the *convert2bed* tool transforms a number of incongruent genomic formats to BED using a 0-based index with an appropriate half-open range. Also, the *bedops* utility can help by moving data between 0-based and 1-based indexing as needed.

2.2 Pipelining with Streams

BEDOPS adopts common Unix design principles [6] in segregating the functionality of individual tools and using standard Unix data streams to handle input and output between tools. A single process works on input data and prints output and error messages to data streams (Fig. 1). Multiple processes can be glued or chained together with the Unix "pipe" operator to construct larger and more complex analyses (Fig. 2). In a Unix pipeline, chunks of input data are operated upon in one process, and generated results are progressively fed to the next process, and so on to any number of processes, until there is no more input data to consume. In this fashion, Unix pipe operators allow a person to create an expressive and self-documenting unit of code.

The BEDOPS utilities work with and produce results in sorted BED format. Results can be directed to a file or sent directly to the next application through a piped data stream. Since pipes do not involve disk accesses, streaming data between utilities offers performance benefits over constructing and maintaining temporary, intermediate files. Further, on modern systems with multiple

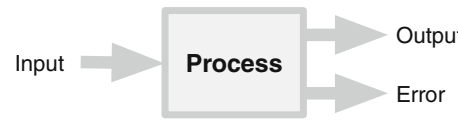

Fig. 1 A single, generic Unix process with three standard data streams: input ("*stdin*"), output ("*stdout*") and error ("*stderr*"). The process receives data on *stdin*, printing its normal output to *stdout* and any error and log messages to *stderr*

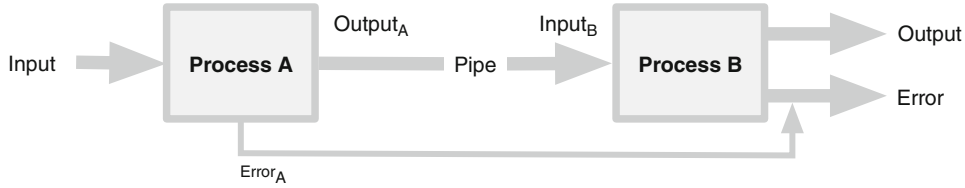

Fig. 2 Two generic Unix processes connected with a Unix pipe. *Process A* receives data on *stdin*, passes its output data stream to the input stream of *Process B*, which in turn prints normal output to the final *stdout* stream. Error and log messages from *Processes A* and *B* are both printed to *stderr*

Fig. 3 Example of a single Unix process, passing forward-stranded transcription start sites (TSSs) to the standard input stream (*stdin*) of *bedops*. In turn, *bedops* makes 1 kb upstream windows of each TSS and prints them to its standard output stream (*stdout*). Any error messages are printed to the standard error stream (*stderr*)

processors, tools linked via pipes can run simultaneously to improve overall throughput.

As an example, the *bedops* program can produce a filtered subset of some genomic input data which is then "piped" downstream to any of a number of BEDOPS or Unix tools. The refined subset might be piped into *bedmap* to investigate statistical relationships with another data set, or to *closest-features* to find the nearest gene annotations. Those extended results could further be piped on to the *starch* utility to be compressed and archived for later retrieval, or to the Unix *tee* utility before finding the genomic complement with yet another call to *bedops*.

To give a more concrete example of how one might progressively build an analysis set, we start by creating a 1 kilobase window upstream of a subset of forward-stranded transcription start sites (TSSs) found in *TSSs.bed*, writing this to a file called *1kb_windows_upstream_of_forward_stranded_TSSs.bed* (Fig. 3).

```
$ awk '$6 == "+"' TSSs.bed \
  | bedops --range 1000:0 --everything - \
  >1kb_windows_upstream_of_forward_stranded_TSSs.bed
```

The *awk* statement, a standard tool found on popular Unix-like systems, shows that strand information for each row is expected to be in the 6th column for this example (though BEDOPS has no such restriction). The pipe operator ('|') sets up a Unix pipe in between the utilities. A hyphen ('-') used in a *bedops* call tells the program to check its standard input stream for incoming data. Reverse strand TSS upstream padding is similar, and all stranded results can be combined together by calling *bedops* once more.

```
$ awk '$6 == "-"' TSSs.bed \
  | bedops --range 0:1000 --everything - \
  | bedops --everything 1kb_windows_upstream_of_forward_stranded_TSSs.bed - \
  >1kb_upstream_padding_TSSs.bed
```

We are likely less interested in the windowed regions themselves than in their biologically relevant attributes. Perhaps the selected genes found in *TSSs.bed* harbor an enrichment of predicted motif binding sites in their upstream regulatory regions, which may highlight the importance of a particular transcription factor in regulating many of the genes.

Rather than writing all of the windowed regions to disk as shown, we can pipe them directly into yet another operation to map on motif predictions found in a BED-formatted text file called *motif_predictions.bed*, and save that extended result to file. The *motif_predictions.bed* file has at least these four columns of information: chromosome, start coordinate, end coordinate, and the predicted motif model's name.

```
$ awk '$6 == "-"' TSSs.bed \
  | bedops --range 0:1000 --everything - \
  | bedops --everything 1kb_windows_upstream_of_forward_stranded_TSSs.bed - \
  | bedmap --echo --echo-map-id-uniq - motif_predictions.bed \
  > motif_IDs_1kb_upstream_padding_TSSs.bed
```

The use of the hyphen character with *bedmap* tells the program that data are available from its standard input (*stdin*) stream. Here, those data come from the standard output (*stdout*) stream of the upstream *bedops* process (Fig. 4). The *--echo* option for *bedmap* causes the program to repeat the information found in its first input (the *Reference*) to its output. The *--echo-map-id-uniq* option instructs *bedmap* to add another output column that lists all unique motif model identifiers, separated by semicolons, that overlap a windowed region of the *Reference* input.

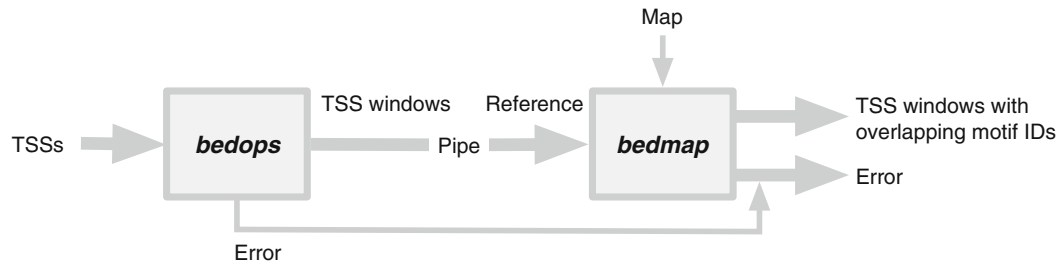

Fig. 4 Two Unix processes running *bedops* and *bedmap*, connected with a Unix pipe. The *bedops* process operates on TSS data received on *stdin* and passes padded TSS windows over its output data stream. These data are received on the input stream of the *bedmap* process, which is configured to treat them as the source of *Reference* set elements. Selected mapping operations are applied and the end result—padded windows with unique motif model names—is written to the final *stdout* stream. Error and log messages from *bedops* and *bedmap* are both printed to the *stderr* stream

Representing standard input with the hyphen character is also the convention in many core Unix utilities, and we can just as easily pipe the standard output from BEDOPS applications to these utilities. For example, we can add calls to the *tr*, *sort*, and *uniq* utilities to build a unique list of motif identifiers that overlap at least one buffered region in our entire TSS data set.

```
$ awk '$6 == "-"' TSSs.bed \
  | bedops --range 0:1000 --everything - \
  | bedops --everything 1kb_windows_upstream_of_forward_stranded_TSSs.bed - \
  | bedmap --echo-map-id-uniq - motif_predictions.bed \
  | tr ';' '\n' \
  | sort - \
  | uniq - \
  > overlapping_motif_IDs_unique_windowed_TSSs.txt
```

Piping data in between BEDOPS and Unix applications is a common and powerful technique to construct complex, yet readable analysis pipelines. With the exception of *unstarch*, all BEDOPS applications can read from a standard input stream, and all write to a standard output stream.

The Unix *bash* shell supports a process substitution feature which makes it syntactically simple to read data streams from more than one upstream process. For instance, if we have BAM-formatted sequencing reads which we want to relate to VCF-formatted variants, say to retrieve SNP names that overlap our sequencing reads, we can use a pair of process substitutions in one BEDOPS command.

```
$ bedmap --echo --echo-map-id-uniq <(bam2bed < reads.bam) <(vcf2bed < variants.vcf) \
  > reads_with_unique_IDs_of_overlapping_SNPs.bed
```

Process substitution in the *bash* shell allows passing the standard output streams of a pair of format conversion tasks directly to *bedmap*, without the wasteful generation and subsequent cleanup of intermediate BED files.

In conjunction with GNU *make*, *rake*, *snakemake*, Luigi, or other pipeline construction toolkits, we can easily abstract an arrangement of piped operations and process substitutions to automate larger and longer-running data analysis tasks (Fig. 5). Operational units ("targets" in GNU *make* terminology) can process data in streams to improve overall performance. They can help with the readability, development, and maintenance of pipelines. One might design long-running tasks to be restartable from an intermediate run point upon interruption, and modifications to downstream pipeline parameters can be tested without regenerating upstream input data.

```
all: all_snps_in_genes.bed

all_snps.bed: snps_1.bed snps_2.bed snps_3.bed
    bedops -m $^ > $@

all_snps_in_genes.bed: all_snps.bed genes.bed
    bedops -e 1 $^ > $@

...
```

Fig. 5 Example snippet of a GNU *make* makefile. Running *make all* attempts to build the file dependency *all_snps_in_genes.bed* from dependencies *all_snps.bed* and *genes.bed*, but the *all_snps.bed* file does not yet exist. The build process therefore looks at the *all_snps.bed* target and creates it from merging files *snps_1.bed*, *snps_2.bed*, and *snps_3.bed* with *bedops*, before creating *all_snps_in_genes.bed*

3 Core Functionality

Relating and operating on genomic intervals is the principal business of BEDOPS. Overlap and proximity relationships among data sets are determined, allowing for general annotations, statistical evaluations, and set-based operations.

3.1 Set-Based Operations

The *bedops* tool offers a host of efficient overlap and set-based operations that can be applied across any number of sorted BED inputs in a single program call. Operations to filter data based upon overlap relationships, generate regions between existing data points, merge intervals across data sets, and many others are all readily available.

Determining the overlaps between genomic data sets is useful for answering research questions, some specific real-world examples of which include:

- Generating SNP variants located within the open chromatin regions as measured with DNaseI hypersensitivity [7].
- Building a list of exemplar regions to represent partially overlapping regions in two or more input data sets [8].
- Filtering out elements that overlap unmappable regions across the genome [9].
- Determining the degree of overlap in peak signal regions between heterogeneous high-throughput experiments [10].
- Building a list of TFs that overlap a set of proximal promoters [11].

With *bedops*, we can define explicitly what it means for two regions to overlap, either through a number of nucleotides specification or as a fraction of a feature's length. For example, in order to use higher-confidence regions in downstream analyses, we can pull out the subset of ChIP-seq peaks in a data set that overlap a replicate experiment's peak regions by 50 % or more.

In addition to determining overlaps, which can be viewed as an element membership operation, *bedops* offers a host of set-like operations for genomic regions. These include the multiset union, element non-membership, complement, symmetric difference, and intersection, among others. All *bedops* operations efficiently handle any number of data inputs at once. For example, one can determine the set of genomic intervals specific to any one of 100 input files using:

```
$ bedops --symmdiff file1.bed file2.bed ... file100.bed \
> intervals_only_from_any_single_file.bed
```

Range adjustments can be made in conjunction with any set-like operations of *bedops*. These coordinate modifications can be symmetric or asymmetric to shrink or expand input regions before applying an operation. The *bedops* utility offers flexibility and very low memory overhead by working with and producing sorted data. More information about sorting is provided in the section on *sort-bed*.

3.2 Proximity

The *closest-features* utility identifies the nearest upstream and downstream elements between two sorted data sources. It can report the signed distances, choose the single nearest element, or exclude overlapping elements in its report. This can help to:

- Build a histogram of the distances between a set of SNPs of interest and their closest DNase-seq peaks in the K562 cell line.
- Identify the nearest predicted motif binding site for a set of NFE2 ChIP-seq peaks in HepG2 cell line, where the distances between features can help to differentiate between direct and indirect binding of the TF.
- Locate the nearest origin of replication to a set of OMIM genes in a disease study.
- Determine the minimum distance between features of any two data sets, optionally ignoring interactions that directly overlap.

As discussed in the previous section on pipelines, the *closest-features* tool can read and write standard input and output. We can pipe the results of calling this tool back into the tool itself to find progressively further elements with a little help from *bedops*. To demonstrate, we show the process for searching for the name of the nearest gene (found in *genes.bed*) to each element in a set of regions of interest (*roi.bed*).

```
$ closest-features --closest --no-ref roi.bed genes.bed \
  | cut -f4 - \
  > closest_gene_name_to_roi.txt
```

We can extend this procedure to find only the next closest gene.

```
$ closest-features --closest --no-ref roi.bed genes.bed \
  | sort-bed - \
  | bedops --not-element-of genes.bed - \
  | closest-features --closest --no-ref roi.bed - \
  | cut -f4 - \
  > second_closest_gene_name_to_roi.txt
```

This process can be repeated until the *n*th nearest element is located, and columns from the resulting text files can be joined into a matrix of regions-of-interest and their nearest *n* genes.

3.3 Statistics and Annotations

The *bedmap* tool associates an element from a *Reference* data set with any number of elements from a second set, *Map*, where associations are determined using a rich set of overlap options. This program is the most feature-rich and general tool for analysis in BEDOPS. Like the *bedops* program, *bedmap* supports overlap specifications using either a number of nucleotides or the proportion of a feature's size, though the latter includes several powerful variants available only with *bedmap*. The program includes options to compute statistics from the overlapping elements and to group the annotations of those elements.

The *bedmap* program can process data for further downstream signal analysis and answer questions about related overlapping features, such as:

- Count how many disease-associated SNPs are found within the open chromatin regions of a cancer tissue sample for comparison against a similar count in a matched sample.
- Smooth experimental sequencing results into binned windows for further signal analysis on a broader scale [12].
- Group TF names that have predicted binding sites at intron-exon boundaries.
- Calculate statistical features of ChIP-seq signals over methylated sites.

Any score-based *bedmap* operation is applied on the measurement or score components of mapped BED elements. For example, one might want to calculate the median or standard deviation of values of all elements from the *Map* input that overlap an element from the *Reference* set. The *bedmap* tool offers these and other common score-based operations, such as trimmed means, variance,

coefficient of variation, mean, median absolute deviation, the *k*th-order statistic, sum, and others.

Other *bedmap* operations report the mapped BED elements themselves or selected aspects of the mapped elements, such as a list of their measurement values or identifiers. Yet other operations count how many elements map onto each *Reference* element, determine the genomic range of all overlapping elements, or calculate the fraction of a *Reference* element's bases covered by *Map* element bases.

There exists a core distinction between what the *bedops --element-of* operator produces and what *bedmap* reports. The *bedops --element-of* operator is a filter that gives back a subset of the elements from a *Reference* data set, specifically the subset that overlaps entries from other inputs. In contrast, *bedmap* reports, per *Reference* element, information about the overlapping elements from the *Map* file.

Like the other utilities, *bedmap* works efficiently with sorted inputs of any size. A person can choose any number of options to compute in a single call to the program, which reduces input/output (I/O) overhead and helps to simplify pipeline scripts. Each specified calculation leads to another appended output column, separated by a default '|' symbol. The next program call calculates a number of phyloP [13] conservation score statistics (*phylop_conservation.bed*) over selected intronic regions (*introns.bed*) to produce a six-column output table. The phyloP conservation file has at least five columns, and the conservation score value is in the fifth column.

```
$ bedmap --mean --stdev --mad --kth 0.25 --median --kth 0.75 \
  introns.bed \
  phylop_conservation.bed \
  > phylop_sequence_conservation_statistics_of_introns.txt
```

4 Working Efficiently with Big Data

There are strategies to working more effectively on whole-genome and similarly large-scale data sets, colloquially described as "Big Data", and BEDOPS enables several practical methodologies. For instance, partitioning input data sets into smaller disjoint subsets often can make it possible to solve focused subproblems in parallel, reducing the overall run-time. If the inputs are ordered, then many algorithms can capitalize on that structured information and avoid reading in all data at once. Finally, if inputs are compressed and indexed, then disk access and network I/O overhead can be reduced. Employing some or all of these strategies can make finding solutions to large-scale bioinformatics problems more tractable.

4.1 Parallelization

In combination with popular open-source distributed computing tools like GNU Parallel, Open Grid Scheduler and Hadoop, BEDOPS applications can quickly partition large, whole-genome analyses into smaller, per-chromosome problems that can be distributed and computed separately, by simply specifying the *--chrom* option along with a chromosome name. With this option, a BEDOPS application immediately jumps to and works with data only from the requested chromosome.

Once all of the smaller calculations are complete, a suitable utility can collate results. Performing tasks in parallel can reduce overall computation time to nearly that required to complete the largest subproblem. Often, this is the time taken to process the largest chromosome, which typically contains the greatest number of input elements in practice.

A generic *map-reduce* approach employs three steps to parallelize work:

1. Build a list of distinct chromosome names in the input file(s).
2. Loop through the list, launching a BEDOPS task on a computational unit or node with the *--chrom* operand, which focuses work on a specified chromosome.
3. Collate all smaller results into a final answer.

BEDOPS includes scripts that show this technique in speeding up the conversion of an indexed BAM file to a BED or Starch (compressed and indexed BED) file, as well as quickly generating a Starch-formatted archive from a BED input.

More concretely, when converting indexed BAM to BED with the parallelized version of *bam2bed*, the procedure is as follows:

1. Use *samtools* [5] on the BAM input to generate a list of chromosome names.
2. Loop through the list of names, farming out per-chromosome BAM-to-SAM extraction tasks to a computational node. On each node, convert the extracted per-chromosome SAM data to BED with *sam2bed*.
3. Concatenate the per-chromosome BED files into one final result with the Unix core utility, *cat*.

4.2 Sorting

The principal utilities of BEDOPS require ordered BED inputs. In exchange, operations are efficient in both time and memory, and they usually produce sorted results that are immediately available for further BEDOPS operations. As an additional benefit to working with sorted data, the toolkit works with any species' genome without modification.

The BEDOPS toolkit provides *sort-bed*, a tool to put data in the requisite order. In practice, the tool is considerably faster than the

equivalent operation using GNU *sort* or similar tools, and it includes an option to limit the use of RAM so that even the largest data sets can be sorted safely. The program accepts any practical number of input data sets and sends a single, final result to its output data stream. BEDOPS defines proper order first by a lexicographical sort on chromosome names, followed by a numerical sort on the start coordinates, with a further numerical sort on the end coordinates when needed.

```
$ sort-bed unsorted_dataset.bed > sorted_dataset.bed
```

The tool also works with data coming from standard input, where a hyphen instructs the program to read from its standard input data stream.

```
$ cat unsorted_dataset.bed \
  | sort-bed - \
  > sorted_dataset.bed
```

While sorting can be a relatively expensive operation in extreme cases, it is almost always a one-time, upfront cost applied to initial, unsorted data. The types of computations offered by the toolkit simply require sorted data. Rather than imposing those costs each time a utility is called, the toolkit requires ordered inputs, with which it keeps overhead minimal while also producing ordered outputs. This means that results generated from one BEDOPS utility can be used directly as an input to any subsequent BEDOPS operation without any overhead due to further sorting.

By default, *sort-bed* consumes as much system memory as needed to sort its input data. A person can temper this behavior by specifying a maximum allowed memory size with the *--max-mem* option, at the cost of a longer run-time.

```
$ sort-bed --max-mem 1G really_large_unsorted_dataset.bed > sorted_dataset.bed
```

4.3 Compression

BEDOPS introduces an archive format called Starch, which provides two useful features: lossless compression and indexing. Efficient compression is obtained by removing redundancy in BED coordinate data before applying compression with a choice between two common, open-source algorithms (Fig. 6). An index is created to provide random access to the start of each chromosome of the archive upon data extraction. The index additionally stores per-chromosome statistics.

The *starch* and *unstarch* tools, respectively, compress and extract data. The *starch* tool allows the person to choose the back-end compression method, as well as append useful metadata to the archive. An annotation note may be as simple as a sentence to describe the file's provenance or purpose, or as complex as a barcode or other machine-readable key that can be useful in identifying the file in an automated pipeline. On the other end, the *unstarch* tool extracts archive data and prints metadata to its standard

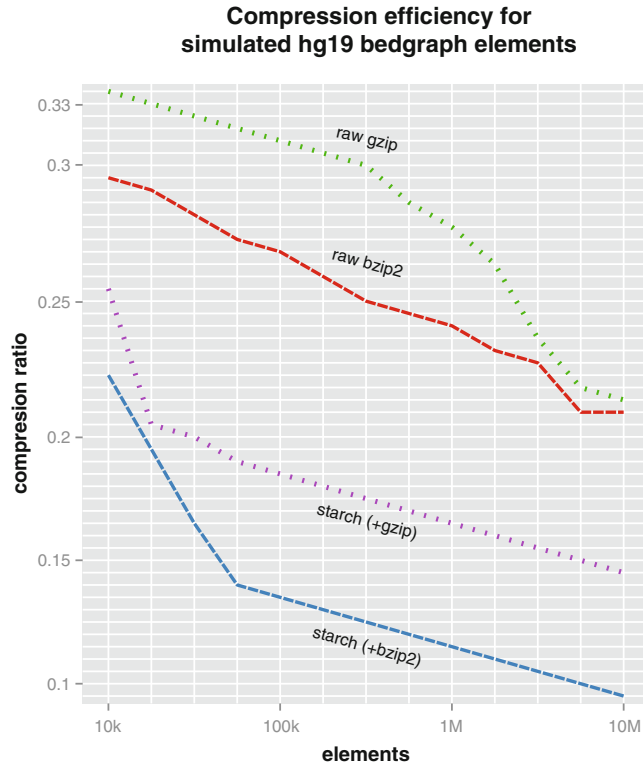

Fig. 6 Compression of 10k to 10 million simulated hg19 bedGraph (UCSC BED3 with a floating-point value) elements, with efficiency measured by comparing file sizes of raw *bzip2*, raw *gzip*, and *bzip2*- and *gzip*-backed Starch archives against the original dataset. Use of *starch* to reduce larger datasets provides considerable space savings over the associated raw compression approach

output. The metadata includes the base coverage and how many elements are contained in an archive, for each chromosome separately or altogether, as well as any note annotation.

Compressing sorted BED data with *starch* is very simple:

```
$ starch dataset.bed > dataset.starch
```

As with other tools in the BEDOPS suite, however, *starch* also accepts data from an upstream Unix process and writes a compressed archive to standard output, which facilitates integration with analysis pipelines.

```
$ awk '$6 == "-"' TSSs.bed \
  | bedops --range 0:1000 --everything - \
  | bedops --everything 1kb_windows_upstream_of_forward_stranded_TSSs.bed - \
  | bedmap --echo --echo-map-id-uniq - motif_predictions.bed \
  | starch --note "Experiment 27B/6 | JASPAR CORE 5.0_ALPHA | GENCODE v19 TSS" - \
  > motif_IDs_1kb_upstream_padding_TSSs.starch
```

Answers to metadata queries with *unstarch* are returned immediately.

```
$ unstarch --note motif_IDs_1kb_upstream_padding_TSSs.starch
Experiment 27B/6 | JASPAR CORE 5.0_ALPHA | GENCODE v19 TSS
$ unstarch --elements motif_IDs_1kb_upstream_padding_TSSs.starch 529
```

While *unstarch* is used to extract information from a Starch-formatted archive and put it on a standard output stream as BED, other BEDOPS utilities work directly with the archive files. Operations can be applied to the entire contents of the archive, or to just one specific chromosome, which can facilitate faster, more focused analyses and parallelization.

Compressed Starch archives can be merged efficiently into one larger archive with *starchcat*. This is highly useful for parallelized analysis pipelines, where results for individual per-chromosome Starch files can be concatenated into one final archive. The *starchcat* tool can also upgrade older Starch archives to add newer metadata annotation features as they become available.

5 Further Reading

The BEDOPS toolkit may be downloaded from its main site (*http://bedops.readthedocs.org/en/latest/*), which includes extensive documentation on all programs and their options, basic and advanced pipeline examples, and performance tips. Additional help from authors and community members is available through an online forum (*http://bedops.uwencode.org/forum/*).

References

1. Kahn SD (2011) On the future of genomic data. Science 331:728–729
2. ENCODE Project Consortium (2012) An integrated encyclopedia of DNA elements in the human genome. Nature 489:57–74
3. Neph S, Kuehn MS, Reynolds AP et al (2012) BEDOPS: high-performance genomic feature operations. Bioinformatics 28(14):1919–1920
4. Kent WJ, Sugnet CW, Furey TS et al (2002) The human genome browser at UCSC. Genome Res 12:996–1006
5. Li H, Handsaker B, Wysoker A et al (2009) The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics 25 (16):2078–2079
6. McIlroy MD, Pinson EN, Tague BA (1978) Unix time-sharing system foreword. Bell Syst Tech J 57(6)
7. Maurano MT, Humbert R, Rynes E et al (2012) Systematic localization of common disease-associated variation in regulatory DNA. Science 337:1190–1195
8. Stergachis AB, Neph S, Reynolds A et al (2013) Developmental fate and cellular maturity encoded in human regulatory DNA landscapes. Cell 154(4):888–903
9. Neph S, Vierstra J, Stergachis AB et al (2012) An expansive human regulatory lexicon encoded in transcription factor footprints. Nature 489:83–90
10. Thurman RE, Rynes E, Humbert R et al (2012) The accessible chromatin landscape of the human genome. Nature 489:75–82
11. Neph S, Stergachis AB, Reynolds A et al (2012) Circuitry and dynamics of human transcription factor regulatory networks. Cell 150 (6):1274–1286
12. John S, Sabo PJ, Thurman RE et al (2011) Cell-specific chromatin landscapes determine cell-selective glucocorticoid receptor occupancy. Nat Genet 43:264–268
13. Pollard KS, Hubisz MJ, Rosenbloom KR et al (2010) Detection of nonneutral substitution rates on mammalian phylogenies. Genome Res 20:110–121