BEDOPS Documentation

Release 2.4.37

Shane Neph, Alex Reynolds

Oct 12, 2019

Contents

BEDOPS is an open-source command-line toolkit that performs highly efficient and scalable Boolean and other set operations, statistical calculations, archiving, conversion and other management of genomic data of arbitrary scale. Tasks can be easily split by chromosome for distributing whole-genome analyses across a computational cluster.

You can read more about **BEDOPS** and how it can be useful for your research in the *Overview* documentation, as well as in the original manuscript.

CHAPTER 1

Citation

If you use **BEDOPS** in your research, please cite the following manuscript:

> Shane Neph, M. Scott Kuehn, Alex P. Reynolds, et al. **BEDOPS: high-performance genomic feature operations**. *Bioinformatics* (2012) 28 (14): 1919-1920. doi: 10.1093/bioinformatics/bts277

CHAPTER 2

Contents

2.1 Overview

2.1.1 About BEDOPS

BEDOPS is an open-source command-line toolkit that performs highly efficient and scalable Boolean and other set operations, statistical calculations, archiving, conversion and other management of genomic data of arbitrary scale.

The suite includes tools for set and statistical operations (*bedops*, *bedmap* and *closest-features*) and compression of large inputs into a novel lossless format (*starch*) that can provide greater space savings and faster data extractions than current alternatives. BEDOPS offers native support for this deep compression format, in addition to BED.

BEDOPS also offers logarithmic time search to per-chromosome regions in sorted BED data (in *bedextract* and core BEDOPS tools). This feature makes whole-genome analyses "embarassingly parallel", in that per-chromosome computations can be distributed onto separate work nodes, with results collated at the end in map-reduce fashion.

Sorting arbitrarily large BED files is easy with *sort-bed*, which easily scales beyond available system memory, as needed. We also offer portable conversion scripts that transform data in common genomic formats (SAM/BAM, GFF/GTF, PSL, WIG, and VCF) to sorted BED data that are ready to use with core BEDOPS utilities.

All of these tools are made to be glued together with common UNIX input and output streams. This helps make your pipeline design and maintenance easy, fast and flexible.

2.1.2 Why you should use BEDOPS

BEDOPS tools are flexible

Our tools fit easily into analysis pipelines, allow practically unlimited inputs, and reduce I/O overhead through standard UNIX input and output streams:

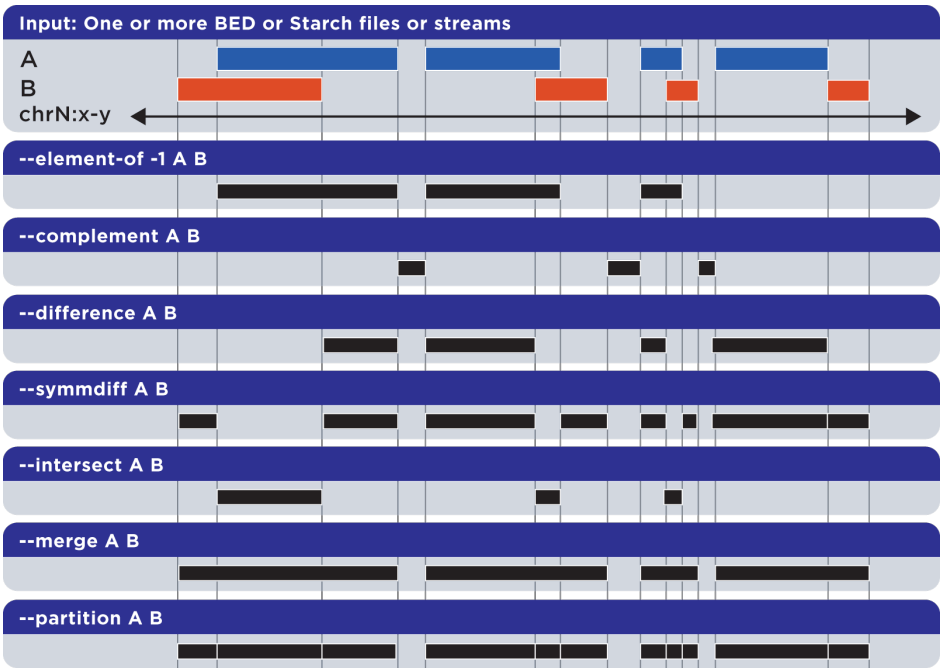
```
$ bedops --intersect A.bed B.bed C.bed \
    | bedmap --echo --mean - D.bed \
```

(continues on next page)

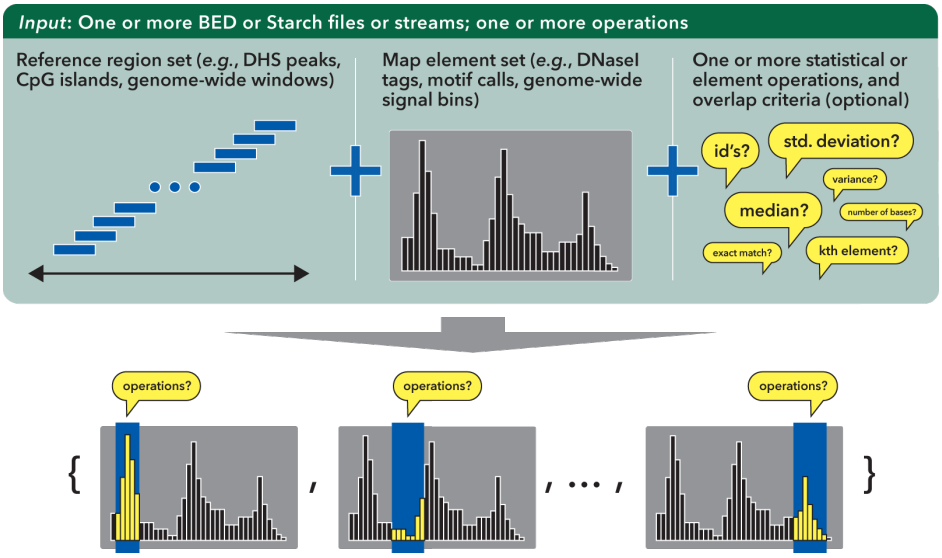
(continued from previous page)

```
    | ... \
    > Answer.bed
```

Our *bedops* core tool offers numerous set operations of all kinds, including those in the slide below:



The *bedmap* core tool applies a wide variety of statistical and mapping operations to genomic inputs:



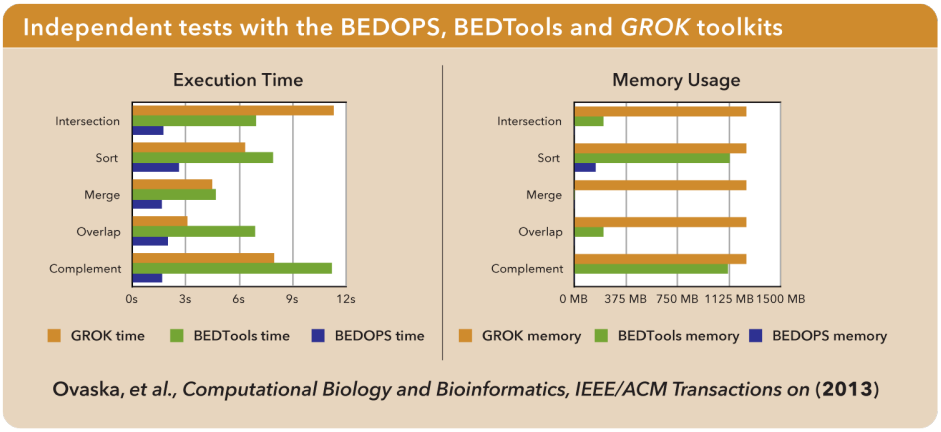
These and other tools send their results to the output stream, ready for consumption by processes downstream along your pipeline.

BEDOPS tools are fast and efficient

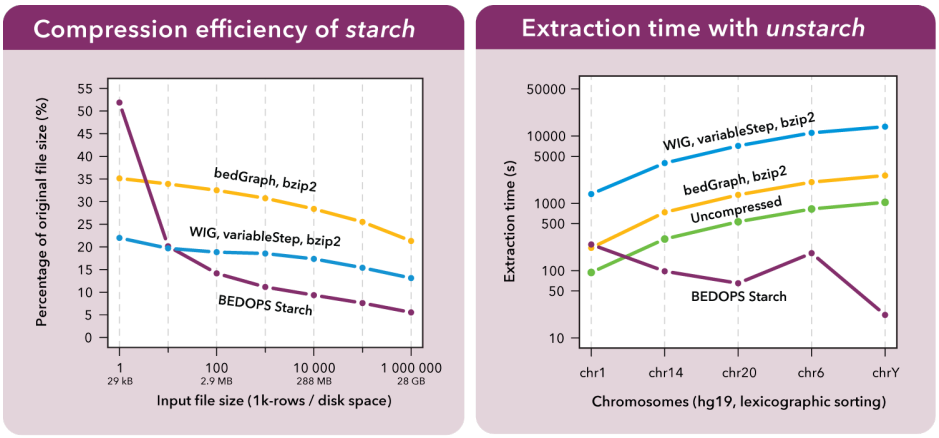
BEDOPS tools take advantage of the information in a sorted BED file to use only what data are needed to perform the analysis. Our tools are agnostic about genomes: Run BEDOPS tools on genomes as small as *Circovirus* or as large as

Polychaos dubium!

Independent tests comparing various kits show that BEDOPS offers the fastest operations with the lowest memory overhead:



BEDOPS also introduces a novel and **lossless** compression format called *Starch* that reduces whole-genome BED datasets to **~5%** of their original size (and BAM datasets to roughly 35% of their original size), while adding useful metadata and random access, allowing instantaneous retrieval of any compressed chromosome:



BEDOPS tools make your work embarrassingly easy to parallelize

BEDOPS tools introduce the `--chrom` option to efficiently locate a specified chromosome within a sorted BED file, useful for "embarrassingly parallel" whole-genome analyses, where work can be logically divided by units of chromosome in a "map-reduce" fashion.

BEDOPS tools are open, documented and supported

BEDOPS is available as GPL-licensed source code and precompiled binaries for Linux and Mac OS X. We offer support through online forums such as our own and Biostars and recipes showing BEDOPS tools in use for answering common research questions.

2.2 Installation

BEDOPS is available to users as *pre-built binaries* and *source code*.

2.2.1 Via pre-built packages

Pre-built binaries offer the easiest and fastest installation option for users of BEDOPS. At this time, we offer binaries for 64-bit versions of Linux and OS X (Intel) platforms. 32-bit binaries can be built via source code by adjusting compile-time variables.

Linux

1. Download the current 64-bit package for Linux from Github BEDOPS Releases.
2. Extract the package to a location of your choice. In the case of 64-bit Linux:

   ```
   $ tar jxvf bedops_linux_x86_64-vx.y.z.tar.bz2
   ```

   Replace `x`, `y` and `z` with the version number of BEDOPS you have downloaded.
3. Copy the extracted binaries to a location of your choice which is in your environment's `PATH`, *e.g.* `/usr/local/bin`:

   ```
   $ cp bin/* /usr/local/bin
   ```

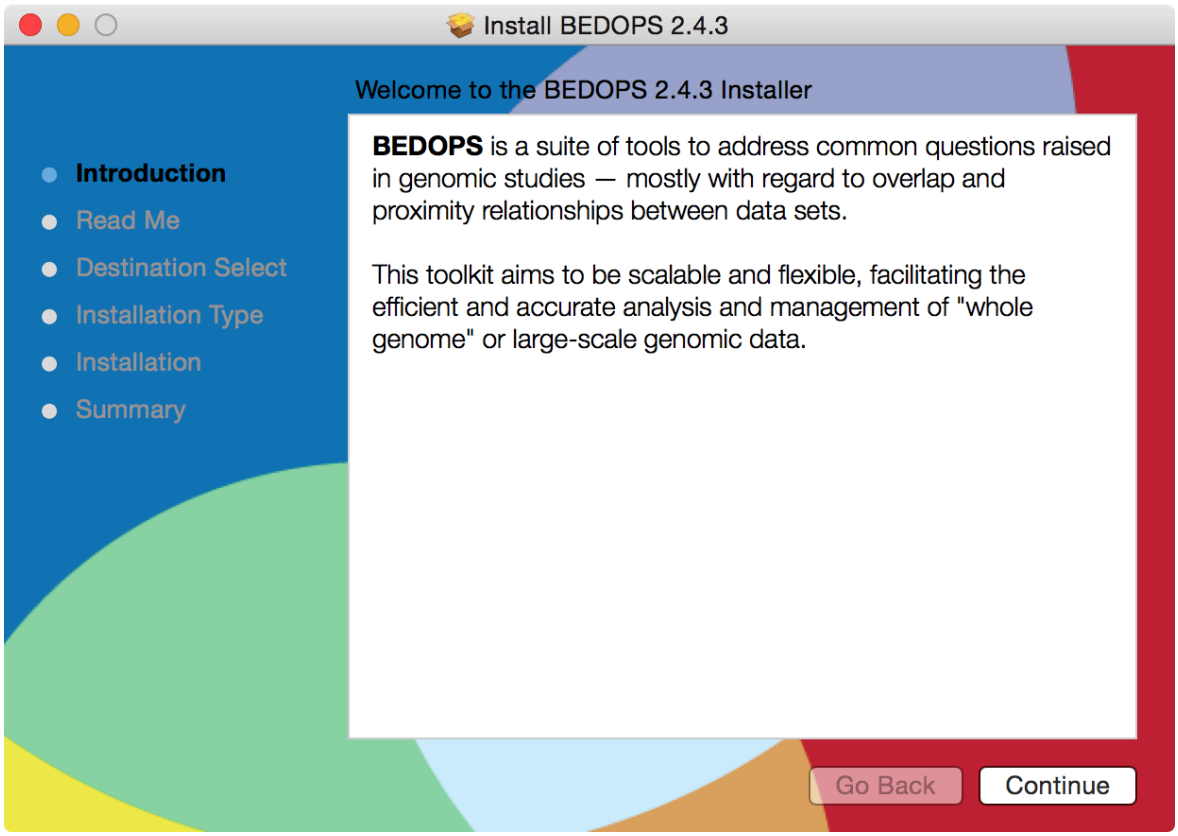
   Change this destination folder, as needed.

Mac OS X

1. Download the current Mac OS X package for BEDOPS from Github BEDOPS Releases.
2. Locate the installer package (usually located in `~/Downloads` – this will depend on your web browser configuration):

Name	BEDOPS 2.2.0b.mpkg
Kind	Installer package
Size	4.8 MB
Created	Wednesday, June 12, 2013 11:42 AM
Modified	Wednesday, June 12, 2013 11:42 AM
Last opened	Wednesday, June 12, 2013 11:42 AM

3. Double-click to open the installer package. It will look something like this:



4. Follow the instructions to install BEDOPS and library dependencies to your Mac. (If you are upgrading from a previous version, components will be overwritten or removed, as needed.)

2.2.2 Via source code

Linux

Compilation of BEDOPS on Linux requires GCC 4.8.2 (both `gcc` and `g++` and related components) or greater, which includes support for C++11 features required by core BEDOPS tools. Other tools may be required as described in the installation documentation that follows.

1. If you do not have GCC 4.8.2 or greater installed (both `gcc` and `g++`), first install these tools. You can check the state of your GCC installation with `gcc --version` and `g++ --version`, *e.g.*:

```
$ gcc --version
gcc (GCC) 4.8.2 20140120 (Red Hat 4.8.2-15)
...
```

   If you lack a compiler or have a compiler that is older than 4.8.2, use your favorite package manager to install or upgrade the newer package. For example, in Ubuntu, you might run the following:

```
$ sudo apt-get install gcc-4.8
$ sudo apt-get install g++-4.8
$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 50
$ sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 50
```

   The specifics of this process will depend on your distribution and what you want to install. Please check with your system administration or support staff if you are unsure what your options are.

You may also need to install static libraries. For instance, in a CentOS- or RH-like environment:

```
$ sudo yum install libstdc++-static
$ sudo yum install glibc-static
```

In Ubuntu, you might instead do:

```
$ sudo apt-get install libc6-dev
$ sudo apt-get install build-essentials
```

2. Install a `git` client of your choice, if you do not already have one installed. Github offers an installation guide.

   Alternatively, use `apt-get` or another package manager to install one, *e.g.* in Ubuntu:

   ```
   $ sudo apt-get install git
   ```

   And in CentOS:

   ```
   $ sudo yum install git
   ```

3. Clone the BEDOPS Git repository in an appropriate local directory:

   ```
   $ git clone https://github.com/bedops/bedops.git
   ```

4. Enter the top-level of the local copy of the BEDOPS repository and run `make` to begin the build process:

   ```
   $ cd bedops
   $ make
   ```

   Running `make` on its own will build so-called "typical" BEDOPS binaries, which make assumptions about line length for most usage scenarios.

   Use `make megarow` or `make float128` to build support for longer-length rows, or BED data which requires statistical or measurement operations with *bedmap* with 128-bit precision floating point support.

   If you want all build types, run `make all`.

Tip: BEDOPS supports parallel builds, which speeds up compilation considerably. If you are compiling on a multi-core or multiprocessor workstation, edit the `JPARALLEL` variable in the top-level Makefile, or override it, specifying the number of cores or processors you wish to use to compile.

---

5. Once the build is complete, install compiled binaries and scripts to a local `bin` directory:

   ```
   $ make install
   ```

   If you ran `make megarow` or `make float128`, instead use `make install_megarow` or `make install_float128`, respectively, to install those binaries.

   If you ran `make all`, use `make install_all` to install all binaries of the three types (typical, megarow, and float128) to the `./bin` directory. You can use the `switch-BEDOPS-binary-type` script to switch symbolic links to one of the three binary types.

6. Copy the extracted binaries to a location of your choice that is in your environment's `PATH`, *e.g.* `/usr/local/bin`:

   ```
   $ cp bin/* /usr/local/bin
   ```

   Change this destination folder, as needed.

Mac OS X

In Mac OS X, you have a few options to install BEDOPS via source code: Compile the code manually, or use the Bioconda or Homebrew package manager to manage installation.

Compilation of BEDOPS on Mac OS X requires Clang/LLVM 3.5 or greater, which includes support for C++11 features required by core BEDOPS tools. Other tools may be required as described in the installation documentation that follows. GNU GCC is no longer required for compilation on OS X hosts.

Manual compilation

1. If you do not have Clang/LLVM 3.5 or greater installed, first do so. You can check this with `clang -v`, *e.g.*:

   ```
   $ clang -v
   Apple LLVM version 8.0.0 (clang-800.0.42.1)
   ...
   ```

   For Mac OS X users, we recommend installing Apple Xcode and its Command Line Tools, via the `Preferences > Downloads` option within Xcode. At the time of this writing, Xcode 8.2.1 (8C1002) includes the necessary command-line tools to compile BEDOPS.

2. Install a `git` client of your choice, if you do not already have one installed. Github offers an installation guide.

3. Clone the BEDOPS Git repository in an appropriate local directory:

   ```
   $ git clone https://github.com/bedops/bedops.git
   ```

4. Run `make` in the top-level of the local copy of the BEDOPS repository:

   ```
   $ cd bedops
   $ make
   ```

   Running `make` on its own will build so-called "typical" BEDOPS binaries, which make assumptions about line length for most usage scenarios.

   Use `make megarow` or `make float128` to build support for longer-length rows, or BED data which requires statistical or measurement operations with *bedmap* with 128-bit precision floating point support.

   If you want all build types, run `make all`.

Tip: BEDOPS supports parallel builds, which speeds up compilation considerably. If you are compiling on a multi-core or multiprocessor workstation, edit the `JPARALLEL` variable in the top-level Makefile, or override it, specifying the number of cores or processors you wish to use to compile.

---

5. Once the build is complete, install compiled binaries and scripts to a local `bin` folder:

   ```
   $ make install
   ```

   If you ran `make megarow` or `make float128`, instead use `make install_megarow` or `make install_float128`, respectively, to install those binaries.

   If you ran `make all`, use `make install_all` to install all binaries of the three types (typical, megarow, and float128) to the `./bin` directory.

   You can use the `switch-BEDOPS-binary-type` script to switch symbolic links to one of the three binary types.

6. Copy the extracted binaries to a location of your choice that is in your environment's `PATH`, *e.g.* `/usr/local/bin`:

```
$ cp bin/* /usr/local/bin
```

   Change this destination folder, as needed.

Installation via Bioconda

Bioconda is a bioinformatics resource that extends the Conda package manager with scientific software packages, including BEDOPS. We aim to keep the recipe concurrent with the present release; occasionally, it may be a minor version behind.

What follows are steps taken from the Bioconda installation page. Use this guide for the most current set of instructions, which we briefly cover here:

1. Follow the instructions on Conda's website to install the Miniconda package, which installs the `conda` command-line tool.
2. If you have not already done so, add the Conda channels that Bioconda depends upon:

```
$ (conda config --add channels r)
$ conda config --add channels defaults
$ conda config --add channels conda-forge
$ conda config --add channels bioconda
```

3. Install the BEDOPS package:

```
$ conda install bedops
```

Other recipes are available for installation, as well.

Installation via Homebrew

Homebrew is a popular package management toolkit for Mac OS X. It facilitates easy installation of common scientific and other packages. Homebrew can usually offer a version of BEDOPS concurrent with the present release; occasionally, it may be one or two minor versions behind.

1. If you do not have Clang/LLVM 3.5 or greater installed, first do so. You can check this with `clang -v`, *e.g.*:

```
$ clang -v
Apple LLVM version 8.0.0 (clang-800.0.42.1)
...
```

   For Mac OS X users, we recommend installing Apple Xcode and its Command Line Tools, via the `Preferences > Downloads` option within Xcode. At the time of this writing, Xcode 8.2.1 (8C1002) includes the necessary command-line tools to compile BEDOPS.

2. Follow the instructions listed on the Homebrew site to install the basic package manager components.
3. Run the following command:

```
$ brew install bedops
```

Docker

Docker containers wrap up a piece of software (such as BEDOPS) in a complete, self-contained VM.

To set up a CentOS 7-based Docker container with BEDOPS binaries, you can use the following steps:

```
$ git clone https://github.com/bedops/bedops.git
$ cd bedops
$ make docker
...
$ docker run -i -t bedops
```

The following then generates a set of RPMs using the CentOS 7 image, which can run in CentOS 6 and Fedora 21 containers:

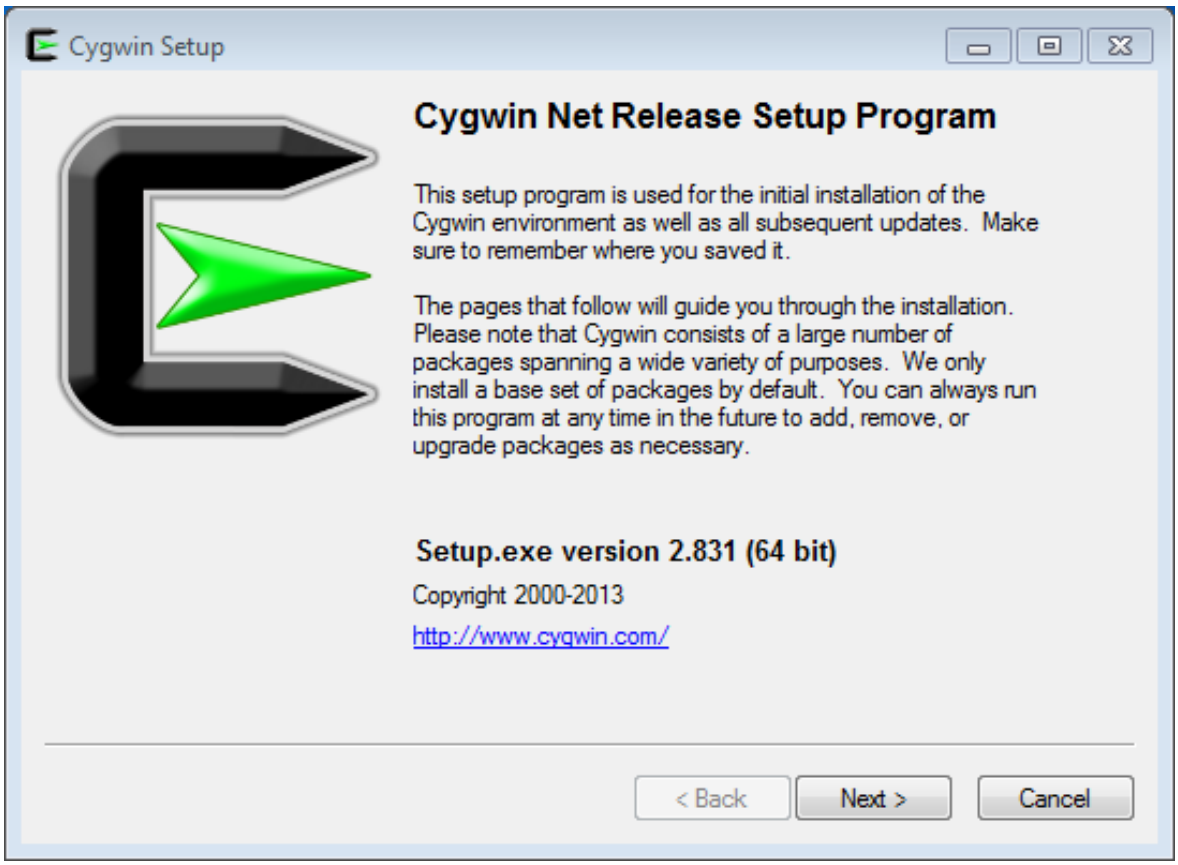
```
$ make rpm
```

Thanks go to Leo Comitale for his efforts here.

Cygwin

1. Make sure you are running a 64-bit version of Cygwin. Compilation of BEDOPS on 32-bit versions of Cygwin is not supported.

   To be sure, open up your Cywin installer application (separate from the Cygwin terminal application) and look for the **64 bit** marker next to the setup application version number:

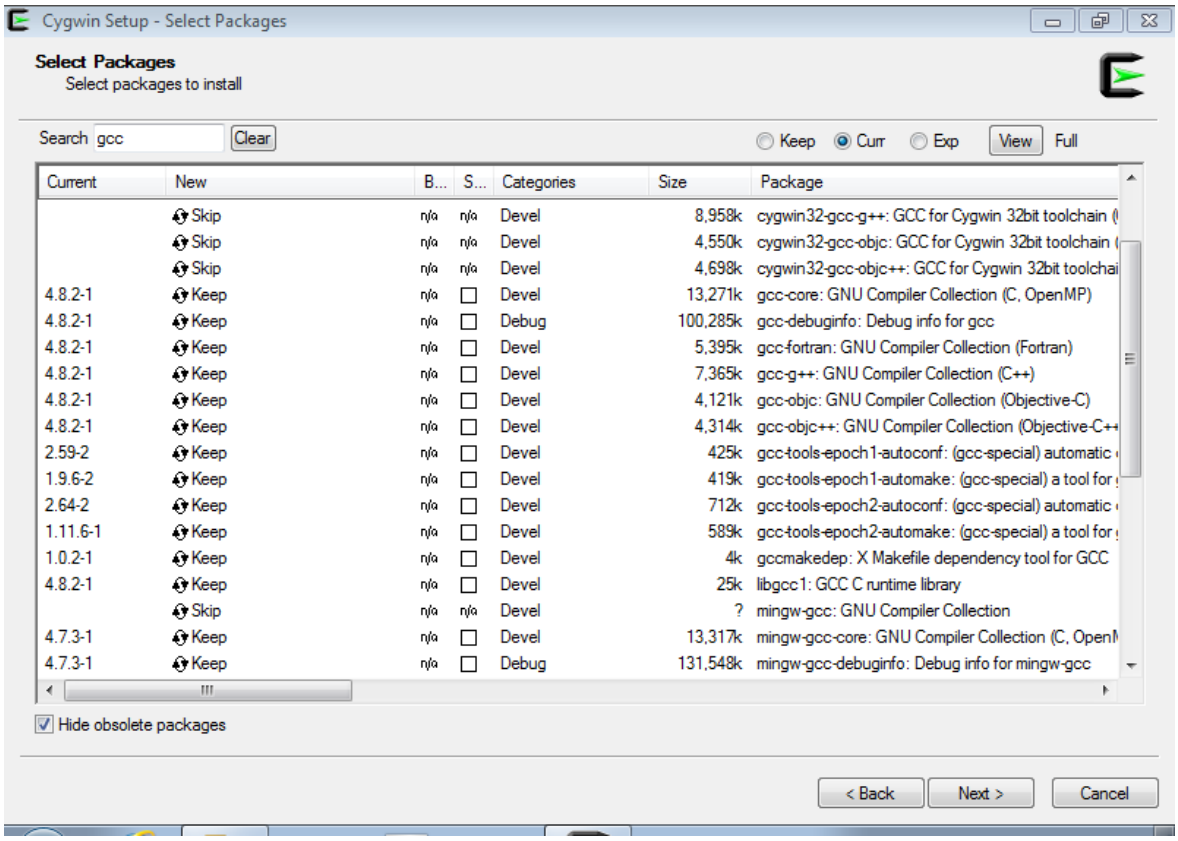


   For instance, this Cygwin installer is version 2.831 and is 64-bit.

2. Check that you have GCC 4.8.2 or greater installed. You can check this by opening the Cygwin terminal window (note that this is not the same as the Cygwin installer application) and typing `gcc --version`, *e.g.*:

```
$ gcc --version
gcc (GCC) 4.8.2
...
```

If you do not have `gcc` installed, then open the Cygwin (64-bit) installer application again, navigate through the current setup options, and then mark the GCC 4.8.* packages for installation:

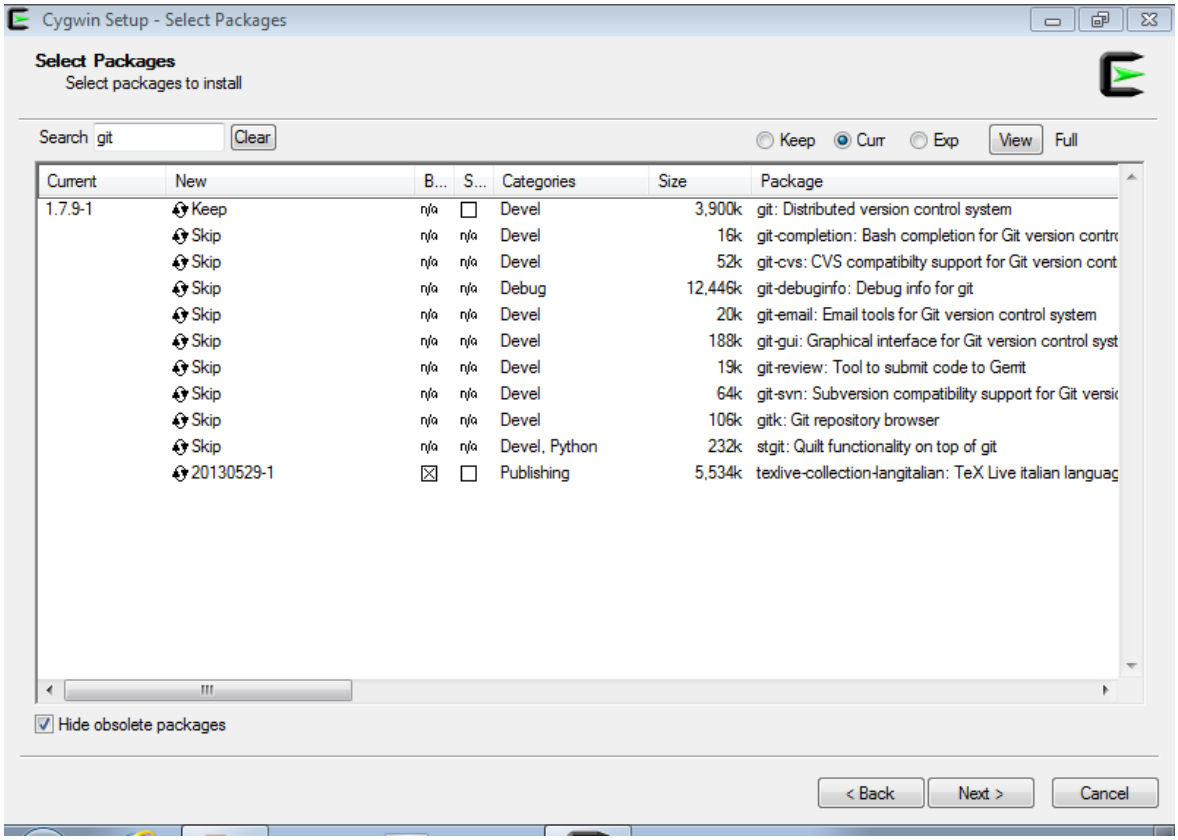


If it helps, type in `gcc` into the search field to filter results to GCC-related packages. Make sure to mark the following packages for installation, at least:

- **gcc-core**
- **gcc-debuginfo**
- **gcc-g++**
- **gcc-tools-xyz**
- **libgcc1**

Click "Next" to follow directives to install those and any other selected package items. Then run `gcc --version` as before, to ensure you have a working GCC setup.

3. Install a `git` client of your choice. You can compile one or use the precompiled `git` package available through the Cygwin (64-bit) installer:



If it helps, type in `git` into the search field to filter results to Git-related packages. Make sure to install the following package, at least:

- **git**

4. In a Cygwin terminal window, clone the BEDOPS Git repository to an appropriate local directory:

```
$ git clone https://github.com/bedops/bedops.git
```

4. Enter the top-level of the local copy of the BEDOPS repository and run `make` to begin the build process:

```
$ cd bedops
$ make
```

Tip: BEDOPS now supports parallel builds. If you are compiling on a multicore or multiprocessor workstation, use `make -j N` where `N` is `2`, `4` or however many cores or processors you have, in order to parallelize and speed up the build process.

5. Once the build is complete, install compiled binaries and scripts to a local `bin` folder:

```
$ make install
```

6. Copy the extracted binaries to a location of your choice that is in your environment's `PATH`, *e.g.* `/usr/bin`:

```
$ cp bin/* /usr/bin
```

Change this destination folder, as needed.

2.2.3 Building an OS X installer package for redistribution

1. Follow steps 1-3 and step 5 from the *Via Source Code* documentation.
2. Run `make install_osx_packaging_bins` in the top-level of the local copy of the BEDOPS repository:

```
$ make install_osx_packaging_bins
```

3. Install WhiteBox Packages.app, an application for building OS X installers, if not already installed.

   On 10.13 hosts, it may be necessary to install a more recent development build of `Packages.app` via Packages Q&A #6.

4. Create a `build` directory to store the installer and open the `BEDOPS.pkgproj` file in the top-level of the local copy of the BEDOPS repository, in order to open the BEDOPS installer project, *e.g.*:

```
$ mkdir -p packaging/os_x/build && open packaging/os_x/BEDOPS.pkgproj
```

   This will open up the installer project with the `Packages.app` application.

5. Within `Packages.app`, modify the project to include the current project version number or other desired changes, as applicable. Make sure the project is set up to build a "flat"-formatted (xar) package, not a bundle, otherwise the digital signing step will fail.
6. Run the `Build > Build` menu selection to construct the installer package, located in the `packaging/os_x/build` subdirectory. Move this installer to the `/tmp` directory:

```
$ mv packaging/os_x/build/BEDOPS\ X.Y.Z.pkg /tmp/BEDOPS.X.Y.Z.unsigned.pkg
```

7. Find the `Developer ID Installer` name that will be used to digitally sign the installer `pkg` file, *e.g.*:

```
$ security find-certificate -a -c "Developer ID Installer" | grep "alis"
    "alis"<blob>="Developer ID Installer: Foo B. Baz (ABCD12345678)"
```

   Here, the name is `Developer ID Installer: Foo B. Baz`.

   (This certificate name is unique to the developer. If necessary, you may need to sign up for a Mac Developer Program account with Apple to set up required certificates.)

8. Sign the package installer, *e.g.*:

```
$ productsign --timestamp --sign "Developer ID Installer: Foo B. Baz" /tmp/BEDOPS.
↪X.Y.Z.unsigned.pkg /tmp/BEDOPS.X.Y.Z.signed.pkg
```

9. Compress the signed `pkg` file (via OS X zip, for instance) and publish via GitHub releases (see *release preparation* for information about publishing the installer).

2.3 Revision history

This page summarizes some of the more important changes between releases.

2.3.1 Current version

v2.4.37

Released: **TBD**

- *bedmap*
  - Running `bedmap --version` now exits with a zero (non-error/success) status.
- *starch*
  - When a Starch file with a header is provided as input to `bedops` or `bedmap`, the line is errantly processed as a BED interval. Thanks to André M. Ribeiro-dos-Santos for patching the Starch C++ API to skip headers.
  - Added a unit test to `tests/starch` to test headered Starch mapped against itself.
- General
  - Applied a placeholder workaround to whatever stupid bug was introducted in Issue 5709 that broke image serving for the document index (front page).
  - Improved speed of generating random intervals in `tests/starch` unit tests.

2.3.2 Previous versions

v2.4.36

Released: **May 2, 2019**

- *bedmap*
  - Resolved an issue preventing use of a `bash` process substitution or Unix named pipe in the reference position: *i.e.*, `bedmap --options <(processToGenerateReferenceElements) map.bed` and similar would issue incorrect output. Thanks to Wouter Meuleman and others for reports and test input.
  - To avoid mapping problems, map elements should not contain spaces in the ID or subsequent non-interval fields. Use of the `--ec` can help identify problems in map input, at the cost of a longer runtime. The documentation is clarified to warn users about avoiding spaces in map input. Thanks to Wouter Meuleman for the report and providing test input.
  - Added `--unmapped-val <val>` option, where `<val>` replaces the empty string output of `--echo-map*` operations when there are no mapped elements. The `--min/max-element` operators will give results as before (the empty string).
- General
  - Reduced `warning: zero as null pointer constant [-Wzero-as-null-pointer-constant]` compiler warnings via Clang.
  - Begun work on a comprehensive test suite for BEDOPS applications. Tests are available via source distribution in `${root}/tests` and can be run by entering `make` in this directory.

v2.4.35

Released: **May 2, 2018**

- *starch*
  - When compressing records, if the last interval in the former chromosome is identical to the first interval of the next chromosome, then a test on the sort order of the remainder string of that interval is applied (incorrectly). This is patched to test that chromosome names are identical before applying sort order rules. Thanks to Andrew Nishida for the report and for providing test input.

v2.4.34

Released: **April 26, 2018**

- *convert2bed*
  - In Issue 208 builds of *convert2bed* would exit with an error state when converting SAM input with newline-delimited records longer than the 5 MB per-thread I/O buffer. The `C2B_THREAD_IO_BUFFER_SIZE` constant is now set to the suite-wide `TOKENS_MAX_LENGTH` value, which should make it easier to compile custom builds of BEDOPS that support very-long line lengths. Thanks to Erich Schwarz for the initial report.
- starchstrip
  - When *starchstrip* is compiled with a C compiler, `qsort` uses a comparator that works correctly on the input chromosome list. When compiled with a C++ compiler (such as when building the larger BEDOPS toolkit), a different comparator is used that does not make variables of the correct type, and so the `qsort` result is garbage, leading to missing chromosomes. Thanks to Jemma Nelson for the initial report.

v2.4.33

Released: **April 9, 2018**

- *convert2bed*
  - Resolved Issue 207 where a megarow build of *convert2bed* would raise segmentation faults when converting SAM input. This build uses constants that define a longer BED line length. Arrays ended up using more stack than available, resulting in segmentation faults. This issue could potentially affect conversion of any data with the megarow build of *convert2bed*. Arrays using megarow-constants were replaced with heap- or dynamically-allocated pointers. Thanks to Erich Schwarz for the initial report.

v2.4.32

Released: **March 14, 2018**

- New build type (128-bit precision floating point arithmetic, `float128`)
  - A new build type adds support for `long double` or 128-bit floating point operations on measurement values in *bedmap*, such as is used with score operators like: `--min`, `--max`, `--min-element`, `--max-element`, `--mean`, and so on.
  - This build includes support for measurements on values ranging from approximately ±6.48e4966 to ±6.48e4966 (subnormal), or ±1.19e-4932 to ±1.19e4932 (normal), which enables *bedmap* to handle, for example, lower p-values without log- or other transformation preprocessing steps. The article on quadruple precision can be useful for technical review.
  - For comparison, the current "non-float128" typical and megarow builds allow measurements on values from approximately ±5e324 to ±5e324 (subnormal) or ±2.23e-308 to ±2.23e308 (normal). Please refer to the article on double precision for more technical detail.
  - Please use `make float128 && make install_float128` to install this build type.
  - This build type combines support for quadruple, 128-bit precision floats with the `typical` build type for handling "typical" BED4+ style line lengths. At this time, "megarow" support is not enabled with higher precision floats.
  - This build will use more memory to store floating-point values with higher precision, and processing those data will require more computation time. It is recommended that this build be used only if analyses require a higher level of precision than what the `double` type allows.

- OS X (Darwin) megarow build
  - Some applications packaged in the OS X installer or compiled via the OS X command-line developer toolkit lacked megarow build support, despite those flags being specified in the parent Makefile. These applications specifically were affected: `bedextract`, `bedmap`, and `convert2bed`. These binaries rely on wider suite-wide constants and data types that the megarow build variety specifies. The Darwin-specific Makefiles have been fixed to resolve this build issue, so that all OS X BEDOPS binaries should now be able to compile in the correct megarow-specific settings.

v2.4.31

Released: **March 8, 2018**

- User forum
  - BEDOPS user forum moved domains from http://bedops.stamlab.org to https://bedops.altius.org
  - Testing out administrator approval requirement for new forum accounts, to help try to reduce visits from spammers.
- Documentation
  - Updated Homebrew installation instructions per issue 202 (thanks to user EricFromCanada).
- *wig2bed*
  - Increased maximum length of chromosome name buffer to suite-wide `TOKEN_CHR_MAX_LENGTH` value, to reduce likelihood of segmentation faults (thanks to user ma-diroma).
- General
  - Updated copyright dates in source and headers.

v2.4.30

Released: **November 25, 2017**

- *bedmap*
  - Errors are no longer reported when error checking is enabled and running in non-fast mode, when a fully-nested element is detected. This follows up on issue 199.
- *starch*
  - Previously, a chromosome record in a Starch archive would result in corrupted metadata, if the chromosome is larger than `UINT32_MAX` bytes (~4.3GB) in size when compressed. This limitation is now removed, and a single chromosome (when compressed in a Starch archive) can be up to `UINT64_MAX` bytes in size.
  - The `starch` binary does more stringent input checks for the character lengths of ID and remainder strings, which must be no larger than $2^{\text{ID\_EXPONENT}}$ - 1 and $2^{\text{REST\_EXPONENT}}$ - 1 characters in length. (These constants are specific to the build-time settings in the Makefile and in the app-wide constants.) This follows up on issue 195.
- *starchcat*
  - Previously, a chromosome record in a Starch archive would result in corrupted metadata, if the chromosome is larger than `UINT32_MAX` bytes (~4.3GB) in size when compressed. This limitation is now removed, and a single chromosome (when compressed in a Starch archive) can be up to `UINT64_MAX` bytes in size.

  - More stringent memory management and stricter adherance to BEDOPS-wide constants, to help reduce likelihood of pointer access out of bounds and incidence of segfaults.

- *unstarch*
  - The `unstarch` binary implements the character length constants of ID and remainder strings, specific to the build-time settings in the Makefile and in the app-wide constants. This follows up on issue 195.

- *sort-bed*
  - Added `--unique` (`-u`) and `--duplicates` (`-d`) options to only print unique and duplicate in sorted output, to mimic the behavior of `sort -u` and `uniq -d` Unix tools. This follows up on issue 196.
  - Switched compile-time, stack-allocated `char` arrays to runtime, heap-based pointers. Timing tests on shuffled FIMO datasets suggest the impact from having to allocate space for buffers at runtime is very minimal. Moving from stack to heap will help avoid segfaults from running into OS-level stack limits, when BEDOPS-level constants change the maximum line length to something larger than the stack.

- Revision testing
  - Starch suite tests were updated for v2.2 archives created via v2.4.30 binaries (Linux, libc 2.22).

v2.4.29

Released: **September 26, 2017**

- *bedmap*
  - Increased megarow build ID length up to 2^{18}.
  - Changed behavior of mapping to return mapped items in sort order provided in inputs. This follows up on issue 198.

- *unstarch*
  - Changed behavior of `--is-starch` option to always return a successful exit code of `0` whether or not the input file is a Starch archive. It will now be up to the person running this option to test the 0 (false) or 1 (true) value printed to the standard output stream. This follows up on issue 197.

v2.4.28

Released: **August 18, 2017**

- *bedmap*
  - Patched issue 191 where `--wmean` option was not recognized.

- *bedextract*
  - Updated documentation with fixed usage statement.

- *sort-bed*
  - Patched typo in `update-sort-bed-starch-slurm.py` script.
  - Fixed bug with `--max-mem` on properly ordering things on fourth and subsequent columns, when the genomic intervals are the same.

- *starch*
  - Updated Makefiles to remove *lib* on *clean* target and to help prevent `ARCH` variable from getting clobbered by third-party package managers.

- Build process

- Updated the OS X installer XML to resolve missing asset links.
- Updated the `module_binaries` target to copy over `starchcluster_*` and `starch-diff` assets for `modules` distributions.

v2.4.27

Released: **July 17, 2017**

This revision of BEDOPS includes significant performance improvements for core tools: `bedops`, `bedmap`, and `closest-features`. Performance tests were done with whole-genome TRANSFAC FIMO scans, with cache purges in between trials.

Pre-built binaries for Darwin and GNU/Linux targets include both the default `typical` and `megarow` builds of BEDOPS. The program names that you are accustomed to will remain as-is, but the binaries will exist as symbolic links pointing to the `typical` builds. These links can be repointed to the `megarow` builds by calling `switch-BEDOPS-binary-type --megarow`, which will set the usual BEDOPS binaries to link to the `megarow` builds. One can run `switch-BEDOPS-binary-type --typical` at any time to revert to the default (`typical`) builds.

The top-level Makefile includes some new variables for those who choose to build from source. The `JPARALLEL` variable sets the number of CPUs to use in parallel when compiling BEDOPS, which can speed compilation time dramatically. The `MASSIVE_REST_EXP`, `MASSIVE_ID_EXP`, and `MASSIVE_CHROM_EXP` are used when building the `megarow` to support any required row lengths (build using `make megarow`). These are the exponents (the n in 2^n) for holding all characters after chromosome, start, and stop fields, the ID field (column 4, typically), and the chromosome field (column 1).

To simplify distribution and support, we have removed pre-built 32-bit program versions in this release. These can be built from source by specifying the correct `ARCH` value in the top-level Makefile. For OS X, our package installer now requires OS X version 10.10 or greater.

Application-level notes follow:

- *bedops*
  - Performance of `bedops` tool improved, doing typical work in **76.5%** of the time of all previous versions.
  - Performance of `-u`/`--everything` has improved, doing the same work in only **55.6%** of the time of previous versions when given a large number of input files.
  - The `megarow` build of this application handles input files with very long rows (4M+ characters). Such input might arise from conversion of very-long-read BAM files to BED via `bam2bed`, such as those that may come from Nanopore or PacBio MinION platforms. This build requires more runtime memory than the default (`typical`) build. Pertinent variables for `megarow` execution can be modified through the make system without changing source.
- *bedmap*
  - Performance of `bedmap` tool improved, doing the same work in **86.7%** of the time of all previous versions.
  - Automatically use `--faster` option when `--exact` is used as the overlap criterion, or if the input files are formatted as Starch archives, no fully-nested elements exist in the archives, and the overlap criterion supports `--faster` (such as `--bp-ovr`, `--exact`, and `--range`).
  - The `megarow` build target handles input files with very long rows (4M+ characters). Such input might arise from conversion of very-long-read BAM files to BED via `bam2bed`, such as those that may come from Nanopore or PacBio MinION platforms. This build requires more runtime memory than the default (`typical`) build. Pertinent variables for `megarow` execution can be modified through the make system without changing source.

  - New `--min-memory` option for use when the reference file has very large regions, and the map file has many small regions that fall within those larger regions. One example is when `--range 100000` is used and the map file consists of whole-genome motif scan hits. Memory overhead can be reduced to that used by all previous versions, up to and including v2.4.26.
  - Added `--faster` automatically when `--exact` is used, which is robust even when nested elements exist in inputs. Similarly, `--faster` is used automatically when inputs are Starch-formatted archives, none of which have nested elements (see `unstarch --has-nested`) when the overlap criterion allows for `--faster`.

- *closest-features*
  - Performance of `closest-features` tool has been improved, doing the same work in **87.7%** of the time of all previous versions.
  - The `megarow` build target is available to compile a version of the program that can handle input files with very long rows (4M+ characters). This requires more runtime memory than the default build. Pertinent variables can be modified through the make system without editing source.

- *convert2bed*

  Numerous internal changes, including giving line functors the ability to resize the destination (write) buffer in mid-stream, along with increased integration with BEDOPS-wide constants. Destination buffer resizing is particularly useful when converting very-long-read BAM files containing numerous D (deletion) operations, such as when used with the new `--split-with-deletions` option.

  - *psl2bed*
    - Migrated storage of PSL conversion state from stack to heap, which helps address segmentation faults on OS X (thanks to rmartson@Biostars for the bug report).
  - *bam2bed* and *sam2bed*
    - Increased thread I/O heap buffer size to reduce likelihood of overflows while parsing reads from Nanopore and PacBio platforms.
    - Added `--split-with-deletions` option to split spliced junctions by `N` and `D` CIGAR operations. The `--split` option now splits only on `N` operations.
    - Added `--reduced` option to print first six columns of BED data to standard output.
  - *gff2bed*
    - Resolved issue parsing GFF input with `##FASTA` directive.

- *sort-bed*
  - The `megarow` build target is available to compile a version of the program that can handle input files with very long rows (4M+ characters). This requires more runtime memory than the default build. The pertinent variables can be modified through the make system without changing source. This is useful for converting ultra-long reads from Nanopore and PacBio sequencing platforms to BED via `bam2bed` / `convert2bed`.

- *starch*
  - Fixed a potential segmentation fault result with `--header` usage.

- Starch C++ API
  - Fixed output from `bedops -u` (`--everything`, or multiset union) on two or more Starch archives, where the remainder string was not being cleared correctly.

- *starch-diff*
  - Improved usage statement to clarify output (cf. Issue 180).

- Clang warnings
  - Resolved compilation warnings for several binaries.

v2.4.26

Released: **March 14, 2017**

- starchstrip
  - New utility to efficiently filter a Starch archive, including or excluding records by specified chromosome names, without doing expensive extraction and recompression. This follows up on internal discussion on the Altius Slack channel.
- *starch-diff*
  - Fixed testing logic in `starch-diff` for certain archives. Thanks to Shane Neph for the report.
- *starchcat*
  - Fixed possible condition where too many variables on the stack can cause a stack overload on some platforms, leading to a fatal segmentation fault. Improved logic for updating v2.1 to v2.2 Starch archives.
- Starch C++ API
  - Patched gzip-backed Starch archive extraction issue. Thanks to Matt Maurano for the bug report.
- *update-sort-bed-migrate-candidates*
  - Added detailed logging via `--debug` option.
  - Added `--bedops-root-dir` option to allow specifying where all BEDOPS binaries are stored. This setting can be overruled on a per-binary basis by adding `--bedextract-path`, `--sort-bed-path`, etc.
  - Added `--non-recursive-search` option to restrict search for BED and Starch candidates to the top-level of the specified parent directory `--parent-dir` option.
  - Further simplification and customization of parameters sent to `update-sort-bed-slurm` and `update-sort-bed-starch-slurm` cluster scripts, as well as logging and variable name improvements to those two scripts.
  - Thanks again to Matt Maurano for ongoing feedback and suggestions on functionality and fixes.
- *gtf2bed*
  - Resolved segmentation fault with certain inputs, in follow-up to this BEDOPS Forum post. Thanks to zebasilio for the report and feedback.

v2.4.25

Released: **February 15, 2017**

- *convert2bed*
  - Patch for RepeatMasker inputs with blank lines that have no spaces. This follows up on Issue 173. Thanks to saketkc for the bug report.
- *update-sort-bed-migrate-candidates*

  The `update-sort-bed-migrate-candidates` utility recursively searches into the specified directory for BED and Starch files which fail a `sort-bed --check-sort` test. Those files which fail this test can have their paths written to a text file for further downstream processing, or the end user can decide to apply

an immediate resort on those files, either locally or via a SLURM-managed cluster. Grateful thanks to Matt Maurano for input and testing.

See `update-sort-bed-migrate-candidates --help` for more information, or review the *sort-bed* documentation.

- *update-sort-bed-starch-slurm*

  This is an adjunct to the `update-sort-bed-slurm` utility, which resorts the provided Starch file and writes a new file. (The `update-sort-bed-slurm` utility only takes in BED files as input and writes BED as output.)

v2.4.24

Released: **February 6, 2017**

- *starch-diff*
  - The `starch-diff` utility compares signatures of two or more v2.2+ Starch archives. This tool tests all chromosomes or one specified chromosome. It returns a zero exit code, if the signature(s) are identical, or a non-zero error exit code, if one or more signature(s) are dissimilar.
- *update-sort-bed-slurm*
  - The `update-sort-bed-slurm` convenience utility provides a parallelized update of the sort order on BED files sorted with pre-v2.4.20 sort-bed, for users with a SLURM job scheduler and associated cluster. See `update-sort-bed-slurm --help` for more details.
- *convert2bed*
  - Patched a memory leak in VCF conversion. Thanks to ehsueh for the bug report.

v2.4.23

Released: **January 30, 2017**

- *unstarch*
  - Fixed bug where missing signature from pre-v2.2 Starch archives would cause a fatal metadata error. Thanks to Shane Neph and Eric Rynes for the bug report.
  - Improved logic reporting signature mismatches when input v2.2 archive lacks signature (*e.g.*, for a v2.2 archive made with `--omit-signature`).
- *starch* and *starchcat*
  - Added `--omit-signature` option to compress without creating a per-chromosome data integrity signature. While this reduces compression time, this eliminates the verification benefits of the data integrity signature.

v2.4.22

Released: **January 25, 2017**

- *convert2bed*
  - Fixed heap corruption in GFF conversion. Thanks to J. Miguel Mendez (ObjectiveTruth) for the bug report.

v2.4.21

Released: **January 23, 2017**

- *bedmap*
  - New `--wmean` operation offers a weighted mean calculation. The "weight" is derived from the proportion of the reference element covered by overlapping map elements: *i.e.*, a map element that covers more of the reference element has its signal given a larger weight or greater impact than another map element with a shorter overlap.
  - Measurement values in `bedmap` did not allow + in the exponent (both − worked and no + for a positive value. Similarly, out in front of the number, + was previously not allowed. Shane Neph posted the report and the fix.
  - The `--min-element` and `--max-element` operations in *bedmap* now process elements in unambiguous order. Former behavior is moved to the operations `--min-element-rand` and `--max-element-rand`, respectively.
  - Fixed issue with use of `--echo-overlap-size` with `--multidelim` (cf. Issue 165). Shane Neph posted the fix. Thanks to Jeff Vierstra for the bug report!
- *bedops*
  - Fixed issue with `--chop` where complement operation could potentially be included. Shane Neph posted the fix.
  - The `bedops --everything` or `bedops -u` (union) operation now writes elements to standard output in unambiguous sort order. If any data are contained in fourth or subsequent fields, a lexicographical sort on that data is applied for resolving order of interval matches.
- *sort-bed*
  - Improved sort times from replacing quicksort (`std::qsort`) with inlined C++ `std::sort`.
  - Sorting of BED input now leads to unambiguous result when two or more elements have the same genomic interval (chromosome name and start and stop position), but different content in remaining columns (ID, score, etc.).

    Formerly, elements with the same genomic interval that have different content in fourth and subsequent columns could be printed in a non-consistent ordering on repeated sorts. A deterministic sort order facilitates the use of data integrity functions on sorted BED and Starch data.
- *starchcluster*
  - A SLURM-ready version of the `starchcluster` script was added to help SLURM job scheduler users with parallelizing the creation of Starch archives.
- Parallel *bam2bed* and *bam2starch*
  - SLURM-ready versions of these scripts were added to help parallelize the conversion of BAM to BED files (`bam2bed_slurm`) or to Starch archives (`bam2starch_slurm`).
- *unstarch*
  - Added `--signature` option to report the Base64-encoded SHA-1 data integrity signature of the Starch-transformed bytes of a specified chromosome, or to report the signature of the metadata string as well as the signatures of all chromosomes, if unspecified.
  - Added `--verify-signature` option to compare the "expected" Base64-encoded SHA-1 data integrity signature stored within the archive's metadata with the "observed" data integrity signature generated from extracting the specified chromosome.

If the observed and expected signatures differ, then this suggests that the chromosome record may be corrupted in some way; `unstarch` will exit with a non-zero error code. If the signatures agree, the archive data should be intact and *unstarch* will exit with a helpful notice and a zero error code.

If no chromosome is specified, `unstarch` will loop through all chromosomes in the archive metadata, comparing observed and expected values for each chromosome record. Upon completion, error and progress messages will be reported to the standard error stream, and `unstarch` will exit with a zero error code, if all signatures match, or a non-zero exit state, if one or more signatures do not agree.

- The output from the `--list` option includes a `signature` column to report the data integrity signature of all Starch-transformed chromosome data.
- The output from the `--list-json` option includes a `signature` key in each chromosome record in the archive metadata, reporting the same information.
- The `--is-starch` option now quits with a non-zero exit code, if the specified input file is not a Starch archive.
- The `--elements-max-string-length` option reports the length of the longest string within the specified chromosome, or the longest string over all chromosomes (if no chromosome name is specified).

- *starch*
  - Added `--report-progress=N` option to (optionally) report compression of the Nth element of the current chromosome to standard error stream.
  - As a chromosome is compressed, the input Starch-transform bytes are continually run through a SHA-1 hash function. The resulting data integrity signature is stored as a Base64-encoded string in the output archive's metadata. Signatures can be compared between and within archives to help better ensure the data integrity of the archive.
  - Fixed `--header` transform bug reported in Issue 161. Thanks to Shane Neph for the bug report!
  - Added chromosome name and "remainder" order tests to `STARCH2_transformHeaderlessBEDInput` and `STARCH2_transformHeaderedBEDInput` functions.

    Compression with `starch` ends with a fatal error, should any of the following comparison tests fail:

    1. The chromosome names are not lexicographically ordered (*e.g.*, `chr1` records coming after `chr2` records indicates the data are not correctly sorted).
    2. The start position of an input element is less than the start position of a previous input element on the same chromosome (*e.g.*, `chr1:1000-1234` coming after `chr1:2000-2345` is not correctly sorted).
    3. The stop positions of two or more input elements are not in ascending order when their start positions are equal (*e.g.*, `chr1:1000-1234` coming after `chr1:1000-2345` is not correctly sorted).
    4. The start and stop positions of two or more input elements are equivalent, and their "remainders" (fourth and subsequent columns) are not in ascending order (*e.g.*, `chr1:1000-1234:id-0` coming after `chr1:1000-1234:id-1` is not correctly sorted).

    If the sort order of the input data is unknown or uncertain, simply use `sort-bed` to generate the correct ordering and pipe the output from that to `starch`, *e.g.* `$ cat elements.bed | sort-bed - | starch - > elements.starch`.

- *starchcat*
  - Added `--report-progress=N` option to (optionally) report compression of the *N* th element of the current chromosome to standard error stream.

  - As in `starch`, at the conclusion of compressing a chromosome made from one or more input Starch archives, the input Starch-transform bytes are continually run through a SHA-1 hash function. The resulting data integrity signature is stored as a Base64-encoded string in the chromosome's entry in the new archive's metadata.
  - As in `starch`, if data should need to be extracted and recompressed, the output is written so that the order is unambiguous: ascending lexicographic ordering on chromosome names, numerical ordering on start positions, the same ordering on stop positions where start positions match, and ascending lexicographic ordering on the remainder of the BED element (fourth and subsequent columns, where present).

- *convert2bed*
  - Improvements in support for BAM/SAM inputs with larger-sized reads, as would come from alignments made from data collected from third-generation sequencers. Simulated read datasets were generated using SimLoRD. Tests have been performed on simulated hg19 data up to 100kb read lengths.

    Improvements allow:

    * conversion of dynamic number of CIGAR operations (up to system memory)
    * conversion of dynamically-sized read fields (up to system memory and inter-thread buffer allocations)

    These patches follow up on bug reports in Issue 157.
  - Improvements in support for VCF inputs, to allow aribtrary-sized fields (up to system memory and inter-thread buffer allocations), which should reduce or eliminate segmentation faults from buffer overruns on fields larger than former stack defaults.
  - Improvements in support for GFF inputs, to allow aribtrary-sized fields (up to system memory and inter-thread buffer allocations), which should reduce or eliminate segmentation faults from buffer overruns on fields larger than former stack defaults.
  - Improvements in support for GTF inputs, to allow aribtrary-sized fields (up to system memory and inter-thread buffer allocations), which should reduce or eliminate segmentation faults from buffer overruns on fields larger than former stack defaults.
- Testing
  - Our use of Travis CI to automate testing of builds now includes Clang on their OS X environment.

v2.4.20

Released: **July 27, 2016**

- *convert2bed*
  - Increased memory allocation for maximum number of per-read CIGAR operations in BAM and SAM conversion to help improve stability. Thanks to Adam Freedman for the report!
  - Improved reliability of gene ID parsing from GTF input, where `gene_id` field may be positioned at start, middle, or end of attributes string, or may be empty. Thanks to blaiseli for the report!

v2.4.19

Released: **May 9, 2016**

- *convert2bed*
  - Fixed bug in BAM and SAM parallel conversion scripts (`*_gnuParallel` and `*_sge`) with inputs containing chromosome names without `chr` prefix. Thanks to Eric Haugen for the bug report!
- Starch C++ API

  - Fixed bug with extraction of bzip2- and gzip-backed archives with all other non-primary Starch tools (all tools except `starch`, `unstarch`, `starchcat`, and `sort-bed`). Thanks to Eric Haugen for the bug report!

v2.4.18

Released: **April 28, 2016**

- *convert2bed*
  - Fixed compile warnings.
  - Fixed bug in BAM and SAM conversion with optional field line overflow. Thanks to Jemma Nelson for the bug report!
- General documentation improvements
  - Updated OS X Installer and Github release instructions
  - Added thank-you to Feng Tian for bug report

v2.4.17

Released: **April 26, 2016**

- *bam2bed* and *sam2bed*
  - Improved parsing of non-split BAM and SAM inputs.
- Docker container build target added for Debian
  - Thanks to Leo Comitale (Poldo) for writing a Makefile target and spec for creating a BEDOPS Docker container for the Debian target.
- Starch C++ API
  - Fixed bug with extraction of bzip2- and gzip-backed archives with all other non-primary Starch tools (all tools except `starch`, `unstarch`, `starchcat`, and `sort-bed`). Thanks to Feng Tian for reports.

v2.4.16

Released: **April 5, 2016**

- *bedmap*
  - Added new `--echo-ref-row-id` option to report reference row ID elements.
- Starch C++ API
  - Fixed bug with extraction of archives made with `starch --gzip` (thanks to Brad Gulko for the bug report and Paul Verhoeven and Peter Weir for compile and testing assistance).
- General improvements
  - Small improvements to build cleanup targets.

v2.4.15

Released: **January 21, 2016**

- Docker container build target added for CentOS 7
  - Thanks to Leo Comitale (Poldo) for writing a Makefile target and spec for creating a BEDOPS Docker container for CentOS 7.
- *convert2bed*
  - Fixed buffer overflows in `convert2bed` to improve conversion reliability for VCF files (thanks to Jared Andrews and Kousik Kundu for bug reports).
- General improvements
  - Improved OS X 10.11 build process.

v2.4.14

Released: **April 21, 2015**

- *convert2bed*
  - Fixed missing `samtools` variable references in cluster conversion scripts (thanks to Brad Gulko for the bug report).
- General suite-wide improvements
  - Fixed exception error message for `stdin` check (thanks to Brad Gulko for the bug report).

v2.4.13

Released: **April 20, 2015**

- *bedops*
  - Resolved issue in using `--ec` with `bedops` when reading from `stdin` (thanks to Brad Gulko for the bug report).
- General suite-wide improvements
  - Addressed inconsistency with constants defined for the suite at the extreme end of the limits we allow for coordinate values (thanks again to Brad Gulko for the report).

v2.4.12

Released: **March 13, 2015**

- *bedops*
  - Checks have been added to determine if an integer argument is a file in the current working directory, before interpreting that argument as an overlap criterion for `-e` and `-n` options.

    To reduce ambiguity, if an integer is used as a file input, `bedops` issues a warning of the interpretation and provides guidance on how to force that value to instead be used as an overlap specification, if desired (thanks to E. Rynes for the pointer).
- *bedmap*

  - Added support for `--prec` / `--sci` with `--min-element` and `--max-element` operations (thanks to E. Rynes for the pointer).

- *bedops* | *bedmap* | *closest-features*
  - Added support for `bash` process substitution/named pipes with specification of `--chrom` and/or `--ec` options (thanks to B. Gulko for the bug report).
  - Fixed code that extracts `gzip`-backed Starch archives from `bedops` and other core tools (thanks again to B. Gulko for the bug report).
- *convert2bed*
  - Switched `matches` and `qSize` fields in order of `psl2bed` output. Refer to documentation for new field order.
  - Added null sentinel to GTF ID value.
  - To help reduce the chance of buffer overflows, the `convert2bed` tool increases the maximum field length from 8191 to 24575 characters to allow parsing of inputs with longer field length, such as very long attributes from mosquito GFF3 data (thanks to T. Karginov for the bug report).

v2.4.11

Released: **February 24, 2015**

- *convert2bed*
  - Fixed bug in `psl2bed` where `matches` column value was truncated by one character. Updated unit tests. Thanks to M. Wirthlin for the bug report.

v2.4.10

Released: **February 23, 2015**

- *starch*
  - In addition to checking chromosome interleaving, the `starch` tool now enforces `sort-bed` sort ordering on BED input and exits with an `EINVAL` POSIX error code if the data are not sorted correctly.
- *convert2bed*
  - Added `--zero-indexed` option to `wig2bed` and `wig2starch` wrappers and `convert2bed` binary, which converts WIG data that are zero-indexed without any coordinate adjustments. This is useful for WIG data sourced from the UCSC Kent tool `bigWigToWig`, where the `bigWig` data can potentially be sourced from 0-indexed BAM- or bedGraph-formatted data.
  - If the WIG input contains any element with a start coordinate of 0, the default use of `wig2bed`, `wig2starch` and `convert2bed` will exit early with an error condition, suggesting the use of `--zero-indexed`.
  - Updated copyright date range of wrapper scripts

v2.4.9

Released: **February 17, 2015**

- *sort-bed*
  - Added support for `--check-sort` to report if input is sorted (or not)

- Starch
  - Improved support for `starch --header`, where header contains tab-delimited fields
- Starch C++ API
  - Fixed bug with `starch --header` functionality, such that BEDOPS core tools (`bedops`, etc.) would be unable to extract correct data from headered Starch archive

v2.4.8

Released: **February 7, 2015**

- Mac OS X packaging
  - Installer signed with productsign to pass OS X Gatekeeper
- Linux packaging
  - SHA1 hashes of each tarball are now part of the BEDOPS Releases description page, going forwards
- Updated copyright dates in source code

v2.4.7

Released: **February 2, 2015**

- *convert2bed* fixes and improvements
  - Fixed `--split` support in `psl2bed` (thanks to Marco A.)
  - Fixed compilation warning regarding comparison of signed and unsigned values
  - Fixed corrupted `psl2bed` test inputs

v2.4.6

Released: **January 30, 2015**

- *convert2bed* fixes and improvements
  - Added support for conversion of the GVF file format, including wrapper scripts and unit tests. Refer to the `gvf2bed` documentation for more information.
  - Fixed bug in string copy of zero-length element attribute for `gff2bed` and `gtf2bed` (GFF and GTF) formats
- General fixes and improvements
  - Fixed possibly corrupt bzip2, Jansson and zlib tarballs (thanks to rekado, Shane N. and Richard S.)
  - Fixed typo in `bedextract` documentation
  - Fixed broken image in *Overview*
  - Removed 19 MB `_build` intermediate result directory (which should improve overall `git clone` time considerably!)

v2.4.5

Released: **January 28, 2015**

- *convert2bed* improvements
  - Addition of RepeatMasker annotation output (`.out`) file conversion support, `rmsk2bed` and `rmsk2starch` wrappers, and unit tests

v2.4.4

Released: **January 25, 2015**

- Documentation improvements
  - Implemented substantial style changes via A Better Sphinx Theme and various customizations. We also include responsive web style elements to help improve browsing on mobile devices.
  - Fixes to typos in conversion and other documents.

v2.4.3

Released: **December 18, 2014**

- Compilation improvements
  - Shane Neph put in a great deal of work to enable parallel builds (*e.g.*, `make -j N` to build various targets in parallel). Depending on the end user's environment, this can speed up compilation time by a factor of 2, 4 or more.
  - Fixed numerous compilation warnings of debug builds of `starch` toolkit under RHEL6/GCC and OS X 10.10.1/LLVM.
- New *bedops* features
  - Added `--chop` and `--stagger` options to "melt" inputs into contiguous or staggered disjoint regions of equivalent size.
  - For less confusion, arguments for `--element-of`, `--chop` and other `bedops` operations that take numerical modifiers no longer require a leading hyphen character. For instance, `--element-of 1` is now equivalent to the former usage of `--element-of -1`.
- New *bedmap* features
  - The `--sweep-all` option reads through the entire map file without early termination and can help deal with `SIGPIPE` errors. It adds to execution time, but the penalty is not as severe as with the use of `--ec`. Using `--ec` alone will enable error checking, but will now no longer read through the entire map file. The `--ec` option can be used in conjunction with `--sweep-all`, with the associated time penalties. (Another method for dealing with issue this is to override how `SIGPIPE` errors are caught by the interpreter (`bash`, Python, *etc.*) and retrapping them or ignoring them. However, it may not a good idea to do this as other situations may arise in production pipelines where it is ideal to trap and handle all I/O errors in a default manner.)
  - New `--echo-ref-size` and `--echo-ref-name` operations report genomic length of reference element, and rename the reference element in `chrom:start-end` (useful for labeling rows for input for `matrix2png` or `R` or other applications).
- *bedextract*
  - Fixed upper bound bug that would cause incorrect output in some cases

- *conversion scripts*
  - Brand new C99 binary called `convert2bed`, which wrapper scripts (`bam2bed`, *etc.*) now call. No more Python version dependencies, and the C-based rewrite offers massive performance improvements over old Python-based scripts.
  - Added `parallel_bam2starch` script, which parallelizes creation of *Starch* archive from very large BAM files in SGE environments.
  - Added bug fix for missing code in *starchcluster.gnu_parallel* script, where the final collation step was missing.
  - The `vcf2bed` script now accepts the `--do-not-split` option, which prints one BED element for all alternate alleles.
- *Starch* archival format and compression/extraction tools
  - Added duplicate- and *nested-element* flags in v2.1 of Starch metadata, which denote if a chromosome contains one or more duplicate and/or nested elements. BED files compressed with `starch` v2.5 or greater, or Starch archives updated with `starchcat` v2.5 or greater will include these values in the archive metadata. The `unstarch` extraction tool offers `--has-duplicate` and `--has-nested` options to retrieve these flag values for a specified chromosome (or for all chromosomes).
  - Added `--is-starch` option to `unstarch` to test if specified input file is a Starch v1 or v2 archive.
  - Added bug fix for compressing BED files with `starch`, where the archive would not include the last element of the BED input, if the BED input lacked a trailing newline. The compression tools now include a routine for capturing the last line, if there is no newline.
- Documentation improvements
  - Remade some image assets throughout the documents to support Retina-grade displays

v2.4.2

Released: **April 10, 2014**

- *conversion scripts*
  - Added support for `sort-bed --tmpdir` option to conversion scripts, to allow specification of alternative temporary directory for sorted results when used in conjunction with `--max-mem` option.
  - Added support for GFF3 files which include a FASTA directive in `gff2bed` and `gff2starch` (thanks to Keith Hughitt).
  - Extended support for Python-based conversion scripts to support use with Python v2.6.2 and forwards, except for `sam2bed` and `sam2starch`, which still require Python v2.7 or greater (and under Python3).
  - Fixed `--insertions` option in `vcf2bed` to now report a single-base BED element (thanks to Matt Maurano).

v2.4.1

Released: **February 26, 2014**

- *bedmap*
  - Added `--fraction-both` and `--exact` (`--fraction-both 1`) to list of compatible overlap options with `--faster`.
  - Added 5% performance improvement with `bedmap` operations without `--faster`.

  - Fixed scenario that can yield incorrect results (cf. Issue 43).

- *sort-bed*
  - Added `--tmpdir` option to allow specification of an alternative temporary directory, when used in conjunction with `--max-mem` option. This is useful if the host operating system's standard temporary directory (*e.g.*, `/tmp` on Linux or OS X) does not have sufficient space to hold intermediate results.
- All *conversion scripts*
  - Improvements to error handling in Python-based conversion scripts, in the case where no input is specified.
  - Fixed typos in `gff2bed` and `psl2bed` documentation (cf. commit a091e18).
- OS X compilation improvements
  - We have completed changes to the OS X build process for the remaining half of the BEDOPS binaries, which now allows direct, full compilation with Clang/LLVM (part of the Apple Xcode distribution).

    All OS X BEDOPS binaries now use Apple's system-level C++ library, instead of GNU's `libstdc++`. It is no longer required (or recommended) to use GNU `gcc` to compile BEDOPS on OS X.

    Compilation is faster and simpler, and we can reduce the size and complexity of Mac OS X builds and installer packages. By using Apple's C++ library, we also eliminate the likelihood of missing library errors.

    In the longer term, this gets us closer to moving BEDOPS to using the CMake build system, to further abstract and simplify the build process.
- Cleaned up various compilation warnings found with `clang` / `clang++` and GCC kits.

v2.4.0

Released: **January 9, 2014**

- *bedmap*
  - Added new `--echo-map-size` and `--echo-overlap-size` options to calculate sizes of mapped elements and overlaps between mapped and reference elements.
  - Improved performance for all `--echo-map-*` operations.
  - Updated documentation.
- Major enhancements and fixes to *sort-bed*:
  - Improved performance.
  - Fixed memory leak.
  - Added support for millions of distinct chromosomes.
  - Improved internal estimation of memory usage with `--max-mem` option.
- Added support for compilation on Cygwin (64-bit). Refer to the *installation documentation* for build instructions.
- *starchcat*
  - Fixed embarassing buffer overflow condition that caused segmentation faults on Ubuntu 13.
- All *conversion scripts*
  - Python-based scripts no longer use temporary files, which reduces file I/O and improves performance. This change also reduces the need for large amounts of free space in a user's `/tmp` folder, particularly relevant for users converting multi-GB BAM files.

  - We now test for ability to locate `starch`, `sort-bed`, `wig2bed_bin` and `samtools` in user environment, quitting with the appropriate error state if the dependencies cannot be found.
  - Improved documentation. In particular, we have added descriptive tables to each script's documentation page which describe how columns map from original data input to BED output.
  - *bam2bed* and *sam2bed*
    * Added `--custom-tags <value>` command-line option to support a comma-separated list of custom tags (cf. Biostars discussion), *i.e.*, tags which are not part of the original SAMtools specification.
    * Added `--keep-header` option to preserve header and metadata as BED elements that use `_header` as the chromosome name. This now makes these conversion scripts fully "non-lossy".
  - *vcf2bed*
    * Added new `--snvs`, `--insertions` and `--deletions` options that filter VCF variants into three separate subcategories.
    * Added `--keep-header` option to preserve header and metadata as BED elements that use `_header` as the chromosome name. This now makes these conversion scripts fully "non-lossy".
  - *gff2bed*
    * Added `--keep-header` option to preserve header and metadata as BED elements that use `_header` as the chromosome name. This now makes these conversion scripts fully "non-lossy".
  - *psl2bed*
    * Added `--keep-header` option to preserve header and metadata as BED elements that use `_header` as the chromosome name. This now makes these conversion scripts fully "non-lossy".
  - *wig2bed*
    * Added `--keep-header` option to `wig2bed` binary and `wig2bed` / `wig2starch` wrapper scripts, to preserve header and metadata as BED elements that use `_header` as the chromosome name. This now makes these conversion scripts fully "non-lossy".
- Added OS X uninstaller project to allow end user to more easily remove BEDOPS tools from this platform.
- Cleaned up various compilation warnings found with `clang` / `clang++` and GCC kits.

v2.3.0

Released: **October 2, 2013**

- Migration of BEDOPS code and documentation from Google Code to Github.
  - Due to changes with Google Code hosting policies at the end of the year, we have decided to change our process for distributing code, packages and documentation. While most of the work is done, we appreciate feedback on any problems you may encounter. Please email us at bedops@stamlab.org with details.
  - Migration to Github should facilitate requests for code by those who are familiar with `git` and want to fork our project to submit pull requests.
- *bedops*
  - General `--ec` performance improvements.
- *bedmap*

  - Adds support for the new `--skip-unmapped` option, which filters out reference elements which do not have mapped elements associated with them. See the end of the *score operations* section of the *bedmap* documentation for more detail.
  - General `--ec` performance improvements.

- *starch*
  - Fixed bug with `starch` where zero-byte BED input (*i.e.*, an "empty set") created a truncated and unusable archive. We now put in a "dummy" chromosome for zero-byte input, which `unstarch` can now unpack.

    This should simplify error handling with certain pipelines, specifically where set or other BEDOPS operations yield an "empty set" BED file that is subsequently compressed with `starch`.

- *unstarch*
  - Can now unpack zero-byte ("empty set") compressed `starch` archive (see above).
  - Changed `unstarch --list` option to print to `stdout` stream (this was previously sent to `stderr`).

- *starch* metadata library
  - Fixed array overflow bug with BEDOPS tools that take *starch* archives as inputs, which affected use of archives as inputs to `closest-features`, `bedops` and `bedmap`.

- All *conversion scripts*
  - Python scripts require v2.7+ or greater.
  - Improved (more "Pythonic") error code handling.
  - Disabled support for `--max-mem` sort parameter until *sort-bed* issue is resolved. Scripts will continue to sort, but they will be limited to available system memory. If you are processing files larger than system memory, please contact us at bedops@stamlab.org for details of a temporary workaround.

- *gff2bed* conversion script
  - Resolved `IndexError` exceptions by fixing header support, bringing script in line with v1.21 GFF3 spec.

- *bam2bed* and *sam2bed* conversion scripts
  - Rewritten `bam2*` and `sam2*` scripts from `bash` into Python (v2.7+ support).
  - Improved BAM and SAM input validation against the v1.4 SAM spec.
  - New `--split` option prints reads with `N` CIGAR operations as separated BED elements.
  - New `--all-reads` option prints all reads, mapped and unmapped.

- *bedextract*
  - Fixed `stdin` bug with `bedextract`.

- New documentation via readthedocs.org.
  - Documentation is now part of the BEDOPS distribution, instead of being a separate download.
  - We use readthedocs.org to host indexed and searchable HTML.
  - PDF and eBook documents are also available for download.
  - Documentation is refreshed and simplified, with new installation and compilation guides.

- OS X compilation improvements

  - We have made changes to the OS X build process for half of the BEDOPS binaries, which allows direct compilation with Clang/LLVM (part of the Apple Xcode distribution). Those binaries now use Apple's system-level C++ library, instead of GNU's `libstdc++`.

    This change means that we require Mac OS X 10.7 ("Lion") or greater—we do not support 10.6 at this time.

    Compilation is faster and simpler, and we can reduce the size and complexity of Mac OS X builds and installer packages. By using Apple's C++ library, we also reduce the likelihood of missing library errors. When this process is completed for the remaining binaries, it will no longer be necessary to install GCC 4.7+ (by way of MacPorts or other package managers) in order to build BEDOPS on OS X, nor will we have to bundle `libstdc++` with the installer.

v2.2.0b

- Fixed bug with OS X installer's post-installation scripts.

v2.2.0

Released: **May 22, 2013**

- Updated packages
  - Precompiled packages are now available for Linux (32- and 64-bit) and Mac OS X 10.6-10.8 (32- and 64-bit) hosts.
- *Starch v2 test suite*
  - We have added a test suite for the Starch archive toolkit with the source download. Test inputs include randomized BED data generated from chromosome and bounds data stored on UCSC servers as well as static FIMO search results. Tests put `starch`, `unstarch` and `starchcat` through various usage scenarios. Please refer to the Starch-specific Makefiles and the test target and subfolder's *README* doc for more information.
- *starchcat*
  - Resolves bug with `--gzip` option, allowing updates of `gzip` -backed v1.2 and v1.5 archives to the *v2 Starch format* (either `bzip2` - or `gzip` -backed).
- *unstarch*
  - Resolves bug with extraction of *Starch* archive made from BED files with four or more columns. A condition where the total length of additional columns exceeds a certain number of characters would result in extracted data in those columns being cut off. As an example, this could affect Starch archives made from the raw, uncut output of GTF- and GFF- *conversion scripts*.
- *conversion scripts*
  - We have partially reverted `wig2bed`, providing a Bash shell wrapper to the original C binary. This preserves consistency of command-line options across the conversion suite, while making use of the C binary to recover performance lost from the Python-based v2.1 revision of `wig2bed` (which at this time is no longer supported). (Thanks to Matt Maurano for reporting this issue.)

v2.1.1

Released: **May 3, 2013**

- *bedmap*

  - Major performance improvements made in v2.1.1, such that current `bedmap` now operates as fast or faster than the v1.2.5 version of `bedmap`!

- *bedops*
  - Resolves bug with `--partition` option.
- *conversion scripts*
  - All v2.1.0 Python-based scripts now include fix for `SIGPIPE` handling, such that use of `head` or other common UNIX utilities to process buffered standard output no longer yields `IOError` exceptions. (Thanks to Matt Maurano for reporting this bug.)
- 32-bit Linux binary support
  - Pre-built Linux binaries are now available for end users with 32-bit workstations.

Other issues fixed:

- Jansson tarball no longer includes already-compiled libraries that could potentially interfere with 32-bit builds.
- Minor changes to conversion script test suite to exit with useful error code on successful completion of test.

v2.1.0

Released: **April 22, 2013**

- *bedops*
  - New `--partition` operator efficiently generates disjoint segments made from genomic boundaries of all overlapping inputs.
- *conversion scripts*
  - All scripts now use `sort-bed` behind the scenes to output sorted BED output, ready for use with BEDOPS utilities. It is no longer necessary to pipe data to or otherwise post-process converted data with `sort-bed`.
  - New `psl2bed` conversion script, converting PSL-formatted UCSC BLAT output to BED.
  - New `wig2bed` conversion script written in Python.
  - New `*2starch` *conversion scripts* offered for all `*2bed` scripts, which output Starch v2 archives.
- *closest-features*
  - Replaced `--shortest` option name with `--closest`, for clarity. (Old scripts which use `--shortest` will continue to work with the deprecated option name for now. We advise editing pipelines, as needed.)
- *starch*
  - Improved error checking for interleaved records. This also makes use of `*2starch` conversion scripts with the `--do-not-sort` option safer.
- Improved Mac OS X support
  - New Mac OS X package installer makes installation of BEDOPS binaries and scripts very easy for OS X 10.6 - 10.8 hosts.
  - Installer resolves fatal library errors seen by some end users of older OS X BEDOPS releases.

v2.0.0b

Released: **February 19, 2013**

- Added `starchcluster` script variant which supports task distribution with GNU Parallel.
- Fixed minor problem with `bam2bed` and `sam2bed` conversion scripts.

v2.0.0a

Released: **February 7, 2013**

- *bedmap*
  - Takes in Starch-formatted archives as input, as well as raw BED (i.e., it is no longer required to extract a Starch archive to an intermediate, temporary file or named pipe before applying operations).
  - New `--chrom` operator jumps to and operates on information for specified chromosome only.
  - New `--echo-map-id-uniq` operator lists unique IDs from overlapping mapping elements.
  - New `--max-element` and `--min-element` operators return the highest or lowest scoring overlapping map element.
- *bedops*
  - Takes in Starch-formatted archives as input, as well as raw BED.
  - New `--chrom` operator jumps to and operates on information for specified chromosome only.
- *closest-features*
  - Takes in Starch-formatted archives as input, as well as raw BED.
  - New `--chrom` operator jumps to and operates on information for specified chromosome only.
- *sort-bed* and `bbms`
  - New `--max-mem` option to limit system memory on large BED inputs.
  - Incorporated `bbms` functionality into `sort-bed` with use of `--max-mem` operator.
- *starch*, *starchcat* and *unstarch*
  - New metadata enhancements to Starch-format archival and extraction, including: `--note`, `--elements`, `--bases`, `--bases-uniq`, `--list-chromosomes`, `--archive-timestamp`, `--archive-type` and `--archive-version` (see `--help` to `starch`, `starchcat` and `unstarch` binaries, or view the documentation for these applications for more detail).
  - Adds 20-35% performance boost to creating Starch archives with `starch` utility.
  - New documentation with technical overview of the Starch format specification.
- *conversion scripts*
  - New `gtf2bed` conversion script, converting GTF (v2.2) to BED.
- Scripts are now part of main download; it is no longer necessary to download the BEDOPS companion separately.

v1.2.5b

Released: **January 14, 2013**

- Adds support for Apple 32- and 64-bit Intel hardware running OS X 10.5 through 10.8.
- Adds `README` for companion download.
- Removes some obsolete code.

v1.2.5

Released: **October 13, 2012**

- Fixed unusual bug with `unstarch`, where an extra (and incorrect) line of BED data can potentially be extracted from an archive.
- Updated companion download with updated `bam2bed` and `sam2bed` conversion scripts to address 0-indexing error with previous revisions.

v1.2.3

Released: **August 17, 2012**

- Added `--indicator` option to `bedmap`.
- Assorted changes to conversion scripts and associated companion download.

2.4 Usage examples

The following examples demonstrate the use of BEDOPS in analyzing genomic data. Here, we provide source code and snippets of data to demonstrate "real-world" examples based on daily usage of these tools in the Stamatoyannopoulos lab.

2.4.1 Visualizing the relationship of SNPs and generic genomic features

We want to visualize how genome-wide association study single nucleotide repeats (GWAS SNPs) relate to other genomic features—in this case, these features are DNaseI-hypersensitive sites (DHSs). We could, instead, look at methylated regions, CpG islands, coding sequence or other genomic features. Normally, we might do this for all sites in the genome, but to reduce the file sizes we only look at a subset of data here and we have taken a subset of the real data for the purposes of demonstration.

Roughly speaking, we considered two classes of SNPs: those which are prostate-related (associated with PSA and prostate cancer) and some not (height). We have some BED files with positions of DNaseI-hypersensitive sites for various tissues: two from prostate (LNCaP and PrEC), the rest from other tissues (CACO2, HEPG2, K562, MCF7).

We will use BEDOPS tools to generate per-tissue DHS counts associated with our SNPs, using matrix2png to visualize results as a heatmap.

BEDOPS tools in use

For this example, we use *sort-bed* to sort the input SNP data, and *bedmap* to count the number of single-base or greater overlaps between a SNP and a tissue-specific DHS. A modified version of this script uses loops and other shell features.

Script

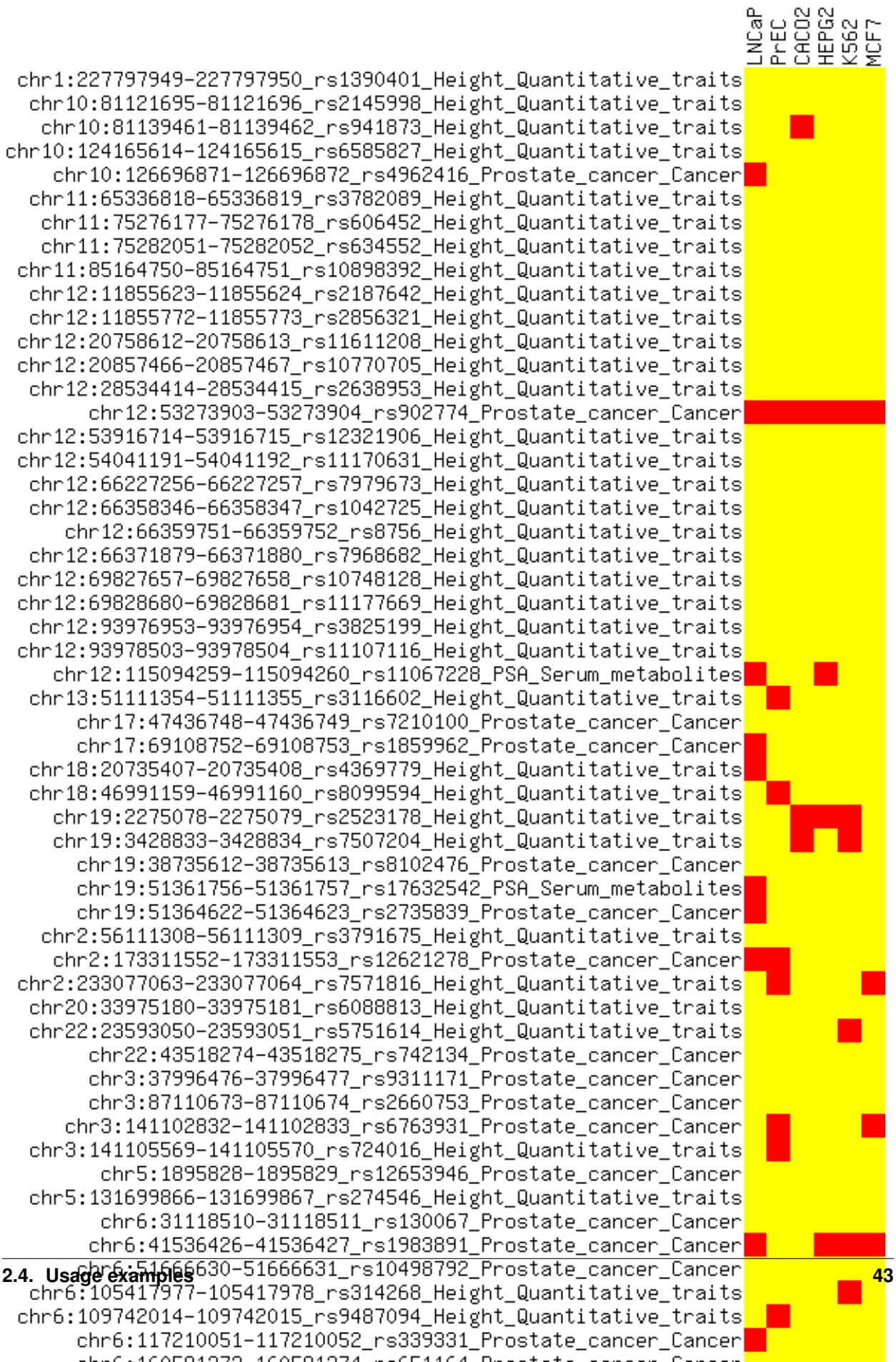
```
#!/bin/tcsh -efx

sort-bed GWAS_SNPs.bed > GWAS_SNPs.sorted.bed
bedmap --ec --delim "\t" --bp-ovr 1 --echo --count GWAS_SNPs.sorted.bed LNCaP_DHS.bed␣
↪> SNP_DHS_matrix.bed

# add PrEC DHS overlap counts to matrix
bedmap --ec --delim "\t" --bp-ovr 1 --count GWAS_SNPs.sorted.bed PrEC_DHS.bed >␣
↪counts.txt
paste SNP_DHS_matrix.bed counts.txt > new_SNP_DHS_matrix.bed
mv new_SNP_DHS_matrix.bed SNP_DHS_matrix.bed

# add CACO2 DHS overlap counts to matrix
bedmap --ec --delim "\t" --bp-ovr 1 --count GWAS_SNPs.sorted.bed CACO2_DHS.bed >␣
↪counts.txt
paste SNP_DHS_matrix.bed counts.txt > new_SNP_DHS_matrix.bed
mv new_SNP_DHS_matrix.bed SNP_DHS_matrix.bed

# add HEPG2 DHS overlap counts to matrix
bedmap --ec --delim "\t" --bp-ovr 1 --count GWAS_SNPs.sorted.bed HEPG2_DHS.bed >␣
↪counts.txt
paste SNP_DHS_matrix.bed counts.txt > new_SNP_DHS_matrix.bed
mv new_SNP_DHS_matrix.bed SNP_DHS_matrix.bed

# add K562 DHS overlap counts to matrix
bedmap --ec --delim "\t" --bp-ovr 1 --count GWAS_SNPs.sorted.bed K562_DHS.bed >␣
↪counts.txt
paste SNP_DHS_matrix.bed counts.txt > new_SNP_DHS_matrix.bed
mv new_SNP_DHS_matrix.bed SNP_DHS_matrix.bed

# add MCF7 DHS overlap counts to matrix
bedmap --ec --delim "\t" --bp-ovr 1 --count GWAS_SNPs.sorted.bed MCF7_DHS.bed >␣
↪counts.txt
paste SNP_DHS_matrix.bed counts.txt > new_SNP_DHS_matrix.bed
mv new_SNP_DHS_matrix.bed SNP_DHS_matrix.bed

# cleanup and sort by disease trait
rm counts.txt
sort -k5d SNP_DHS_matrix.bed > new_SNP_DHS_matrix.bed
mv new_SNP_DHS_matrix.bed SNP_DHS_matrix.bed

# condense data fields into matrix2png form
awk '{print $1":"$2"-"$3"_"$4"_"$5"_"$6"\t"$7"\t"$8"\t"$9"\t"$10"\t"$11"\t"$12}' SNP_
↪DHS_matrix.bed > new_SNP_DHS_matrix.txt
mv new_SNP_DHS_matrix.txt SNP_DHS_matrix.txt

# add header
echo -e "0\tLNCaP\tPrEC\tCACO\tHEPG2\tK562\tMCF7" | cat - SNP_DHS_matrix.txt > new_
↪SNP_DHS_matrix.txt
mv new_SNP_DHS_matrix.txt SNP_DHS_matrix.txt

# make heatmap
matrix2png -r -c -g -size 16:16 -mincolor yellow -midcolor black -maxcolor red -data␣
↪SNP_DHS_matrix.txt > SNP_DHS_matrix.png
```

Discussion

Each use of *bedmap* is identical: the options `--ec --delim "\t" --bp-ovr 1 --count` add several settings:

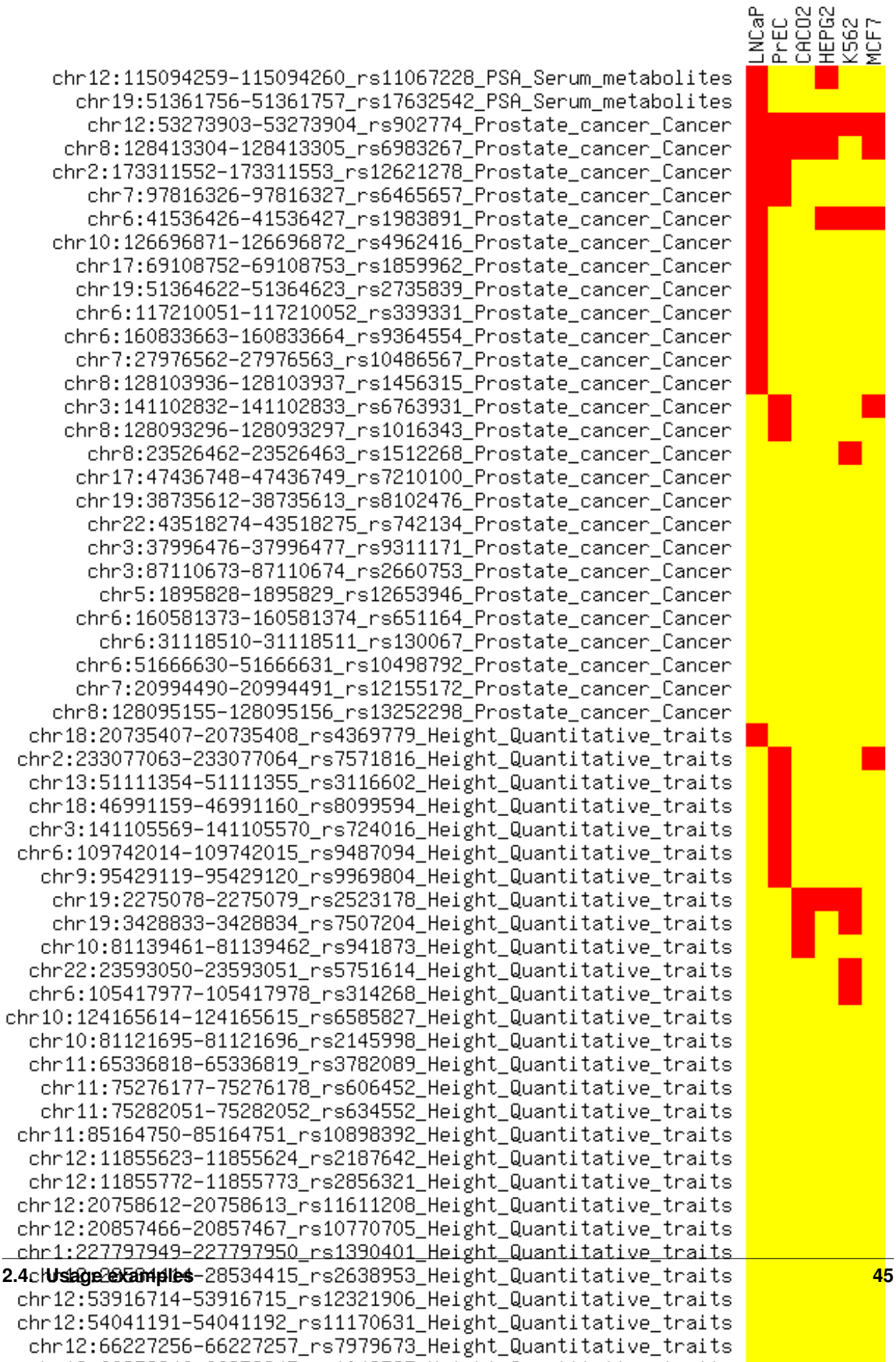
- Error checking/correction (`--ec`)
- Use of the tab character as a custom field delimiter (`--delim "\t"`) to make results easier to parse with `awk` further downstream
- Manual specification of a single base-pair criteria for overlap (`--bp-ovr 1`, although this is the default)
- Using `--count`, counting the number of mapping elements (DHSs) which overlap a reference instance (a given SNP)

These results are calculated for each of the seven cell types and collated into matrix form to run through matrix2png. We show it here to give an idea of what kind of data *bedmap* generates, to help create these quantitative visualizations:


LNCaP
PrEC
CACO2
HEPG2
K562
MCF7
chr1:227797949-227797950_rs1390401_Height_Quantitative_traits
chr10:81121695-81121696_rs2145998_Height_Quantitative_traits
chr10:81139461-81139462_rs941873_Height_Quantitative_traits
chr10:124165614-124165615_rs6585827_Height_Quantitative_traits
chr10:126696871-126696872_rs4962416_Prostate_cancer_Cancer
chr11:65336818-65336819_rs3782089_Height_Quantitative_traits
chr11:75276177-75276178_rs606452_Height_Quantitative_traits
chr11:75282051-75282052_rs634552_Height_Quantitative_traits
chr11:85164750-85164751_rs10898392_Height_Quantitative_traits
chr12:11855623-11855624_rs2187642_Height_Quantitative_traits
chr12:11855772-11855773_rs2856321_Height_Quantitative_traits
chr12:20758612-20758613_rs11611208_Height_Quantitative_traits
chr12:20857466-20857467_rs10770705_Height_Quantitative_traits
chr12:28534414-28534415_rs2638953_Height_Quantitative_traits
chr12:53273903-53273904_rs902774_Prostate_cancer_Cancer
chr12:53916714-53916715_rs12321906_Height_Quantitative_traits
chr12:54041191-54041192_rs11170631_Height_Quantitative_traits
chr12:66227256-66227257_rs7979673_Height_Quantitative_traits
chr12:66358346-66358347_rs1042725_Height_Quantitative_traits
chr12:66359751-66359752_rs8756_Height_Quantitative_traits
chr12:66371879-66371880_rs7968682_Height_Quantitative_traits
chr12:69827657-69827658_rs10748128_Height_Quantitative_traits
chr12:69828680-69828681_rs11177669_Height_Quantitative_traits
chr12:93976953-93976954_rs3825199_Height_Quantitative_traits
chr12:93978503-93978504_rs11107116_Height_Quantitative_traits
chr12:115094259-115094260_rs11067228_PSA_Serum_metabolites
chr13:51111354-51111355_rs3116602_Height_Quantitative_traits
chr17:47436748-47436749_rs7210100_Prostate_cancer_Cancer
chr17:69108752-69108753_rs1859962_Prostate_cancer_Cancer
chr18:20735407-20735408_rs4369779_Height_Quantitative_traits
chr18:46991159-46991160_rs8099594_Height_Quantitative_traits
chr19:2275078-2275079_rs2523178_Height_Quantitative_traits
chr19:3428833-3428834_rs7507204_Height_Quantitative_traits
chr19:38735612-38735613_rs8102476_Prostate_cancer_Cancer
chr19:51361756-51361757_rs17632542_PSA_Serum_metabolites
chr19:51364622-51364623_rs2735839_Prostate_cancer_Cancer
chr2:56111308-56111309_rs3791675_Height_Quantitative_traits
chr2:173311552-173311553_rs12621278_Prostate_cancer_Cancer
chr2:233077063-233077064_rs7571816_Height_Quantitative_traits
chr20:33975180-33975181_rs6088813_Height_Quantitative_traits
chr22:23593050-23593051_rs5751614_Height_Quantitative_traits
chr22:43518274-43518275_rs742134_Prostate_cancer_Cancer
chr3:37996476-37996477_rs9311171_Prostate_cancer_Cancer
chr3:87110673-87110674_rs2660753_Prostate_cancer_Cancer
chr3:141102832-141102833_rs6763931_Prostate_cancer_Cancer
chr3:141105569-141105570_rs724016_Height_Quantitative_traits
chr5:1895828-1895829_rs12653946_Prostate_cancer_Cancer
chr5:131699866-131699867_rs274546_Height_Quantitative_traits
chr6:31118510-31118511_rs130067_Prostate_cancer_Cancer
chr6:41536426-41536427_rs1983891_Prostate_cancer_Cancer
chr6:51666630-51666631_rs10498792_Prostate_cancer_Cancer
chr6:105417977-105417978_rs314268_Height_Quantitative_traits
chr6:109742014-109742015_rs9487094_Height_Quantitative_traits
chr6:117210051-117210052_rs339331_Prostate_cancer_Cancer

Rows are presented in *sort-bed* order. Cells in red show greatest relative number of counts, while yellow shows the least. Examining this heatmap, DHS elements appear to associate with prostate disease-related GWAS SNPs.

To make this clearer, here is the same result, with rows sorted by disease name:


LNCaP
PrEC
CACO2
HEPG2
K562
MCF7
chr12:115094259-115094260_rs11067228_PSA_Serum_metabolites
chr19:51361756-51361757_rs17632542_PSA_Serum_metabolites
chr12:53273903-53273904_rs902774_Prostate_cancer_Cancer
chr8:128413304-128413305_rs6983267_Prostate_cancer_Cancer
chr2:173311552-173311553_rs12621278_Prostate_cancer_Cancer
chr7:97816326-97816327_rs6465657_Prostate_cancer_Cancer
chr6:41536426-41536427_rs1983891_Prostate_cancer_Cancer
chr10:126696871-126696872_rs4962416_Prostate_cancer_Cancer
chr17:69108752-69108753_rs1859962_Prostate_cancer_Cancer
chr19:51364622-51364623_rs2735839_Prostate_cancer_Cancer
chr6:117210051-117210052_rs339331_Prostate_cancer_Cancer
chr6:160833663-160833664_rs9364554_Prostate_cancer_Cancer
chr7:27976562-27976563_rs10486567_Prostate_cancer_Cancer
chr8:128103936-128103937_rs1456315_Prostate_cancer_Cancer
chr3:141102832-141102833_rs6763931_Prostate_cancer_Cancer
chr8:128093296-128093297_rs1016343_Prostate_cancer_Cancer
chr8:23526462-23526463_rs1512268_Prostate_cancer_Cancer
chr17:47436748-47436749_rs7210100_Prostate_cancer_Cancer
chr19:38735612-38735613_rs8102476_Prostate_cancer_Cancer
chr22:43518274-43518275_rs742134_Prostate_cancer_Cancer
chr3:37996476-37996477_rs9311171_Prostate_cancer_Cancer
chr3:87110673-87110674_rs2660753_Prostate_cancer_Cancer
chr5:1895828-1895829_rs12653946_Prostate_cancer_Cancer
chr6:160581373-160581374_rs651164_Prostate_cancer_Cancer
chr6:31118510-31118511_rs130067_Prostate_cancer_Cancer
chr6:51666630-51666631_rs10498792_Prostate_cancer_Cancer
chr7:20994490-20994491_rs12155172_Prostate_cancer_Cancer
chr8:128095155-128095156_rs13252298_Prostate_cancer_Cancer
chr18:20735407-20735408_rs4369779_Height_Quantitative_traits
chr2:233077063-233077064_rs7571816_Height_Quantitative_traits
chr13:51111354-51111355_rs3116602_Height_Quantitative_traits
chr18:46991159-46991160_rs8099594_Height_Quantitative_traits
chr3:141105569-141105570_rs724016_Height_Quantitative_traits
chr6:109742014-109742015_rs9487094_Height_Quantitative_traits
chr9:95429119-95429120_rs9969804_Height_Quantitative_traits
chr19:2275078-2275079_rs2523178_Height_Quantitative_traits
chr19:3428833-3428834_rs7507204_Height_Quantitative_traits
chr10:81139461-81139462_rs941873_Height_Quantitative_traits
chr22:23593050-23593051_rs5751614_Height_Quantitative_traits
chr6:105417977-105417978_rs314268_Height_Quantitative_traits
chr10:124165614-124165615_rs6585827_Height_Quantitative_traits
chr10:81121695-81121696_rs2145998_Height_Quantitative_traits
chr11:65336818-65336819_rs3782089_Height_Quantitative_traits
chr11:75276177-75276178_rs606452_Height_Quantitative_traits
chr11:75282051-75282052_rs634552_Height_Quantitative_traits
chr11:85164750-85164751_rs10898392_Height_Quantitative_traits
chr12:11855623-11855624_rs2187642_Height_Quantitative_traits
chr12:11855772-11855773_rs2856321_Height_Quantitative_traits
chr12:20758612-20758613_rs11611208_Height_Quantitative_traits
chr12:20857466-20857467_rs10770705_Height_Quantitative_traits
chr1:227797949-227797950_rs1390401_Height_Quantitative_traits
chr12:28534414-28534415_rs2638953_Height_Quantitative_traits
chr12:53916714-53916715_rs12321906_Height_Quantitative_traits
chr12:54041191-54041192_rs11170631_Height_Quantitative_traits
chr12:66227256-66227257_rs7979673_Height_Quantitative_traits

While there are some DHSs associated with non-disease SNPs, the majority accumulate with the prostate SNPs.

Downloads

- The `example` script, after modification to use loops and other shell features.
- `Data` for this example are contained in a tarball (use `tar -xzf` to extract files).

The *bedmap* tool can operate directly on Starch-formatted archives. Alternatively, use the *unstarch* tool to decompress Starch data files to sorted BED format.

Note that these are not the full datasets that went into the original research, but snippets that should otherwise demonstrate the disease-DHS association phenomenon and the use of parts of the BEDOPS toolset.

2.4.2 Collapsing multiple BED files into a master list by signal

Given a list of five-column UCSC BED files, where scores are kept in the fifth column, we want to build a "master list" of non-overlapping elements from all the inputs. Elements that initially overlap are ranked by score, and the highest scoring element is added to the master list.

BEDOPS tools in use

In the following example, we want to merge hotspot peaks for five fetal adrenal tissues, picking the highest scoring element where there are overlapping peaks. We'll use a mix of *bedmap* and its `--max-element` operation with *bedops* set operations to accomplish this.

Script

```
#!/bin/bash
# author : Bob Thurman

beds=(fAdrenal-DS12528.dhs.bed
      fAdrenal-DS15123.dhs.bed
      fAdrenal-DS17319.dhs.bed
      fAdrenal-DS17677.dhs.bed
      fAdrenal-DS20343.dhs.bed)

out=fAdrenal.master.merge.bed

tmpd=/tmp/tmp$$
mkdir -p $tmpd

## First, union all the peaks together into a single file.
bedlist=""
for bed in ${beds[*]}
do
    bedlist="$bedlist $bed"
done

bedops -u $bedlist > $tmpd/tmp.bed

## The master list is constructed iteratively.  For each pass through
## the loop, elements not yet in the master list are merged into
```

(continues on next page)

(continued from previous page)

```
## non-overlapping intervals that span the union (this is just bedops
## -m).  Then for each merged interval, an original element of highest
## score within the interval is selected to go in the master list.
## Anything that overlaps the selected element is thrown out, and the
## process then repeats.
iters=1
solns=""
stop=0
while [ $stop == 0 ]
do
    echo "merge steps..."

    ## Condense the union into merged intervals. This klugey bit
    ## before and after the merging is because we don't want to merge
    ## regions that are simply adjacent but not overlapping
    bedops -m --range 0:-1 $tmpd/tmp.bed \
        | bedops -u --range 0:1 - \
        > $tmpd/tmpm.bed

    ## Grab the element with the highest score among all elements forming each␣
↪interval.
    ## If multiple elements tie for the highest score, just grab one of them.
    ## Result is the current master list.  Probably don't need to sort, but do it␣
↪anyway
    ## to be safe since we're not using --echo with bedmap call.
    bedmap --max-element $tmpd/tmpm.bed $tmpd/tmp.bed \
        | sort-bed - \
        > $tmpd/$iters.bed
    solns="$solns $tmpd/$iters.bed"
    echo "Adding `awk 'END { print NR }' $tmpd/$iters.bed` elements"

    ## Are there any elements that don't overlap the current master
    ## list?  If so, add those in, and repeat.  If not, we're done.
    bedops -n 1 $tmpd/tmp.bed $tmpd/$iters.bed \
       > $tmpd/tmp2.bed

    mv $tmpd/tmp2.bed $tmpd/tmp.bed

    if [ ! -s $tmpd/tmp.bed ]
    then
        stop=1
    fi

    ((iters++))
done

## final solution
bedops -u $solns \
   > $out

## Clean up
rm -r $tmpd

exit 0
```

Discussion

A broad array of human cell tissue hotspot data for testing this example are available for public download from the UCSC Genome Browser:

- http://genome.ucsc.edu/cgi-bin/hgFileUi?db=hg19&g=wgEncodeUwDnase

This includes hotspot data for `DS12528`, `DS15123`, `DS17319`, `DS17677` and `DS20343` lines.

2.4.3 Measuring the frequency of signed distances between SNPs and nearest DHSes

In this example, we would like to find the **signed** distance between a single nucleotide repeat and the DNase-hypersensitive site nearest to it, as measured in base pairs (bp).

BEDOPS tools in use

To find nearest elements, we will use *closest-features* with the `--dist`, `--closest`, and `--no-ref` options.

Script

SNPs are in a BED-formatted file called `SNPs.bed` sorted lexicographically with *sort-bed*. The DNase-hypersensitive sites are stored in a sorted BED-formatted file called `DHSs.bed`. These two files are available in the *Downloads* section.

```
# author : Eric Rynes
closest-features --dist --closest --no-ref SNPs.bed DHSs.bed \
    | cut -f2 -d '|' \
    | grep -w -F -v -e "NA" \
    > answer.bed
```

Discussion

The `--dist` option returns signed distances between input elements and reference elements, `--closest` chooses the single closest element, and `--no-ref` keeps SNP coordinates from being printed out.

The output from *closest-features* contains coordinates and the signed distance to the closest DHS, separated by the pipe (`|`) character. Such output might look something like this:

```
chr1    2513240 2513390 MCV-11  97.201400|25
```

This type of result is chopped up with the standard UNIX utility `cut` to get at the distances to the closest elements. Finally, we use `grep -v` to throw out any non-distance, denoted by `NA`. This can occur if there exists some chromosome in the SNP dataset that does not exist in the DHSs.

Thus, for every SNP, we have a corresponding distance to nearest DHS. As an example, from this data we could build a histogram showing the frequencies of distances-to-nearest-DHS.

Downloads

- `SNP` elements
- `DNase-hypersensitive` elements

The *closest-features* tool can operate directly on Starch-formatted archives. Alternatively, use the *unstarch* tool to decompress Starch data files to sorted BED format.

2.4.4 Finding the subset of SNPs within DHSes

In this example, we would like to identify the set of SNPs that are within a DHS, printing out both the SNP element *and* the DHS it is contained within.

BEDOPS tools in use

We use *bedmap* to answer this question, as it traverses a *reference* BED file (in this example, SNPs), and identifies overlapping elements from the *mapping* BED file (in this example, DHSs).

Script

SNPs are in a BED-formatted file called `SNPs.bed` sorted lexicographically with *sort-bed*. The DNase-hypersensitive sites are stored in a sorted BED-formatted file called `DHSs.bed`. These two files are available in the *Downloads* section.

```
bedmap --skip-unmapped --echo --echo-map SNPs.bed DHSs.bed \
  > subsetOfSNPsWithinAssociatedDHS.bed
```

Discussion

The output of this *bedmap* statement might look something like this:

```
chr1    10799576    10799577    rs12.4.378  Systolic_blood_pressure␣
↪Cardiovascular|chr1 10799460    10799610    MCV-1   9.18063
```

The output is delimited by pipe symbols (`|`), showing the reference element (SNP) and the mapped element (DHS).

If multiple elements are mapped onto a single reference element, the mapped elements are further separated by semi-colons, by default.

Downloads

- `SNP` elements
- `DNase-hypersensitive` elements

The *bedmap* tool can operate directly on Starch-formatted archives. Alternatively, use the *unstarch* tool to decompress Starch data files to sorted BED format.

2.4.5 Smoothing raw tag count data across the genome

In this example, we generate smoothed density signal by binning the genome into 20 bp intervals and counting the number of non-paired-end tag reads falling within 75 bp of each interval. A simple follow-on script marks up results to wig or bigWig format for loading into a track of a local UCSC Genome Browser.

BEDOPS tools in use

For this script, we use *bam2bed* to convert a BAM file to BED, then we use *bedmap* to run a sliding density window over input genomic regions. Finally *starch* compresses the results.

Script

```
#!/bin/tcsh -ef
# author : Richard Sandstrom

if ( $#argv != 5 ) then
  printf "Wrong number of arguments\n"
  printf "<bam-file> <out-file> <window-size> <step-size> <chromosome-file>\n"
  printf "  where <chromosome-file> contains whole chromosome BED items for the\n"
  printf "  genome, e.g., sort-bed formatted output from the UCSC hg19.chromInfo␣
↪table.\n"
  exit -1
endif

# BAM file
set inBam = $argv[1]
# resulting density file
set outDensity = $argv[2]
# +/- window for counting read 5' ends
set window = $argv[3]
# step size across genome
set binI = $argv[4]
# chromosome file for organism of interest
set chromsfile = $argv[5]

set outDir = $outDensity:h
mkdir -p $outDir

set tmpDir = /tmp/`whoami`/scratch/$$
if ( -d $tmpDir ) then
  rm -rf $tmdDir
endif
mkdir -p $tmpDir

# clip tags to single 5' end base
bam2bed < $inBam \
    | awk '{if($6=="+"){s=$2; e=$2+1}else{s=$3-1; e=$3}print $1"\t"s"\t"e}' \
    | sort-bed --max-mem 2G - \
   >! $tmpDir/tags.bed

# create genome-wide bins and count how many tags fall within range of each
awk -v binI=$binI -v win=$window \
      '{ \
         i = 0; \
         for(i = $2; i <= $3-binI; i += binI) { print $1"\t"i"\t"i + binI } \
         # end of chrome may include a bin of size < binI \
         if ( i < $3 ) { print $1"\t"i"\t"$3; } \
      }' $chromsfile \
    | bedmap --faster --range $window --echo --count --delim "\t" - $tmpDir/tags.bed \
    | starch - \
   >! $outDensity
```

(continues on next page)

(continued from previous page)

```
rm -rf $tmpDir

exit 0
```

2.4.6 Efficiently creating Starch-formatted archives with a cluster

In this example, we demonstrate how to use *bedextract* and *starchcat* to efficiently generate Starch-formatted archives from BED datasets.

BEDOPS tools in use

For this script, we use *bedextract* to quickly build a list of chromosomes in an input BED dataset and extract records for each chromosome to separate files. We then use *starch* to compress each per-chromosome file and *starchcat* to concatenate per-chromosome Starch archives into one file.

Script

Three versions of the `starchcluster` script are included with the source and package distributions of BEDOPS (see *Installation* for more detail).

One version makes use of an Oracle Grid Engine (or Sun Grid Engine) cluster environment to distribute per-chromosome tasks, another script uses GNU Parallel to split the workload over cores or processors on the local host. Finally, we include a version that implements a SLURM -capable script.

Discussion

The overview that follows applies to the Grid Engine-based version of the *starchcluster* script. However, the general algorithm is identical for the Grid Engine-, GNU Parallel-, and SLURM-based compression scripts.

Splitting BED files

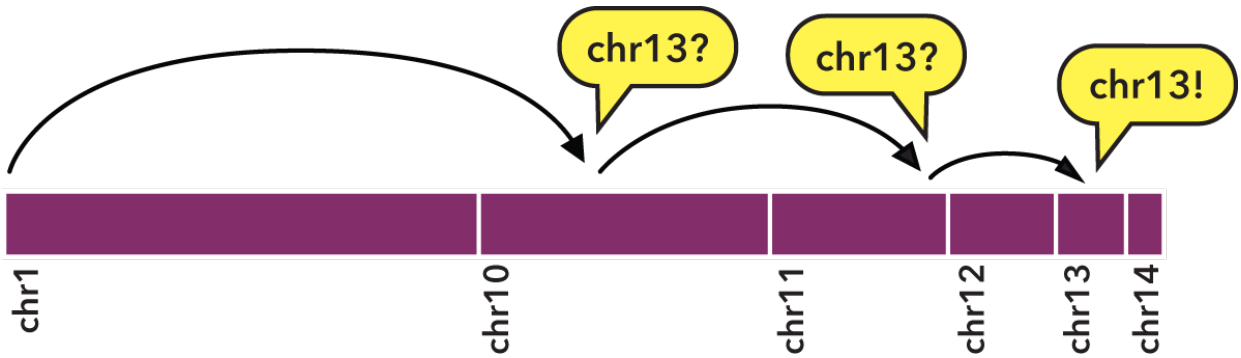
Whole-genome analyses are often "embarassingly parallel", in that per-chromosome computations can be placed onto separate work nodes of a computational cluster, with results collated at the end in "map-reduce" fashion.

If we want to filter any BED file to retrieve elements from a specific chromosome (say, to compress a BED file, one chromosome at a time), to arrange this kind of analysis, one trivial—but very slow—way to do this involves sequentially walking line by line through the file to parse and test each element. This can take a while to do.

However, just as BEDOPS tools use the information in *sorted data* to apply efficient set and statistical operations, we can use this same information to jump quickly through our data of interest.

Specifically, sorting allows us to perform a binary search:

1. We jump to the middle byte of the BED file, stream to the nearest element, then parse and test the chromosome name.
2. Either we have a match, or we jump to the middle of the remaining left or right half (decided by dictionary order), parse and test again.
3. We repeat steps 1 and 2 until we have matches that define the bounds of the target chromosome.



To indicate the kind of speed gain that the *bedextract* tool provides, in local testing, a naïve listing of chromosomes from a 36 GB BED input using UNIX `cut` and `uniq` utilities took approximately 20 minutes to complete on a typical Core 2 Duo-based Linux workstation. Retrieval of the same chromosome listing with `bedextract --list-chr` took only 2 seconds (cache flushed—no cheating!).

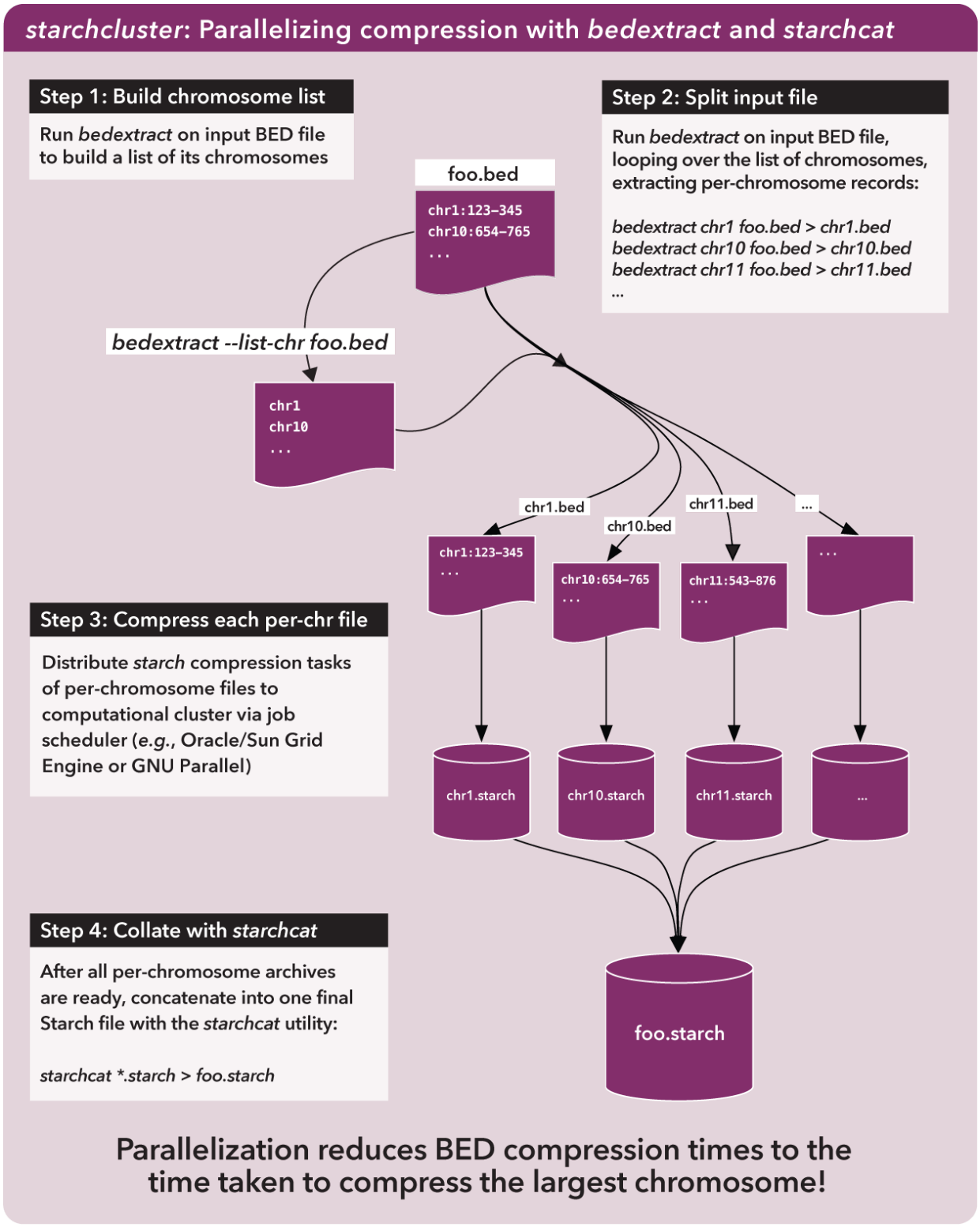
Compressing BED subsets

Now we can very quickly demarcate where chromosomes start and stop in a BED file, we can apply *starch* on those subsets on separate cluster nodes.

Stitching together compressed sets

Once we have per-chromosome Starch-formatted archives, we need some way to put them all together into one archive. This is where *starchcat* comes in, taking all the per-chromosome archives as inputs and creating a new archive as output.

The big picture view is like this:



As the figure notes, the compression time for a very large BED file is reduced roughly to the time taken to compress the largest chromosome in the original file. Parallelization of this process is an order of magnitude faster than compressing chromosomes in serial.

2.4.7 Working with many input files at once with `bedops` and `bedmap`

BEDOPS is designed to work with as many input files at once as you need, either through the *bedops* program, or through a combined use of that program with others in the suite.

Discussion

Say we have five input BED files (`A`, `B`, `C`, `D`, `E`), and we need to identify those regions where any two (or more) of the input files (`{A,B}`, `{A,C}`, `{A,D}`, `{A,E}`, `{B,C}`, ...) overlap reciprocally by 30% or more.

One concrete application may be where we have multiple biological replicates, and we take any repeatable result (in two or more inputs, in this case) as true signal. Similarly, we might be interested in a problem like this if we have multiple related (or even unrelated) cell type samples and we want to be confident in peak calls for DNaseI sequencing of ChIP-seq experiments.

These sorts of problems often have efficient solutions in BEDOPS. Here, the solution is independent of how many inputs we start with, what overlap criteria we use, and whether the requirement calls for two or more files of overlap (or whether it is 4 or more files in the overlap, or 9, or whatever).

Consider a case study of one such problem that utilizes both *bedops* and *bedmap* together to create an efficient solution:

```
$ bedops -u file1.bed file2.bed ... fileN.bed \
    | bedmap --echo --echo-map-id-uniq --fraction-both 0.5 - \
    | awk -F"|" '(split($2, a, ";") > 1)' \
    > answer.bed
```

Here, we pass in as many files as we have to *bedops*. The requirement of elements overlapping reciprocally is met by using `--fraction-both`, and the requirement that overlapping elements must come from two or more (distinct) files is satisfied by checking how many elements there are via the `--echo-map-id-uniq` operator.

The requirements for `file1.bed` through `fileN.bed` are that each is properly *sorted* (as expected for any BEDOPS input) and that their respective fourth-column ID fields identify the file. For example:

```
$ head -2 file1.bed
chr1 1   50   1  anything-else
chr1 230 400  1  whatever-you-like

$ head -2 file2.bed
chr1 23  78  2  other-fields
chr1 56  98  2  5.678  +  peak-2
```

As a nice side-effect, `answer.bed` will show from which file each entry originated. If we don't want that extra information, we simply cut it out:

```
cut -f1-3,5- answer.bed >! my-final-answer.bed
```

There is also a column that shows exactly which files are part of the per-row intersection. If we don't want that information, then we just cut that:

```
cut -f1 -d'|' my-final-answer.bed
```

While this is just one example of how the tools can be used together to answer complicated questions efficiently, it demonstrates why it is worthwhile to learn about the relatively few core programs in BEDOPS.

If we look at what is required to answer this kind of question using other tool suites, we will quickly find that solutions do not scale to the number of files, nor with the requirement that overlaps must come from k or more distinct input files. Even in the simplest case of just requiring the regions overlap in 2 of n inputs, we must build on the order of $n^2/2$

intermediate files (and sweep through the n original inputs n^2 times as well). If our requirement is 3 of n inputs, the polynomials increase accordingly.

The solution with BEDOPS is far more efficient than this and requires no intermediate results.

2.5 Performance

In this document, we compare the performance of our set operations and compression utilities with common alternatives. In-house performance measures include speed, memory usage, and compression efficiency on a dual-core machine with 18 GB of virtual memory. Additionally, we report independently-generated performance statistics collected by a research group that has recently released a similar analysis toolkit.

2.5.1 Test environment and data

Timed results were derived using actual running times (also known as wall-clock times), averaged over 3 runs. All timed tests were performed using a single 64-bit Linux machine with a dual-core 3 GHz Intel Xeon processor, 8 GB of physical RAM, and 18 GB of total virtual memory. All caches were purged in between sequential program runs to remove hardware biases.

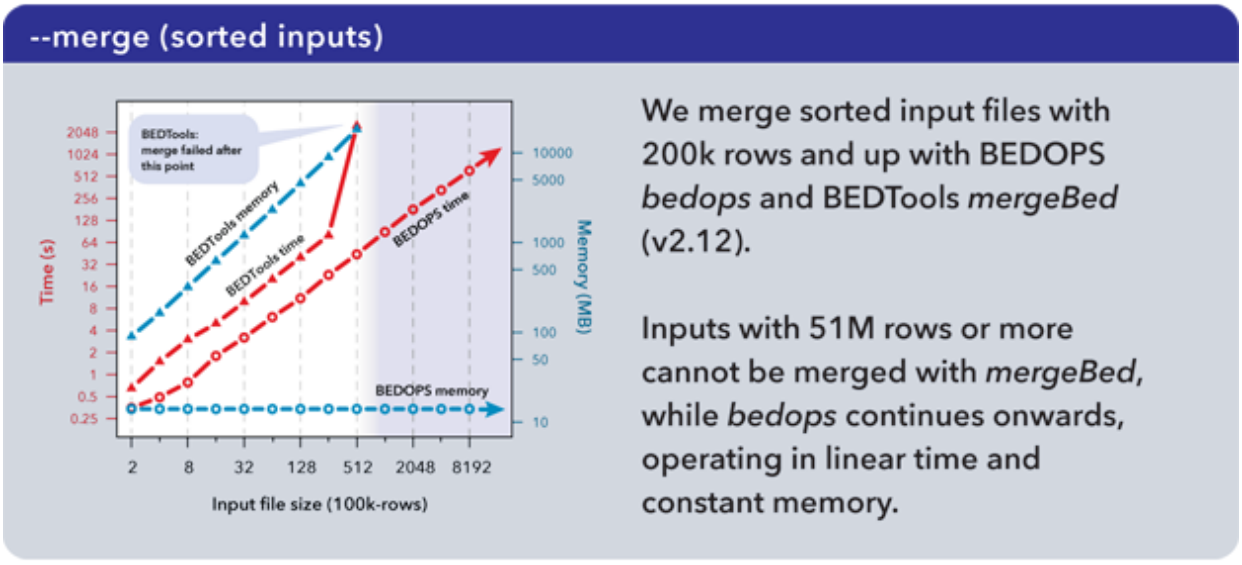
Random subsamples of phyloP conservation for the human genome were used as inputs for testing whenever the full phyloP results were not used. The full phyloP results were downloaded from UCSC.

2.5.2 Set operations with `bedops`

In this section, we provide time and memory measurements of various *bedops* operations against analogous BEDTools utilities.

Direct merge (sorted)

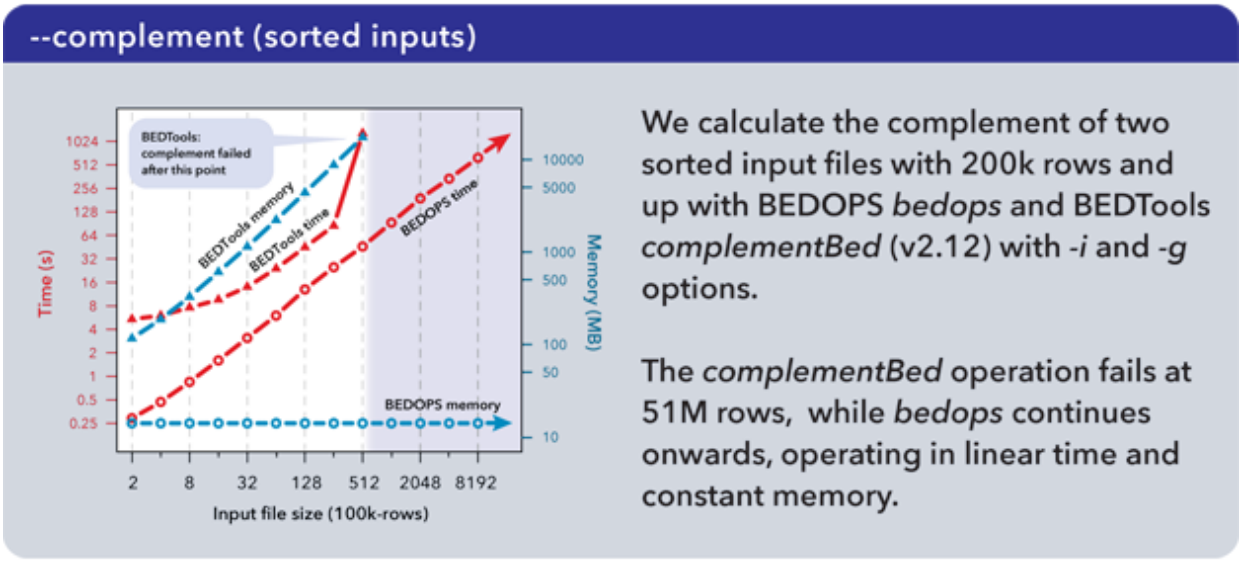
The performance of the `mergeBed` program (with the `-i` option) from the BEDTools suite (v2.12.0) was compared with that of the `--merge` option of our *bedops* utility.

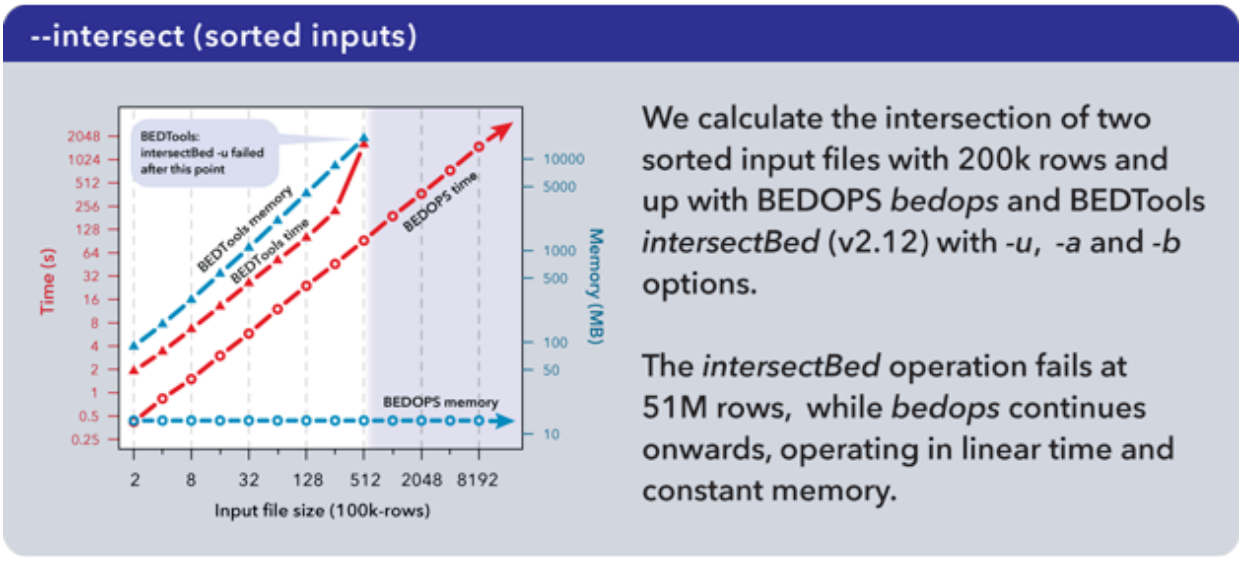


As measured, the `mergeBed` program loads all data from a file into memory and creates an index before computing results, incurring longer run times and higher memory costs that can lead to failures. The *bedops* utility minimizes memory consumption by retaining only the information required to compute the next line of output.

Complement and intersection

The `complementBed` (with `-i` and `-g` options) and `intersectBed` (with `-u`, `-a`, and `-b` options) programs from the BEDTools suite (v2.12.0) also were compared to our *bedops* program.





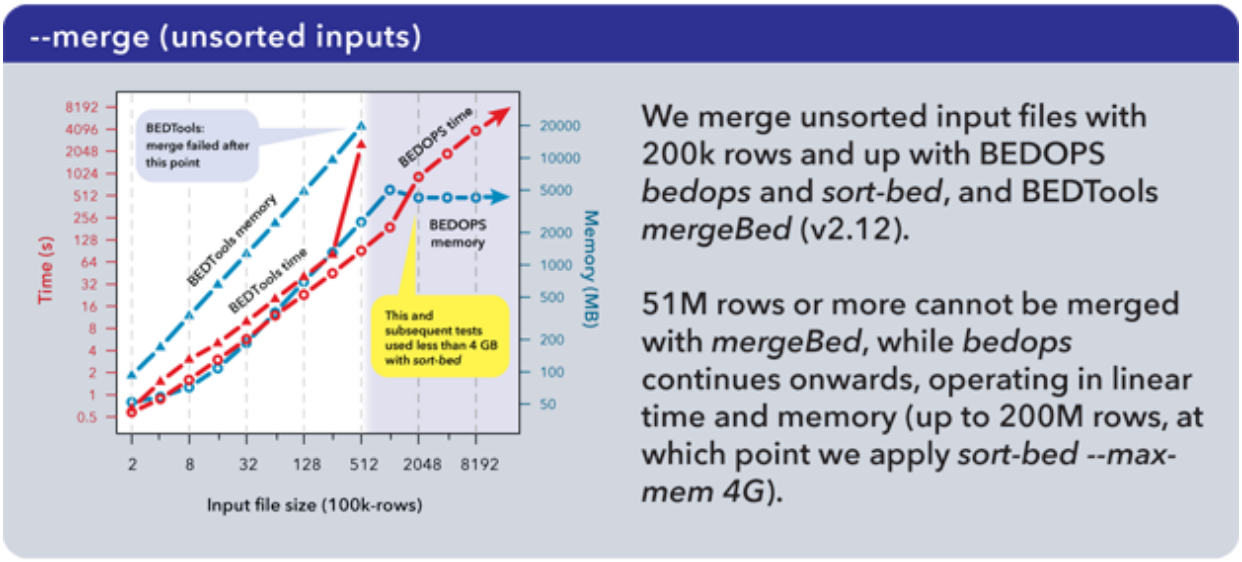
Both BEDTools programs were unable to complete operations after 51M elements with the allocated 18 GB of memory. The *bedops* program continued operating on the full dataset.

Important: It is our understanding that the BEDTools' `intersectBed` program was modified to accept (optionally) sorted data for improved performance some time after these results were published.

A *more recent study* suggests `bedops --intersect` still offers better memory and running time performance characteristics than recent versions of BEDTools.

Direct merge (unsorted)

In typical pipelines, where utilities are chained together to perform more complex operations, the performance and scalability gaps between BEDOPS and competitive tool suites widen. We show here the use of *sort-bed* on unsorted BED input, piping it to BEDOPS tools:



Time performance of *bedops* stays under that of `mergeBed` (BEDTools v2.12), while continuing past the point where `mergeBed` fails. Memory limitations of the system are easily overcome by using the `--max-mem` operator with *sort-bed*, allowing the `--merge` operation to continue unimpeded even with ever-larger unsorted BED inputs.

Discussion

The *bedops* utility performs a wide range of set operations (merge, intersect, union, symmetric difference, and so forth). As with all main utilities in BEDOPS, the program requires *sorted* inputs and creates sorted results on output. As such, sorting is, at most, a *one-time cost* to operate on data any number of times in the most efficient way. Also, as shown in an *independent study*, BEDOPS also sorts data more efficiently than other tools. Further, our utility can sort BED inputs of any size.

Another important feature of *bedops* that separates it from the competition is its ability to work with *any number of inputs* at once. Every operation (union, difference, intersection, and so forth) accepts an arbitrary number of inputs, and each input can be of any size.

2.5.3 Compression characteristics of `starch`

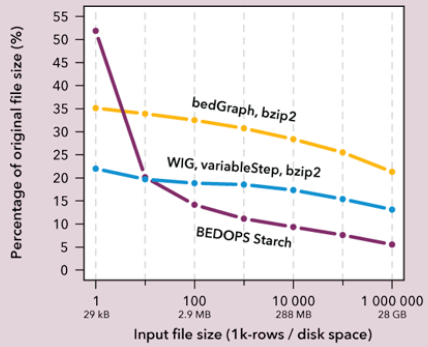
The *starch* utility offers high-quality BED compression into a format with a smaller footprint than common alternatives. The format is designed to help manage data bloat in this genomic era. Further, the format actually enables improved access times to the vast majority of datasets, as compared with raw (uncompressed) and naively-compressed data.

Here, we provide two measures of this format's utility: comparing the compression efficiency of the `bzip2`-backed Starch format against common, "naive" `bzip2`-compression of UCSC BedGraph and WIG forms of BED data, and by comparing the time required to extract the records for any one chromosome from these formats as well as from a raw (uncompressed) BED file.

Compression efficiency

After just 10K rows (roughly 300 kB of raw BED data storing phyloP conservation scores), compression into the Starch format begins to consistently outperform `bzip2` compression of the same data stored in either variable-step WIG or UCSC BedGraph formats.

Compression efficiency of *starch*



After just 10k rows (roughly 300 kB of raw BED data storing phyloP conservation scores), compression into the Starch format begins to consistently outperform bzip2 compression of the same data stored in either variable-step WIG or UCSC BedGraph formats.

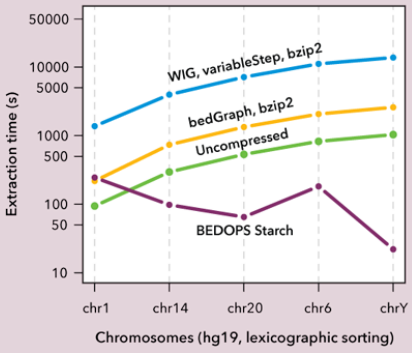
For very large, raw BED datasets, the Starch format stores the original data in approximately 5% of the original input size. These improved compression results generalize to compressed versions of the fixed-step WIG format, as well.

For very large raw BED datasets, the Starch format stores the original data in approximately 5% of the original input size. These improved compression results generalize to compressed versions of the fixed-step WIG format, as well. For more information, refer to the Supplemental Data in our Bioinformatics paper.

Extraction time

Data were sorted per sort-bed with chromosomes in lexicographical order. Extractions by chromosomes were significantly faster in general with the Starch format, even over raw (sequentially-processed) BED inputs:

Extraction time with *unstarch*



Under the assumption that chromosomes create natural partitions of genomic data, random access to data within Starch archives uses a chromosome-indexing scheme. Extractions by chromosomes are significantly faster in general with the Starch format, even over uncompressed BED inputs.

This design improves data processing times for analyses focused on specific areas of whole-genome datasets, or where whole-genome experiments can be reduced to per-chromosome analyses that can be split and distributed on a computational cluster.

Under the assumption that chromosomes create very natural partitions of the data, the Starch format was designed using a chromosome-indexing scheme. This mechanism for random access further helps to improve data processing times

within a clustered environment. Again, for more information, refer to the Supplemental Data in our Bioinformatics paper.

Important: Our *bedextract* program similarly makes it possible to extract data quickly by chromosome in any properly sorted BED file. However, for large (or many) data sets, deep compression has serious benefit. In our lab, more than 99% of all files are not touched (even) on a monthly basis—and new results are generated every day. Why would we want to keep all of that data in fully-bloated BED form? The workhorse programs of BEDOPS accept inputs in Starch format directly, just as they do raw BED files, to help manage 'big data'.

2.5.4 Independent testing

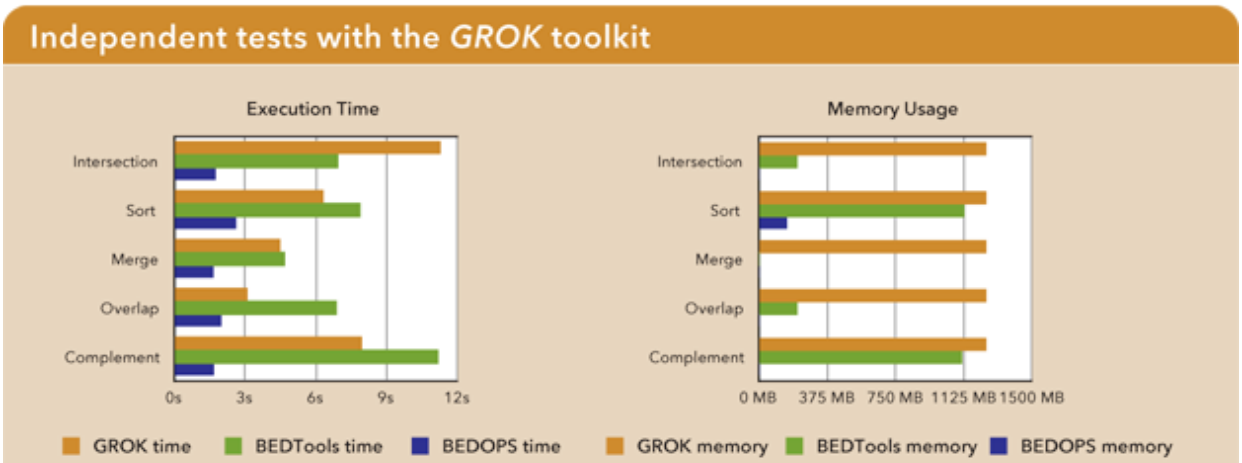
Genomic Region Operation Kit (GROK)

Ovaska, et al. independently developed a genomic analysis toolkit called Genomic Region Operation Kit (GROK), which is described in more detail in their publication in IEEE/ACM Transactions on Computational Biology and Bioinformatics.

In it, they compare the performance characteristics of their GROK toolkit with their analogs in the BEDTools and BEDOPS suites, which they summarize as follows:

> **Results**
>
> Results of the benchmark analyses are shown in Table VII. GROK and BEDTools perform at comparable levels for speed and memory efficiency. In this benchmark BEDOPS is the fastest and least memory consuming method, which was expected due to performance optimized implementation of its operations [9]. The optimized performance of BEDOPS, however, entails stronger assumptions for the input than GROK and BEDTools, in particular the requirement for pre-sorting the input BED files.

Operational input was a 14 MB BED file containing annotations of human gene and exon coordinates, totaling ~423k records. We summarize the results of operations on that input here:



Remember that with BEDOPS, sorting is, at most, a *one-time cost* to operate on data any number of times in the most efficient way. Since the programs in BEDOPS produce sorted outputs, you never need to sort results before using them in downstream analyses.

2.5.5 Worst-case memory performance

Non-sorting utilities operate efficiently with large inputs by keeping memory overhead low. The worst-case design scenario, however, causes the *bedops* or *bedmap* programs to load all data from a single chromosome from a single input file into memory. For *bedops*, the worst-case scenario applies only to the `--element-of` and `--not-element-of` options.

Fortunately, worst-case situations are conceptually easy to understand, and their underlying questions often require no windowing logic to answer, so simpler approaches can sometimes be used. Conceptually, any summary analysis over an entire chromosome triggers the worst-case scenario. For example, to determine the number of sequencing tags mapped to a given chromosome, *bedmap* loads all tag data for that one chromosome into memory, whereas a one-line `awk` statement can provide the answer with minimal memory overhead.

We note that the worst case memory performance of non-sorting BEDOPS utilities still improves upon the best case performance of current alternatives.

2.6 Reference

2.6.1 Set operations

bedops

`bedops` is a core tool for finding relationships between two or more genomic datasets.

This is an important category of problems to solve. As examples, one might want to:

- Know how much overlap exists between the elements of two datasets, to quantitatively establish the degree to which they are similar.
- Merge or filter elements. For example, retrieving non-overlapping, "unique" elements from multiple BED files.
- Split elements from multiple BED files into disjoint subsets.

The *bedops* program offers several Boolean set and multiset operations, including union, subset, and difference, to assist investigators with answering these types of questions.

Importantly, *bedops* handles any number of any-size inputs at once when computing results in order to maximize efficiency. This use case has *serious practical consequences* for many genomic studies.

One can also use *bedops* to symmetrically or asymmetrically pad coordinates.

Inputs and outputs

Input

The *bedops* program reads *sorted* BED data and BEDOPS *Starch-formatted* archives as input.

Finally, *bedops* requires specification of a set operation (and, optionally, may include modifier options).

Support for common headers (including UCSC track headers) is offered through the `--header` option. Headers are stripped from output.

Output

The *bedops* program returns *sorted* BED results to standard output. This output can be redirected to a file or piped to other utilities.

Usage

The *bedops* program takes sorted BED-formatted data as input, either from a file or streamed from standard input. It will process any number of input files in parallel.

If your data are unsorted, use BEDOPS *sort-bed* to prepare data for *bedops*. You only need to sort once, as all BEDOPS tools read and write sorted BED data.

Because memory usage is very low, one can use sorted inputs of any size. Processing times generally follow a simple linear relationship with input sizes (*e.g.*, as the input size doubles, the processing time doubles accordingly).

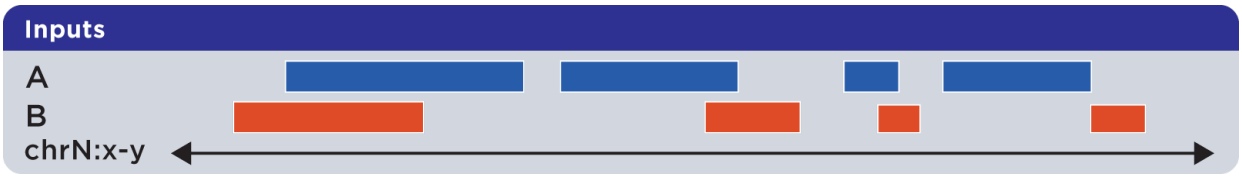
The `--help` option describes the set operation and other options available to the end user:

```
bedops
  citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
  version:  2.4.37 (typical)
  authors:  Shane Neph & Scott Kuehn

      USAGE: bedops [process-flags] <operation> <File(s)>*

          Every input file must be sorted per the sort-bed utility.
          Each operation requires a minimum number of files as shown below.
            There is no fixed maximum number of files that may be used.
          Input files must have at least the first 3 columns of the BED specification.
          The program accepts BED and Starch file formats.
          May use '-' for a file to indicate reading from standard input (BED format␣
↪only).

      Process Flags:
          --chrom <chromosome> Process data for given <chromosome> only.
          --ec                 Error check input files (slower).
          --header             Accept headers (VCF, GFF, SAM, BED, WIG) in any input␣
↪file.
          --help               Print this message and exit successfully.
          --help-<operation>   Detailed help on <operation>.
                                 An example is --help-c or --help-complement
          --range L:R          Add 'L' bp to all start coordinates and 'R' bp to end
                                 coordinates. Either value may be + or - to grow or
                                 shrink regions.  With the -e/-n operations, the first
                                 (reference) file is not padded, unlike all other␣
↪files.
          --range S            Pad or shink input file(s) coordinates symmetrically␣
↪by S.
                                 This is shorthand for: --range -S:S.
          --version            Print program information.

      Operations: (choose one of)
          -c, --complement [-L] File1 [File]*
          -d, --difference ReferenceFile File2 [File]*
          -e, --element-of [number% | number] ReferenceFile File2 [File]*
                 by default, -e 100% is used.  'bedops -e 1' is also popular.
          -i, --intersect File1 File2 [File]*
          -m, --merge File1 [File]*
          -n, --not-element-of [number% | number] ReferenceFile File2 [File]*
                 by default, -n 100% is used.  'bedops -n 1' is also popular.
          -p, --partition File1 [File]*
          -s, --symmdiff File1 File2 [File]*
          -u, --everything File1 [File]*
          -w, --chop [bp] [--stagger [bp]] [-x] File1 [File]*
                 by default, -w 1 is used with no staggering.

      Example: bedops --range 10 -u file1.bed
      NOTE: Only operations -e|n|u preserve all columns (no flattening)
```

Note: Extended help is available for all operations in *bedops*. For example, the `--help-symmdiff` option in *bedops* gives detailed information on the `--symmdiff` operation.

Operations

To demonstrate the various operations in *bedops*, we start with two simple datasets `A` and `B`, containing genomic elements on generic chromsome `chrN`:

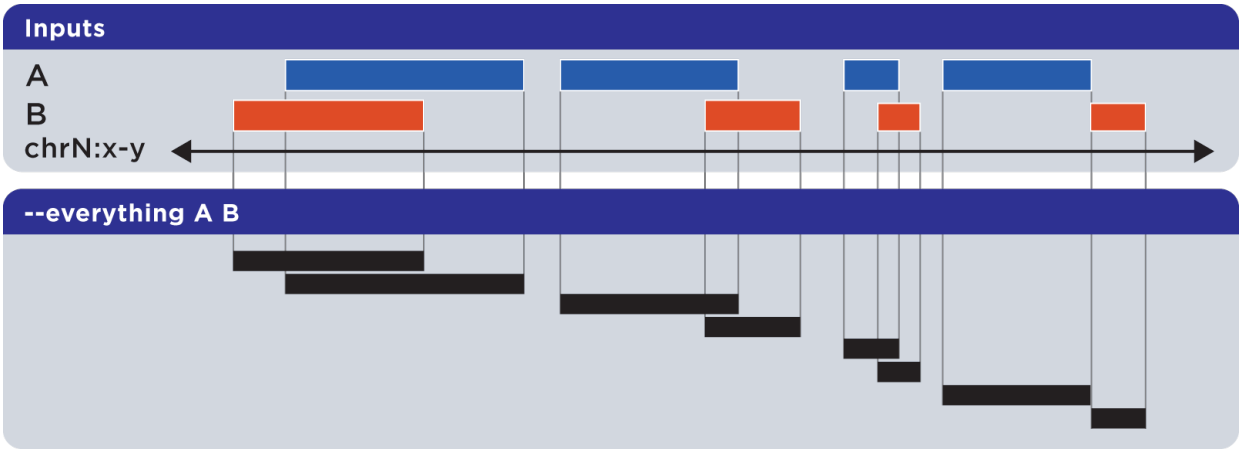


These datasets can be *sorted* BED or *Starch-formatted* files or streams.

Note: The *bedops* tool can operate on two or more multiple inputs, but we show here the results of operations acting on just two or three sets, in order to help demonstrate the basic principles of applying set operations.

Everything (-u, –everything)

The `--everything` option is equivalent to concatenating and sorting BED elements from multiple files, but works much faster:



As with all BEDOPS tools and operations, the output of this operation is *sorted*.

Note: The `--everything` option preserves all columns from all inputs. This is useful for multiset unions of datasets with additional ID, score or other metadata.

Example

To demonstrate the use of `--everything` in performing a multiset union, we show three sorted sets `First.bed`, `Second.bed` and `Third.bed` and the result of their union with `bedops`:

```
$ more First.bed
chr1      100     200
chr2      150     300
chr2      200     250
chr3      100     150
```

```
$ more Second.bed
chr2      50      150
chr2      400     600
```

```
$ more Third.bed
chr3      150     350
```

```
$ bedops --everything First.bed Second.bed Third.bed > Result.bed
```

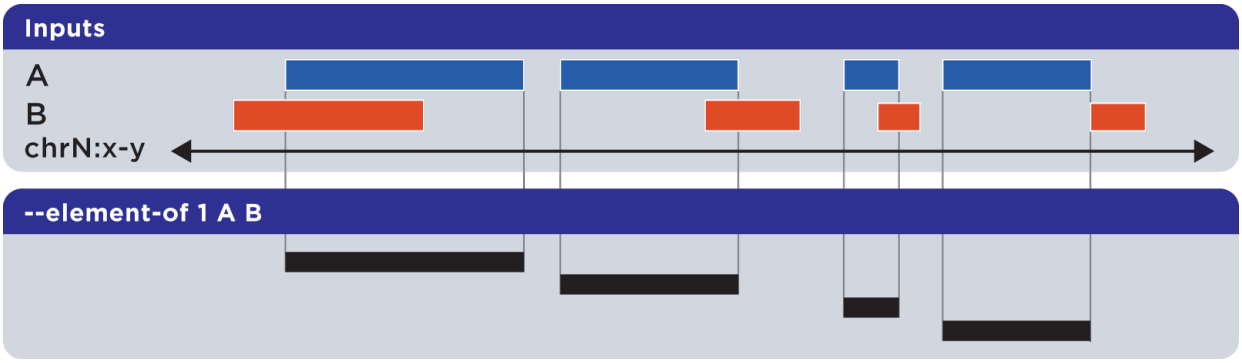
```
$ more Result.bed
chr1      100     200
chr2      50      150
chr2      150     300
chr2      200     250
chr2      400     600
chr3      100     150
chr3      150     350
```

This example uses three input sets, but you can specify two, four or even more sets with `--everything` to take their union.

Element-of (-e, –element-of)

The `--element-of` operation shows the elements of the first ("*reference*") file that overlap elements in the second and subsequent "*query*" files by the specified length (in bases) or by percentage of length.

In the following example, we search for elements in the reference set `A` which overlap elements in query set `B` by at least one base:



Elements that are returned are always from the reference set (in this case, set `A`).

Note: The `--element-of` option preserves all columns from the first (reference) input.

Example

The argument to `--element-of` is a value that species to degree of overlap for elements. The value is either integral for per-base overlap, or fractional for overlap measured by length.

Here is a demonstration of the use of `--element-of 1` on two sorted sets `First.bed` and `Second.bed`, which looks for elements in the `First` set that overlap elements in the `Second` set by one or more bases:

```
$ more First.bed
chr1      100     200
chr1      150     160
chr1      200     300
chr1      400     475
chr1      500     550
```

```
$ more Second.bed
chr1      120     125
chr1      150     155
chr1      150     160
chr1      460     470
chr1      490     500
```

```
$ bedops --element-of 1 First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1      100     200
chr1      150     160
chr1      400     475
```

One base is the least stringent (default) integral criterion. We can be more restrictive about our overlap requirement by increasing this value, say to 15 bases:

```
$ bedops --element-of 15 First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1      100     200
```

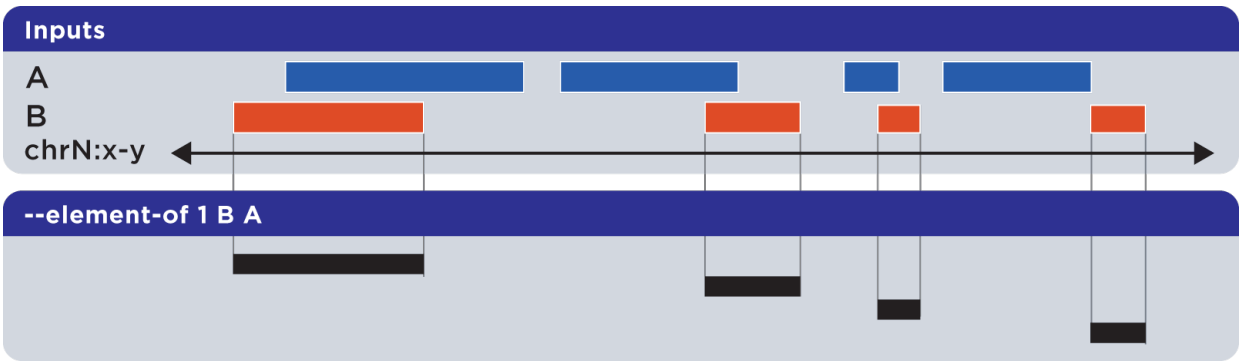
Only this element from the `First` set overlaps one or more elements in the `Second` set by a total of fifteen or more bases.

We can also use percentage of overlap as our argument. Let's say that we only want elements from the `First` set, which overlap half their length or more of a qualifying element in the `Second` set:

```
$ bedops --element-of 50% First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1      150     160
```

Note that *–element-of* is *not* a symmetric operation, as demonstrated by reversing the order of the reference and query set:



Example

As we show here, by inverting the usual order of our sample sets `First` and `Second`, we retrieve elements from the `Second` set:

```
$ bedops --element-of 1 Second.bed First.bed > Result.bed
```

```
$ more Result.bed
chr1      120     125
chr1      150     155
chr1      150     160
chr1      460     470
```

While this operation is not symmetric with respect to ordering of input sets, `--element-of` (`-e`) does produce exactly everything that `--not-element-of` (`-n`) does not, given the same overlap criterion and ordered input sets.

Note: We show usage examples with two files, but `--element-of` supports three or more input sets. For a more in-depth discussion of `--element-of` and how overlaps are determined with three or more input files, please review the BEDOPS forum discussion on this subject.

Not-element-of (-n, –not-element-of)

The `--not-element-of` operation shows elements in the reference file which do not overlap elements in all other sets. For example:

Example

We again use sorted sets `First.bed` and `Second.bed` to demonstrate `--not-element-of`, in order to look for elements in the `First` set that *do not* overlap elements in the `Second` set by one or more bases:

```
$ more First.bed
chr1      100     200
chr1      150     160
chr1      200     300
chr1      400     475
chr1      500     550
```

```
$ more Second.bed
chr1      120     125
chr1      150     155
```

(continues on next page)

(continued from previous page)

```
chr1      150     160
chr1      460     470
chr1      490     500
```

```
$ bedops --not-element-of 1 First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1      200     300
chr1      500     550
```

As with the `--element-of` (`-e`) operator, the overlap criterion for `--not-element-of` (`-n`) can be specified either by length in bases, or by percentage of length.

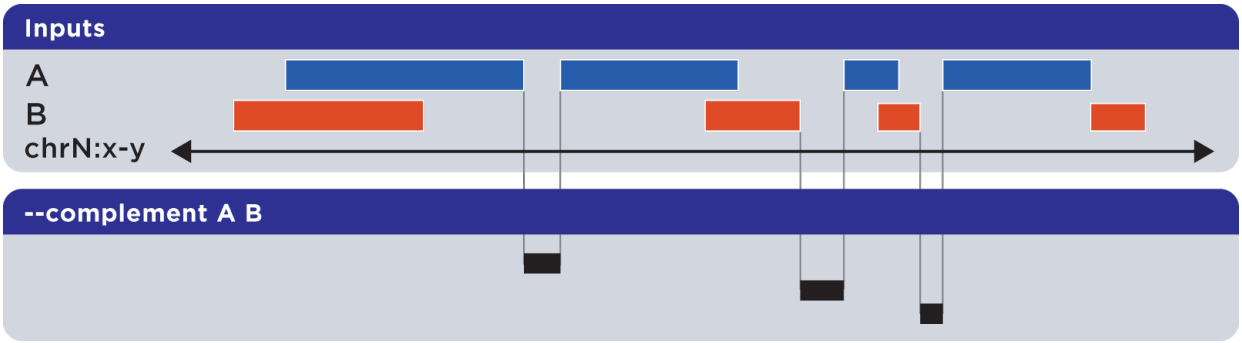
Similarly, this operation is not symmetric – the order of inputs will specify the reference set, and thus the elements in the result (if any).

Note: The `--not-element-of` operatior preserves columns from the first (reference) dataset.

Note: The same caveat applies to use of `--not-element-of` (`-n`) as with `--element-of` (`-e`), namely that the second and all subsequent input files are merged before the set operation is applied. Please review the BEDOPS forum discussion thread on this topic for more details.

Complement (-c, –complement)

The `--complement` operation calculates the genomic regions in the gaps between the contiguous per-chromosome ranges defined by one or more inputs. The following example shows the use of two inputs:



Note this **computed result** will lack ID, score and other columnar data other than the first three columns that contain positional data. That is, computed elements will not come from any of the input sets, but are new elements created from the input set space.

Example

To demonstrate `--complement`, we again use sorted sets `First.bed` and `Second.bed`, in order to compute the "gaps" between their inputs:

```
$ more First.bed
chr1       100      200
chr1       150      160
chr1       200      300
chr1       400      475
chr1       500      550
```

```
$ more Second.bed
chr1       120      125
chr1       150      155
chr1       150      160
chr1       460      470
chr1       490      500
```

```
$ bedops --complement First.bed Second.bed > Result.bed
```

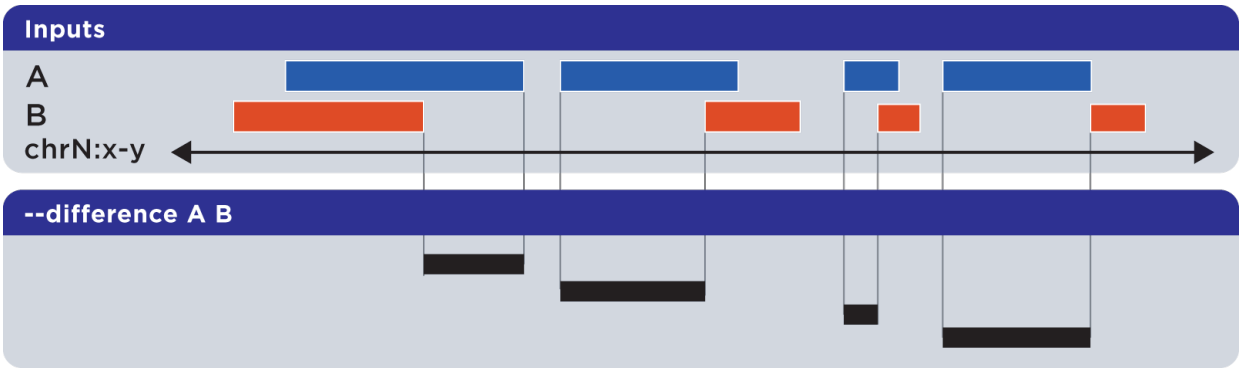
```
$ more Result.bed
chr1       300      400
chr1       475      490
```

As we see here, for a given chromosome, gaps are computed between the leftmost and rightmost edges of elements in the union of elements across all input sets.

Note: For a more in-depth discussion on using `--complement` with left and right bounds of input chromosomes, please review the BEDOPS forum discussion on this subject.

Difference (-d, –difference)

The `--difference` operation calculates the genomic regions found within the first (reference) input file, excluding regions in all other input files:



Example

To demonstrate `--difference`, we use sorted sets `First.bed` and `Second.bed` and compute the genomic space in `First` that excludes (or "subtracts") ranges from `Second`:

```
$ more First.bed
chr1       100      200
```

(continues on next page)

(continued from previous page)

```
chr1       150      160
chr1       200      300
chr1       400      475
chr1       500      550
```

```
$ more Second.bed
chr1       120      125
chr1       150      155
chr1       150      160
chr1       460      470
chr1       490      500
```

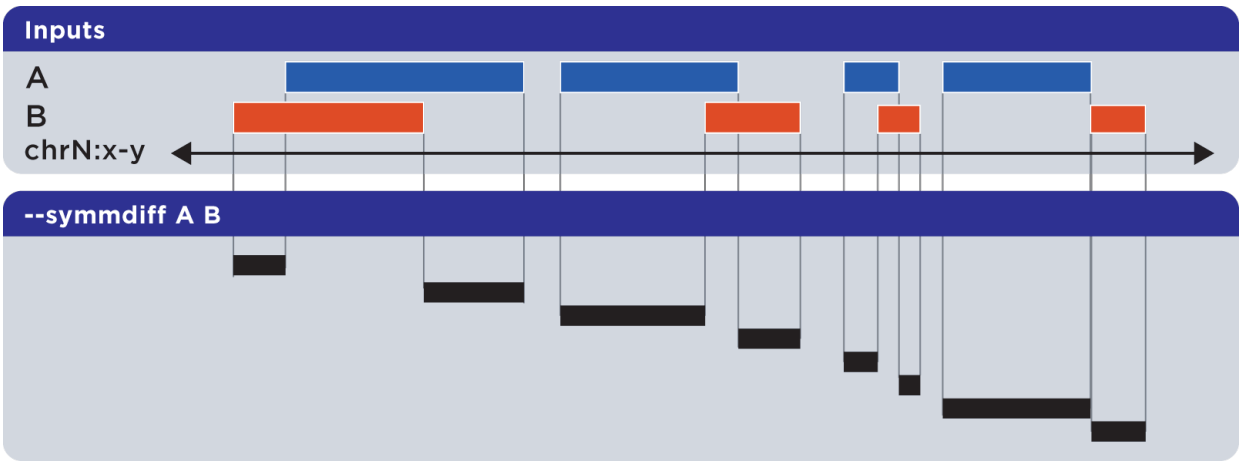
```
$ bedops --difference First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1       100      120
chr1       125      150
chr1       160      300
chr1       400      460
chr1       470      475
chr1       500      550
```

Note: As with `--element-of` and `--not-element-of`, this operation is not symmetric. While `--not-element-of` preserves all columns of elements found in the reference input and allows one to define overlaps, the `--difference` operator simply reports every genomic range as three-column BED, which does not overlap elements found in the second and subsequent input files by any amount.

Symmetric difference (-s, –symmdiff)

The `--symmdiff` operation calculates the genomic range that is exclusive to each input, excluding any ranges shared across inputs:



Example

To demonstrate `--symmdiff`, we use sorted sets `First.bed` and `Second.bed` and compute the genomic space that is unique to `First` and `Second`:

```
$ more First.bed
chr1       100      200
chr1       150      160
chr1       200      300
chr1       400      475
chr1       500      550
```

```
$ more Second.bed
chr1       120      125
chr1       150      155
chr1       150      160
chr1       460      470
chr1       490      500
```

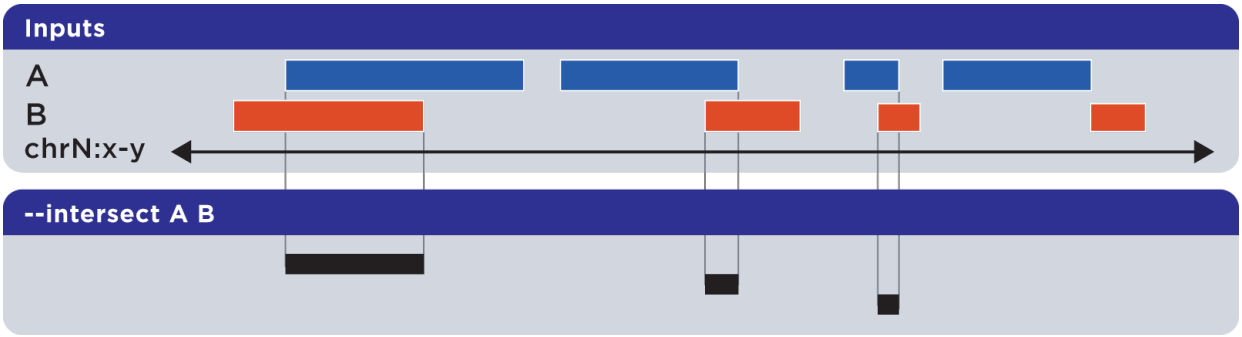
```
$ bedops --symmdiff First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1       100      120
chr1       125      150
chr1       160      300
chr1       400      460
chr1       470      475
chr1       490      550
```

Tip: It has been observed that `--symmdiff` (`-s`) is the same as the union of `--difference A B` with `--difference B A`, but `--symmdiff` runs faster in practice.

Intersect (-i, –intersect)

The `--intersect` operation determines genomic regions common to all input sets:



Example

To demonstrate `--intersect`, we use sorted sets `First.bed` and `Second.bed` and compute the genomic space that is common to both `First` and `Second`:

```
$ more First.bed
chr1       100      200
chr1       150      160
chr1       200      300
chr1       400      475
chr1       500      550
```

```
$ more Second.bed
chr1       120      125
chr1       150      155
chr1       150      160
chr1       460      470
chr1       490      500
```

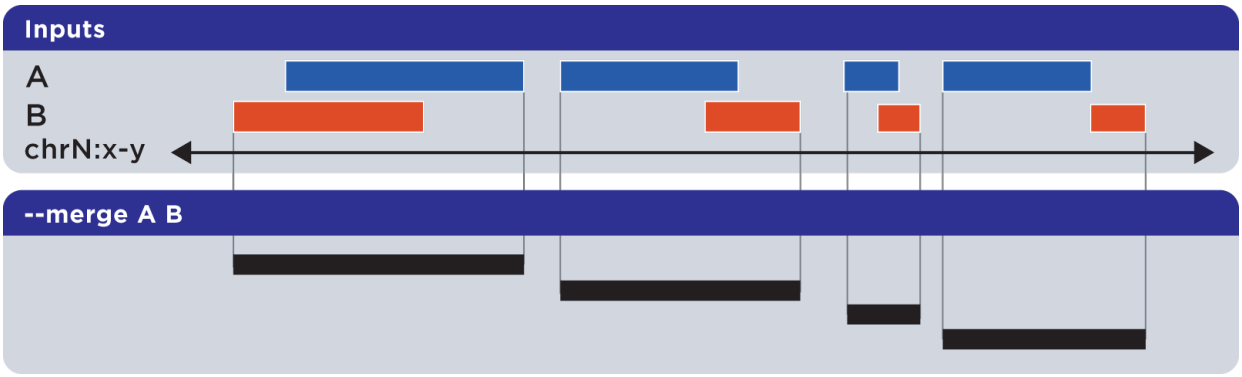
```
$ bedops --intersect First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1       120      125
chr1       150      160
chr1       460      470
```

Notice how this computed result is quite different from that of `--element-of N`, which functions more like a LEFT JOIN operation in SQL.

Merge (-m, –merge)

The `--merge` operation flattens all disjoint, overlapping, and adjoining element regions into contiguous, disjoint regions:



Example

To demonstrate `--merge`, we use sorted sets `First.bed` and `Second.bed` and compute the contiguous genomic space across both `First` and `Second`:

```
$ more First.bed
chr1       100      200
chr1       150      160
chr1       200      300
chr1       400      475
chr1       500      550
```

```
$ more Second.bed
chr1       120      125
chr1       150      155
chr1       150      160
chr1       460      470
chr1       490      500
```

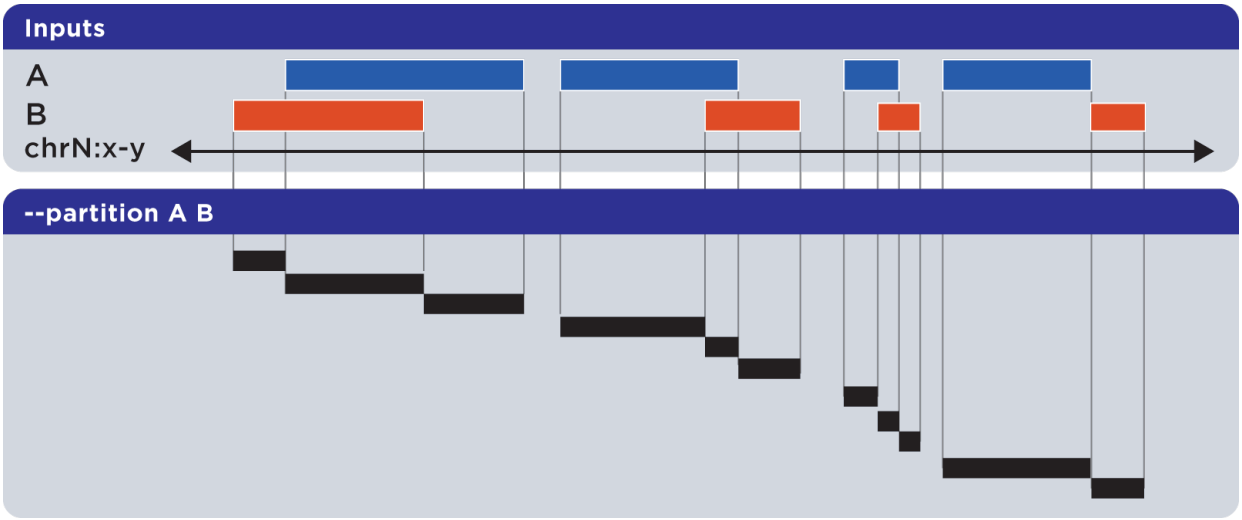
```
$ bedops --merge First.bed Second.bed > Result.bed
```

```
$ more Result.bed
chr1       100      300
chr1       400      475
chr1       490      550
```

Tip: The preceding example shows use of `--merge` (`-m`) with two inputs, but the merge operation works just as well with one input, collapsing elements within the file that overlap or which are directly adjoining.

Partition (-p, –partition)

The `--partition` operator splits all overlapping input regions into a set of disjoint segments. One or more input files may be provided; this option will segment regions from all inputs:



Example

To demonstrate `--partition`, we use sorted sets `First.bed` and `Second.bed` and compute disjointed genomic regions across both `First` and `Second`:

```
$ more First.bed
chr1       100      200
chr1       150      160
chr1       200      300
chr1       400      475
chr1       500      550
```

```
$ more Second.bed
chr1       120      125
chr1       150      155
chr1       150      160
chr1       460      470
chr1       490      500
```

```
$ bedops --partition First.bed Second.bed > Result.bed
```

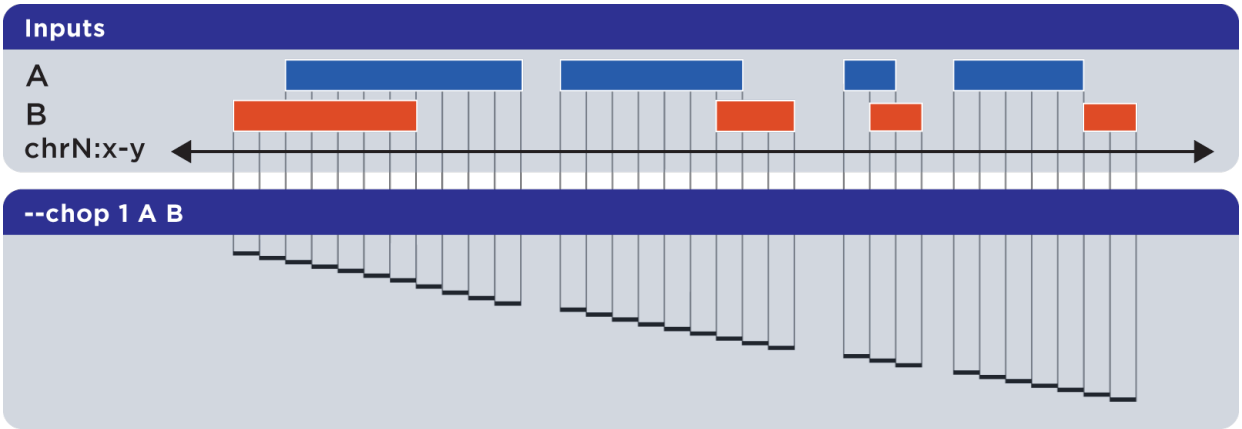
```
$ more Result.bed
chr1       100      120
chr1       120      125
chr1       125      150
chr1       150      155
chr1       155      160
chr1       160      200
chr1       200      300
chr1       400      460
chr1       460      470
chr1       470      475
chr1       490      500
chr1       500      550
```

Notice that the result set of partitioned elements excludes any duplicates from input regions, thus enforcing the disjoint nature of the computed result.

Note: As with `--merge`, `--complement` and other "computing" operations, note the lack of ID, score and other columnar data in this computed result.

Chop (-w, –chop)

The `--chop` operator merges all overlapping input regions and "chops" them up into a set of disjoint segments of identical length (with a default of one base). One or more input files may be provided; this option will segment regions from all inputs:



Example

To demonstrate `--chop`, we use a sorted set called `Regions.bed` and compute a set of one-base genomic regions constructed from the merged input elements:

```
$ more Regions.bed
chr1      100      105
chr1      120      127
chr1      122      124
```

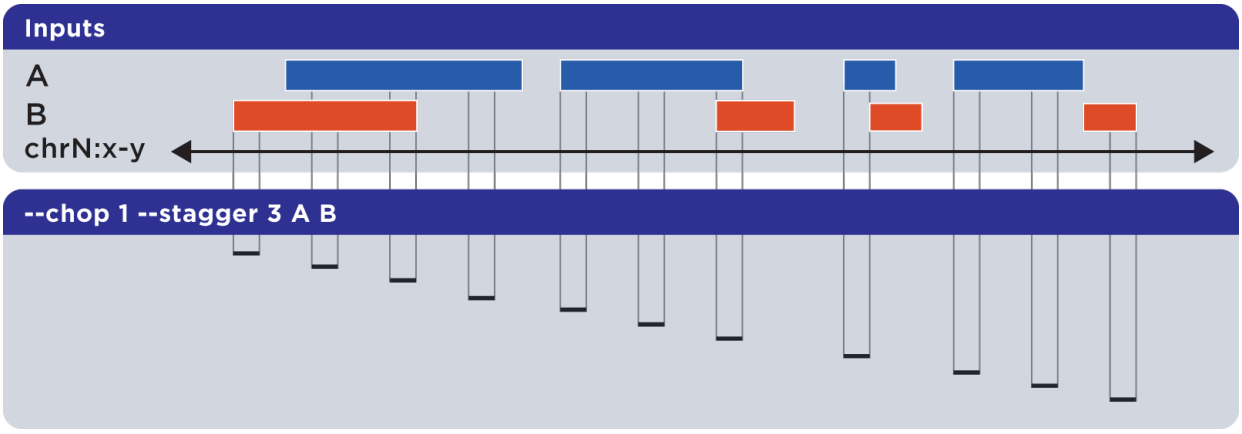
```
$ bedops --chop 1 Regions.bed > Result.bed
```

```
$ more Result.bed
chr1      100      101
chr1      101      102
chr1      102      103
chr1      103      104
chr1      104      105
chr1      120      121
chr1      121      122
chr1      122      123
chr1      123      124
chr1      124      125
chr1      125      126
chr1      126      127
```

Note: Overlapping and nested regions are merged into contiguous ranges before chopping. The end result contains unique, non-overlapping elements.

Stagger (–stagger)

The `--stagger` operator works in conjunction with *–chop*. While `--chop` sets the size of each cut, the `--stagger` operator moves the start position of each cut by the specified number of bases, across each merged interval.



Example

To demonstrate `--stagger`, we use a sorted set called `Regions.bed` and compute a set of one-base genomic regions constructed from the merged input elements, but move the start position across the merged regions by three

bases, before generating the next chop:

```
$ more Regions.bed
chr1      100     105
chr1      120     127
chr1      122     124
```

```
$ bedops --chop 1 --stagger 3 Regions.bed > Result.bed
```

```
$ more Result.bed
chr1      100     101
chr1      103     104
chr1      120     121
chr1      123     124
chr1      126     127
```

Note: Overlapping and nested regions are merged into contiguous ranges before chopping and staggering. The end result contains unique, non-overlapping elements.

Exclude (-x)

Like `--stagger`, `-x` is a sub-option of the *–chop* operator, and it may be used with or without `--stagger`. This option will remove any remainder genomic chunk that is smaller than that specified with `--chop`. For example, if you start with a 10 nt region and use `--chop 4`, the final segment would be 2 nt in length if `-x` is not specified. With `-x`, that last segment does not go to output. With `-x`, the `chop` operation produces output regions that are all the same size.

Per-chromosome operations (–chrom)

All operations on inputs can be restricted to one chromosome, by adding the `--chrom <val>` operator.

Note: This operator is highly useful for parallelization, where operations on large BED inputs can be split up by chromosome and pushed to separate nodes on a computational cluster. See the *Efficiently creating Starch-formatted archives with a cluster* documentation for a demonstration of this technique in action.

Example

To demonstrate the use of `--chrom` to restrict operations to a chromosome (such as `chr3`), we perform a per-chromosome union of elements from three sorted sets `First.bed`, `Second.bed` and `Third.bed`, each with elements from multiple chromosomes:

```
$ more First.bed
chr1      100     200
chr2      150     300
chr2      200     250
chr3      100     150
```

```
$ more Second.bed
chr2      50      150
chr2      400     600
```

```
$ more Third.bed
chr3      150     350
```

```
$ bedops --chrom chr3 --everything First.bed Second.bed Third.bed > Result.bed
```

```
$ more Result.bed
chr3      100     150
chr3      150     350
```

Range (–range)

The `--range` operation works in conjunction with other operations.

When used with one value (`--range S`), this operation **symmetrically** pads all elements of input sets by the specified integral value `S`. When the specified value is positive, every genomic segment grows in size. An element will grow asymmetrically to prevent growth beyond base position 0, if needed. Otherwise, when negative, elements shrink, and any element with zero (or less) length is discarded.

Alternatively, when used with two values (`--range L:R`), this operation **asymmetrically** pads elements, adding `L` to each start coordinate, and adding `R` to each stop coordinate. Negative values may be specified to grow or shrink the region, accordingly.

This option is immediately useful for adjusting the coordinate index of BED files. For example, to shift from 1-based to 0-based coordinate indexing:

```
$ bedops --range -1:-1 --everything my1BasedCoordinates.bed > my0BasedCoordinates.bed
```

And, likewise, for 0-based to 1-based indexing:

```
$ bedops --range 1:1 --everything my0BasedCoordinates.bed > my1BasedCoordinates.bed
```

Note: The `--range` value is applied to inputs prior to the application of other operations (such as `--intersect` or `--merge`, etc.).

Padding elements with *bedops* is much more efficient that doing so with `awk` or some other script, *and you do not need to go back and resort your data*. Even symmetric padding can cause data to become unsorted in non-obvious ways. Using `--range` ensures that your data remain sorted and it works efficiently with any set operation.

Also, note that the `--element-of` and `--not-element-of` operations behave differently with `--range`, in that only the second and subsequent input files are padded.

Starch support

The *bedops* application supports use of *Starch*-formatted archives as inputs, as well as text-based BED data. One or multiple inputs may be Starch archives.

Tip: By combining the `--chrom` operator with operations on *Starch* archives, the end user can achieve improved computing performance and disk space savings, particularly where *bedops*, *bedmap* and *closest-features* operations are applied with a computational cluster on separate chromosomes.

Error checking (–ec)

Use the `--ec` option in conjunction with any aforementioned operation to do more stringent checking of the inputs' compliance to *bedops* requirements, including sorting checks, delimiter checks, among others.

To demonstrate, we can deliberately introduce a typo in dataset *A*, using the `--ec` option to try to catch it:

```
$ bedops --ec --everything BEDFileA
May use bedops --help for more help.

Error: in BEDFileA
First column should not have spaces.  Consider 'chr1' vs. 'chr1 '.  These are␣
↪different names.
See row: 3
```

The typo introduced was the addition of a space within the third line of dataset `A`.

Note: Use of the `--ec` option will roughly *double* the running times of set operations, but it provides stringent error checking to ensure inputs and outputs are valid. `--ec` can help check problematic input and offers helpful hints for any needed corrections, when problems are detected.

Tips

Chaining operations

You can efficiently chain operations together, *e.g.*:

```
$ bedops --range 50 --merge A | bedops --intersect - B > answer.bed
```

In this example, elements from `A` are padded 50 bases up- and downstream and merged, before intersecting with coordinates in `B`.

Sorting inputs

For unsorted input, be sure to first use *sort-bed* to presort the data stream before using with *bedops*. Unsorted input will not work properly with BEDOPS tools.

Tip: If you will use an initially-unsorted file more than once, save the results of sorting. You only need to sort once! BEDOPS tools take in and export sorted data.

bedextract

The `bedextract` utility performs three primary tasks, with the goal of doing them very quickly:

1. Lists all the chromosomes in a sorted input BED file.
2. Extracts all the elements in a sorted input BED file, for a given chromosome.
3. Finds elements of one BED file, which overlap elements in a second, reference BED file (when specific element criteria are satisfied).

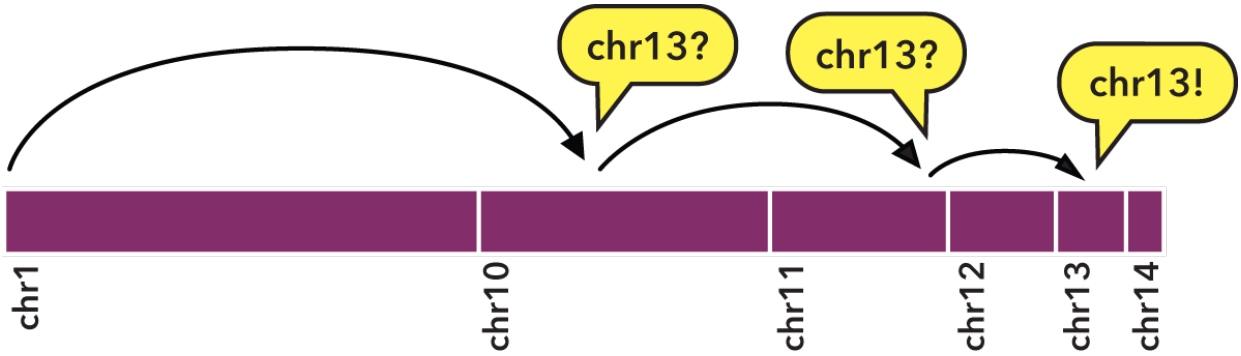
One might ask why use this utility, when the first two tasks can already be performed with common UNIX text processing tools, such as `cut`, `sort`, `uniq`, and `awk`, and the third task can be performed with *bedops* with the `--element-of 1` options?

The `bedextract` utility does the work of all those tools without streaming through an entire BED file, resulting in massive performance improvements. By using the hints provided by sorted BED input, the *bedextract* tool can jump around, seeking very quick answers to these questions about your data.

How it works

Specifically, sorting with *sort-bed* allows us to perform a binary search:

1. We jump to the middle byte of the BED file, stream to the nearest element, then parse and test the chromosome name.
2. Either we have a match, or we jump to the middle of the remaining left or right half (decided by dictionary order), parse and test again.
3. We repeat steps 1 and 2 until we have matches that define the bounds of the target chromosome.



To indicate the kind of speed gain that the *bedextract* tool provides, in local testing, a naïve listing of chromosomes from a 36 GB BED input using UNIX `cut` and `uniq` utilities took approximately 20 minutes to complete on a typical Core 2 Duo-based Linux workstation. Retrieval of the same chromosome listing with `bedextract --list-chr` took only 2 seconds (cache flushed—no cheating!).

Tip: While listing chromosomes is perhaps a trivial task, 1200 seconds to 2 seconds is a **600-fold** speedup. Similar improvements are gained from using `--chrom` and `--faster` options with other core BEDOPS tools like *bedops* and *bedmap*. If your data meet the criteria for using this approach—and a lot of genomic datasets do—we strongly encourage adding this to your toolkit.

Inputs and outputs

Input

Depending on specified options, *bedextract* requires one or two *sorted* BED files.

Note: It is critical that inputs are *sorted* as the information in a sorted file allows *bedextract* to do its work correctly. If your datasets are output from other BEDOPS tools, then they are already sorted!

Output

Depending on specified options, the *bedextract* program will send a list of chromosomes or BED elements to standard output.

Tip: The use of UNIX-like standard streams allows easy downstream analysis or post-processing with other tools and scripts, including other BEDOPS utilities.

Usage

The `--help` option describes the functionality available to the end user:

```
bedextract
  citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
  version:  2.4.37 (typical)
  authors:  Shane Neph & Alex Reynolds

    Every input file must be sorted per sort-bed.

 USAGE:
   0) --help or --version           Print requested info and exit successfully.
   1) --list-chr <input.bed>        Print all unique chromosome names found in <input.
↪bed>.
   2) <chromosome> <input.bed>      Retrieve all rows for chr8 with:  bedextract chr8
↪<input.bed>.
   3) <query.bed> <target>          Grab elements from the <query.bed> that overlap␣
↪elements in <target>. Same as
                                     `bedops -e 1 <query.bed> <target>`, except that␣
↪this option fails silently
                                     if <query.bed> contains fully-nested BED␣
↪elements.  If no fully-nested
                                     element exists, bedextract can vastly improve␣
↪upon the performance of bedops.
                                     <target> may be a BED or Starch file (with or␣
↪without fully-nested elements).
                                     Using '-' for <target> indicates input (in BED␣
↪format) comes from stdin.
```

Listing chromosomes

Use the `--list-chr` option to quickly retrieve a listing of chromosomes from a given sorted BED input.

For example, the following lists the chromosomes in an example BED file of FIMO motif hits (see the *Downloads* section):

```
$ bedextract --list-chr motifs.bed
chr1
chr10
chr11
chr12
...
chr9
chrX
```

Note: The `bedextract --list-chr` operation only works on BED files. If you have a Starch file, use `unstarch --list-chr` to list its chromosomes.

Retrieving elements from a specific chromosome

To quickly retrieve the subset of elements from a sorted BED file associated with a given chromosome, apply the second usage case and specify the chromosome as the argument.

For example, to retrieve `chrX` from the same motif sample:

```
$ bedextract chrX motifs.bed
chrX    6775077 6775092 +V_SPZ1_01      4.92705e-06     +       GTTGGAGGGAAGGGC
chrX    6775168 6775179 +V_ELF5_01      8.57585e-06     +       TCAAGGAAGTA
chrX    6777790 6777799 +V_CKROX_Q2     8.90515e-06     +       TCCCTCCCC
...
```

Note: The `bedextract <chromosome>` operation only works on BED files. If you have a Starch file, use `unstarch <chromosome>` to list the elements associated with that chromosome.

Retrieving elements which overlap target elements

A common *bedops* query involves asking which elements overlap one or more bases between two BED datasets, which we will call here `Query` and `Target`.

One can already use `bedops --element-of 1` to accomplish this task, but if certain specific criteria are met (which we will describe shortly) then a much faster result can often be obtained by instead using *bedextract*.

Three criteria make the use of *bedextract* in this mode very successful in practice, with potentially massive speed improvements:

1. `Query` is a huge file.
2. There are relatively few regions of interest in `Target` (say, roughly 30,000 or fewer).
3. There are **no fully-nested elements** in `Query` (but duplicate coordinates are fine).

Note: With some extra work, it is possible to use this mode of bedextract with a huge `Query` BED file that includes fully-nested elements. The technique requires that you create a merged version of `Query` and keep that result, `Query-Index`, around along with `Query`.

```
$ bedops -m Query > Query-Index
$ bedextract Query-Index Target \
    | bedextract Query - \
    | bedops --element-of 1 - Target \
    > answer.bed
```

Note: You may change the final overlap criterion to the *bedops –element-of* as you see fit for your final answer. Adding –range 10, for example, to *bedops -m Query* > *Query-Index* can often greatly reduce the size of Query-Index without much of an increase in the running time of this approach.

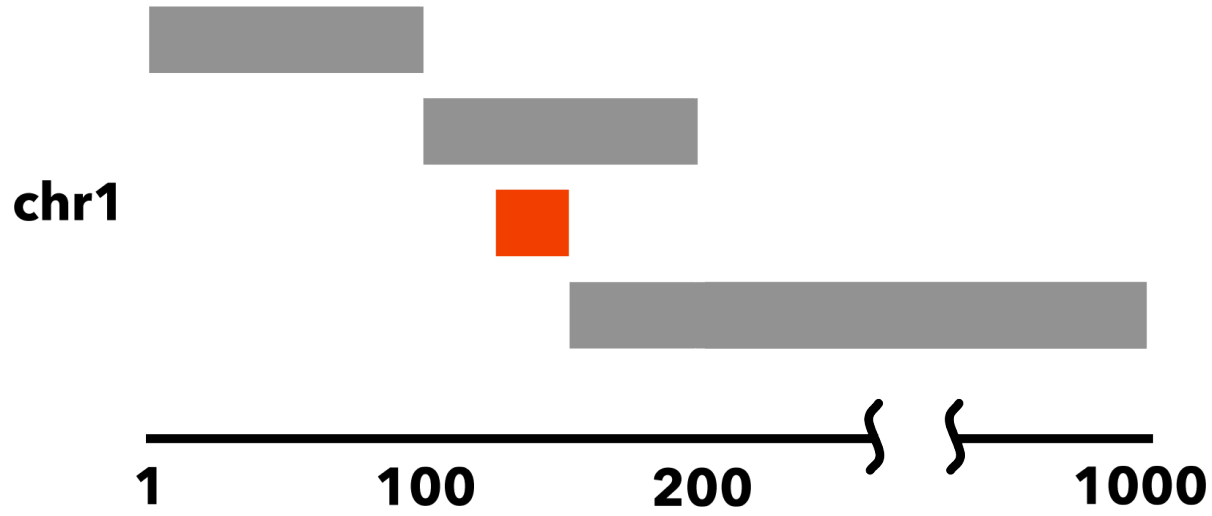
What are nested elements?

For a precise definition of a nested element, refer to the *documentation on nested elements*.

For an example, we show the following sorted BED file:

```
chr1    1       100
chr1    100     200
chr1    125     150
chr1    150     1000
```

In this sorted dataset, the element `chr1:125-150` is entirely nested within `chr1:100-200`:



Note: Fully-nested elements are not a problem for the other two *bedextract* features: 1) Listing all chromosomes, and 2) Retrieving all information for a single chromosome.

Fully-nested elements are only an issue for *bedextract* if they exist in the `Query` dataset. Results are not affected if the `Target` dataset contains nested elements. Overlapping (but not fully-nested) elements in the `Query` input file are fine, as are duplicated genomic positions.

Note: Our lab works with BED data of various types: cut-counts, hotspots, peaks, footprints, etc. These data generally

do not contain nested elements and so are amenable to use with *bedextract* for extracting overlapping elements.

However, other types of `Query` datasets can be problematic. FIMO search results, for example, might cause trouble, where the boundaries of one motif hit can be contained within another larger hit. Or paired-end sequence data, where tags are not of a fixed length. Be sure to consider the makeup of your BED data before using *bedextract*.

Demonstration

To demonstrate this use of *bedextract*, for our `Query` dataset we will use the `Map` example from our *bedmap* documentation, which contains raw DNaseI hypersensitivity signal from a human K562 cell line (see the *Downloads* section for sample data):

```
$ cat query.bed
chr21   33031165        33031185        map-1   1.000000
chr21   33031185        33031205        map-2   3.000000
chr21   33031205        33031225        map-3   3.000000
chr21   33031225        33031245        map-4   3.000000
...
chr21   33032445        33032465        map-65  5.000000
chr21   33032465        33032485        map-66  6.000000
```

Our `Target` data is simply an *ad-hoc* BED region which overlaps part of the `Query` dataset, stored in a *Starch-formatted* archive:

```
$ unstarch target.starch
chr21   33031600        33031700
```

We can now ask which elements of `Query` overlap the element in `Target`:

```
$ bedextract query.bed target.starch
chr21   33031585        33031605        map-22  26.000000
chr21   33031605        33031625        map-23  27.000000
chr21   33031625        33031645        map-24  29.000000
chr21   33031645        33031665        map-25  31.000000
chr21   33031665        33031685        map-26  31.000000
chr21   33031685        33031705        map-27  37.000000
```

Our `Target` dataset is a Starch-formatted file. Note that we can also use "-" to denote standard input for the `Target` dataset, as well as a regular BED- or Starch-formatted file. In other words, we can pipe target elements from another process to *bedextract*, *e.g.* we can query for an ad-hoc element as follows:

```
$ echo -e "chr21\t33031590\t33031600" | bedextract query.bed -
chr21   33031585        33031605        map-22  26.000000
```

Instead of an *ad-hoc* element as in this example, however, target elements could just as easily be piped in from upstream *bedmap* or *bedops* operations, or extracted elements from a Starch archive, etc.

Tip: The output of this particular use of *bedextract* is made up of elements from the `Query` dataset and is therefore *sorted* BED data, which can be piped to *bedops*, *bedmap* and other BEDOPS utilities for further downstream processing.

Note: Though *bedextract* only supports the overlap equivalent of `bedops --element-of 1`, other overlap criteria are efficiently supported by combining *bedextract* with *bedops*.

Specifically, we can quickly filter through just the results given by *bedextract* and implement other overlap criteria with *bedops*, *e.g.*:

```
$ bedextract query.bed target.bed | bedops -e 50% - target.bed
```

Downloads

- Sample `FIMO motifs`
- Sample `Query` dataset: `DHS signal`
- Sample `Target` dataset: `ad-hoc coordinates`

closest-features

The `closest-features` program efficiently associates nearest features between two sorted inputs, based upon genomic distance measures.

An application of this tool in our own research is *finding the nearest DNase hypersensitive sites* upstream and downstream from a given SNP, as well as signed distances. The *closest-features* program can report both results.

As another example of what one can do with this utility, we can identify the closest transcriptional start site for a given putative replication origin. Suppose we have a sorted BED file named `TSS.bed` that contains all transcriptional start sites of all genes in some genome. Further, suppose that we have a set of measurements showing probable replication origins for the same species in a sorted BED file named `RepOrigins.bed`. The following command gives the closest TSS to each origin:

```
$ closest-features --closest RepOrigins.bed TSS.bed
```

By default, the program will echo each entry from `RepOrigins.bed`, followed by the two closest elements in `TSS.bed` (the closest element to each side of the entry from `RepOrigins.bed`), with output columns separated by a pipe (`|`). With the `--shortest` option, the echoed entry from `RepOrigins.bed` and only the single nearest element in `TSS.bed` will be part of the output.

Inputs and outputs

Input

The *closest-features* program takes two sorted BED files (a so-called *reference* file and a *map* file), as well as optional arguments for modifying behavior and outputs.

Alternatively, *closest-features* can accept *Starch-formatted archives* as inputs, with no need to extract archive data to intermediate BED files!

Support for common headers (such as UCSC track headers) is offered through the `--header` option. Headers are stripped from output.

Output

The *closest-features* program returns summary data to standard output, which may include reference and nearest elements and distance values (depending on provided options).

Usage

The `--help` option describes the various operations and options available to the end user:

```
closest-features
  citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
  version:  2.4.37 (typical)
  authors:  Shane Neph & Scott Kuehn

USAGE: closest-features [Process-Flags] <input-file> <query-file>
   All input files must be sorted per sort-bed.
   The program accepts BED and Starch file formats
   May use '-' for a file to indicate reading from standard input (BED format only).

   For every element in <input-file>, determine the two elements from <query-file> falling
     nearest to its left and right edges (See NOTES below).  By default, echo the <input-file>
     element, followed by those left and right elements found in <query-file>.

  Process Flags:
    --chrom <chromosome>    : Process data for given <chromosome> only.
    --closest               : Choose the closest element for output only.  Ties go the left element.
    --delim <delim>         : Change output delimiter from '|' to <delim> between columns (e.g. '\t')
    --dist                  : Print the signed distances to the <input-file> element as additional
                                columns of output.  An overlapping element has a distance of 0.
    --ec                    : Error check all input files (slower).
    --header                : Accept headers (VCF, GFF, SAM, BED, WIG) in any input file.
    --help                  : Print this message and exit successfully.
    --no-overlaps           : Overlapping elements from <query-file> will not be reported.
    --no-ref                : Do not echo elements from <input-file>.
    --version               : Print program information.

  NOTES:
    If an element from <query-file> overlaps the <input-file> element, its distance is zero.
      An overlapping element takes precedence over all non-overlapping elements.  This is true
      even when the overlapping element's edge-to-edge distance to the <input-file>'s element
      is greater than the edge-to-edge distance from a non-overlapping element.
    Overlapping elements may be ignored completely (no precedence) with --no-overlaps.
    Elements reported as closest to the left and right edges are never the same.
    When no qualifying element from <query-file> exists as a closest feature, 'NA' is reported.
```

Per-chromosome operations (–chrom)

All operations on inputs can be restricted to one chromosome, by adding the `--chrom <val>` operator.

Tip: This option is highly useful for cluster-based work, where operations on large BED inputs can be split up by chromosome and pushed to separate cluster nodes.

To demonstrate the use of this option, we take two sample Starch-archived BED datasets `A` and `B` (refer to the *Downloads* section for sample inputs) which contain regions from multiple chromosomes:

```
$ unstarch A.starch
chr1    100     200     id-001A
chr1    400     500     id-002A
chr2    100     300     id-003A

$ unstarch B.starch
chr1    150     300     id-001B
chr1    500     600     id-002B
chr2    100     150     id-003B
chr2    180     500     id-004B
```

Now we want to ask, what is the closest element from `chr2` in `A`, to `chr2` elements in `B`:

```
$ closest-features --chrom chr2 --closest A.starch B.starch
chr2    100     300     id-003A|chr2    100     150     id-003B
```

As we expect, element `id-003A` is closest to element `id-003B` between the two datasets.

Error checking

For performance reasons, no error checking of input is done, by default. Add `--ec` for stringent error checking and debugging purposes.

Note: Using `--ec` will slow down analysis considerably. We recommend using this option to test and debug pipelines and then removing it for use in production.

Downloads

- Sample dataset `A`
- Sample dataset `B`

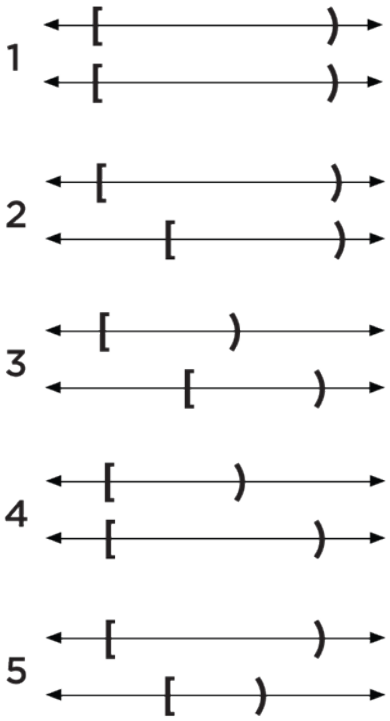
Nested elements

This page describes nested BED elements, their impact on the performance of BEDOPS tools, and how we can identify them beforehand.

Definition

A *nested element* is defined as a BED element from a sorted BED file, where a genomic range is entirely enclosed by the previous element's range.

Loosely speaking, consider the following five overlap cases for pairings of generic, half-open intervals:



Of these five interval pairs, the fifth overlap pairing identifies a nested element, where the second interval is nested within the first.

More rigorously, we define two generic, half-open BED elements A and B, both located on the same generic chromosome N, each with ranges $[a_{start}, a_{stop})$ and $[b_{start}, b_{stop})$, respectively.

These two elements A and B have the following relations:

1. $a_{start} < a_{stop}$
2. $b_{start} < b_{stop}$
3. $a_{start} <= b_{start}$
4. $a_{stop} <= b_{stop}$

Note: The third and fourth conditions place elements A and B into sort order, as applied by the *sort-bed* application.

If we further restrict these ranges: $a_{start} < b_{start}$ and $b_{stop} < a_{stop}$, then for the purposes of BEDOPS we call the element B a *nested element*, one which is contained or *nested* within element A.

It can be useful to be able to identify nested elements in an input set. Here's a method that uses `awk`:

```
#!/usr/bin/env awk -f
{
    if (NR > 1) {
        currentChr = $1
        currentStart = $2
        currentStop = $3
        if ((previousStart < currentStart) && (previousStop > currentStop)) {
            print $0;
        }
        else {
```

(continues on next page)

(continued from previous page)

```
            previousChr = currentChr
            previousStart = currentStart
            previousStop = currentStop
        }
    }
    else {
        previousChr = $1
        previousStart = $2
        previousStop = $3
    }
}
```

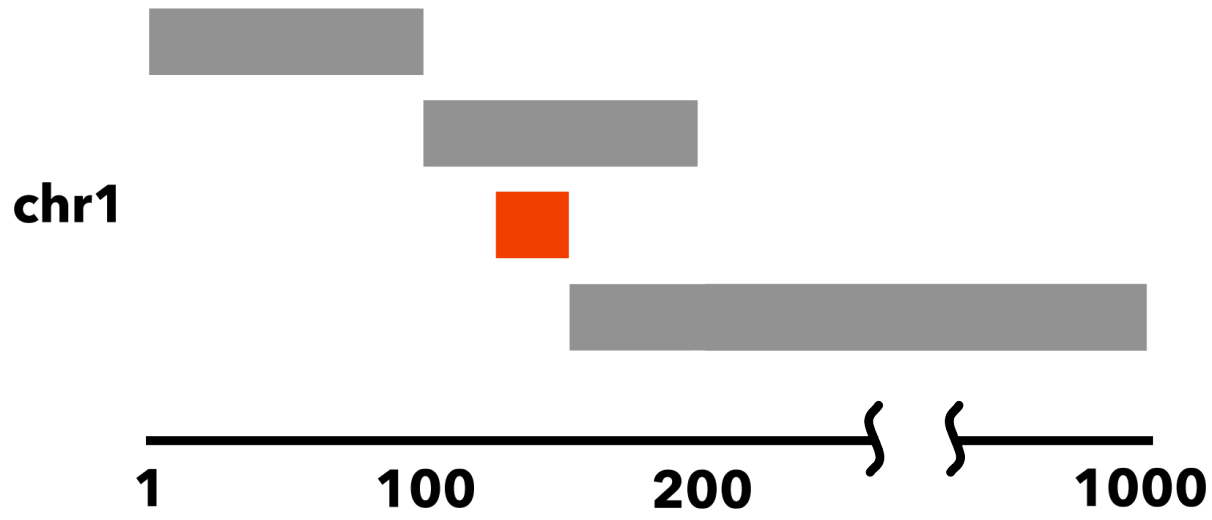
If this script is given the name `getNestedElements.awk` and is made executable, one could filter a BED file via `./getNestedElements.awk foo.bed > nested.bed`, or similar.

Example

A more concrete example of a sorted BED file which contains a nested element follows. Consider the following simple, sorted BED dataset:

```
chr1    1       100
chr1    100     200
chr1    125     150
chr1    150     1000
```

Here, the element `chr1:125-150` is entirely nested within `chr1:100-200`:



Why nested elements matter

BEDOPS *bedmap* and *bedextract* tools offer the `--faster` option to perform very fast retrieval of overlapping elements, so long as input datasets do not contain nested elements, as defined above.

To extract maximum performance out of the use of the BEDOPS toolkit, therefore, it is very useful to know if the input datasets contain such elements — if they do not, then we can apply this optimization.

Common datasets we work with do not contain nested elements: reads, peaks, footprints, and others. However, other datasets do, such as motif hits or paired-end sequencing reads.

How can we find out if our inputs have nested elements, before we start applying any operations?

The compression tool *starch* (v2.5 and greater) will look for these elements in an input BED file and store this condition as a flag in the output archive's metadata. This value can be retrieved in constant time with *unstarch* and other tools which make use of the Starch C++ API.

Additionally, the `--ec` (error-correction) option in *bedmap* will also report if inputs contain nested elements. This option doubles execution time, but when used in conjunction with the `--faster` option, the speed gains are more than recovered.

2.6.2 Statistics

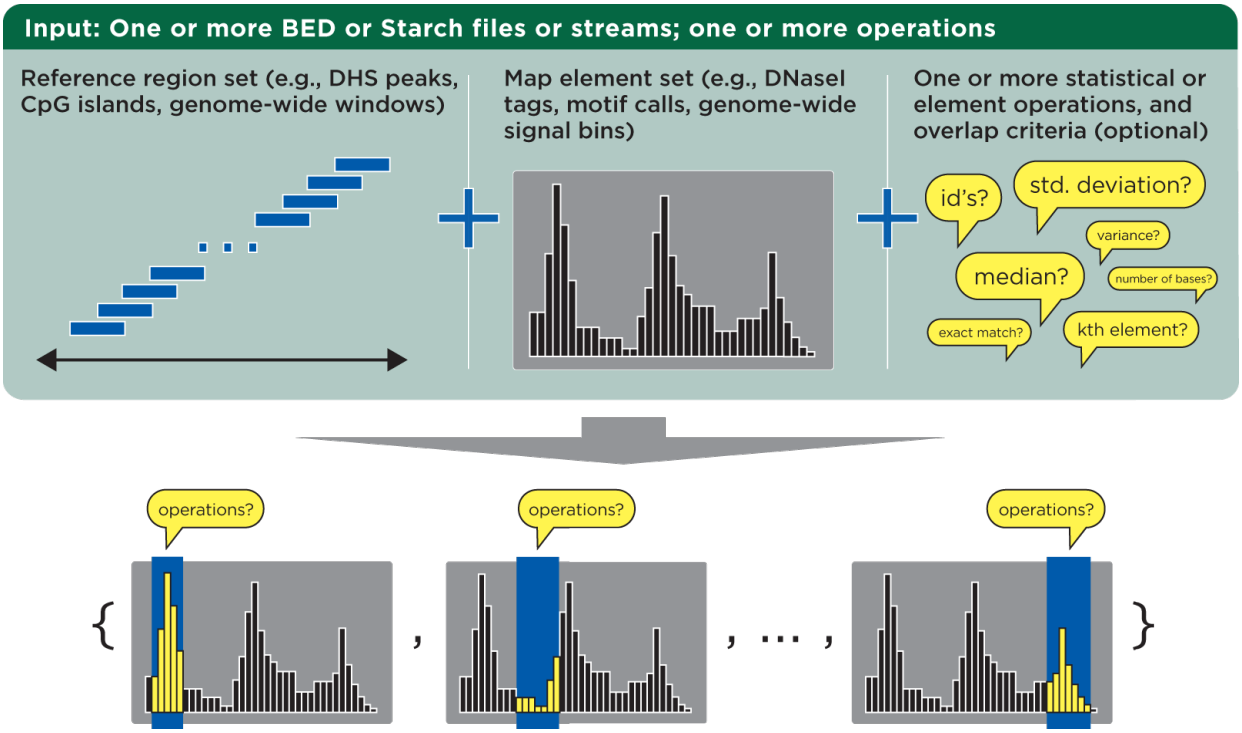
bedmap

The `bedmap` program is used to retrieve and process signal or other features over regions of interest in BED files (including DNase hypersensitive regions, SNPs, transcription factor binding sites, etc.), performing tasks such as: *smoothing raw tag* count signal in preparation for uploading to the UCSC Genome Browser, *finding subsets of elements* within a larger coordinate set, *filtering multiple BED files* by signal, *finding multi-input overlap* solutions, and much, much more.

Inputs and outputs

Input

The *bedmap* program takes in *reference* and *mapping* files and calculates statistics for each reference element. These calculations—*operations*—are applied to overlapping elements from the mapped file:



The *bedmap* program requires files in a relaxed variation of the BED format as described by UCSC's browser documentation. The chromosome field can be any non-empty string, the score field can be any valid numeric value, and information is unconstrained beyond the minimum number of columns required by the chosen options.

Note: Information is unconstrained, with one important exception: Map input (defined below) should not contain spaces in the ID or in subsequent fields. Running *bedmap* with `--ec` will identify problematic input.

An `awk` script can help with translating ID spaces to another non-whitespace delimiter, *e.g.*, `bedmap (--options...) reference.bed <(awk -vOFS="\t" -vFS="\t" '{gsub(" ", "%%", $4); print;}' map.bed)` and then *awk -vOFS="t" -vFS="t" '{gsub("%%", " "); print;}' result.bed* to convert delimiters back to spaces.

Alternatively, *bedmap* can accept *Starch-formatted archives* of BED data as input—it is no longer necessary to extract Starch archive data to intermediate BED files!

Support for common headers (including UCSC browser track headers) is available with the `--header` option, although headers are stripped from output.

Most importantly, *bedmap* expects *sorted* inputs. You can use the BEDOPS *sort-bed* program to ensure your inputs are properly sorted.

Note: You only need to sort once, and only if your input data are unsorted, as all BEDOPS tools take in and export sorted BED data.

Operations are applied over map elements that overlap the coordinates of each reference element. You can use the default overlap criterion of one base, or define your own criteria using the *overlap criteria operators*.

Once you have overlapping elements, you can either perform *numerical calculations* on their scores or return identifiers or other *non-score information*. Additional *modifier operators* allow customization of how output is presented, to assist with downstream processing in a pipeline setting.

Output

Depending on specified options, the *bedmap* program can send a variety of delimited information about the reference and mapped elements (as well as analytical results) to standard output. If the `--echo` option is used, the output will be at least a three-column BED file. The use of predictable delimiters (which are customizable) and the use of UNIX-like standard streams allows easy downstream analysis or post-processing with other tools and scripts.

Usage

The `--help` option describes the various mapping and analytical operations and other options available to the end user:

```
bedmap
  citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
  version:  2.4.37 (typical)
  authors:  Shane Neph & Scott Kuehn

 USAGE: bedmap [process-flags] [overlap-option] <operation(s)...> <ref-file> [map-
↪file]
     Any input file must be sorted per the sort-bed utility.
     The program accepts BED and Starch file formats.
     You may use '-' for a BED file to indicate the input comes from stdin.

     Traverse <ref-file>, while applying <operation(s)> on qualified, overlapping␣
↪elements from
       <map-file>.  Output is one line for each line in <ref-file>, sent to standard␣
↪output.  There
       is no limit on the number of operations you can specify to compute in one␣
↪bedmap call.
     If <map-file> is omitted, the given file is treated as both the <ref-file> and
↪<map-file>.
       This usage is more efficient than specifying the same file twice.
     Arguments may be given in any order before the input file(s).

    Process Flags:
     --------
      --chrom <chromosome>  Jump to and process data for given <chromosome> only.
      --delim <delim>       Change output delimiter from '|' to <delim> between␣
↪columns (e.g. '\t').
      --ec                  Error check all input files (slower).
      --faster              (advanced) Strong input assumptions are made.  Compatible␣
↪with:
                              --bp-ovr, --range, --fraction-both, and --exact overlap␣
↪options only.
      --header              Accept headers (VCF, GFF, SAM, BED, WIG) in any input␣
↪file.
      --help                Print this message and exit successfully.
      --min-memory          Minimize memory usage (slower).
      --multidelim <delim>  Change delimiter of multi-value output columns from ';'␣
↪to <delim>.
      --prec <int>          Change the post-decimal precision of scores to <int>.  0
↪<= <int>.
      --sci                 Use scientific notation for score outputs.
      --skip-unmapped       Print no output for a row with no mapped elements.
      --sweep-all           Ensure <map-file> is read completely (helps to prevent␣
↪broken pipes).
```

(continues on next page)

(continued from previous page)

```
      --unmapped-val <val>  Print <val> on unmapped --echo-map* and --min/max-
↪element* operations.
                              The default is to print nothing.
      --unmapped-val <val>  Use <val> in place of the empty string on unmapped --echo-
↪map* ops.
      --version             Print program information.

   Overlap Options (At most, one may be selected.  By default, --bp-ovr 1 is used):
    --------
      --bp-ovr <int>           Require <int> bp overlap between elements of input␣
↪files.
      --exact                  First 3 fields from <map-file> must be identical to
↪<ref-file>'s.
      --fraction-ref <val>     The fraction of the element's size from <ref-file>␣
↪that must overlap
                                 the element in <map-file>.  Expect 0 < val <= 1.
      --fraction-map <val>     The fraction of the element's size from <map-file>␣
↪that must overlap
                                 the element in <ref-file>.  Expect 0 < val <= 1.
      --fraction-both <val>    Both --fraction-ref <val> and --fraction-map <val>␣
↪must be true to
                                 qualify as overlapping.  Expect 0 < val <= 1.
      --fraction-either <val>  Either --fraction-ref <val> or --fraction-map <val>␣
↪must be true to
                                 qualify as overlapping.  Expect 0 < val <= 1.
      --range <int>            Grab <map-file> elements within <int> bp of <ref-file>
↪'s element,
                                 where 0 <= int.  --range 0 is an alias for --bp-ovr␣
↪1.

   Operations:  (Any number of operations may be used any number of times.)
    ----------
     SCORE:
      <ref-file> must have at least 3 columns and <map-file> 5 columns.

      --cv                The result of --stdev divided by the result of --mean.
      --kth <val>         Generalized median. Report the value, x, such that the␣
↪fraction <val>
                            of overlapping elements' scores from <map-file> is less␣
↪than x,
                            and the fraction 1-<val> of scores is greater than x.  0
↪< val <= 1.
      --mad <mult=1>      The median absolute deviation of overlapping elements in
↪<map-file>.
                            Multiply mad score by <mult>.  0 < mult, and mult is 1 by␣
↪default.
      --max               The highest score from overlapping elements in <map-file>.
      --max-element       A (non-random) highest-scoring and overlapping element in
↪<map-file>.
      --max-element-rand  A random highest-scoring and overlapping element in <map-
↪file>.
      --mean              The average score from overlapping elements in <map-file>.
      --median            The median score from overlapping elements in <map-file>.
      --min               The lowest score from overlapping elements in <map-file>.
      --min-element       A (non-random) lowest-scoring and overlapping element in
↪<map-file>.
      --min-element-rand  A random lowest-scoring and overlapping element in <map-
↪file>.
```

(continues on next page)

(continued from previous page)

```
      --stdev              The square root of the result of --variance.
      --sum                Accumulated scores from overlapping elements in <map-file>.
      --tmean <low> <hi>   The mean score from overlapping elements in <map-file>,␣
↪after
                             ignoring the bottom <low> and top <hi> fractions of those␣
↪scores.
                             0 <= low <= 1.  0 <= hi <= 1.  low+hi <= 1.
      --variance           The variance of scores from overlapping elements in <map-
↪file>.
      --wmean              Weighted mean, scaled in proportion to the coverage of the
↪<ref-file>
                             element by each overlapping <map-file> element.

    ----------
     NON-SCORE:
      <ref-file> must have at least 3 columns.
      For --echo-map-id/echo-map-id-uniq, <map-file> must have at least 4 columns.
      For --echo-map-score, <map-file> must have at least 5 columns.
      For all others, <map-file> requires at least 3 columns.

      --bases              The total number of overlapping bases from <map-file>.
      --bases-uniq         The number of distinct bases from <ref-file>'s element␣
↪covered by
                             overlapping elements in <map-file>.
      --bases-uniq-f       The fraction of distinct bases from <ref-file>'s element␣
↪covered by
                             overlapping elements in <map-file>.
      --count              The number of overlapping elements in <map-file>.
      --echo               Print each line from <ref-file>.
      --echo-map           List all overlapping elements from <map-file>.
      --echo-map-id        List IDs from all overlapping <map-file> elements.
      --echo-map-id-uniq   List unique IDs from overlapping <map-file> elements.
      --echo-map-range     Print genomic range of overlapping elements from <map-file>.
      --echo-map-score     List scores from overlapping <map-file> elements.
      --echo-map-size      List the full length of every overlapping element.
      --echo-overlap-size  List lengths of overlaps.
      --echo-ref-name      Print the first 3 fields of <ref-file> using chrom:start-
↪end format.
      --echo-ref-row-id    Print 'id-' followed by the line number of <ref-file>.
      --echo-ref-size      Print the length of each line from <ref-file>.
      --indicator          Print 1 if there exists an overlapping element in <map-file>
↪, 0 otherwise.
```

Operations

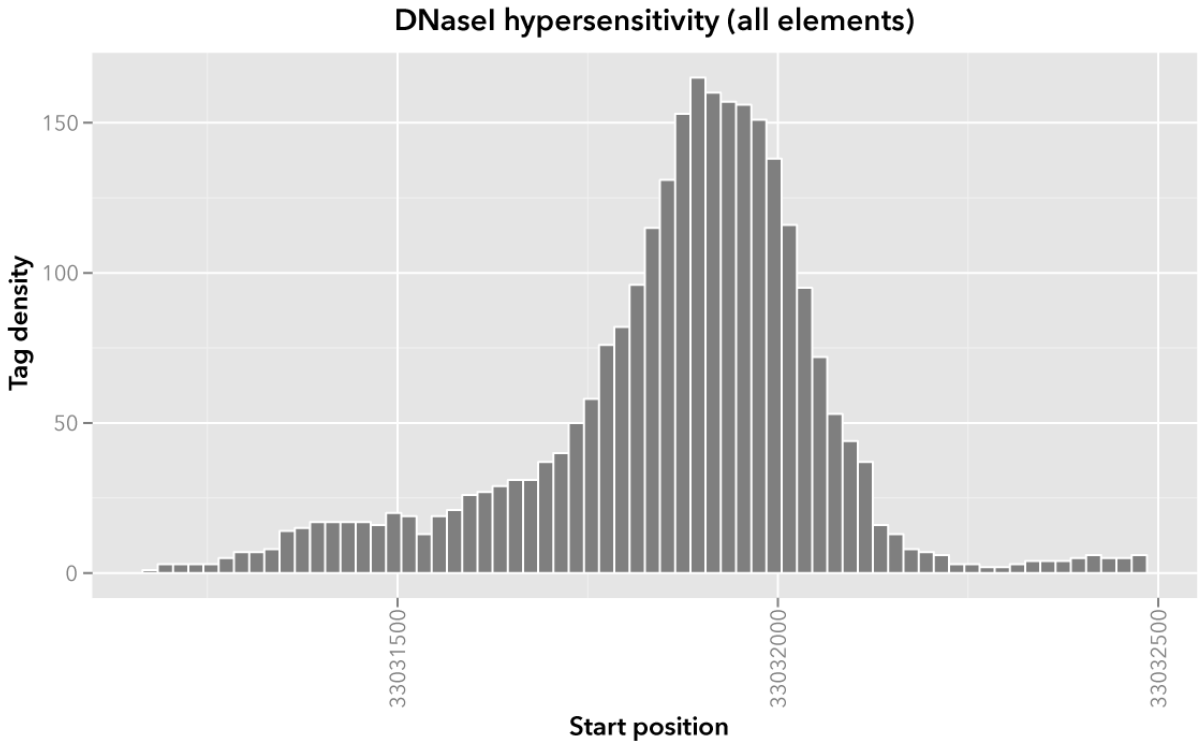
To demonstrate the various operations in *bedmap*, we start with two simple, pre-sorted BED files that we label as `Map` and `Reference` (see the *Downloads* section for files you can use to follow along).

Our `Map` file is a snippet of real-world BED data derived from ENCODE experiments conducted by our lab: specifically, raw DNaseI hypersensitivity signal for the human K562 cell line (region `chr21:33031165-33032485`, assembly `GRCh37/h19` and table `wgEncodeUwDnaseK562RawRep1` from the UCSC Genome Browser).

This raw signal is the density of sequence tags which map within a 150 bp sliding window, at 20 bp steps across the genome—a smoothed picture of DNaseI hypersensitivity:

```
chr21   33031165        33031185        map-1   1.000000
chr21   33031185        33031205        map-2   3.000000
chr21   33031205        33031225        map-3   3.000000
chr21   33031225        33031245        map-4   3.000000
chr21   33031245        33031265        map-5   3.000000
chr21   33031265        33031285        map-6   5.000000
chr21   33031285        33031305        map-7   7.000000
chr21   33031305        33031325        map-8   7.000000
chr21   33031325        33031345        map-9   8.000000
chr21   33031345        33031365        map-10  14.000000
chr21   33031365        33031385        map-11  15.000000
chr21   33031385        33031405        map-12  17.000000
chr21   33031405        33031425        map-13  17.000000
...
chr21   33032425        33032445        map-64  5.000000
chr21   33032445        33032465        map-65  5.000000
chr21   33032465        33032485        map-66  6.000000
```

When visualized, the signal data has the following appearance:



Note: Rectangles colored in grey represent each of the sixty-six `map` elements. The x-axis represents the start coordinate of the `map` element, while the y-axis denotes the tag density, or sum of tags over that element's 20-base window.

Our sample `Reference` file is not as exciting. It is just three BED elements which span portions of this density file:

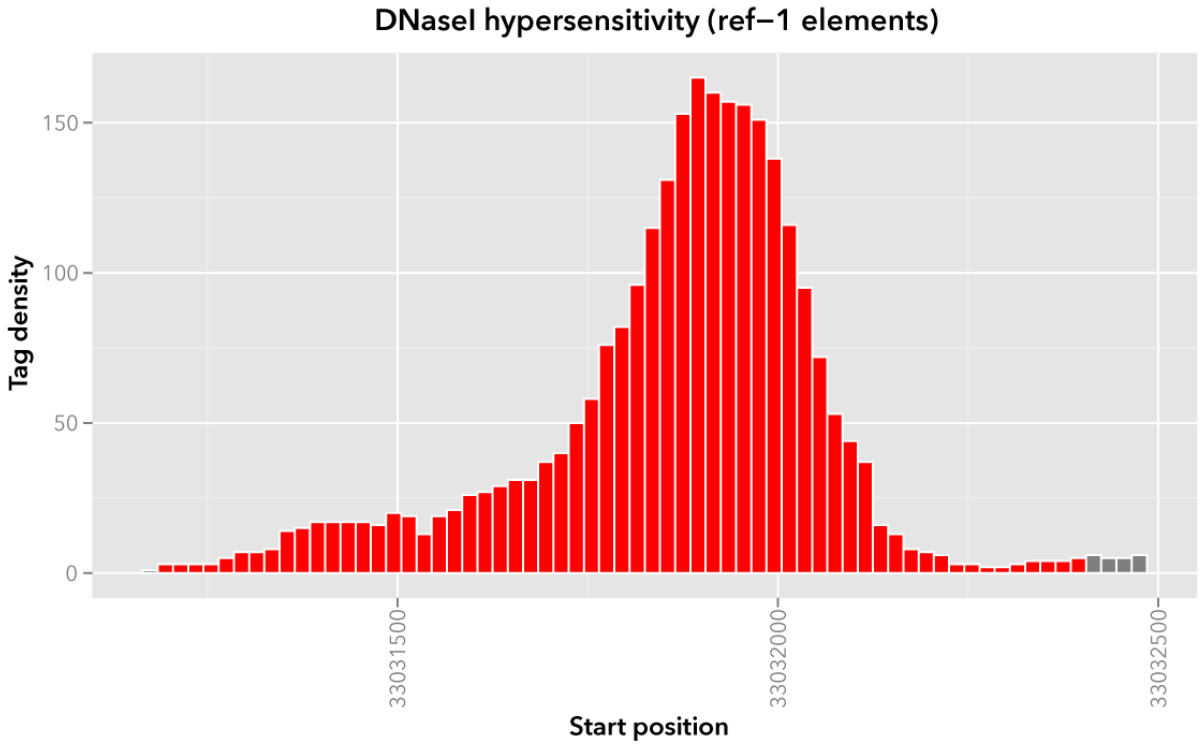
```
chr21   33031200    33032400    ref-1
chr21   33031400    33031800    ref-2
chr21   33031900    33032000    ref-3
```

These reference elements could be exons, promoter regions, etc. It doesn't matter for purposes of demonstration here, except to say that we can use *bedmap* to ask some questions about the `Reference` set.

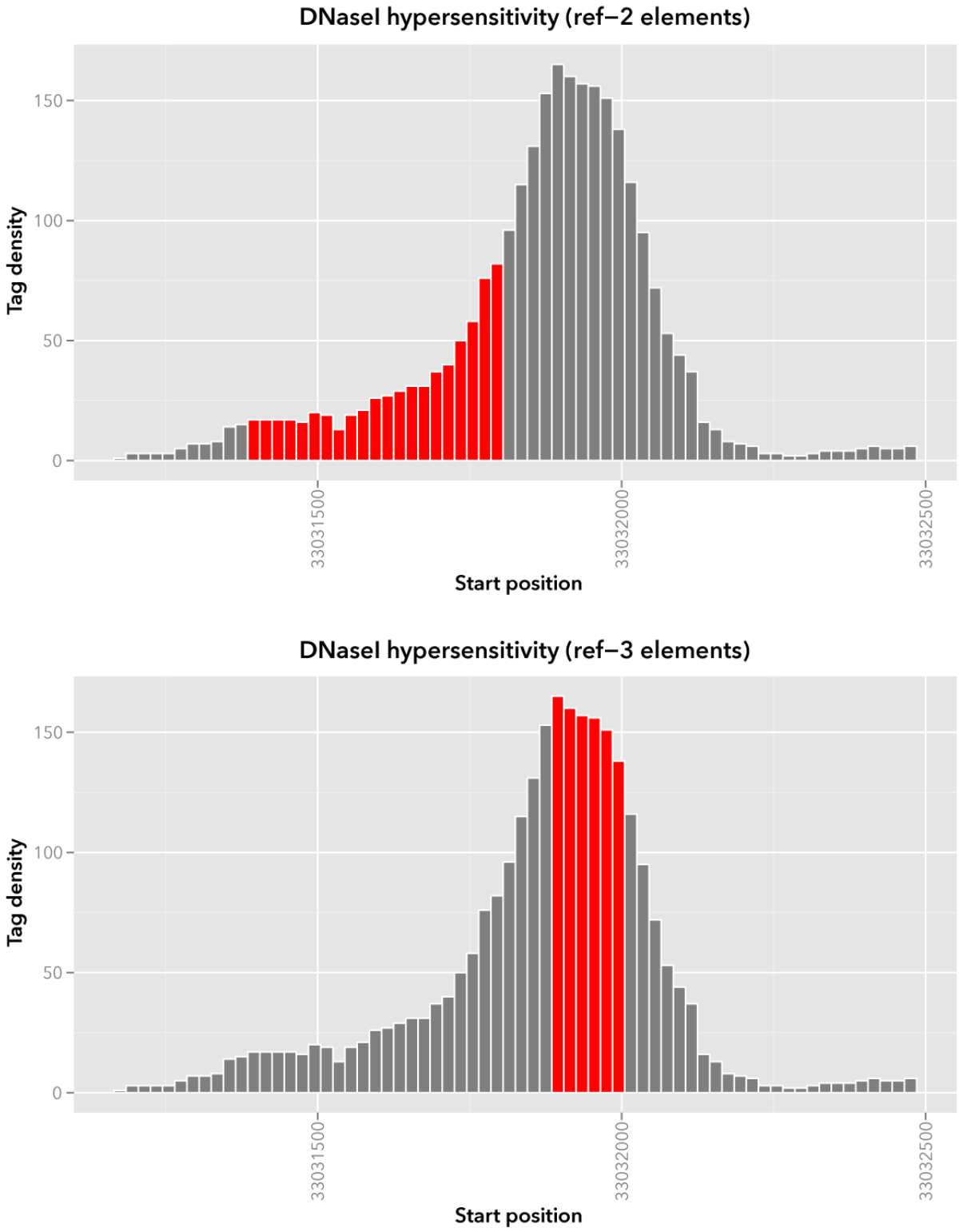
Among them, what are the quantitative and qualitative features of the `map` elements that span over these three reference regions? For example, we might want to know the mean DNase hypersensitivity across each—the answer may have some biological significance.

It may help to first visualize the reference regions and the mapped elements associated with them. A default *bedmap* task will operate on the following set of mapped (red-colored) elements, for each reference element `ref-1`, `-2` and `-3`.

Here we show elements from the `Map` set which overlap the `ref-1` region `chr21:33031200-33032400`, colored in red:



Likewise, here are elements of the `Map` set which overlap the `ref-2` element `chr21:33031400-33031800` and `ref-3` element `chr21:33031900-33032000`, respectively, with the same coloring applied:



In these sample files, we provide the `Map` file with ID and score columns, and the `Reference` file with an ID column. These extra columns are not required by *bedmap*, but we can use the information in these columns in conjunction with the options provided by *bedmap* to identify matches, retrieve matched signals, and summarize data about signal across mapped elements.

Overlap criteria

The default overlap criterion that *bedmap* uses is *one base*. That is, one or more bases of overlap between reference and mapping elements is sufficient for inclusion in operations. This value can be adjusted with the `--bp-ovr` option. The `--range` overlap option implicitly applies `--bp-ovr 1` after symmetrically padding elements.

If a fractional overlap is desired, the `--fraction-{ref,map,both,either}` options provide the ability to filter on overlap by a specified percentage of the length of either or both the reference and mapping elements.

Finally, the `--exact` flag enforces exact matches between reference and mapping elements.

Note: The `--exact` option is an alias for `--fraction-both 1`.

Using `--faster` with `--bp-ovr`, `--fraction-both`, `--exact` or `--range`

The `--faster` modifier works with the `--bp-ovr`, `--fraction-both` and `--exact` (`--fraction-both 1`) overlap and `--range` specifiers to dramatically increase the performance of *bedmap*, when the following input restriction is met:

- No *fully-nested elements* in any input mapping file (duplicate elements and other overlapping elements are okay).

Note: The details of this restriction are explained in more detail in the *nested element documentation*.

This option also works with the `--ec` error checking flag, which indicates if the data contain nested elements. Using `--ec` carries its usual overhead, but as it only doubles the much-improved execution time, it may be worth using.

Tip: To give an idea of the speed improvement, a `--range 100000 --echo --count` operation on 8.4 million, non-nested mapping elements (DNaseI footprints across multiple cell types) took *2 minutes and 55 seconds* without speed-up. By adding the `--faster` flag, the same calculation took *10 seconds*. That is an **18-fold** speed improvement.

One scenario where this option can provide great speed gains is where `--range` is used with a large numerical parameter. Another scenario where this option is very useful is where the reference file has large elements, and the mapping file is made up of many small elements—specifically, where a number of small elements overlap each big element from the reference file.

An example of a research application for our lab which benefits from this flag is where we perform statistical analysis of large numbers of small sequence tags that fall in hotspot regions.

If your data meet the *non-nesting criteria*, using `--faster` with `--bp-ovr`, `--fraction-both`, `--exact` or `--range` is *highly recommended*.

Note: Our lab works with BED data of various types: cut-counts, hotspots, peaks, footprints, etc. These data generally do not contain nested elements and so are amenable to use with *bedmap's* `--faster` flag for extracting overlapping elements.

However, other types of data can be problematic. FIMO search results, for example, may cause trouble, where the boundaries of one motif hit can be contained within another larger hit. Or paired-end sequence data, where tags are not of a fixed length.

Be sure to consider the makeup of your BED data before using `--faster`.

Tip: Using `--ec` with `--faster` will report if any nested elements exist in your data.

Score operations

Score operators apply a numerical calculation on the values of the score column of mapping elements. Per UCSC specifications, *bedmap* assumes the score data are stored in the fifth column.

The variety of score operators include common statistical measures:

- mean (`--mean`)
- trimmed mean (`--tmean`)
- weighted mean (`--wmean`)
- standard deviation (`--stdev`)
- variance (`--variance`)
- coefficient of variance (`--cv`)
- median (`--median`)
- median absolute deviation (`--mad`)
- k-th order statistic (`--kth`)

One can also take the sum of scores (`--sum`), find the minimum or maximum score over a region (`--min` and `--max`, respectively), or retrieve the map element with the least or greatest signal over the reference region (`--min-element` and `--max-element`, respectively).

Note: Map input should not contain spaces in the ID or in subsequent fields. Running *bedmap* with `--ec` will identify problematic input. Spaces in these fields will cause problems with `--min-element`, `--max-element`, and other options that require parsing of the fourth and subsequent columns of the map input.

An `awk` script can help with translating ID spaces to another non-whitespace delimiter, *e.g.*, `bedmap (--options...) reference.bed <(awk -vOFS="\t" -vFS="\t" '{gsub(" ", "%%", $4); print;}' map.bed)` and then *awk -vOFS="t" -vFS="t" '{gsub("%%", " "); print;}' result.bed* to convert delimiters back to spaces.

We will demonstrate some of these operators by applying them to the `Reference` and `Map` datasets (see the *Downloads* section for sample inputs).

As a reminder, the `Map` file contains regions of DNaseI-seq tag density. If we want the mean of the density across *Reference* elements, we use the `--mean` option:

```
$ bedmap --echo --mean reference.bed map.bed > mappedReferences.mean.bed
```

The `--echo` flag prints each `Reference` element, while the `--mean` flag calculates the mean signal of the `Map` elements which overlap the reference element:

```
$ more mappedReferences.mean.bed
chr21   33031200    33032400    ref-1|43.442623
chr21   33031400    33031800    ref-2|31.571429
chr21   33031900    33032000    ref-3|154.500000
```

This result tells us that the mean density across regions `ref-1`, `ref-2` and `ref-3` is `44.442623`, `31.571429` and `154.5`, respectively.

Note: The pipe character (`|`) delimits the results of each specified option (with the exception of the so-called "multi" operators that return multiple results — this is discussed in the section on `--echo` flags). In the provided example, the delimiter divides the reference element from the mean score across the reference element.

Tip: Because we used the `--echo` flag in this example, we are guaranteed output that is at least three-column BED format and which is *sorted*, which can be useful for pipeline design, where results are piped downstream to *bedmap*, *bedops* and other BEDOPS and UNIX utilities.

If we simply want the mean values and don't care about the reference data, we can skip `--echo`:

```
$ bedmap --mean reference.bed map.bed
43.442623
31.571429
154.500000
```

While not very detailed, this single-column representation can be useful for those who use UNIX utilities like `paste` or need to do additional downstream calculations with `R` or other utilities, where the reference information is unnecessary (or, at least, more work to excise).

If a reference element does not overlap any map element, then a `NAN` is returned for any operation on that entry, *e.g.*, we know that the *ad hoc* element `chr21:1000-2000` does not overlap any member of our `Map` dataset, and there is therefore no mean value that can be calculated for that element:

```
$ echo -e "chr21\t1000\t2000\tfoo-1" | bedmap --echo --mean - map.bed
chr21   1000    2000    foo-1|NAN
```

Tip: For this example, we use `echo -e` to send *bedmap* a sample reference coordinate by way of standard input. The *bedmap* program can process any BED data from the standard input stream, either as the reference or map data, by placing the dash character (`-`) where the file name would otherwise go.

In the example above, we sent *bedmap* a single reference element via standard input, but multiple lines of BED data can come from other upstream processes.

Using standard streams is useful for reducing file I/O and improving performance, especially in situations where one is using *bedmap* in the middle of an extended pipeline.

While *bedmap* returns a `NAN` if there are no mapped elements that associate with a reference element, we may want to filter these lines out. We can apply the `--skip-unmapped` option to leave out reference elements without mapped elements:

```
$ echo -e "chr21\t1000\t2000\tfoo-1" | bedmap --echo --mean --skip-unmapped - map.bed
$
```

Note: Some operations may yield a reference element with one or more mapped elements, which still return a `NAN` value by virtue of the calculation result. The `--skip-unmapped` operand will still allow these reference elements to be printed out!

For instance, consider the `--variance` operator, which requires two or more map elements to calculate a variance. Where there is only one mapped element associated with the reference element, a `--variance` calculation will

return a `NAN`. In this case, `--skip-unmapped` will still print this element, even though the result is `NAN`.

Given the following statement:

```
$ bedmap --skip-unmapped --variance file1 file2
```

This is functionally equivalent to the following statement:

```
$ bedmap --indicator --variance --delim "\t" file1 file2 | awk '($1==1) {print $2}'
```

The `--indicator` operand calculates whether there are any mapped elements (or none)—see the *indicator* section for more detail. The `awk` statement then prints results which have one or more mapped elements, effectively filtering unmapped references.

It should therefore be more convenient to use `--skip-unmapped` where unmapped reference elements are not needed.

Another option is to retrieve the mapping element with the highest or lowest score within the reference region, using the `--max-element` or `--min-element` operators, respectively.

Going back to our sample `Reference` and `Map` data, we can search for the highest scoring mapping elements across the three reference elements:

```
$ bedmap --echo --max-element --prec 0 reference.bed map.bed
chr21   33031200        33032400        ref-1|chr21     33031885        33031905
↪  map-37  165
chr21   33031400        33031800        ref-2|chr21     33031785        33031805
↪  map-32  82
chr21   33031900        33032000        ref-3|chr21     33031885        33031905
↪  map-37  165
```

Over reference elements `ref-1` and `ref-3`, the mapping element `map-37` has the highest score. Over reference element `ref-2`, the highest scoring mapping element is `map-32`.

Likewise, we can repeat this operation, but look for the lowest scoring elements, instead:

```
$ bedmap --echo --min-element --prec 0 reference.bed map.bed
chr21   33031200        33032400        ref-1|chr21     33032265        33032285
↪  map-56  2
chr21   33031400        33031800        ref-2|chr21     33031525        33031545
↪  map-19  13
chr21   33031900        33032000        ref-3|chr21     33031985        33032005
↪  map-42  138
```

Note: Where there are ties in score values, using `--max-element` or `--min-element` now selects the lexicographically smallest element amongst the set of tied elements. This generally means that the first element in the lexicographic ordering of the ID fields (fourth column) will determine the selection.

A random selection process was used for `--max-element` and `--min-element` in versions 2.4.20 and previous. If you wish to randomly sample a maximum- or minimum-scoring element from amongst tied elements (say, to reproduce the procedure of prior analyses), you may use the `--max-element-rand` or `--min-element-rand` options, respectively.

We can also perform multiple score operations, which are summarized on one line, *e.g.*, to show the mean, standard deviation, and minimum and maximum signal over each `Reference` element, we simply add the requisite options in series:

```
$ bedmap --echo --mean --stdev --min --max reference.bed map.bed
chr21   33031200    33032400    ref-1|43.442623|50.874527|2.000000|165.000000
chr21   33031400    33031800    ref-2|31.571429|19.638155|13.000000|82.000000
chr21   33031900    33032000    ref-3|154.500000|9.311283|138.000000|165.000000
```

Multiple score-operational results are ordered identically with the command-line options. The section on *formatting score output* demonstrates how one can change the precision and general format of numerical score results.

Non-score operations

Sometimes it is useful to get summary or non-score statistics about the map elements. This category of operators returns information from the ID column of mapping elements, or can return counts and base overlap totals.

Note: As with score data, we follow the UCSC convention for the BED format and retrieve ID data from the fourth column.

Echo

The ID, score and coordinate columns of the reference and map files are very useful for reading and debugging results, or reporting a more detailed mapping.

We can use the `--echo`, `--echo-map`, `--echo-map-id`, `--echo-map-id-uniq`, `--echo-map-score`, `--echo-map-range`, `--echo-map-size`, `--echo-overlap-size`, `--echo-ref-name`, `--echo-ref-row-id`, and `echo-ref-size` flags to tell *bedmap* to report additional details about the reference and map elements.

The `--echo` flag reports each reference element. We have already seen the application of `--echo` in earlier examples. The option helps to clearly associate results from other chosen operations with specific reference elements. Additionally, `--echo` enables the output from *bedmap* to be used as input to additional BEDOPS utilities, including *bedmap* itself.

The `--echo-map` flag gathers overlapping mapped elements for every reference element. The option is useful for debugging and detailed downstream processing needs. This is the most general operation in *bedmap* in that overlapping elements are returned in full detail, for every reference element. While results are well-defined and easily parsed, the output can be very large and difficult to read.

As an example of using the `--echo-map-id` operator in a biological context, we examine a FIMO analysis that returns a subset of transcription factor binding sites in BED format, with TRANSFAC motif names listed in the ID column:

```
chr1    4534161 4534177 -V_GRE_C        4.20586e-06     -       CGTACACACAGTTCTT
chr1    4534192.4.374205 -V_STAT_Q6      2.21622e-06     -       AGCACTTCTGGGA
chr1    4534209 4534223 +V_HNF4_Q6_01   6.93604e-06     +       GGACCAGAGTCCAC
chr1    4962522.4.372540 -V_GCNF_01      9.4497e-06      -       CCCAAGGTCAAGATAAAG
chr1    4962529 4962539 +V_NUR77_Q5     8.43564e-06     +       TTGACCTTGG
...
```

This input is available from the *Downloads* section as the `Motifs` dataset.

We will treat this as a map file, asking which motif IDs are associated with a region of interest (`chr1:4534150-4534300`). To do this using *bedmap*, we use the `--echo-map-id` option to summarize the IDs of mapped elements:

```
$ echo -e "chr1\t4534150\t4534300\tref-1" | bedmap --echo --echo-map-id - motifs.bed
chr1    4534150 4534300 ref-1|-V_GRE_C;-V_STAT_Q6;+V_HNF4_Q6_01
```

Note: To expand on the types of questions one can answer with *bedmap* in this context, in conjunction with the `--count` operator (described below), one can quantify predicted transcription factor binding sites by sliding a reference window across the entire genome.

One could determine, for example, where predicted sites are most prevalent and investigate the distribution of factors or other genomic features at or around these dense regions.

The `--echo-map-id-uniq` operator works exactly like `--echo-map-id`, except that duplicate IDs are removed from the result. For example, we can pull all the motifs hits from a wide region on `chr2`:

```
$ echo -e "chr2\t1000\t10000000\tref-1" | bedmap --echo --echo-map-id - motifs.bed
chr2    1000    10000000        ref-1|+V_OCT1_05;+V_OCT_C;-V_CACD_01;+V_IRF_Q6;-V_
↪BLIMP1_Q6;-V_IRF2_01;-V_IRF_Q6_01;+V_SMAD_Q6_01;-V_TATA_01;-V_TATA_C;-V_CEBP_01;-V_
↪HNF6_Q6;+V_MTF1_Q4;+V_MYOD_Q6_01;-V_KROX_Q6;+V_EGR1_01;-V_SP1SP3_Q4;+V_EGR_Q6;+V_
↪SP1_Q6;-V_SP1_Q2_01;-V_CKROX_Q2;+V_SP1_Q6_01;-V_SREBP1_Q5;+V_VDR_Q3;-V_DMRT1_01;-V_
↪DMRT7_01;+V_DMRT1_01;-V_DMRT1_01;-V_DMRT7_01;+V_DMRT1_01;-V_DMRT1_01;-V_DMRT7_01
```

However, some hits (*e.g.*, `-V_DMRT7_01`) show up two or more times. If we want a non-redundant list, we replace `--echo-map-id` with `--echo-map-id-uniq`:

```
$ echo -e "chr2\t1000\t10000000\tref-1" | bedmap --echo --echo-map-id-uniq - motifs.
↪bed
chr2    1000    10000000        ref-1|+V_DMRT1_01;+V_EGR1_01;+V_EGR_Q6;+V_IRF_Q6;+V_
↪MTF1_Q4;+V_MYOD_Q6_01;+V_OCT1_05;+V_OCT_C;+V_SMAD_Q6_01;+V_SP1_Q6;+V_SP1_Q6_01;+V_
↪VDR_Q3;-V_BLIMP1_Q6;-V_CACD_01;-V_CEBP_01;-V_CKROX_Q2;-V_DMRT1_01;-V_DMRT7_01;-V_
↪HNF6_Q6;-V_IRF2_01;-V_IRF_Q6_01;-V_KROX_Q6;-V_SP1SP3_Q4;-V_SP1_Q2_01;-V_SREBP1_Q5;-
↪V_TATA_01;-V_TATA_C
```

The `--echo-map-score` flag works in a similar fashion to `--echo-map-id`, reporting scores instead of IDs. The *formatting score output* section demonstrates how one can use `--echo-map-score` to summarize score data from mapped elements.

Note: Both the `--echo-map-id` and `--echo-map-score` flags use the semi-colon (`;`) as a default delimiter, which may be changed with the `--multidelim` option (see the *Delimiters* section for more information on this and other modifier operators).

The `--echo-map-range` flag tells *bedmap* to report the genomic range of overlapping mapped elements. If we apply this flag to the `Reference` and `Map` datasets (see *Downloads*), we get the following result:

```
$ bedmap --echo --echo-map-range reference.bed map.bed
chr21   33031200    33032400    ref-1|chr21 33031185    33032405
chr21   33031400    33031800    ref-2|chr21 33031385    33031805
chr21   33031900    33032000    ref-3|chr21 33031885    33032005
```

Note: The `--echo-map-range` option produces three-column BED results that are not always guaranteed to be sorted. The `--echo` operation is independent, and it produces reference elements in proper BEDOPS order, as shown. If the results of the `--echo-map-range` option will be used directly as BED coordinates in downstream BEDOPS analyses (*i.e.*, no `--echo` operator), first pipe them to *sort-bed* to ensure proper sort order.

Note: Map input should not contain spaces in the ID or in subsequent fields. Running *bedmap* with `--ec` will identify problematic input. Spaces in these fields will cause problems with `--echo-map-id`, `--echo-map-id-uniq`, and other options that require parsing of the fourth and subsequent columns of the map input.

An `awk` script can help with translating ID spaces to another non-whitespace delimiter, *e.g.*, `bedmap (--options...) reference.bed <(awk -vOFS="\t" -vFS="\t" '{gsub(" ", "%%", $4); print;}' map.bed)` and then *awk -vOFS="t" -vFS="t" '{gsub("%%", " "); print;}' result.bed* to convert delimiters back to spaces.

The `--echo-ref-size` flag reports the difference between the stop and start coordinates of the reference element. The `--echo-ref-name` flag produces a converted format for the first 3 BED fields, A:B-C, where A is the chromosome name, B is the start coordinate, and C is the stop coordinate for that reference element.

The `--echo-ref-row-id` flag prints the prefix `id-` with the line number of the reference element.

Finally, the `--echo-map-size` flag reports the difference between the stop and start coordinates of each mapped element, while the `--echo-overlap-size` flag reports the length of the overlap between the reference element and each mapped element.

Element and overlap statistics

Looking back at the `Map` and `Reference` datasets, let's say we want to count the number of elements in `Map` that overlap a given `Reference` element, as well as the extent of that overlap as measured by the total number of overlapping bases from mapped elements. For this, we use the `--count` and `--bases` flags, respectively:

```
$ bedmap --echo --count --bases reference.bed map.bed
chr21   33031200    33032400    ref-1|61|1200
chr21   33031400    33031800    ref-2|21|400
chr21   33031900    33032000    ref-3|6|100
```

This result tells us that there are 61 elements in `Map` that overlap `ref-1`, and 1200 total bases from the 61 elements overlap bases of `ref-1`. Similarly, 21 elements overlap `ref-2`, and 400 total bases from the 21 elements overlap bases of `ref-2`, etc.

The `--bases` operator works on `Map` elements. If, instead, we want to quantify the degree to which `Reference` elements overlap `Map` , we can use the `--bases-uniq` and `--bases-uniq-f` flags to count the number of bases and, respectively, the fraction of total bases within `Reference` which are covered by overlapping elements in `Map`.

This last example uses `Motifs` elements and all of the options: `--bases`, `--bases-uniq` and `--bases-uniq-f`, to illustrate their different behaviors:

```
$ echo -e "chr1\t4534161\t4962550\tadhoc-1" | bedmap --echo --bases --bases-uniq --
↪bases-uniq-f - motifs.bed
chr1    4534161     4962550     adhoc-1|169|71|0.000166
```

Indicator

If we simply want to know if a reference element overlaps one or more map elements, we can use the `--indicator` operator, which returns a `1` or `0` value, depending on whether there is or is not an overlap, respectively. For example:

```
$ bedmap --echo --indicator reference.bed map.bed
chr21   33031200    33032400    ref-1|1
chr21   33031400    33031800    ref-2|1
chr21   33031900    33032000    ref-3|1
```

All three of our reference elements have mapped elements associated with them. If we, instead, test a reference element that we know ahead of time does not contain overlapping map elements, we get a `0` result, as we expect:

```
$ echo -e "chr21\t1000\t2000\tfoo-1" | bedmap --echo --indicator - map.bed
chr21   1000    2000    foo-1|0
```

Note: The `--indicator` option is equivalent to testing if the result from `--count` is equal to or greater than `0`:

```
$ bedmap --count foo bar | awk '{ print ($1 > 0 ? "1" : "0") }' -
```

This option eliminates the need for piping *bedmap* results to `awk`.

Modifiers

These options can modify the coordinates used for generating the set of mapped regions, as well as alter the presentation of results. These modifiers can be useful, depending on how *bedmap* is used in your own workflow.

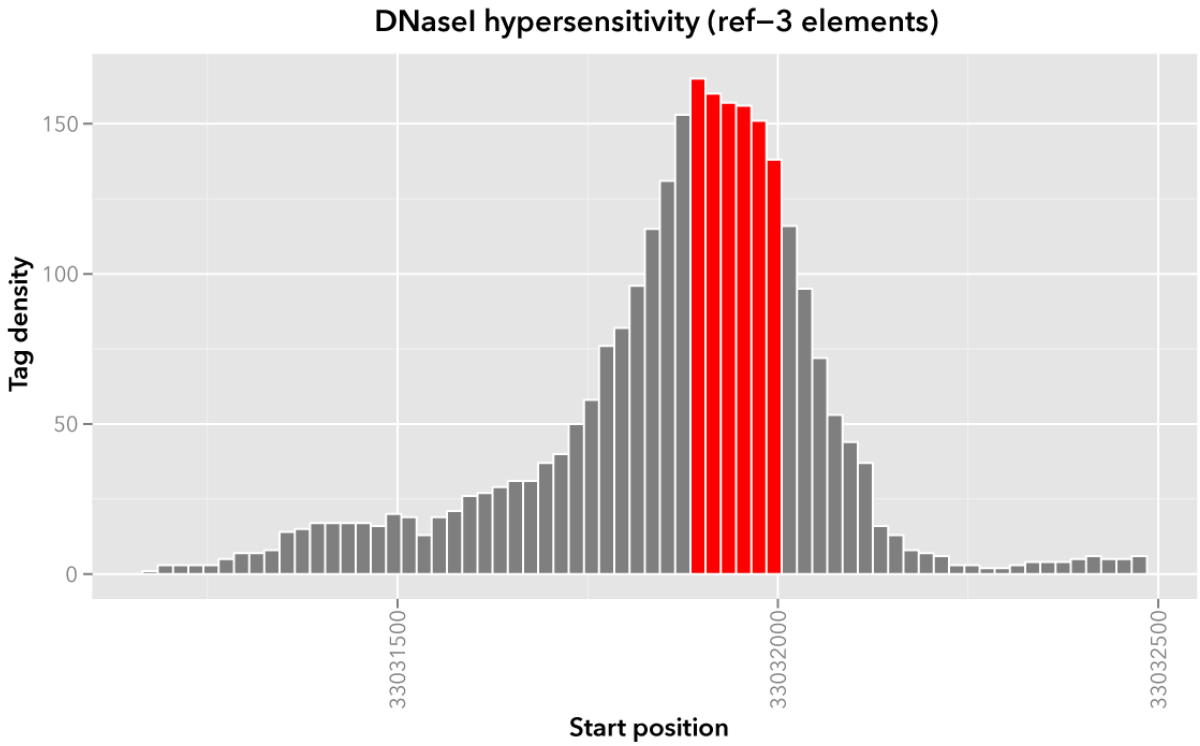
Range

The `--range` option uses `--bp-ovr 1` (*i.e.*, *one base of overlap*) after internally and symmetrically padding reference coordinates by a specified positive integer value. The larger reference elements are used to determine overlapping mapped elements, prior to applying chosen operations.

Tip: To change the coordinates of a BED file on *output* (symmetrically or asymmetrically), see the `--range` option applied with *bedops –everything*.

As an example, we look again at element `ref-3` from the `Reference` dataset and where it overlaps with `Map`:

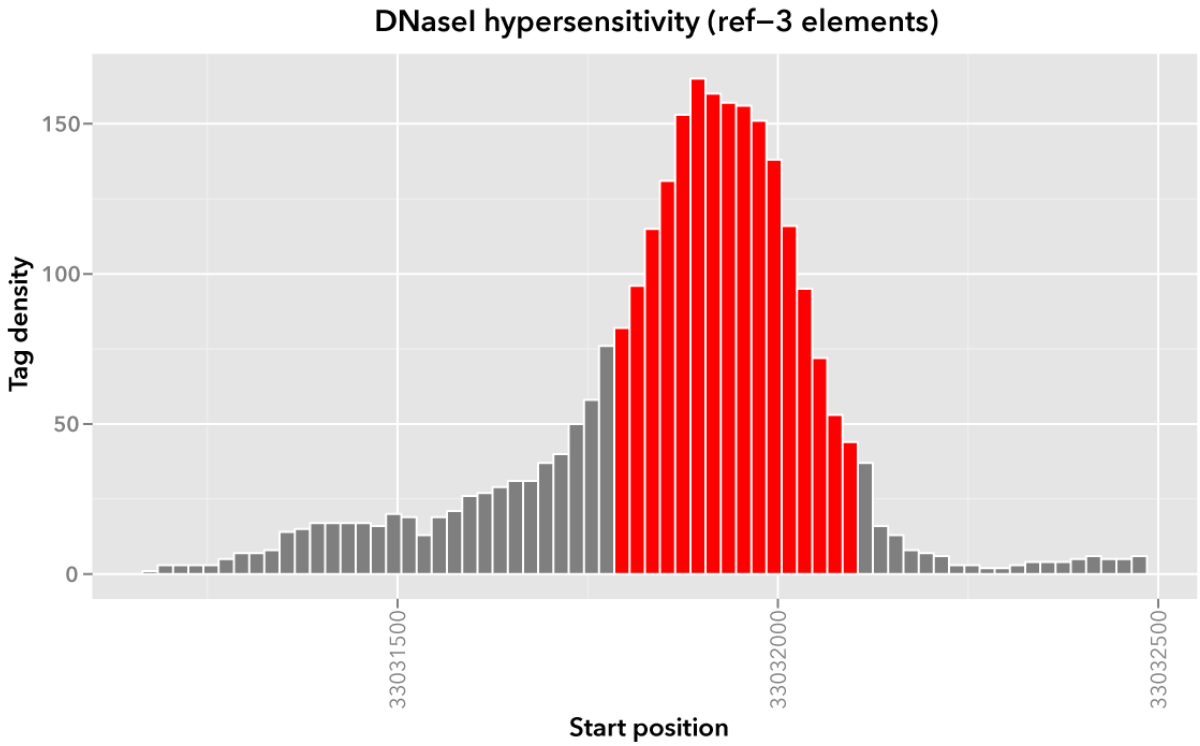
```
chr21   33031900    33032000    ref-3
```



If we want to apply an operation on 100 bp upstream and downstream of this and the other reference elements, we can use the `--range` option:

```
$ bedmap --echo --echo-map-id --range 100 reference.bed map.bed > mappedReference.
↪padded.bed
```

Any operation will now be applied to a broader set of mapped elements, as visualized here with a "padded" version of `ref-3`:



We can compare mean densities, in order to see the effect of using `--range`. Here is the mean density across the original, unpadded `ref-3`:

```
$ bedmap --echo --mean reference.bed map.bed
...
chr21   33031900    33032000    ref-3|154.500000
```

And here is the mean density across the padded `ref-3`:

```
$ bedmap --echo --range 100 --mean reference.bed map.bed
...
chr21   33031900    33032000    ref-3|117.750000
```

Looking at the visualizations above, we would expect the mean density to be lower, as the expanded reference region includes map elements with lower tag density, which pushes down the overall mean.

Note: The `--range` option is classified as an overlap option (like `--fraction-map` or `--exact`) that implicitly uses `--bp-ovr 1` after padding reference elements. As shown above, the extended padding is an internal operation and it is not reflected in the output with the `--echo` option. Real padding can be added by using `bedops --range 100 --everything reference.bed` and piping results to *bedmap*.

Note: Because `--range` is an internal operation, some statistical operations like `--bases` and `--bases-uniq` do not work as one might expect.

As an example, we might want to count the number of overlapping, unique bases between a 1000-base window around a reference element and a set of mapped elements. The following command will not work:

```
$ bedmap --echo --range 1000 --bases-uniq reference.bed map.bed
```

Instead, use *bedops* to build the window, piping it into a downstream *bedmap* command. The result of this operation can be piped into the core utility `paste` with the original reference set, in order to associate reference elements with the windowed operation result:

```
$ bedops --range 1000 --everything reference.bed \
    | bedmap --bases-uniq - map.bed \
    | paste reference.bed -
```

To extend this demonstration even further, let's say we are interested in calculations of unique base counts across 1, 2.5 and 5 kilobase windows around reference elements. We can build a matrix-like result through a judicious use of UNIX pipes that progressively expand windows:

```
$ bedops --range 1000 --everything reference.bed \
  | bedmap --echo --bases-uniq - map.bed \
  | bedops --range 1500 --everything - \
  | bedmap --echo --bases-uniq - map.bed \
  | bedops --range 2500 --everything - \
  | bedmap --echo --bases-uniq - map.bed \
  | cut -f2- -d'|' \
  | paste reference.bed - \
  | tr '|' '\t'
```

To explain how this works, we first build a 1 kilobase window around reference elements with *bedops* and pipe these windows to *bedmap*, which does two things:

1. Use `--echo` to print the windowed element.
2. Use `--bases-uniq` to print the number of uniquely-mapped bases across the window.

In turn, this result is passed to the second *bedops* operation, which expands the 1-kilobase window from *bedmap* by another 1.5 kilobases. This creates a window that is now 2.5 kilobases around the original reference element. We pipe this to the second *bedmap* operation, which prints the 2.5 kb window and the number of bases across *that* window.

In the third and last round of operations, we expand the 2.5 kb window by another 2.5 kb, creating a 5000-base window around the original reference element. We repeat the same mapping operation.

At this point, each line of the output consists of a windowed reference element, and pipe characters (the default *bedmap* delimiter) which separate the unique base counts across the 1, 2.5 and 5 kilobase windows. The final `cut`, `paste` and `tr` operations strip out the windows, paste in the original reference elements and replace default delimiters with tab characters, creating a matrix-like output.

To make this analysis run quickly, use the `--faster` modifier on each of the *bedmap*, if the data allow it. See the following section for more details on where and how `--faster` can be used.

Using `--faster` with `--range`

The `--faster` modifier works with the `--bp-ovr`, `--fraction-both` and `--exact` (`--fraction-both 1`) overlap and `--range` specifiers to dramatically increase the performance of *bedmap*, when the following input restriction is met:

- No *fully-nested elements* in any input mapping file (duplicate elements and other overlapping elements are okay).

Note: The details of this restriction are explained in more detail in the *nested element documentation*.

This option also works with the `--ec` error checking flag, which indicates if the data contain nested elements. Using `--ec` carries its usual overhead, but as it only doubles the much-improved execution time, it may be worth using.

Tip: To give an idea of the speed improvement, a `--range 100000 --echo --count` operation on 8.4 million, non-nested mapping elements (DNaseI footprints across multiple cell types) took *2 minutes and 55 seconds* without speed-up. By adding the `--faster` flag, the same calculation took *10 seconds*. That is an **18-fold** speed improvement.

One scenario where this option can provide great speed gains is where `--range` is used with a large numerical parameter. Another scenario where this option is very useful is where the reference file has large elements, and the mapping file is made up of many small elements—specifically, where a number of small elements overlap each big element from the reference file.

An example of a research application for our lab which benefits from this flag is where we perform statistical analysis of large numbers of small sequence tags that fall in hotspot regions.

If your data meet the *non-nesting criteria*, using `--faster` with `--bp-ovr`, `--exact` or `--range` is **highly recommended**.

Note: Our lab works with BED data of various types: cut-counts, hotspots, peaks, footprints, etc. These data generally do not contain nested elements and so are amenable to use with *bedmap's* `--faster` flag for extracting overlapping elements.

However, other types of data can be problematic. FIMO search results (motif hits), for example, may cause trouble, where the boundaries of one motif hit can be contained within another larger hit. Or paired-end sequence data, where tags are not of a fixed length.

Be sure to consider the makeup of your BED data before using `--faster`.

Tip: Using `--ec` with `--faster` will report if any nested elements exist in your data. Using `--ec` carries its usual overhead, but as it only doubles the much-improved execution time, it may be worth using.

Formatting score output

The `--prec` and `--sci` process flags are useful for controlling the arithmetic precision and notation of score output, when used with the `--echo-map-score`, `--sum`, `--mean` and other numerical score operators. This will also format results from the non-score operator `--bases-uniq-f`.

To demonstrate their use, we revisit the `Motifs` dataset, which includes *p*-values reporting the statistical significance of putative transcription factor binding sites:

```
chr1    4534161 4534177 -V_GRE_C        4.20586e-06     -       CGTACACACAGTTCTT
chr1    4534192.4.374205 -V_STAT_Q6      2.21622e-06     -       AGCACTTCTGGGA
chr1    4534209 4534223 +V_HNF4_Q6_01   6.93604e-06     +       GGACCAGAGTCCAC
chr1    4962522.4.372540 -V_GCNF_01      9.4497e-06      -       CCCAAGGTCAAGATAAAG
chr1    4962529 4962539 +V_NUR77_Q5     8.43564e-06     +       TTGACCTTGG
...
```

Let's say we want a list of motifs and associated *p*-values mapped to a coordinate range of interest (`chr1:4534150-4534300`). In order to conserve space, however, we only want two significant figures for the score data. So we use `--prec 2` to try to reformat the score output:

```
$ echo -e "chr1\t4534150\t4534300\tref-1" \
    | bedmap --prec 2 --echo --echo-map-id --echo-map-score - motifs.bed \
    > motifsForRef1.bed
```

Here is the output:

```
chr1    4534150 4534300 ref-1|-V_GRE_C;-V_STAT_Q6;+V_HNF4_Q6_01|0.00;0.00;0.00
```

It looks like our *p*-values were rounded down to zeroes, which is not what we want. But we remember that the binding site *p*-values are listed in scientific notation, and so we add the `--sci` flag to preserve the format of the score data in scientific notation:

```
$ echo -e "chr1\t4534150\t4534300\tref-1" \
    | bedmap --prec 2 --sci --echo --echo-map-id --echo-map-score - motifs.bed \
    > correctedMotifsForRef1.bed
```

Here is the corrected output:

```
chr1    4534150 4534300 ref-1|-V_GRE_C;-V_STAT_Q6;+V_HNF4_Q6_01|4.21e-06;2.22e-06;6.
↪94e-06
```

Rounding of the mantissa is done to the precision specified in `--prec`.

Obviously, the `--sci` flag is useful for very small or large score data. You probably wouldn't use `--sci` with most integer signal (*e.g.*, raw tag counts or most discrete measurements).

Delimiters

As shown in the examples above, the pipe (`|`) and semi-colon (`;`) characters are used to split operational and `echo`-ed results, respectively. The `--delim` and `--multidelim` flags change these delimiters to characters of your choice, which let you pick what makes most sense for your custom post-processing or other downstream pipelining work (for instance, in our lab `--delim "\t"` is a popular alternative to the default `|` character).

As an example, the following *bedmap* result is obtained from using the `--echo`, `--echo-map-id`, `--echo-map-score` and `--max` options on the `Motifs` dataset:

```
chr1    4534150 4534300 ref-1|-V_GRE_C;-V_STAT_Q6;+V_HNF4_Q6_01|4.21e-06;2.22e-06;6.
↪94e-06|6.94e-06
```

For this result, the *bedmap* program organizes data using the default set of delimiters:

```
[reference-line] | [map-IDs] | [map-scores] | [maximum-map-score]
```

Here, you can use the `--delim` option to replace the pipe character with an alternative delimiter.

Within the `map-IDs` and `map-scores` subgroups, individual results are split further by semi-colon:

```
[id-1] ; [id-2] ; ... ; [id-N]
```

```
[score-1] ; [score-2] ; ... ; [score-N]
```

You can use the `--multidelim` option to replace the semi-colon with another delimiter, *e.g.*:

```
$ echo -e "chr1\t4534150\t4534300\tref-1" | bedmap --multidelim '$' --echo --echo-map-
↪id - motifs.bed
chr1    4534150 4534300 ref-1|-V_GRE_C$-V_STAT_Q6$+V_HNF4_Q6_01
```

Note: Grouped results derived with the `--echo-map`, `--echo-map-id`, and `--echo-map-score` options are listed in identical order. In other words, ID results line up at the same position as their score result counterparts when

both `--echo-map-id` and `--echo-map-score` are chosen together. The same applies to the `--echo-map` option.

I/O event handling

During normal use of `bedmap`, the application will usually terminate when it is determined that no more map data needs to be processed. This improves performance by limiting execution time to only that which is required to do actual work. However, closing early can trigger `SIGPIPE` or broken pipe errors that can cause batch scripts that use the standard input stream to pass data to `bedmap` to terminate early with an error state (even though there is often no functional problem from this early termination of `bedmap`).

When adding `--ec`, `bedmap` will go into *error checking mode* and read through the entire map dataset.

One method for dealing with this is to override how `SIGPIPE` errors are caught by the interpreter (bash, Python, etc.) and retrapping them or ignoring them. However, it may not a good idea to do this as other situations may arise in production pipelines where it is ideal to trap and handle all I/O errors in a default manner.

Until now, we have proposed using the `--ec` (error checking) option in `bedmap` as one way to prevent raising `SIGPIPE` events when chaining commands via pipes, by forcing all inputs to be read entirely. Early pipe termination can cause scripts to stop processing when certain flags are set (for example, when -e is used with tcsh). This hidden behavior of `--ec` has been replaced with the explicit option `--sweep-all`.

The `--ec` and `--sweep-all` options work independently, and `--ec` no longer has the `--sweep-all` side-effect. These options may be used in conjunction. The `--sweep-all` option can add significant execution time in cases where early termination is possible.

Per-chromosome operations (–chrom)

All operations on inputs described so far can be restricted to one chromosome, by adding the `--chrom <val>` operator. This is highly useful for cluster-based work, where operations on large BED inputs can be split up by chromosome and pushed to separate cluster nodes.

Here, we use the `--echo` and `--echo-map-id` operators on our `Motifs` dataset, but we limit operations to those on elements on chromosome `chr2`:

```
$ echo -e "chr2\t1000000\t5000000\tref-1" | bedmap --chrom chr2 --echo --echo-map-id -
↪ motifs.bed
chr2    1000000 5000000 ref-1|+V_OCT1_05;+V_OCT_C;-V_CACD_01;+V_IRF_Q6;-V_BLIMP1_Q6;-
↪V_IRF2_01;-V_IRF_Q6_01
```

If the reference elements are not on the specified chromosome provided to `--chrom`, then no output is generated. In the following example, our reference element is on `chr2`, but we ask for operations to be limited to `chr3`, yielding an empty set:

```
$ echo -e "chr2\t1000000\t5000000\tref-1" | bedmap --chrom chr3 --echo --echo-map-id -
↪ motifs.bed
$
```

Starch support

The *bedmap* application supports use of *Starch-formatted archives* as inputs, as well as text-based BED data. One or multiple inputs may be Starch archives.

For example, we can repeat the overlapping-motif example from the *Echo section*, using a Starch archive made from the regions in `Motifs`:

```
$ echo -e "chr1\t4534150\t4534300\tref-1" | bedmap --echo --echo-map-id - motifs.bed.
↪starch
chr1    4534150 4534300 ref-1|-V_GRE_C;-V_STAT_Q6;+V_HNF4_Q6_01
```

By combining the `--chrom` operator with operations on Starch archives, the end user can achieve improved computing performance and disk space savings, particularly where *bedops*, *bedmap* and *closest-features* operations are applied with a computational cluster on separate chromosomes.

Error checking

The bedmap program does not perform error checking by default, but it offers an `--ec` option for comprehensive checks.

Note: Use of the `--ec` option will roughly double the running time, but it provides stringent error checking to ensure all inputs are valid. `--ec` can help check problematic input and offers helpful hints for any needed corrections, when problems are detected.

Endlines

The *bedmap* program expects endlines (`\n`) appropriate to Linux and Mac OS X operating systems. Microsoft Windows uses different characters for endlines. In UNIX-like environments, you can quickly check to see if your file contains the native endlines with this command:

```
$ head myData.bed | cat -et
```

The appropriate endlines will show up as a `$` character at the end of each line. See the `dos2unix` program (sometimes called `fromdos`) to convert newlines from files saved on Microsoft Windows. The `unix2dos` (or `todos`) program can convert files in the other direction, if needed.

Downloads

- Sample `Reference` dataset: `reference elements`
- Sample `Map` dataset: `map elements`
- Sample `Motifs` dataset: `motif elements`

2.6.3 File management

Sorting

sort-bed

The `sort-bed` utility sorts BED files of any size, even larger than system memory. BED files that are in lexicographic-chromosome order allow BEDOPS utilities to work efficiently with data from any species without software modifications. Further, sorted files can be traversed very quickly.

Sorted BED order is defined first by lexicographic chromosome order, then ascending integer start coordinate order, and finally by ascending integer end coordinate order. To make the sort order unambiguous, a lexicographical sort is applied on fourth and subsequent columns, where present in the input BED dataset.

Other utilities in the BEDOPS suite require data in sorted order as described. You only need to sort once: BEDOPS utilities all read and write data in sorted order.

Migrating older BED and Starch files

The utility `update-sort-bed-migrate-candidates` recursively locates BED and pre-v2.4.20 Starch files in the specified parent directory, tests if they require re-sorting to conform to the updated, post-v2.4.20 'sort-bed' order, and offers actions to log candidate files, or immediately apply a resort action that is performed locally or via a SLURM-managed cluster.

The convenience utilities `update-sort-bed-slurm` and `update-sort-bed-starch-slurm` update the sort order of BED or Starch files sorted with pre-v2.4.20 `sort-bed` via a SLURM-based cluster. See `update-sort-bed-slurm --help` or `update-sort-bed-starch-slurm --help` for more details. These utilities can be used standalone or in conjunction with the `update-sort-bed-migrate-candidates` utility.

Inputs and outputs

Input

The `sort-bed` utility requires one or more three-column BED file(s). Support for common headers (such as UCSC BED track headers) is included, although headers will be stripped from the output.

Output

The `sort-bed` utility sends sorted BED data to standard output, which can be redirected to a file or piped to other utilities, including core BEDOPS utilities like *bedops* and *bedmap*. Sort order is defined by a lexicographical sort on chromosome name, a numerical sort on start coordinates, a numerical sort on stop coordinates where there are start matches, and finally a lexicographical sort on the remainder of the BED element (if additional columns are present). Additional options may be specified to print only unique or duplicate elements, or check the sort order of input.

Usage

The `--help` option is fairly basic, but describes the usage:

```
sort-bed
  citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
  version:  2.4.37 (typical)
  authors:  Scott Kuehn

USAGE: sort-bed [--help] [--version] [--check-sort] [--max-mem <val>] [--tmpdir <path>
↪] [--unique] [--duplicates] <file1.bed> <file2.bed> <...>
        Sort BED file(s).
        May use '-' to indicate stdin.
        Results are sent to stdout.

        <val> for --max-mem may be 8G, 8000M, or 8000000000 to specify 8 GB of memory.
```

(continues on next page)

(continued from previous page)

```
        --tmpdir is useful only with --max-mem.
        --unique can be used to print only unique BED elements (similar to "sort -u").
        --duplicates can be used to print only duplicated or repeated elements␣
↪(similar to "uniq -d").
```

A simple example of using `sort-bed` would be:

```
$ sort-bed unsortedData.bed > sortedData.bed
```

The `sort-bed` program efficiently sorts BED inputs. By default, all input records are read into system memory and sorted. If your BED dataset is larger than available system memory, use the `--max-mem` option to limit the amount of memory `sort-bed` uses to do its work:

```
$ sort-bed --max-mem 2G reallyHugeUnsortedData.bed > reallyHugeSortedData.bed
```

This option allows `sort-bed` to scale to input of any size.

The `--tmpdir` option allows specification of an alternative temporary directory, when used in conjunction with `--max-mem` option. This is useful if the host operating system's standard temporary directory (*e.g.*, `/tmp` on Linux or OS X) does not have sufficient space to hold intermediate results.

For example, to use the current working directory to store temporary data, one could use the `$PWD` environment variable:

```
$ sort-bed --max-mem 2G --tmpdir $PWD reallyHugeUnsortedData.bed >␣
↪reallyHugeSortedData.bed
```

Use of the `--check-sort` option returns a message if the input is sorted, or not.

The `--unique` and `--duplicates` options print only unique or duplicated elements in sorted output, respectively. These options mimic `sort -u` and `uniq -d` commands, respectively.

Compression

starch

With high-throughput sequencing generating large amounts of genomic data, archiving can be a critical part of an analysis toolkit. BEDOPS includes the `starch` utility to provide a method for efficient and lossless compression of UCSC BED-formatted data into the *Starch v2 format*.

Starch v2 archives can be extracted with *unstarch* to recover the original BED input, or processed as inputs to *bedops* and *bedmap*, where set operations and element calculations can be performed directly and without the need for intermediate file extraction.

The *starch* utility includes large file support on 64-bit operating systems, enabling compression of more than 2 GB of data (a common restriction on 32-bit systems).

Data can be stored with one of two open-source backend compression methods, either `bzip2` or `gzip`, providing the end user with a reasonable tradeoff between speed and storage performance that can be useful for working with constrained storage situations or slower hardware.

Inputs and outputs

Input

As with other BEDOPS utilities, *starch* takes in *sorted* BED data as input. You can use *sort-bed* to sort BED data, piping it into *starch* as standard input (see *Example* section below).

Note: While more than three columns may be specified, most of the space savings in the Starch format are derived from from a pre-processing step on the coordinates. Therefore, minimizing or removing unnecessary columnar data from the fourth column on (*e.g.*, with `cut -f1-3` or similar) can help improve compression efficiency considerably.

Output

This utility outputs a *Starch v2-formatted* archive file.

Requirements

The *starch* tool requires data in a relaxed variation of the BED format as described by UCSC's browser documentation. BED data should be sorted before compression, *i.e.* with BEDOPS *sort-bed*.

At a minimum, three columns are required to specify the chromosome name and start and stop positions. Additional columns may be specified, containing up to 128 kB of data per row (including tab delimiters).

Usage

Use the `--help` option to list all options:

```
starch
 citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
 binary version: 2.4.37 (typical) (creates archive version: 2.2.0)
 authors:  Alex Reynolds and Shane Neph

USAGE: starch [ --note="foo bar..." ]
              [ --bzip2 | --gzip ]
              [ --omit-signature ]
              [ --report-progress=N ]
              [ --header ] [ <unique-tag> ] <bed-file>

    * BED input must be sorted lexicographically (e.g., using BEDOPS sort-bed).
    * Please use '-' to indicate reading BED data from standard input.
    * Output must be directed to a regular file.
    * The bzip2 compression type makes smaller archives, while gzip extracts
      faster.

    Process Flags
    -------------------------------------------------------------------------
    --note="foo bar..."   Append note to output archive metadata (optional).

    --bzip2 | --gzip      Specify backend compression type (optional, default
                          is bzip2).

    --omit-signature      Skip generating per-chromosome data integrity signature
                          (optional, default is to generate signature).
```

(continues on next page)

(continued from previous page)

```
  --report-progress=N   Report compression progress every N elements per
                        chromosome to standard error stream (optional)

  --header              Support BED input with custom UCSC track, SAM or VCF
                        headers, or generic comments (optional).

  <unique-tag>          Optional. Specify unique identifier for transformed
                        data.

  --version             Show binary version.

  --help                Show this usage message.
```

Options

Backend compression type

Use the `--bzip2` or `--gzip` operators to use the `bzip2` or `gzip` compression algorithms on transformed BED data. By default, *starch* uses the `bzip2` method.

Note

Use the `--note="xyz..."` option to add a custom string that describes the archive. This data can be retrieved with `unstarch --note`.

Tip: Examples of usage might include a description of the experiment associated with the data, a URL to a UCSC Genome Browser session, or a bar code or other unique identifier for internal lab or LIMS use.

Note: The only limitation on the length of a note is the command-line shell's maximum argument length parameter (as found on most UNIX systems with the command `getconf ARG_MAX`) minus the length of the non- `--note="..."` command components. On most desktop systems, this value will be approximately 256 kB.

Per-chromosome data integrity signature

By default, a data integrity signature is generated for each chromosome. This can be used to verify if chromosome streams from two or more Starch archives are identical, or used to test the integrity of a chromosome, to identify potential data corruption.

Generating this signature adds to the computational cost of compression, or an integrity signature may not be useful for all archives. Add the `--omit-signature` option, if the compression time is too high or the data integrity signature is not needed.

Compression progress

To optionally track the progress of compression, use the `--report-progress=N` option, specifying a positive integer `N` to report the compression of the N -th element for the current chromosome. The report is printed to the

standard error stream.

Note: For instance, specifying a value of `1` reports the compression of every input element of all chromosomes, while a value of `1000` would report the compression of every 1000th element of the current chromosome.

Headers

Add the `--header` flag if the BED data being compressed contain extra header data that are exported from a UCSC Genome Browser session.

Note: If the BED data contain custom headers and `--header` is not specified, *starch* will be unable to read chromosome data correctly and exit with an error state.

Unique tag

Adding a `<unique-tag>` string replaces portions of the *filename* key in the archive's *stream metadata*.

Note: This feature is largely obsolete and included for legacy support. It is better to use the `--note="xyz..."` option to add identifiers or other custom data.

Example

To compress unsorted BED data (or data of unknown sort order), we feed *starch* a *sorted* stream, using the hyphen (`-`) to specify standard input:

```
$ sort-bed unsorted.bed | starch - > sorted.starch
```

This creates the file `sorted.starch`, which uses the `bzip2` algorithm to compress transformed BED data from a sorted permutation of data in `unsorted.bed`. No note or custom tag data is added.

It is possible to speed up the compression of a BED file by using a cluster. Start by reviewing our *starchcluster* script.

unstarch

With high-throughput sequencing generating large amounts of genomic data, archiving can be a critical part of an analysis toolkit. BEDOPS includes the `unstarch` utility to recover original BED input and whole-file or per-chromosome data attributes from archives created with *starch* (these can be v1.x or *v2.x archives*).

The *unstarch* utility includes large file support on 64-bit operating systems, enabling extraction of more than 2 GB of data (a common restriction on 32-bit systems).

Starch data can be stored with one of two open-source backend compression methods, either `bzip2` or `gzip`. The *unstarch* utility will transparently extract data, without the end user needing to specify the backend type.

Inputs and outputs

Input

The *unstarch* utility takes in a Starch v1.x or v2.x archive as input.

Output

The typical output of *unstarch* is *sorted* BED data, which is sent to standard output.

Specifying certain options will instead send *archive metadata* to standard output, either in text or JSON format, or export *whole-file or per-chromosome attributes* (also to standard output).

Requirements

The metadata of a Starch v2.x archive must pass an integrity check before *unstarch* can extract data. Any manual changes to the metadata will cause extraction to fail.

Usage

Use the `--help` option to list all options:

```
unstarch
 citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
 binary version: 2.4.37 (typical) (extracts archive version: 2.2.0 or older)
 authors: Alex Reynolds and Shane Neph

USAGE: unstarch [ <chromosome> ]  [ --elements |
                                    --elements-max-string-length |
                                    --bases | --bases-uniq |
                                    --has-duplicates | --has-nested | --list |
                                    --list-json | --list-chromosomes |
                                    --archive-timestamp | --note |
                                    --archive-version | --is-starch |
                                    --signature | --verify-signature ]
                                    <starch-file>

    Modifiers
    ------------------------------------------------------------------------
    <chromosome>                    Optional. Either unarchives chromosome-
                                    specific records from the starch archive
                                    file or restricts action of operator to
                                    chromosome (e.g., chr1, chrY, etc.).

    Process Flags
    ------------------------------------------------------------------------
    --elements                      Show total element count for archive. If
                                    <chromosome> is specified, the result
                                    shows the element count for the
                                    chromosome.

    --elements-max-string-length    Show the maximum string length over all
                                    elements in <chromosome>, if specified.
```

(continues on next page)

(continued from previous page)

```
                                If <chromosome> is not specified, the
                                maximum string length is shown over all
                                chromosomes.

  --bases,
  --bases-uniq                  Show total and unique base counts,
                                respectively, for archive. If
                                <chromosome> is specified, the count is
                                specific to the chromosome, if available.

  --has-duplicate-as-string,
  --has-duplicate               Show whether there is one or more
                                duplicate elements in the specified
                                chromosome, either as a numerical (1/0)
                                or string (true/false) value. If no
                                <chromosome> is specified, the value
                                given indicates if there is one or more
                                duplicate elements across all chromosome
                                records.

  --has-nested-as-string,
  --has-nested                  Show whether there is one ore more nested
                                elements in the specified chromosome,
                                either as a numerical (1/0) or string
                                (true/false) value. If no <chromosome> is
                                specified, the value given indicates if
                                there is one or more nested elements
                                across all chromosome records.

  --list                        List archive metadata (output is in text
                                format). If chromosome is specified, the
                                attributes of the given chromosome are
                                shown.

  --list-json,
  --list-json-no-trailing-newline  List archive metadata (output is in JSON
                                format)

  --list-chr,
  --list-chromosomes            List all or specified chromosome in
                                starch archive (like "bedextract --list-
                                chr"). If <chromosome> is specified but
                                is not in the output list, nothing is
                                returned.

  --note                        Show descriptive note, if available.

  --signature                   Display the Base64-encoded SHA-1 data
                                integrity signature for specified
                                <chromosome>, or the signatures of the
                                metadata and all available chromosomes,
                                if the <chromosome> is unspecified.

  --verify-signature            Verify data integrity of specified
                                <chromosome>, or the integrity of all
                                available chromosomes, if the
                                <chromosome> is unspecified.
```

(continues on next page)

(continued from previous page)

```
    --archive-timestamp             Show archive creation timestamp (ISO 8601
                                    format).

    --archive-type                  Show archive compression type.

    --archive-version               Show archive version.

    --is-starch                     Test if <starch-file> is a valid archive
                                    and print 0/1 (false/true) to standard
                                    output. Unstarch will also return a non-
                                    zero error code if the input file is not
                                    a valid archive.

    --version                       Show binary version.

    --help                          Show this usage message.
```

Extraction

Specify a specific chromosome to extract data only from that chromosome. This is optional; if a chromosome is not specified, data are extracted from all chromosomes in the archive.

```
$ unstarch chr12 example.starch
...
```

Archive attributes

Archive attributes are described in greater depth in the *Starch specification* page. We provide an overview here of the major points.

Metadata

Use the `--list-json` or `--list` options to export the archive metadata as a JSON- or table-formatted text string, sent to standard output:

```
$ unstarch --list-json example.starch
{
  "archive": {
    "type": "starch",
    "customUCSCHeaders": false,
    "creationTimestamp": "2014-05-01T14:09:29-0700",
    "version": {
      "major": 2,
      "minor": 2,
      "revision": 0
    },
    "compressionFormat": 0
  },
  "streams": [
    {
```

(continues on next page)

(continued from previous page)

```
      "chromosome": "chr1",
      "filename": "chr1.pid31740.fiddlehead.regulomecorp.com",
      "size": "88330",
      "uncompressedLineCount": 10753,
      "nonUniqueBaseCount": 549829,
      "uniqueBaseCount": 548452,
      "duplicateElementExists": false,
      "nestedElementExists": false,
      "signature": "XtnjojM1LyuMnZI4CIneSzgLI5Q="
      "uncompressedLineMaxStringLength": 371
    },
    ...
  ]
}
```

The `--list-chr` (or `--list-chromosomes`) option exports a list of chromosomes stored in the Starch archive.

Note

Using `--note` will export any note stored with the archive, when created.

Tip: One can use *starchcat* to add a new note to an existing Starch archive.

Timestamp

The `--archive-timestamp` option will report the archive's creation date and time as an ISO 8601 -formatted string.

Compression type

The `--archive-type` option will report the compression type of the archive, either `bzip2` or `gzip`:

```
$ unstarch --archive-type example.starch
unstarch
 archive compression type: bzip2
```

Version

The `--version` option reports the Starch archive version. This value is different from the version of the *starch* binary used to create the archive.

Whole-file or per-chromosome attributes

Data integrity

For a specified chromosome, the `--signature` operator reports the very nearly unique "signature" or message digest generated from hashing the extracted, post-transform bytes within the chromosome stream with the SHA-1

hash function, followed with an encoding step with the Base64 scheme to turn it into a human-readable string.

If no chromosome is specified, this operator reports the encoded SHA-1 digests of the archive metadata and the signatures of each chromosome stream.

Signatures can be used to compare chromosome streams between two or more archives.

Further, use of the `--verify-signature` option with a chromosome name will compare the signature stored in the metadata (the "expected" signature) with an "observed" value generated from extracting the bytes of the chromosome record and hashing them, and then Base64-encoding the result.

If the observed and expected signatures or digests are identical, this validates or verifies the integrity of the chromosome record. A mismatch would result in a non-zero exit state and suggest potential data corruption and the need for further investigation.

Elements

The `--elements` operator reports the number of BED elements that were compressed into the chromosome stream, if specified. If no chromosome is specified, the sum of elements over all chromosomes is reported.

Tip: This option is equivalent to a `wc -l` (line count) operation performed on BED elements that match the given chromosome, but is much, much faster as data are precomputed and stored with the archive, retrieved from the metadata in O(1) time.

The `--elements-max-string-length` operator reports the maximum string length of BED elements over the specified chromosome, or the maximum string length over all chromosomes, if no chromosome name is specified.

Bases

The `--bases` and `--bases-uniq` flags return the overall and unique base counts for a specified chromosome, or the sum of counts over all chromosomes, if no one chromosome is specified.

Duplicate element(s)

The `--has-duplicate` operator reports whether the chromosome stream contains one or more duplicate elements, printing a `0` if the chromosome does *not* contain a duplicate element, and a `1` if the chromosome *does* contain a duplicate.

Note: A duplicate element exists if there are two or more BED elements where the chromosome name and start and stop positions are identical. Id, score, strand and any other optional columns are ignored when determining if a duplicate element is present.

Tip: To get a string value of `true` or `false` in place of `1` and `0`, use the `--has-duplicate-as-string` operator, instead.

Note: If the chromosome name argument to `unstarch` is omitted, or set to `all`, the `--has-duplicate` and `--has-duplicate-as-string` operators will return a result for all chromosomes (if any one chromosome has

one or more duplicate elements, the return value is `1` or `true`, respectively). If the chromosome name is provided and the archive does not contain metadata for the given chromosome, these operators will return a `0` or `false` result.

Nested element(s)

The `--has-nested` operator reports whether the chromosome stream contains one or more *nested elements*, printing a `0` if the chromosome does *not* contain a nested element, and a `1` if the chromosome *does* contain a nested element.

Note: The definition of a nested element relies on coordinates and is explained in the *documentation for nested elements*. Id, score, strand and any other optional columns are ignored when determining if a nested element is present.

Tip: To get a string value of `true` or `false` in place of `1` and `0`, use the `--has-nested-as-string` operator, instead.

Note: If the chromosome name argument to `unstarch` is omitted, or set to `all`, the `--has-nested` and `--has-nested-as-string` operators will return a result for all chromosomes (if any one chromosome has one or more nested elements, the return value is `1` or `true`, respectively). If the chromosome name is provided and the archive does not contain metadata for the given chromosome, these operators will return a `0` or `false` result.

Example

To extract a generic Starch file input to a BED file:

```
$ unstarch example.starch > example.bed
```

This creates the *sorted* file `example.bed`, containing BED data from extracting `example.starch`. This can be a `bzip2` or `gzip` -formatted Starch archive—*unstarch* knows how to extract either type transparently.

To list the chromosomes in a Starch v2 archive, use the `--list-chr` (or `--list-chromosomes`) option:

```
$ unstarch --list-chr example.starch
chr1
chr10
chr11
chr11_gl000202_random
chr12
chr13
chr14
chr15
chr16
chr17
...
```

To show the number of BED elements in chromosome `chr13`, use the `--elements` operator:

```
$ unstarch chr13 --elements example.starch
10753
```

To find the number of unique bases in chromosome `chr8`:

```
$ unstarch chr8 --bases-uniq example.starch
545822
```

To report if the chromosome `chr14` contains at least one duplicate BED element:

```
$ unstarch chr14 --has-duplicate-as-string example.starch
true
```

To show when the archive was created:

```
$ unstarch --archive-timestamp example.starch
2014-05-01T14:09:29-0700
```

To get the SHA-1 message digest, or "signature" of chromosome `chr8`, use the `--signature` operator:

```
$ unstarch chr8 --signature example.starch
nZI4CIneSzgLI5QXtnjojM1LyuM=
```

The signature is written to the standard output stream.

To verify the data integrity of the same chromosome, use `--verify-signature`:

```
$ unstarch chr8 --verify-signature example.starch
PROGRESS: Expected and observed data integrity signatures match for chromosome [chr8]
```

Any output from `--verify-signature` is written to the standard error stream.

Note: Some option calls will not work with legacy v1.x or v2.0 archives. For instance, to get a result for nested or duplicate elements, you need to input a v2.1 (or greater) archive. If you have a v1.x or v2.0 archive, use the *starchcat* utility to upgrade an older archive to a Starch v2.2 file, which will recalculate the metadata and make all current attributes available.

---

starchcat

The `starchcat` utility efficiently merges per-chromosome records contained within one or more BEDOPS *Starch-formatted* archives. This is an equivalent operation to `bedops --everything` or `bedops -u` (a *multiset union*), but inputs are *starch* archives rather than uncompressed BED files.

As a further advantage to using this over *bedops*, in the case where a *starch* input contains BED elements exclusive to one chromosome, this utility will directly and quickly copy over compressed elements to a new archive, avoiding the need for costly and wasteful extraction and re-compression.

In the general case, where two or more *starch* inputs contain BED elements from the same chromosome, a *sorted* merge is performed and the stream reprocessed into a *Starch-formatted* archive.

Parallelization

Those with access to a computational cluster such as an Oracle/Sun Grid Engine or a group of hosts running SSH services should find *starchcat* highly useful, as this facilitates:

- Much faster compression of an entire genome of BED data, using nodes of a computational cluster to compress separate chromosomes, followed by a collation step with *starchcat* (see the *Efficiently creating Starch-formatted archives with a cluster* documentation).

- Extraction, manipulation and reintegration of a *starch* -ed chromosome into a larger *starch* archive
- Refreshing metadata or re-compressing the data within a lone *starch* archive.

To demonstrate the first application of this utility, we have packaged a helper script with the BEDOPS suite called *starchcluster*, which archives data much faster than *starch* alone. By distributing work across the nodes of a computational cluster, the upper bound on compression time is almost entirely determined by the largest chromosome, reducing compression time by an order of magnitude.

Inputs and outputs

Input

The input to *starchcat* consists of one or more BEDOPS *Starch-formatted* archive files.

Note: If a single archive is provided as input, it may be reprocessed with specified options. When two or more archives are specified, the output will be the equivalent of a multiset union of the inputs.

Note: This utility does not accept standard input.

Output

The *starchcat* tool outputs a *starch* -formatted archive to standard output, which is usually redirected to a file.

Additionally, an optional compression flag specifies if the final *starch* output should be compressed with either the `bzip2` or `gzip` method (the default being `bzip2`).

Note: If *starch* inputs use a different backend compression method, the input stream is re-compressed before integrated into the larger archive. This will incur extra processing overhead.

Usage

Use the `--help` option to list all options:

```
starchcat
 citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
 version:  2.4.37 (typical)
 authors:  Alex Reynolds and Shane Neph

USAGE: starchcat [ --note="..." ]
                 [ --bzip2 | --gzip ]
                 [ --omit-signature ]
                 [ --report-progress=N ] <starch-file-1> [<starch-file-2> ...]

    * At least one lexicographically-sorted, headerless starch archive is
      required.

    * While two or more inputs make sense for a multiset union operation, you
```

(continues on next page)

(continued from previous page)

```
    can starchcat one file in order to update its metadata, recompress it
    with a different backend method, or add a note annotation.

  * Compressed data are sent to standard output. Use the '>' operator to
    redirect to a file.

  Process Flags
  ------------------------------------------------------------------------
  --note="foo bar..."   Append note to output archive metadata (optional).

  --bzip2 | --gzip      Specify backend compression type (optional, default
                        is bzip2).

  --omit-signature      Skip generating per-chromosome data integrity signature
                        (optional, default is to generate signature).

  --report-progress=N   Report compression progress every N elements per
                        chromosome to standard error stream (optional)

  --version             Show binary version.

  --help                Show this usage message.
```

Per-chromosome data integrity signature

By default, a data integrity signature is generated for each chromosome. This can be used to verify if chromosome streams from two or more Starch archives are identical, or used to test the integrity of a chromosome, to identify potential data corruption.

Generating this signature adds to the computational cost of compression, or an integrity signature may not be useful for all archives. Add the `--omit-signature` option, if the compression time is too high or the data integrity signature is not needed.

Example

Let's say we have a set of 23 *starch* archives, one for each chromosome of the human genome: `chr1.starch`, `chr2.starch`, and so on, to `chrY.starch`. (To simplify this example, we leave out mitochondrial, random, pseudo- and other chromosomes.) We would like to build a new *starch* archive from these 23 separate files:

```
$ starchcat chr1.starch chr2.starch ... chrY.starch > humanGenome.starch
```

The *starchcat* utility parses the metadata from each of the 23 inputs, determines what data to either simple copy or reprocess, and then it performs the merge. Cleanup is performed afterwards, as necessary, and the output is a brand new *starch* file, written to `humanGenome.starch`.

Note: No filtering or processing is performed on extracted BED elements, before they are written to the final output. Thus, *it is possible for duplicate BED elements to occur*. It would be easy to use the `--signature` option to validate the expected content of a new Starch archive.

However, the final archive is sorted per *sort-bed* ordering, so that data extracted from this archive will be ready for use with BEDOPS utilities.

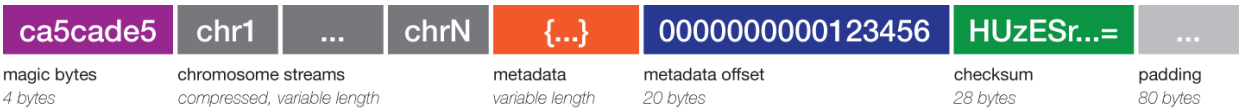
Note: When input archives contain data on disjoint chromosomes, use of *starchcat* is very efficient as data are simply copied, instead of extracted and re-compressed.

Starch (v2.x) specification

This document describes the specification for a "Starch v2.x"-formatted archive, which is created by the *starch* and *starchcat* utilities and extracted with the *unstarch* utility.

Archive structure

A Starch v2.x archive is divided up into six portions:



Each portion is explained below.

Magic bytes

We use four `unsigned char` bytes `ca5cade5` to identify the file as a Starch v2.x archive. BEDOPS utilities and applications which process Starch archives search for these magic bytes at the start of the file to identify it as a v2.x archive.

If the file does not have these bytes, it may still be a legacy (v1, v1.2 or v1.5) Starch archive, which is identified and processed by other means not described in this document.

Chromosome streams

chromosome streams

compressed, variable length

These variable-length data streams contain compressed, transformed BED data separated by chromosome.

Transformation is performed on BED input to remove redundancy in the coordinate data provided in the second and third columns ("start" and "stop" coordinates). Data in any additional columns are left unchanged. Transformed data are highly reduced and compressed further with open-source `bzip2` or `gzip` libraries.

Starch v2 streams extracted with *unstarch*, *bedops*, *bedmap* or *closest-features* are uncompressed with the requisite backend compression library calls and then reverse-transformed to recover the original BED input.

Metadata

The archive metadata is made up of data, offset and hash components, each with different characteristics as described below.

Data

metadata

variable length

This variable-length portion of the archive is a JSON -formatted ASCII string that describes the Starch archive contents. We choose JSON as it provides a human-readable structure, allows easier extensibility for future revisions of BEDOPS and is a common format in web services, facilitating usage with web- and command-line-based bioinformatics pipelines.

The format of a typical Starch v2 JSON object is made up of two key-value pairs, one for archive and the second for streams, which we describe in greater detail below.

Archive

The archive key scheme is described below:

```
{
  "archive": {
    "type": "starch",
    "customUCSCHeaders": (Boolean),
    "creationTimestamp": (string),
    "version": { "major": 2, "minor": 2, "revision": 0 },
    "compressionFormat": (unsigned integer),
    "note": (string, optional)
  },
  ...
}
```

At this time, the `type` key will specify `starch`.

The `customUCSCHeaders` value is either `true` or `false`. If `true`, the `--header` option was provided to *starch* when the archive was created, and the archive may likely contain UCSC headers commonly encountered with UCSC Genome Browser data downloads. Archives created with *starchcat* do not support UCSC headers (*i.e.*, this value is false in archives created with *starchcat*).

The `creationTimestamp` value is an ISO 8601 string that specifies the creation date and time of the archive. Most scripting and programming languages can parse ISO 8601-formatted date strings with little or no extra work.

The `version` is a triplet of integer values specifying the version of the archive. For a v2.x archive, the major version will be set to `2`. Major, minor and revision values need not necessarily be the identical to the version of the *starch* binary used to create the archive.

At this time (November 2016), we offer v2.0, v2.1, and v2.2 archives: Each version makes different stream metadata fields available.

The `compressionFormat` key specifies the backend compression format used for the chromosome streams contained within the archive. We currently use `0` to specify `bzip2` and `1` to specify `gzip`. No other backend formats are available at this time.

The `note` key is an optional string that can contain information if the `--note="abc..."` option is provided to *starch* when the archive is created. If this option is not specified at creation time, this key will not be present in the metadata.

Streams

The `streams` key scheme contains an array of objects, each describing the attributes of an individually-compressed chromosome stream, sorted on chromosome name:

```
{
  ...,
  "streams": [
    {
      "chromosome": (string),
      "filename": (string),
      "size": (unsigned integer),
      "uncompressedLineCount": (unsigned integer),
      "nonUniqueBaseCount": (unsigned integer),
      "uniqueBaseCount": (unsigned integer),
      "duplicateElementExists": (Boolean),
      "nestedElementExists": (Boolean),
      "signature": (string),
      "uncompressedLineMaxStringLength": (integer)
    },
```

(continues on next page)

(continued from previous page)

```
    ...
  ]
}
```

The `chromosome` key specifies the name of the chromosome associated with the compressed regions. For example, this might be `chr1`, `chrX`, etc.

The `filename` key is a string that concatenates the chromosome name, process ID and host strings (unless a `unique-tag` value is given to *starch* when creating an archive, which would replace the process ID and host values). It is a holdover from a procedure for creating legacy archives and exists for backwards-compatibility.

The `size` key specifies the byte-size of the compressed stream and exists for calculating offsets within the archive where a chromosome stream begins (and ends). In this way, *unstarch* and other Starch-capable applications can extract data only from a desired chromosome, without wasteful processing of the remainder of the archive.

The `uncompressedLineCount` key specifies the number of BED elements that were compressed into the chromosome stream. This is a precomputed equivalent to the result of a `wc -l` (line count) operation performed on BED elements that match the given chromosome, without needing to stream through the entire file.

The `nonUniqueBaseCount` key specifies the sum of non-unique bases across all BED elements compressed into the chromosome stream. Non-uniqueness allows multiple counting of bases in elements which overlap.

The `uniqueBaseCount` key specifies the sum of unique bases across all BED elements compressed into the chromosome stream. Uniqueness takes into account overlapping elements and therefore only counts bases once.

The `duplicateElementExists` key specifies if there is a duplicate BED element somewhere within the compressed chromosome stream. A duplicate element is defined by matching chromosome name and start and stop coordinates; id, score, strand and other optional information are ignored when determining if a duplicate element exists.

The `nestedElementExists` key specifies if there is a nested BED element somewhere within the compressed chromosome stream. Refer to BEDOPS documentation to see how *nested elements* are defined.

The `signature` key, available in v2.2 archives, specifies the Base64-encoded SHA-1 data integrity signature generated from the transformed chromosome stream (not the raw BED data, but the reduced or transformed form that is compressed). This can be used to compare the transformed bytes for chromosomes from different archives, or to validate the genomic data in a Starch archive by chromosome, or in entirety. (Note that, if `--omit-signature` was used to create a v2.2 archive, this key will not be present in the stream object.)

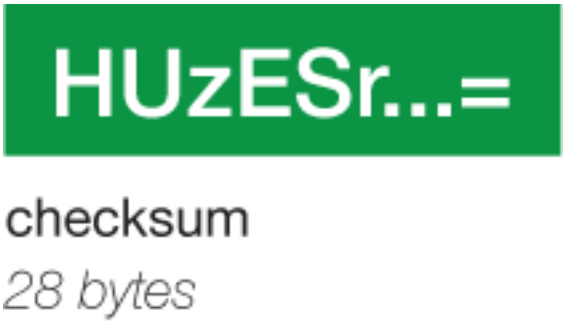
The `uncompressedLineMaxStringLength` key, available in v2.2 archives, specifies the maximum string length over all records in the chromosome stream.

Offset

The metadata offset is a 20-byte long, zero-padded string that specifies the number of bytes into the file where the JSON-formatted metadata string is stored.

The *unstarch* utility and the newer versions of *bedops* and *bedmap* applications use this offset to jump to the correct point in the file where the metadata can be read into memory and processed into an internal data structure.

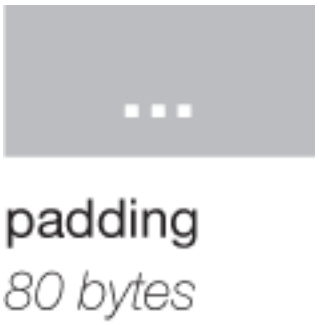
Hash



The metadata hash is a 28-byte long, Base64 -encoded SHA-1 hash of the bytes that make up the JSON-formatted metadata string.

This data is used to validate the integrity of the metadata: Any change to the metadata (*e.g.*, data corruption that changes stream offset values, or the data integrity signatures for each chromosome stream) causes *unstarch* and other Starch utilities and applications to exit early with a fatal, informative error.

Padding



The remainder of the file is made up of 80 bytes of padding, which are unused at this time.

starch-diff

This tool allows the end user to quickly compare the compressed chromosomes in two or more v2.2+ Starch archives.

Inputs and outputs

Input

The *starch-diff* utility takes in two or more Starch v2.2+ archives as input. The end user may also add *–chr <chr>* to compare one chromosome directly; otherwise, all chromosomes in specified archives are compared.

Output

The typical output of *starch-diff* is a message indicating the archives' chromosome(s) are identical or dissimilar.

In addition, if the chromosomes are identical, *starch-diff* exits with a zero status code. Likewise, if any chromosomes are dissimilar, *starch-diff* exits with a non-zero status code.

Requirements

If the user passes in a pre-v2.2 archive, the utility will exit with a fatal error.

Usage

Use the `--help` option to list all options:

```
starch-diff
  citation: http://bioinformatics.oxfordjournals.org/content/28/14/1919.abstract
  version:  2.4.37
  authors:  Alex Reynolds and Shane Neph

  $ starch-diff [ --chr <chr> ] starch-file-1 starch-file-2 [ starch-file-3 ... ]

  The 'starch-diff' utility compares the signatures of two or more specified
  Starch v2.2+ archives for all chromosomes, or for a specified chromosome.
```

Data conversion

Wrapper scripts around the *convert2bed* utility quickly convert a variety of common genomic data types to BED with no loss of information. In using these tools, you can easily prepare data from these formats for use with core BEDOPS tools, whether VCF, GFF/GTF/GVF, BAM/SAM, PSL (Blat), RepeatMasker annotation output, etc.

Some other formats not covered here can be converted with, for instance, the UCSC Kent toolset (*e.g.*, altGraphX, bigWig, bigBed, etc.). Just remember to use the *sort-bed* utility to prepare BED output from external programs for use with BEDOPS core tools.

convert2bed

The `convert2bed` binary converts common binary and text genomic formats (BAM, GFF, GTF, GVF, PSL, RepeatMasker annotation output (OUT), SAM, VCF and WIG) to unsorted or sorted, extended BED or *BEDOPS Starch* (compressed BED) with additional per-format options.

Convenience wrapper bash scripts are provided for each of these input formats, which convert standard input to unsorted or sorted BED, or to BEDOPS Starch (compressed BED). Scripts expose format-specific `convert2bed` options.

We also provide `bam2bed_sge`, `bam2bed_gnuParallel`, `bam2starch_sge` and `bam2starch_gnuParallel` convenience scripts, which parallelize the conversion of indexed BAM to BED or to BEDOPS Starch via a Sun Grid Engine-based computational cluster or local GNU Parallel installation.

Dependencies

Conversion of BAM and SAM input is dependent upon the installation of SAMtools and *convert2bed*. All `*2starch` wrapper scripts are further dependent on the installation of the *starch* binary, part of a typical BEDOPS installation.

Source

The `convert2bed` conversion tool is part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

Generally, to convert data in format `xyz` to sorted BED:

```
$ convert2bed -i xyz < input.xyz > output.bed
```

Add the `-o starch` option to write a BEDOPS Starch file, which stores compressed BED data and feature metadata:

```
$ convert2bed -i xyz -o starch < input.xyz > output.starch
```

Wrappers are available for each of the supported formats to convert to BED or Starch, *e.g.*:

```
$ bam2bed < reads.bam > reads.bed
$ bam2starch < reads.bam > reads.starch
```

Tip: Format-specific options are available for each wrapper; use `--help` with a wrapper script or `--help-bam`, `--help-gff` etc. with `convert2bed` to get a format-specific description of the conversion procedure and options.

Example

Please review documentation for each wrapper script to see format-specific examples of their use.

bam2bed

The `bam2bed` script converts 0-based, half-open `[start-1, end)` Binary (Sequence) Alignment/Map (BAM) to sorted, 0-based, half-open `[start-1, end)` UCSC BED data.

For convenience, we also offer `bam2starch`, which performs the extra step of creating a *Starch-formatted* archive.

The `bam2bed` script is "non-lossy" (with the use of specific options, described below). Other toolkits tend to throw out information from the original BAM input upon conversion; `bam2bed` can retain everything, facilitating reuse of converted data and conversion to other formats.

Tip: Doing the extra step of creating a *Starch-formatted* archive can save a lot of space relative to the original BAM format, up to 33% of the original BAM dataset, while offering per-chromosome random access.

Dependencies

The `bam2bed` wrapper script is dependent upon the installation of SAMtools and *convert2bed*. The `bam2starch` wrapper script is further dependent on the installation of the *starch* binary, part of a typical BEDOPS installation.

Source

The `bam2bed` and `bam2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `bam2bed` script parses BAM data from standard input and prints *sorted* BED to standard output. The `bam2starch` script uses an extra step to parse BAM to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a BAM file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Tip: If you work with RNA-seq data, you may use the `--split` option to process reads with `N`-CIGAR operations, splitting them into separate BED elements.

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If sorting converted data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample binary input called `foo.bam` (see the *Downloads* section to grab this file).

We can convert it to sorted BED data in the following manner (omitting standard error messages):

```
$ bam2bed < foo.bam
seq1    0       36      B7_591:4:96:693:509     99      +       73      36M     *       ␣
↪    0       0       CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG    <<<<<<<<<<<<<<;<<<<<<<<
↪<5<<<<<;:<;7    MF:i:18 Aq:i:73 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    2       37      EAS54_65:7:152:368:113  99      +       73      35M     *       ␣
↪    0       0       CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT     <<<<<<<<<<0<<<<655<<7<<
↪<:9<<3/:<6):     MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    4       39      EAS51_64:8:5:734:57     99      +       137     35M     *       ␣
↪    0       0       AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC     <<<<<<<<<<7;71<<;<;;<7;<
↪<3;);3*8/5     MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
```

(continues on next page)

(continued from previous page)

```
seq1    5       41      B7_591:1:289:587:906    63      +       137     36M     *
↪   0       0       GTGGCTCATTGTAATTTTTTGTTTTAACTCTTCTCT    (-&----,----)-)-),'--)---
↪',+-,),''*,    MF:i:130        Aq:i:63 NM:i:5  UQ:i:38 H0:i:0  H1:i:0
...
```

Note that we strip the header section from the output. If we want to keep this, the use of the `--keep-header` option will preserve the BAM file's header, turning it into BED elements that use `_header` as a chromosome name.

Here's an example:

```
$ bam2bed --keep-header < foo.bam
_header 0       1       @HD     VN:1.0 SO:coordinate
_header 1       2       @SQ     SN:seq1 LN:5000
_header 2       3       @SQ     SN:seq2 LN:5000
_header 3       4       @CO     Example of SAM/BAM file format.
seq1    0       36      B7_591:4:96:693:509     99      +       73      36M     *
↪   0       0       CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG    <<<<<<<<<<<<<<;<<<<<<<<
↪<5<<<<<;:<;7   MF:i:18 Aq:i:73 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    2       37      EAS54_65:7:152:368:113  99      +       73      35M     *
↪   0       0       CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT     <<<<<<<<<<0<<<<655<<7<<
↪<:9<<3/:<6):    MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    4       39      EAS51_64:8:5:734:57     99      +       137     35M     *
↪   0       0       AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC     <<<<<<<<<<<7;71<<;<;;<7;<
↪<3;);3*8/5     MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    5       41      B7_591:1:289:587:906    63      +       137     36M     *
↪   0       0       GTGGCTCATTGTAATTTTTTGTTTTAACTCTTCTCT    (-&----,----)-)-),'--)---
↪',+-,),''*,    MF:i:130        Aq:i:63 NM:i:5  UQ:i:38 H0:i:0  H1:i:0
...
```

With this option, the `bam2bed` and `bam2starch` scripts are completely "non-lossy" (with the exception of unmapped reads; see note below). Use of `awk` or other scripting tools can munge these data back into a SAM-formatted file.

Note: The provided scripts **strip out unmapped reads** from the BAM file. We believe this makes sense under most circumstances. Add the `--all-reads` option if you need unmapped and mapped reads.

Column mapping

In this section, we describe how non-header BAM data (converted to SAM columns) are mapped to BED columns. We start with the first six UCSC BED columns as follows:

SAM field	BED column index	BED field
RNAME	1	chromosome
POS - 1	2	start
POS + length(CIGAR) - 1	3	stop
QNAME	4	id
MAPQ	5	score
16 & FLAG	6	strand

The remaining SAM-converted columns are mapped as-is, in same order, to adjacent BED columns:

SAM field	BED column index	BED field
FLAG	7	
CIGAR	8	
RNEXT	9	
PNEXT	10	
TLEN	11	
SEQ	12	
QUAL	13	

Because we have mapped all columns, we can translate converted BED data back to headered or headerless SAM reads with a simple `awk` statement (or other script) that reverts back to 1-based coordinates and permutes columns to SAM-based ordering.

Downloads

- Sample BAM dataset: `foo.bam`

Parallel *bam2bed*

The `bam2bed_slurm`, `bam2bed_sge`, and `bam2bed_gnuParallel` scripts use a SLURM, Sun or Oracle Grid Engine (SGE/OGE), or GNU Parallel job scheduler, respectively, to parallelize the work of `bam2bed`, which converts an **indexed**, 0-based, half-open `[start-1, end)` Binary (Sequence) Alignment/Map (BAM) file to a sorted, 0-based, half-open `[start-1, end)` UCSC BED dataset.

This script splits the indexed BAM file by chromosome name. Each chromosome of BAM records is converted to a BED-formatted dataset with `bam2bed` (via `convert2bed`). Once all per-chromosome BED files are made, they are collated into one final BED file with a multiset union performed with *bedops –everything*.

Dependencies

This shell script is dependent upon a working computational grid that is managed with SLURM 16.05.0, Sun Grid Engine 6.1u5 (or higher), or installation of GNU Parallel v20130922 or greater.

Source

The `bam2bed_slurm`, `bam2bed_sge`, and `bam2bed_gnuParallel` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

Note: Please review and edit the contents of the relevant script before use with your data, particularly if you use a Sun or Oracle Grid Engine environment and make use of the SGE version of this script. Customization may be required to match your SLURM, SGE/OGE or GNU Parallel installation and environment, as well as the nature of your BAM data.

At minimum, use of this script with an SGE/OGE computational cluster will require editing of the `queue` parameter, possible adjustments to `qsub` options, and may also require adjustments to paths to working BEDOPS and Samtools binaries. If you use the *module* system, running *module add samtools* will take care of this dependency.

You will also need to make sure your BAM data are indexed. There must be a second BAI file with the same prefix name as the BAM file you wish to compress, located in the same working directory. If this index file is not present, the script will exit early with an error.

You may also wish to review other parameters available with the `bam2bed` script, applying them in this script as needed (see the *bam2bed* documentation for more details).

Parallel *bam2starch*

The `bam2starch_slurm`, `bam2starch_sge`, and `bam2starch_gnuParallel` scripts use a SLURM, Sun or Oracle Grid Engine (SGE/OGE), or GNU Parallel job scheduler, respectively, to parallelize the work of `bam2starch`, which converts an **indexed**, 0-based, half-open `[start-1, end)` Binary (Sequence) Alignment/Map (BAM) file to a sorted, 0-based, half-open `[start-1, end)` UCSC BED dataset, and thence converts this to a *Starch-formatted* archive.

This script splits the indexed BAM file by chromosome name. Each chromosome of BAM records is converted to a *Starch-formatted* archive with `bam2starch` (via `convert2bed`). Once all per-chromosome archives are made, they are collated into one final Starch archive with *starchcat*.

Tip: A *Starch-formatted* archive can save a great deal of space relative to the original BAM format, up to 33% of the original BAM dataset, while offering per-chromosome random access. Further, use of a computational grid practically reduces the total compression time to that of the largest chromosome (*e.g.*, `chr1` or similar), an order of magnitude reduction over `bam2starch` alone.

Dependencies

This shell script is dependent upon a working computational grid that is managed with SLURM 16.05.0, Sun Grid Engine 6.1u5 (or higher), or installation of GNU Parallel v20130922 or greater.

Source

The `bam2starch_slurm`, `bam2starch_sge`, and `bam2starch_gnuParallel` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

Note: Please review and edit the contents of this script before use with your data. Customization may be required to match your SLURM, SGE/OGE, or GNU Parallel installation and environment, as well as the nature of your BAM data.

At minimum, use of this script with an SGE/OGE computational cluster will require editing of the `queue` parameter, possible adjustments to `qsub` options, and may require adjustments to paths to working BEDOPS binaries.

You will also need to make sure your BAM data are indexed. There must be a second BAI file with the same prefix name as the BAM file you wish to compress, located in the same working directory. If this index file is not present, the script will exit early with an error.

You may also wish to review other parameters available with the `bam2starch` script, applying them in this script as needed (see the *bam2bed* documentation for more details).

gff2bed

The `gff2bed` script converts 1-based, closed `[start, end]` General Feature Format v3 (GFF3) to sorted, 0-based, half-open `[start-1, end)` extended BED-formatted data.

For convenience, we also offer `gff2starch`, which performs the extra step of creating a *Starch-formatted* archive.

Dependencies

The `gff2bed` script requires *convert2bed*. The `gff2starch` script requires *starch*. Both dependencies are part of a typical BEDOPS installation.

This script is also dependent on input that follows the GFF3 specification. A GFF3-format validator is available here to ensure your input follows specification.

Tip: Conversion of data which are GFF-like, but which do not follow the specification can cause parsing issues. If you run into problems, please check that your input follows the GFF3 specification. Tools such as the GFF3 Online Validator are useful for this task.

Source

The `gff2bed` and `gff2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `gff2bed` script parses GFF3 from standard input and prints sorted BED to standard output. The `gff2starch` script uses an extra step to parse GFF to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a GFF file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If sorting converted data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample GFF input called `foo.gff` (see the *Downloads* section to grab this file).

```
##gff-version 3
chr1    Canada  exon    1300    1300    .       +       .       ID=exon00001;score=1
chr1    USA     exon    1050    1500    .       -       0       ID=exon00002;Ontology_
↪term="GO:0046703";Ontology_term="GO:0046704"
chr1    Canada  exon    3000    3902    .       ?       2       ID=exon00003;score=4;
↪Name=foo
chr1    .       exon    5000    5500    .       .       .       ID=exon00004;Gap=M8␣
↪D3 M6 I1 M6
chr1    .       exon    7000    9000    10      +       1       ID=exon00005;Dbxref=
↪"NCBI_gi:10727410"
```

We can convert it to sorted BED data in the following manner:

```
$ gff2bed < foo.gff3
chr1    1049    1500    exon00002       .       -       USA     exon    0       ␣
↪ID=exon00002;Ontology_term="GO:0046703";Ontology_term="GO:0046704"
chr1    1299    1300    exon00001       .       +       Canada  exon    .       ␣
↪ID=exon00001;score=1;zeroLengthInsertion=True
chr1    2999    3902    exon00003       .       ?       Canada  exon    2       ␣
↪ID=exon00003;score=4;Name=foo
chr1    4999    5500    exon00004       .       .       .       exon    .       ␣
↪ID=exon00004;Gap=M8 D3 M6 I1 M6
chr1    6999    9000    exon00005       10      +       .       exon    1       ␣
↪ID=exon00005;Dbxref="NCBI_gi:10727410"
```

The default usage strips the leading pragma, or header (`##gff-version 3`), but adding the `--keep-header` option will preserve this as a BED element that uses `_header` as a chromosome name:

```
$ gff2bed --keep-header < foo.gff3
_header 0       1       ##gff-version 3
chr1    1049    1500    exon00002       .       -       USA     exon    0       ␣
↪ID=exon00002;Ontology_term="GO:0046703";Ontology_term="GO:0046704"
chr1    1299    1300    exon00001       .       +       Canada  exon    .       ␣
↪ID=exon00001;score=1;zero_length_insertion=True
chr1    2999    3902    exon00003       .       ?       Canada  exon    2       ␣
↪ID=exon00003;score=4;Name=foo
chr1    4999    5500    exon00004       .       .       .       exon    .       ␣
↪ID=exon00004;Gap=M8 D3 M6 I1 M6
chr1    6999    9000    exon00005       10      +       .       exon    1       ␣
↪ID=exon00005;Dbxref="NCBI_gi:10727410"
```

Note: Zero-length insertion elements are given an extra attribute called `zeroLengthInsertion` which lets a BED-to-GFF or other parser know that the element will require conversion back to a right-closed element `[a, b]`, where `a` and `b` are equal.

Note: Note the conversion from 1- to 0-based coordinate indexing, in the transition from GFF3 to BED. *BEDOPS supports operations on input with any coordinate indexing*, but the coordinate change made here is believed to be convenient for most end users.

Column mapping

In this section, we describe how GFF3 columns are mapped to BED columns. We start with the first six UCSC BED columns as follows:

GFF3 field	BED column index	BED field
seqid	1	chromosome
start	2	start
end	3	stop
ID (via attributes)	4	id
score	5	score
strand	6	strand

The remaining columns are mapped as follows:

GFF3 field	BED column index	BED field
source	7	
type	8	
phase	9	
attributes	10	

If we encounter zero-length insertion elements (which are defined where the `start` and `stop` GFF3 field values are equivalent), the `start` coordinate is decremented to convert to 0-based, half-open indexing, and a `zero_length_insertion` attribute is added to the `attributes` GFF3 field value.

Downloads

- Sample GFF dataset: `foo.gff`

gtf2bed

The `gtf2bed` script converts 1-based, closed `[start, end]` Gene Transfer Format v2.2 (GTF2.2) to sorted, 0-based, half-open `[start-1, end)` extended BED-formatted data.

For convenience, we also offer `gtf2starch`, which performs the extra step of creating a Starch-formatted archive.

Dependencies

The `gtf2bed` script requires *convert2bed*. The `gtf2starch` script requires *starch*. Both dependencies are part of a typical BEDOPS installation.

This script is also dependent on input that follows the GTF 2.2 specification. A GTF-format validator is available here to ensure your input follows specification.

Tip: Conversion of data which are GTF-like, but which do not follow the specification can cause parsing issues. If you run into problems, please check that your input follows the GTF specification.

Source

The `gtf2bed` and `gtf2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `gtf2bed` script parses GTF from standard input and prints sorted BED to standard output. The `gtf2starch` script uses an extra step to parse GTF to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If sorting converted data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample GTF input called `foo.gtf` (see the *Downloads* section to grab this file).

```
chr20      protein_coding  exon    9874841 9874841 .       +       .       gene_id
↪"ENSBTAG00000020601"; transcript_id "ENSBTAT0000002.4.37"; gene_name "ZNF366";
chr20      protein_coding  CDS     9873504 9874841 .       +       0       gene_id
↪"ENSBTAG00000020601"; transcript_id "ENSBTAT0000002.4.37"; gene_name "ZNF366";
chr20      protein_coding  exon    9877488 9877679 .       +       .       gene_id
↪"ENSBTAG00000020601"; transcript_id "ENSBTAT0000002.4.37";
```

We can convert it to sorted BED data in the following manner:

```
$ gtf2bed < foo.gtf
chr20   9874840 9874841 ZNF366  .       +       protein_coding  exon    .       gene_
↪id "ENSBTAG00000020601"; transcript_id "ENSBTAT0000002.4.37"; gene_name "ZNF366";␣
↪zero_length_insertion "True";
chr20   9873503 9874841 ZNF366  .       +       protein_coding  CDS     0       gene_
↪id "ENSBTAG00000020601"; transcript_id "ENSBTAT0000002.4.37"; gene_name "ZNF366";
chr20   9877487 9877679 ENSBTAG00000020601      .       +       protein_coding  exon ␣
↪   .       gene_id "ENSBTAG00000020601"; transcript_id "ENSBTAT0000002.4.37";
```

Tip: After, say, performing set or statistical operations with *bedops*, *bedmap* etc., converting data back to GTF is accomplished through an `awk` statement that re-orders columns and shifts the coordinate index:

```
$ awk '{print $1"\t"$7"\t"$8"\t"($2+1)"\t"$3"\t"$5"\t"$6"\t"$9"\t"(substr($0, index(
↪$0,$10)))}' foo_subset.bed > foo_subset.gtf
```

Note: Zero-length insertion elements are given an extra attribute called `zero_length_insertion` which lets a BED-to-GTF or other parser know that the element will require conversion back to a right-closed element `[a, b]`, where `a` and `b` are equal.

Note: Note the conversion from 1- to 0-based coordinate indexing, in the transition from GTF to BED. *BEDOPS supports operations on input with any coordinate indexing*, but the coordinate change made here is believed to be convenient for most end users.

Column mapping

In this section, we describe how GTF2.2 columns are mapped to BED columns. We start with the first six UCSC BED columns as follows:

GFF2.2 field	BED column index	BED field
seqname	1	chromosome
start	2	start
end	3	stop
gene_id	4	id
score	5	score
strand	6	strand

The remaining columns are mapped as follows:

GFF2.2 field	BED column index	BED field
source	7	
feature	8	
frame	9	
attributes	10	

If present in the GTF2.2 input, the following column is also mapped:

GFF2.2 field	BED column index	BED field
comments	11	

If we encounter zero-length insertion elements (which are defined where the `start` and `stop` GFF3 field values are equivalent), the `start` coordinate is decremented to convert to 0-based, half-open indexing, and a `zero_length_insertion` attribute is added to the `attributes` GTF2.2 field value.

Downloads

- Sample GTF dataset: `foo.gtf`

gvf2bed

The `gvf2bed` script converts 1-based, closed `[start, end]` Genome Variation Format (GVF, a type of General Feature Format v3 or GFF3) to sorted, 0-based, half-open `[start-1, end)` extended BED-formatted data.

For convenience, we also offer `gvf2starch`, which performs the extra step of creating a *Starch-formatted* archive.

Dependencies

The `gvf2bed` script requires *convert2bed*. The `gvf2starch` script requires *starch*. Both dependencies are part of a typical BEDOPS installation.

This script is also dependent on input that follows the GFF3 specification. A GFF3-format validator is available here to ensure your input follows specification.

Tip: Conversion of data which are GFF-like, but which do not follow the specification can cause parsing issues. If you run into problems, please check that your input follows the GFF3 specification. Tools such as the GFF3 Online Validator are useful for this task.

Source

The `gvf2bed` and `gvf2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `gvf2bed` script parses GVF from standard input and prints sorted BED to standard output. The `gvf2starch` script uses an extra step to parse GVF to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a GVF file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If sorting converted data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample GVF input called `foo.gvf` (see the *Downloads* section to grab this file).

```
##gvf-version 1.07
##feature-ontology http://www.sequenceontology.org/resources/obo_files/current_
↪release.obo
##multi-individual NA19240,NA18507,NA12878,NA19238
##genome-build NCBI B36.3
```

(continues on next page)

(continued from previous page)

```
##sequence-region chr16 1 88827254

chr16 dbSNP   SNV     49291360        49291360        .       +       .       ID=ID_2;
↪Variant_seq=C,G;Individual=0,1,2,3;Genotype=0:1,0:0,1:1,0:1;
chr16 dbSNP   SNV     49302125        49302125        .       +       .       ID=ID_3;
↪Variant_seq=C,T;Individual=0,1,3;Genotype=0:1,2:2,0:2;
chr16 dbSNP   SNV     49302365        49302365        .       +       .       ID=ID_4;
↪Variant_seq=G;Individual=0,1;Genotype=0:0,0:0;
chr16 dbSNP   SNV     49302700        49302700        .       +       .       ID=ID_5;
↪Variant_seq=C,T;Individual=2,3;Genotype=0:1,0:0;
chr16 dbSNP   SNV     49303084        49303084        .       +       .       ID=ID_6;
↪Variant_seq=T,G,A;Individual=3;Genotype=1,2:;
chr16 dbSNP   SNV     49303427        49303427        .       +       .       ID=ID_8;
↪Variant_seq=T;Individual=0;Genotype=0:0;
chr16 dbSNP   SNV     49303596        49303596        .       +       .       ID=ID_9;
↪Variant_seq=A,G,T;Individual=0,1,3;Genotype=1:2,3:3,1:3;
```

We can convert it to sorted BED data in the following manner:

```
$ gvf2bed < foo.gvf
chr16 49291359        49291360        ID_2    .       +       dbSNP   SNV     .       ␣
↪ID=ID_2;Variant_seq=C,G;Individual=0,1,2,3;Genotype=0:1,0:0,1:1,0:1;zero_length_
↪insertion=True
chr16 49302124        49302125        ID_3    .       +       dbSNP   SNV     .       ␣
↪ID=ID_3;Variant_seq=C,T;Individual=0,1,3;Genotype=0:1,2:2,0:2;zero_length_
↪insertion=True
chr16 49302364        49302365        ID_4    .       +       dbSNP   SNV     .       ␣
↪ID=ID_4;Variant_seq=G;Individual=0,1;Genotype=0:0,0:0;zero_length_insertion=True
chr16 49302699        49302700        ID_5    .       +       dbSNP   SNV     .       ␣
↪ID=ID_5;Variant_seq=C,T;Individual=2,3;Genotype=0:1,0:0;zero_length_insertion=True
chr16 49303083        49303084        ID_6    .       +       dbSNP   SNV     .       ␣
↪ID=ID_6;Variant_seq=T,G,A;Individual=3;Genotype=1,2:;zero_length_insertion=True
chr16 49303426        49303427        ID_8    .       +       dbSNP   SNV     .       ␣
↪ID=ID_8;Variant_seq=T;Individual=0;Genotype=0:0;zero_length_insertion=True
chr16 49303595        49303596        ID_9    .       +       dbSNP   SNV     .       ␣
↪ID=ID_9;Variant_seq=A,G,T;Individual=0,1,3;Genotype=1:2,3:3,1:3;zero_length_
↪insertion=True
```

As shown, the default usage strips the leading pragmas (`##gvf-version 1.07`, *etc.*), but adding the `--keep-header` option will preserve pragmas as BED elements that use `_header` as a chromosome name:

```
$ gvf2bed --keep-header < foo.gvf
_header       0       1       ##gvf-version 1.07
_header       1       2       ##feature-ontology http://www.sequenceontology.org/
↪resources/obo_files/current_release.obo
_header       2       3       ##multi-individual NA19240,NA18507,NA12878,NA19238
_header       3       4       ##genome-build NCBI B36.3
_header       4       5       ##sequence-region chr16 1 88827254
chr16 49291359        49291360        ID_2    .       +       dbSNP   SNV     .       ␣
↪ID=ID_2;Variant_seq=C,G;Individual=0,1,2,3;Genotype=0:1,0:0,1:1,0:1;zero_length_
↪insertion=True
chr16 49302124        49302125        ID_3    .       +       dbSNP   SNV     .       ␣
↪ID=ID_3;Variant_seq=C,T;Individual=0,1,3;Genotype=0:1,2:2,0:2;zero_length_
↪insertion=True
chr16 49302364        49302365        ID_4    .       +       dbSNP   SNV     .       ␣
↪ID=ID_4;Variant_seq=G;Individual=0,1;Genotype=0:0,0:0;zero_length_insertion=True
```

(continues on next page)

(continued from previous page)

```
chr16 49302699        49302700        ID_5    .       +       dbSNP   SNV     .
↪ID=ID_5;Variant_seq=C,T;Individual=2,3;Genotype=0:1,0:0;zero_length_insertion=True
chr16 49303083        49303084        ID_6    .       +       dbSNP   SNV     .
↪ID=ID_6;Variant_seq=T,G,A;Individual=3;Genotype=1,2:;zero_length_insertion=True
chr16 49303426        49303427        ID_8    .       +       dbSNP   SNV     .
↪ID=ID_8;Variant_seq=T;Individual=0;Genotype=0:0;zero_length_insertion=True
chr16 49303595        49303596        ID_9    .       +       dbSNP   SNV     .
↪ID=ID_9;Variant_seq=A,G,T;Individual=0,1,3;Genotype=1:2,3:3,1:3;zero_length_
↪insertion=True
```

Note: Zero-length insertion elements are given an extra attribute called `zero_length_insertion` which lets a BED-to-GVF or other parser know that the element will require conversion back to a right-closed element `[a, b]`, where `a` and `b` are equal.

Note: Note the conversion from 1- to 0-based coordinate indexing, in the transition from GVF to BED. *BEDOPS supports operations on input with any coordinate indexing*, but the coordinate change made here is believed to be convenient for most end users.

Column mapping

In this section, we describe how GVF columns are mapped to BED columns. We start with the first six UCSC BED columns as follows:

GVF field	BED column index	BED field
seqid	1	chromosome
start	2	start
end	3	stop
ID (via attributes)	4	id
score	5	score
strand	6	strand

The remaining columns are mapped as follows:

GVF field	BED column index	BED field
source	7	
type	8	
phase	9	
attributes	10	

When we encounter zero-length insertion elements (which are defined where the `start` and `stop` GVF field values are equivalent), the `start` coordinate is decremented to convert to 0-based, half-open indexing, and a `zero_length_insertion` attribute is added to the `attributes` field value.

Downloads

- Sample GVF dataset: `foo.gvf`

psl2bed

The `psl2bed` script converts 0-based, half-open `[start-1, end)` Pattern Space Layout (PSL) to sorted, 0-based, half-open `[start-1, end)` extended BED-formatted data.

For convenience, we also offer `psl2starch`, which performs the extra step of creating a Starch-formatted archive.

Dependencies

The `psl2bed` script requires *convert2bed*. The `psl2starch` script requires *starch*. Both dependencies are part of a typical BEDOPS installation.

This script is also dependent on input that follows the PSL specification.

Tip: Conversion of data which are PSL-like, but which do not follow the specification can cause parsing issues. If you run into problems, please check that your input follows the PSL specification.

Source

The `psl2bed` and `psl2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `psl2bed` script parses PSL from standard input and prints sorted BED to standard output. The `psl2starch` script uses an extra step to parse GFF to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a headered PSL file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

If your data contains a record with multiple blocks (`block count` is greater than one, and the `tStarts` field has multiple target start positions), you can use the `--split` option to print that record to separate BED elements, each with a start position defined by `tStarts` and a length defined by the associated value in the `blockSizes` string.

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If you are sorting data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample PSL input called `foo.psl` (see the *Downloads* section to grab this file).

```
psLayout version 3

match   mis-    rep.    N's     Q gap   Q gap   T gap   T gap   strand  Q
↪  Q       Q       Q       T               T       T       T       block
↪blockSizes      qStarts  tStarts
        match   match           count   bases   count   bases           name
↪  size    start   end     name            size    start   end     count
-------------------------------------------------------------------------------
↪-------------------------------------------------------------------------
35      0       0       0       0       0       0       0       +       foo     50
↪  15      50      chrX    155270560       40535836        40535871        1
↪35,     15,     40535836,
34      2       0       0       0       0       0       0       +       foo     50
↪  14      50      chrX    155270560       68019028        68019064        1
↪36,     14,     68019028,
33      2       0       0       0       0       0       0       +       foo     50
↪  14      49      chrX    155270560       43068135        43068170        1
↪35,     14,     43068135,
35      2       0       0       0       0       0       0       +       foo     50
↪  13      50      chr8    146364022       131572122       131572159       1
↪37,     13,     131572122,
30      0       0       0       0       0       0       0       +       foo     50
↪  14      44      chr6    171115067       127685756       127685786       1
↪30,     14,     127685756,
30      0       0       0       0       0       0       0       +       foo     50
↪  14      44      chr6    171115067       93161871        93161901        1
↪30,     14,     93161871,
31      0       0       0       0       0       0       0       +       foo     50
↪  13      44      chr5    180915260       119897315       119897346       1
↪31,     13,     119897315,
30      0       0       0       0       0       0       0       +       foo     50
↪  14      44      chr5    180915260       1232.4.37       1232.4.375      1
↪30,     14,     1232.4.37,
...
```

We can convert it to sorted BED data in the following manner:

```
$ psl2bed < foo.psl
chr1    30571100        30571135        foo     50      -       35      0       0
↪  0       0       0       0       0       15      50      249250621       1
↪35,     0,      30571100,
chr1    69592160        69592195        foo     50      -       34      1       0
↪  0       0       0       0       0       15      50      249250621       1
↪35,     0,      69592160,
chr1    107200050       107200100       foo     50      +       50      0       0
↪  0       0       0       0       0       0       50      249250621       1
↪50,     0,      107200050,
chr11   12618347        12618389        foo     50      +       39      3       0
↪  0       0       0       0       0       8       50      135006516       1
↪42,     8,      12618347,
chr11   32933028        32933063        foo     50      +       35      0       0
↪  0       1       1       0       0       8       44      135006516       2       4,
↪31,    8,13,   32933028,32933032,
chr11   80116421        80116457        foo     50      +       35      1       0
↪  0       0       0       0       0       14      50      135006516       1
↪36,     14,     80116421,
chr11   133952291       133952327       foo     50      +       34      2       0
↪  0       0       0       0       0       14      50      135006516       1
↪36,     14,     133952291,
```

(continues on next page)

(continued from previous page)

```
chr13   99729482        99729523        foo     50      +       39      2       0
↪  0       0       0       0       0       8       49      115169878       1
↪41,     8,      99729482,
chr13   111391852       111391888       foo     50      +       34      2       0
↪  0       0       0       0       0       14      50      115169878       1
↪36,     14,     111391852,
chr16   8149657 8149694 foo     50      +       36      1       0       0       0
↪  0       0       0       13      50      90354753        1       37,     13,
↪8149657,
...
```

As you see here, the header data of a headered PSL file is discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Here is a demonstration of conversion of the same headered input, adding the `--keep-header` option:

```
$ psl2bed --keep-header < foo.psl
_header 0       1       psLayout version 3
_header 1       2
_header 2       3       match   mis-    rep.    N's     Q gap   Q gap   T gap   T gap
↪  strand  Q               Q       Q       Q       T               T       T       T
↪      block   blockSizes      qStarts  tStarts
_header 3       4       match   match           count   bases   count   bases
↪  name            size    start   end     name            size    start   end
↪count
_header 4       5       ----------------------------------------------------------------
↪--------------------------------------------------------------------------------
↪--------------
chr1    30571100        30571135        foo     50      -       35      0       0
↪  0       0       0       0       0       15      50      249250621       1
↪35,     0,      30571100,
chr1    69592160        69592195        foo     50      -       34      1       0
↪  0       0       0       0       0       15      50      249250621       1
↪35,     0,      69592160,
chr1    107200050       107200100       foo     50      +       50      0       0
↪  0       0       0       0       0       0       50      249250621       1
↪50,     0,      107200050,
chr11   12618347        12618389        foo     50      +       39      3       0
↪  0       0       0       0       0       8       50      135006516       1
↪42,     8,      12618347,
chr11   32933028        32933063        foo     50      +       35      0       0
↪  0       1       1       0       0       8       44      135006516       2       4,
↪31,    8,13,   32933028,32933032,
chr11   80116421        80116457        foo     50      +       35      1       0
↪  0       0       0       0       0       14      50      135006516       1
↪36,     14,     80116421,
chr11   133952291       133952327       foo     50      +       34      2       0
↪  0       0       0       0       0       14      50      135006516       1
↪36,     14,     133952291,
chr13   99729482        99729523        foo     50      +       39      2       0
↪  0       0       0       0       0       8       49      115169878       1
↪41,     8,      99729482,
chr13   111391852       111391888       foo     50      +       34      2       0
↪  0       0       0       0       0       14      50      115169878       1
↪36,     14,     111391852,
```

(continues on next page)

(continued from previous page)

```
chr16   8149657 8149694 foo     50      +       36      1       0       0       0
↪  0       0       0       13      50      90354753        1       37,     13,
↪8149657,
...
```

With this option, the `psl2bed` and `psl2starch` scripts are completely "non-lossy". Use of `awk` or other scripting tools can munge these data back into a PSL-formatted file.

This example PSL file contains one record with a block count of 2. If we were to add the `--split` option, this record would be split into two separate BED elements that have start positions `32933028` and `32933032`, with lengths `4` and `31`, respectively. These elements fall within the genomic range already defined by the `tStart` and `tEnd` fields (`32933028` and `32933063`).

Note: The `psl2bed` and `psl2starch` scripts work with headered or headerless PSL data.

Note: By default, the `psl2bed` and `psl2starch` scripts assume that PSL data do *not* need splitting. If you expect your data to contain multiple blocks, add the `--split` option.

Column mapping

In this section, we describe how PSL columns are mapped to BED columns. We start with the first six UCSC BED columns as follows:

PSL field	BED column index	BED field
tName	1	chromosome
tStart(*)	2	start
tEnd(*)	3	stop
qName	4	id
matches	5	score
strand	6	strand

The remaining PSL columns are mapped, in order, to the remaining columns of the BED output:

PSL field	BED column index	BED field
qSize	7	
misMatches	8	
repMatches	9	
nCount	10	
qNumInsert	11	
qBaseInsert	12	
tNumInsert	13	
tBaseInsert	14	
qStart	15	
qEnd	16	
tSize	17	
blockCount	18	
blockSizes	19	
qStarts	20	
tStarts	21	

This is a lossless mapping. Because we have mapped all columns, we can translate converted BED data back to headerless PSL with a simple `awk` statement that permutes columns to PSL-based ordering:

```
$ awk 'BEGIN { OFS = "\t" } \
  { \
    print $5" "$8" "$9" "$10" "$11" "$12" "$13" "$14" "$6" "$4" "$7" "$15" "$16" "$1"
↪"$17" "$2" "$3" "$18" "$19" "$20" "$21 }' converted.bed > original.psl
```

In the case where the `--split` option is added, the `tStart` and `tEnd` fields are replaced with each of the values in the larger `tStarts` string, added to the respective values in the larger `blockSizes` string. This is still a lossless conversion, but modifications to the `awk` script printed above would be required to rebuild the original PSL.

Downloads

- Sample PSL dataset: `foo.psl`

rmsk2bed

The `rmsk2bed` script converts 1-based, closed `[start, end]` RepeatMasker annotation output (OUT) to sorted, 0-based, half-open `[start-1, end)` extended BED-formatted data.

For convenience, we also offer `rmsk2starch`, which performs the extra step of creating a *Starch-formatted* archive.

Dependencies

The `rmsk2bed` script requires *convert2bed*. The `rmsk2starch` script requires *starch*. Both dependencies are part of a typical BEDOPS installation.

This script is also dependent on input that follows the RepeatMasker annotation output specification, outlined here: http://www.repeatmasker.org/webrepeatmaskerhelp.html.

Source

The `rmsk2bed` and `rmsk2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `rmsk2bed` script parses RepeatMasker annotation output from standard input and prints sorted BED to standard output. The `rmsk2starch` script uses an extra step to parse RepeatMasker annotation output to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a RepeatMasker annotation output file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If sorting converted data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample RepeatMasker annotation output input called `foo.out` (see the *Downloads* section to grab this file).

```
  SW  perc perc perc  query    position in query     matching repeat      position in␣
↪ repeat
score div. del. ins.  sequence begin  end  (left)    repeat  class/family   begin end␣
↪(left) ID
...
 1320 15.6  6.2  0.0  HSU08988  6563 6781 (22462) C  MER7A   DNA/MER2_type    (0) ␣
↪337  104  20
12279 10.5  2.1  1.7  HSU08988  6782 7718 (21525) C  Tigger1 DNA/MER2_type    (0)␣
↪2418 1486  19
 1769 12.9  6.6  1.9  HSU08988  7719 8022 (21221) C  AluSx   SINE/Alu         (0) ␣
↪317    1  17
12279 10.5  2.1  1.7  HSU08988  8023 8694 (20549) C  Tigger1 DNA/MER2_type  (932)␣
↪1486  818  19
 2335 11.1  0.3  0.7  HSU08988  8695 9000 (20243) C  AluSg   SINE/Alu         (5) ␣
↪305    1  18
12279 10.5  2.1  1.7  HSU08988  9001 9695 (19548) C  Tigger1 DNA/MER2_type (1600) ␣
↪818    2  19
  721 21.2  1.4  0.0  HSU08988  9696 9816 (19427) C  MER7A   DNA/MER2_type  (224) ␣
↪122    2  20
```

We can convert it to sorted BED data in the following manner:

```
$ rmsk2bed < foo.out
HSU08988      6562    6781    MER7A   1320    -       15.6    6.2     0.0     (22462)␣
↪DNA/MER2_type   (0)     337     104     20
HSU08988      6781    7718    Tigger1 12279   -       10.5    2.1     1.7     (21525)␣
↪DNA/MER2_type   (0)     2418    1486    19
HSU08988      7718    8022    AluSx   1769    -       12.9    6.6     1.9     (21221)␣
↪SINE/Alu        (0)     317     1       17
HSU08988      8022    8694    Tigger1 12279   -       10.5    2.1     1.7     (20549)␣
↪DNA/MER2_type   (932)   1486    818     19
HSU08988      8694    9000    AluSg   2335    -       11.1    0.3     0.7     (20243)␣
↪SINE/Alu        (5)     305     1       18
HSU08988      9000    9695    Tigger1 12279   -       10.5    2.1     1.7     (19548)␣
↪DNA/MER2_type   (1600)  818     2       19
HSU08988      9695    9816    MER7A   721     -       21.2    1.4     0.0     (19427)␣
↪DNA/MER2_type   (224)   122     2       20
```

Note: Use *bedops –merge* to merge elements, *e.g.*: `rmsk2bed < foo.out | bedops --merge - > merged_repeatmasker_elements.bed`

As shown above, we strip the header element, but adding the `--keep-header` option will preserve this header as a BED element that uses `_header` as a chromosome name:

```
$ rmsk2bed --keep-header < foo.out
HSU08988      6562    6781    MER7A   1320    -       15.6    6.2     0.0     (22462)␣
↪DNA/MER2_type   (0)     337     104     20
HSU08988      6781    7718    Tigger1 12279   -       10.5    2.1     1.7     (21525)␣
↪DNA/MER2_type   (0)     2418    1486    19
HSU08988      7718    8022    AluSx   1769    -       12.9    6.6     1.9     (21221)␣
↪SINE/Alu        (0)     317     1       17
HSU08988      8022    8694    Tigger1 12279   -       10.5    2.1     1.7     (20549)␣
↪DNA/MER2_type   (932)   1486    818     19
HSU08988      8694    9000    AluSg   2335    -       11.1    0.3     0.7     (20243)␣
↪SINE/Alu        (5)     305     1       18
HSU08988      9000    9695    Tigger1 12279   -       10.5    2.1     1.7     (19548)␣
↪DNA/MER2_type   (1600)  818     2       19
HSU08988      9695    9816    MER7A   721     -       21.2    1.4     0.0     (19427)␣
↪DNA/MER2_type   (224)   122     2       20
_header       0       1       SW  perc perc perc  query    position in query    ␣
↪matching repeat      position in  repeat
_header       1       2       score div. del. ins.  sequence begin  end  (left)   ␣
↪repeat  class/family   begin end (left) ID
_header       2       3       ...
```

Note: Note the conversion from 1- to 0-based coordinate indexing, in the transition from RepeatMasker annotation output to BED. *BEDOPS supports operations on input with any coordinate indexing*, but the coordinate change made here is believed to be convenient for most end users.

Column mapping

In this section, we describe how RepeatMasker annotation columns are mapped to BED columns. We start with the first six UCSC BED columns as follows:

RepeatMasker annotation field	BED column index	BED field
Query sequence	1	chromosome
Query start	2	start
Query end	3	stop
Repeat name	4	id
Smith-Waterman score	5	score
Strand	6	strand

The remaining columns are mapped as follows:

RepeatMasker annotation field	BED column index	BED field
Percentage, substitutions	7	
Percentage, deleted bases	8	
Percentage, inserted bases	9	
Bases in query, past match	10	
Repeat class	11	
Bases in complement of the repeat consensus sequence	12	
Match start	13	
Match end	14	
Unique ID	15	
Higher-scoring match (optional)	16	

Downloads

- Sample RepeatMasker annotation dataset: `foo.out`

sam2bed

The `sam2bed` script converts 1-based, closed `[start, end]` Sequence Alignment/Map (SAM) to sorted, 0-based, half-open `[start-1, end)` UCSC BED data.

For convenience, we also offer `sam2starch`, which performs the extra step of creating a *Starch-formatted* archive.

The `sam2bed` script is "non-lossy" (with the use of specific options, described below). Other toolkits tend to throw out information from the original SAM input upon conversion; `sam2bed` retains everything, facilitating reuse of converted data and conversion to other formats.

Tip: Doing the extra step of creating a *Starch-formatted* archive can save a lot of space relative to the original SAM format, up to 33% of the original SAM dataset, while offering per-chromosome random access.

Dependencies

The `sam2bed` wrapper script is dependent upon the installation of SAMtools and *convert2bed*. The `sam2starch` wrapper script is further dependent on the installation of the *starch* binary, part of a typical BEDOPS installation.

Source

The `sam2bed` and `sam2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `sam2bed` script parses SAM data from standard input and prints *sorted* BED to standard output. The `sam2starch` script uses an extra step to parse SAM to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a SAM file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Tip: If you work with RNA-seq data, you can use the `--split` option to process reads with `N`-CIGAR operations, splitting them into separate BED elements.

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If sorting converted data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample binary input called `foo.sam` (see the *Downloads* section to grab this file).

```
@HD     VN:1.0 SO:coordinate
@SQ     SN:seq1 LN:5000
@SQ     SN:seq2 LN:5000
@CO     Example of SAM/BAM file format.
B7_591:4:96:693:509     73      seq1    1       99      36M     *       0       0
↪  CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG    <<<<<<<<<<<<<<<;<<<<<<<<<5<<<<<;:<;7
↪MF:i:18 Aq:i:73 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
EAS54_65:7:152:368:113  73      seq1    3       99      35M     *       0       0
↪  CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT     <<<<<<<<<<0<<<<655<<7<<<:9<<3/:<6):
↪MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
EAS51_64:8:5:734:57     137     seq1    5       99      35M     *       0       0
↪  AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC     <<<<<<<<<<<7;71<<;<;;<7;<<3;);3*8/5
↪MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
...
```

We can convert it to sorted BED data in the following manner (omitting standard error messages):

```
$ sam2bed < foo.sam
seq1    0       36      B7_591:4:96:693:509     99      +       73      36M     *
↪   0       0       CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG    <<<<<<<<<<<<<<;<<<<<<<<<5
↪<<<<<;:<;7    MF:i:18 Aq:i:73 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    2       37      EAS54_65:7:152:368:113  99      +       73      35M     *
↪   0       0       CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT      <<<<<<<<<<0<<<<655<<7<<<:9
↪<<3/:<6):      MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    4       39      EAS51_64:8:5:734:57     99      +       137     35M     *
↪   0       0       AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC      <<<<<<<<<<7;71<<;<;;<7;<
↪<3;);3*8/5     MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    5       41      B7_591:1:289:587:906    63      +       137     36M     *
↪   0       0       GTGGCTCATTGTAATTTTTTGTTTTAACTCTTCTCT     (-&----,----)-)-),'--)---
↪',+-,),''*,    MF:i:130        Aq:i:63 NM:i:5  UQ:i:38 H0:i:0  H1:i:0
...
```

Note also that we strip the header section from the output. If we want to keep this, the use of the `--keep-header` option will preserve the BAM file's header, turning it into BED elements that use `_header` as a chromosome name.

Here's an example:

```
$ sam2bed --keep-header < foo.sam
_header 0       1       @HD     VN:1.0 SO:coordinate
_header 1       2       @SQ     SN:seq1 LN:5000
_header 2       3       @SQ     SN:seq2 LN:5000
_header 3       4       @CO     Example of SAM/BAM file format.
seq1    0       36      B7_591:4:96:693:509     99      +       73      36M     *
↪   0       0       CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG    <<<<<<<<<<<<<<;<<<<<<<<<5
↪<<<<<;:<;7    MF:i:18 Aq:i:73 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    2       37      EAS54_65:7:152:368:113  99      +       73      35M     *
↪   0       0       CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT      <<<<<<<<<<0<<<<655<<7<<<:9
↪<<3/:<6):      MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    4       39      EAS51_64:8:5:734:57     99      +       137     35M     *
↪   0       0       AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC      <<<<<<<<<<7;71<<;<;;<7;<
↪<3;);3*8/5     MF:i:18 Aq:i:66 NM:i:0  UQ:i:0  H0:i:1  H1:i:0
seq1    5       41      B7_591:1:289:587:906    63      +       137     36M     *
↪   0       0       GTGGCTCATTGTAATTTTTTGTTTTAACTCTTCTCT     (-&----,----)-)-),'--)---
↪',+-,),''*,    MF:i:130        Aq:i:63 NM:i:5  UQ:i:38 H0:i:0  H1:i:0
...
```

With this option, the `sam2bed` and `sam2starch` scripts are completely "non-lossy" (with the exception of unmapped reads; see note below). Use of `awk` or other scripting tools can munge these data back into a SAM-formatted file.

Note: The provided scripts **strip out unmapped reads** from the SAM file. We believe this makes sense under most circumstances. Add the `--all-reads` option if you need unmapped and mapped reads.

Note: Note the conversion from 1- to 0-based coordinates. While BEDOPS fully supports 0- and 1-based coordinates, the coordinate change in BED is believed to be convenient to most end users.

Column mapping

In this section, we describe how SAM columns are mapped to BED columns. We start with the first six UCSC BED columns as follows:

SAM field	BED column index	BED field
RNAME	1	chromosome
POS - 1	2	start
POS + length(CIGAR) - 1	3	stop
QNAME	4	id
MAPQ	5	score
16 & FLAG	6	strand

The remaining SAM columns are mapped as-is, in same order, to adjacent BED columns:

SAM field	BED column index	BED field
FLAG	7	
CIGAR	8	
RNEXT	9	
PNEXT	10	
TLEN	11	
SEQ	12	
QUAL	13	

Because we have mapped all columns, we can translate converted BED data back to headered or headerless SAM reads with a simple `awk` statement (or other script) that reverts back to 1-based coordinates and permutes columns to SAM-based ordering.

Downloads

- Sample SAM dataset: `foo.sam`

vcf2bed

The `vcf2bed` script converts 1-based, closed `[start, end]` Variant Call Format v4.2 (VCF) to sorted, 0-based, half-open `[start-1, start)` extended BED data.

Note: Note that this script converts from `[start, end]` to `[start-1, start)`. Unless the `--snvs`, `--insertions` or `--deletions` options are added, we perform the equivalent of a *single-base insertion* to make BED output that is guaranteed to work with BEDOPS, **regardless of what the actual variant may be**, to allow operations to be performed. The converted output contains additional columns which allow reconstruction of the original VCF data and associated variant parameters.

For convenience, we also offer `vcf2starch`, which performs the extra step of creating a *Starch-formatted* archive.

Dependencies

The `vcf2bed` script requires *convert2bed*. The `vcf2starch` script requires *starch*. Both dependencies are part of a typical BEDOPS installation.

This script is also dependent on input that follows the VCF v4.2 specification.

Tip: Conversion of data which are VCF-like, but which do not follow the specification can cause parsing issues. If you run into problems, please check that your input follows the VCF specification using validation tools, such as those packaged with VCFTools.

Source

The `vcf2bed` and `vcf2starch` conversion scripts are part of the binary and source downloads of BEDOPS. See the *Installation* documentation for more details.

Usage

The `vcf2bed` script parses VCF from standard input and prints sorted BED to standard output. The `vcf2starch` script uses an extra step to parse VCF to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a VCF file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Note: By default, multiple BED annotations are printed if there are multiple alternate alleles in a variant call. Use the `--do-not-split-alt-alleles` option to preserve the alternate allele string and print only one BED element for the variant call.

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If the converted BED output looks truncated or incomplete, and you are converting a VCF file that is larger than the capacity of your `/tmp` folder, then use the `--sort-tmpdir` option to specify an alternative directory to store intermediate sort results. Or, if you do not want to sort the converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If you are sorting data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Customized variant handling

By default, the `vcf2bed` script translates all variants to single-base positions in the resulting BED output. Depending on the category of variant you are interested in, however, you may want more specific categories handled differently.

Based on the VCF v4.2 specification, we also provide three custom options for filtering input for each of the three types of variants listed: `--snvs`, `--insertions` and `--deletions`. In each case, we use the length of the reference and alternate alleles to determine which type of variant is being handled.

In addition, using any of these three custom options automatically results in processing of mixed variant records for a microsatellite, where present. For instance, the following record contains a mixture of a deletion and insertion variant (`GTC -> G` and `GTC -> GTCT`, respectively):

```
#CHROM POS     ID        REF    ALT     QUAL FILTER INFO                              ␣
↪FORMAT      NA00001        NA00002        NA00003
20     12.4.377 microsat1 GTC    G,GTCT  50   PASS   NS=3;DP=9;AA=G                   ␣
↪ GT:GQ:DP    0/1:35:4       0/2:17:2       1/1:40:3
```

When using `--snvs`, `--insertions` or `--deletions`, this record is split into two distinct BED records and filtered depending on which custom option was chosen. The `--insertions` option would only export the single-base position of the insertion in this mixed variant, while `--deletions` would show the deletion.

In this way, you can control what kinds of variants are translated into BED outputs—most importantly, there is also no confusion about what the length of the BED element signifies.

Example

To demonstrate these scripts, we use a sample VCF input called `foo.vcf` (see the *Downloads* section to grab this file).

Note: This data is also publicly available from the Broad Institute.

```
##fileformat=VCFv4.0
##FILTER=<ID=LowQual,Description="QUAL < 50.0">
##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt␣
↪alleles in the order listed">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth (only filtered reads␣
↪used for calling)">
##FORMAT=<ID=GQ,Number=1,Type=Float,Description="Genotype Quality">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=PL,Number=3,Type=Float,Description="Normalized, Phred-scaled likelihoods␣
↪for AA,AB,BB genotypes where A=ref and B=alt; not applicable if site is not␣
↪biallelic">
##INFO=<ID=AC,Number=.,Type=Integer,Description="Allele count in genotypes, for each␣
↪ALT allele, in the same order as listed">
##INFO=<ID=AF,Number=.,Type=Float,Description="Allele Frequency, for each ALT allele,␣
↪in the same order as listed">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called␣
↪genotypes">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP Membership">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=DS,Number=0,Type=Flag,Description="Were any of the samples downsampled?">
##INFO=<ID=Dels,Number=1,Type=Float,Description="Fraction of Reads Containing␣
↪Spanning Deletions">
##INFO=<ID=HRun,Number=1,Type=Integer,Description="Largest Contiguous Homopolymer Run␣
↪of Variant Allele In Either Direction">
##INFO=<ID=HaplotypeScore,Number=1,Type=Float,Description="Consistency of the site␣
↪with two (and only two) segregating haplotypes">
##INFO=<ID=MQ,Number=1,Type=Float,Description="RMS Mapping Quality">
##INFO=<ID=MQ0,Number=1,Type=Integer,Description="Total Mapping Quality Zero Reads">
##INFO=<ID=QD,Number=1,Type=Float,Description="Variant Confidence/Quality by Depth">
##INFO=<ID=SB,Number=1,Type=Float,Description="Strand Bias">
##INFO=<ID=VQSLOD,Number=1,Type=Float,Description="log10-scaled probability of␣
↪variant being true under the trained gaussian mixture model">
##UnifiedGenotyperV2="analysis_type=UnifiedGenotyperV2 input_file=[TEXT CLIPPED FOR␣
↪CLARITY]"
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  NA12878
```

(continues on next page)

(continued from previous page)

```
chr1    873762  .       T       G       5231.78 PASS    AC=1;AF=0.50;AN=2;DP=315;
↪Dels=0.00;HRun=2;HaplotypeScore=15.11;MQ=91.05;MQ0=15;QD=16.61;SB=-1533.02;VQSLOD=-
↪1.5473      GT:AD:DP:GQ:PL  0/1:173,141:282:99:255,0,255
chr1    877664  rs3828047       A       G       3931.66 PASS    AC=2;AF=1.00;AN=2;DB;
↪DP=105;Dels=0.00;HRun=1;HaplotypeScore=1.59;MQ=92.52;MQ0=4;QD=37.44;SB=-1152.13;
↪VQSLOD=0.1185      GT:AD:DP:GQ:PL  1/1:0,105:94:99:255,255,0
chr1    899282  rs28548431      C       T       71.77   PASS    AC=1;AF=0.50;AN=2;DB;
↪DP=4;Dels=0.00;HRun=0;HaplotypeScore=0.00;MQ=99.00;MQ0=0;QD=17.94;SB=-46.55;VQSLOD=-
↪1.9148 GT:AD:DP:GQ:PL  0/1:1,3:4:25.92:103,0,26
chr1    974165  rs9442391       T       C       29.84   LowQual AC=1;AF=0.50;AN=2;DB;
↪DP=18;Dels=0.00;HRun=1;HaplotypeScore=0.16;MQ=95.26;MQ0=0;QD=1.66;SB=-0.98␣
↪GT:AD:DP:GQ:PL  0/1:14,4:14:60.91:61,0,255
```

We can convert VCF to sorted BED data in the following manner:

```
$ vcf2bed < foo.vcf
chr1    873761  873762  .       5231.78 T       G       PASS    AC=1;AF=0.50;AN=2;
↪DP=315;Dels=0.00;HRun=2;HaplotypeScore=15.11;MQ=91.05;MQ0=15;QD=16.61;SB=-1533.02;
↪VQSLOD=-1.5473      GT:AD:DP:GQ:PL  0/1:173,141:282:99:255,0,255
chr1    877663  877664  rs3828047       3931.66 A       G       PASS    AC=2;AF=1.00;
↪AN=2;DB;DP=105;Dels=0.00;HRun=1;HaplotypeScore=1.59;MQ=92.52;MQ0=4;QD=37.44;SB=-
↪1152.13;VQSLOD=0.1185      GT:AD:DP:GQ:PL  1/1:0,105:94:99:255,255,0
chr1    899281  899282  rs28548431      71.77   C       T       PASS    AC=1;AF=0.50;
↪AN=2;DB;DP=4;Dels=0.00;HRun=0;HaplotypeScore=0.00;MQ=99.00;MQ0=0;QD=17.94;SB=-46.55;
↪VQSLOD=-1.9148 GT:AD:DP:GQ:PL  0/1:1,3:4:25.92:103,0,26
chr1    974164  974165  rs9442391       29.84   T       C       LowQual AC=1;AF=0.50;
↪AN=2;DB;DP=18;Dels=0.00;HRun=1;HaplotypeScore=0.16;MQ=95.26;MQ0=0;QD=1.66;SB=-0.98␣
↪GT:AD:DP:GQ:PL  0/1:14,4:14:60.91:61,0,255
```

As you see here, the header data of the VCF file is discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

Here we use `--keep-header` with our example dataset:

```
$ vcf2bed --keep-header < foo.vcf
_header 0       1       ##fileformat=VCFv4.0
_header 1       2       ##FILTER=<ID=LowQual,Description="QUAL < 50.0">
_header 2       3       ##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic␣
↪depths for the ref and alt alleles in the order listed">
_header 3       4       ##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth␣
↪(only filtered reads used for calling)">
_header 4       5       ##FORMAT=<ID=GQ,Number=1,Type=Float,Description="Genotype␣
↪Quality">
_header 5       6       ##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
_header 6       7       ##FORMAT=<ID=PL,Number=3,Type=Float,Description="Normalized,␣
↪Phred-scaled likelihoods for AA,AB,BB genotypes where A=ref and B=alt; not␣
↪applicable if site is not biallelic">
_header 7       8       ##INFO=<ID=AC,Number=.,Type=Integer,Description="Allele count␣
↪in genotypes, for each ALT allele, in the same order as listed">
_header 8       9       ##INFO=<ID=AF,Number=.,Type=Float,Description="Allele␣
↪Frequency, for each ALT allele, in the same order as listed">
_header 9       10      ##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number␣
↪of alleles in called genotypes">
_header 10      11      ##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP Membership
↪">
```

(continues on next page)

(continued from previous page)

```
_header 11      12      ##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
_header 12      13      ##INFO=<ID=DS,Number=0,Type=Flag,Description="Were any of the␣
↪samples downsampled?">
_header 13      14      ##INFO=<ID=Dels,Number=1,Type=Float,Description="Fraction of␣
↪Reads Containing Spanning Deletions">
_header 14      15      ##INFO=<ID=HRun,Number=1,Type=Integer,Description="Largest␣
↪Contiguous Homopolymer Run of Variant Allele In Either Direction">
_header 15      16      ##INFO=<ID=HaplotypeScore,Number=1,Type=Float,Description=
↪"Consistency of the site with two (and only two) segregating haplotypes">
_header 16      17      ##INFO=<ID=MQ,Number=1,Type=Float,Description="RMS Mapping␣
↪Quality">
_header 17      18      ##INFO=<ID=MQ0,Number=1,Type=Integer,Description="Total␣
↪Mapping Quality Zero Reads">
_header 18      19      ##INFO=<ID=QD,Number=1,Type=Float,Description="Variant␣
↪Confidence/Quality by Depth">
_header 19      20      ##INFO=<ID=SB,Number=1,Type=Float,Description="Strand Bias">
_header 20      21      ##INFO=<ID=VQSLOD,Number=1,Type=Float,Description="log10-
↪scaled probability of variant being true under the trained gaussian mixture model">
_header 21      22      ##UnifiedGenotyperV2="analysis_type=UnifiedGenotyperV2 input_
↪file=[TEXT CLIPPED FOR CLARITY]"
_header 22      23      #CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO␣
↪  FORMAT  NA12878
chr1    873761  873762  .       5231.78 T       G       PASS    AC=1;AF=0.50;AN=2;
↪DP=315;Dels=0.00;HRun=2;HaplotypeScore=15.11;MQ=91.05;MQ0=15;QD=16.61;SB=-1533.02;
↪VQSLOD=-1.5473      GT:AD:DP:GQ:PL  0/1:173,141:282:99:255,0,255
chr1    877663  877664  rs3828047       3931.66 A       G       PASS    AC=2;AF=1.00;
↪AN=2;DB;DP=105;Dels=0.00;HRun=1;HaplotypeScore=1.59;MQ=92.52;MQ0=4;QD=37.44;SB=-
↪1152.13;VQSLOD=0.1185      GT:AD:DP:GQ:PL  1/1:0,105:94:99:255,255,0
chr1    899281  899282  rs28548431      71.77   C       T       PASS    AC=1;AF=0.50;
↪AN=2;DB;DP=4;Dels=0.00;HRun=0;HaplotypeScore=0.00;MQ=99.00;MQ0=0;QD=17.94;SB=-46.55;
↪VQSLOD=-1.9148 GT:AD:DP:GQ:PL  0/1:1,3:4:25.92:103,0,26
chr1    974164  974165  rs9442391       29.84   T       C       LowQual AC=1;AF=0.50;
↪AN=2;DB;DP=18;Dels=0.00;HRun=1;HaplotypeScore=0.16;MQ=95.26;MQ0=0;QD=1.66;SB=-0.98␣
↪GT:AD:DP:GQ:PL  0/1:14,4:14:60.91:61,0,255
```

With this option, the `vcf2*` scripts are completely "non-lossy". Use of `awk` or other scripting tools can munge these data back into a VCF-formatted file.

Note: Note the conversion from 1- to 0-based coordinate indexing, in the transition from VCF to BED. While BEDOPS supports 0- and 1-based coordinate indexing, the coordinate change made here is believed to be convenient for most end users.

Column mapping

In this section, we describe how VCF v4.2 columns are mapped to BED columns. We start with the first five UCSC BED columns as follows:

VCF v4.2 field	BED column index	BED field
#CHROM	1	chromosome
POS - 1	2	start
POS (*)	3	stop
ID	4	id
QUAL	5	score

The remaining columns are mapped as follows:

VCF v4.2 field	BED column index	BED field
REF	6	
ALT	7	
FILTER	8	
INFO	9	

If present in the VCF v4.2 input, the following columns are also mapped:

VCF v4.2 field	BED column index	BED field
FORMAT	10	
Sample ID 1	11	
Sample ID 2	12	
...	13, 14, etc.	

When using `--deletions`, the stop value of the BED output is determined by the length difference between ALT and REF alleles. Use of `--insertions` or `--snvs` yields a one-base BED element.

If the ALT field contains more than one allele, multiple BED records will be printed. Use the `--do-not-split` option if you only want one BED record per variant call.

The "meta-information" (starting with `##`) and "header" lines (starting with `#`) are discarded, unless the `--keep-headers` options is specified.

Downloads

- Sample VCF dataset: `foo.vcf`

wig2bed

The `wig2bed` script converts both *variable* - and *fixed* -step, 1-based, closed `[start, end]` UCSC Wiggle format (WIG) to sorted, 0-based, half-open `[start-1, end)` extended BED data.

In the case where WIG data are sourced from `bigWigToWig` or other tools that generate 0-based, half-open `[start-1, end)` WIG, a `--zero-indexed` option is provided to generate coordinate output without any re-indexing.

For convenience, we also offer `wig2starch`, which performs the extra step of creating a *Starch-formatted* archive.

The utility also supports multiple embedded WIG sections in a single file, which are output to the BED file with modified ID fields, using the `--multisplit` option.

Source

The `wig2bed` script requires *convert2bed*. The `wig2starch` script requires *starch*. Both dependencies are part of a typical BEDOPS installation.

Usage

The `wig2bed` script parses WIG from standard input and prints sorted BED to standard output. The `wig2starch` script uses an extra step to parse WIG to a compressed BEDOPS *Starch-formatted* archive, which is also directed to standard output.

The header data of a WIG file is usually discarded, unless you add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

If the input data contain WIG elements with a start position of 0, the default use of `wig2bed` and `wig2starch` will exit early with an `EINVAL` error. Add the `--zero-indexed` option to denote that the input WIG data are zero-indexed, and re-run the conversion tool to print unmodified output coordinates.

Tip: If your WIG input is potentially zero-indexed, *e.g.*, if derived from `bigWigToWig`, where the `bigWig` data are themselves sourced from BAM- or bedGraph-formatted data, then it is recommended to use the `--zero-indexed` option as a safety measure.

If your data contain multiple WIG sections, use the `--multisplit <basename>` option to split sections out to BED elements with modified ID fields. This option can be used in conjunction with the `--keep-header` option to preserve metadata.

Tip: By default, all conversion scripts now output sorted BED data ready for use with BEDOPS utilities. If you do not want to sort converted output, use the `--do-not-sort` option. Run the script with the `--help` option for more details.

Tip: If sorting converted data larger than system memory, use the `--max-mem` option to limit sort memory usage to a reasonable fraction of available memory, *e.g.*, `--max-mem 2G` or similar. See `--help` for more details.

Example

To demonstrate these scripts, we use a sample multi-section WIG input called `foo.wig` (see the *Downloads* section to grab this file). We can convert WIG to sorted BED data in the following manner:

```
$ wig2bed < foo.wig
chr1    147971108       147971158       id-1    -0.590000
chr1    147971146       147971196       id-2    0.120000
chr1    147971184       147971234       id-3    0.110000
chr1    147971222       147971272       id-4    -0.760000
...
```

Note: Even though our WIG input `foo.wig` has multiple sections, we can omit the use of `--multisplit`, because conversion and sorting puts everything into one sorted BED file. However, the header data of the WIG file is discarded.

If we want to preserve the header data, we can add the `--keep-header` option. In this case, BED elements are created from these data, using the chromosome name `_header` to denote content. Line numbers are specified in the start and stop coordinates, and unmodified header data are placed in the fourth column (ID field).

In the case of the sample input `foo.wig`, we will also need to add the `--multisplit` option, as header BED elements from each section will otherwise be collated in a non-sensical way. Adding `--multisplit` ensures that header data are converted and stored in separate BED files.

To demonstrate, we next repeat the above conversion, adding the `--keep-header` and `--multisplit` options:

```
$ wig2bed --multisplit bar --keep-header < foo.wig > foo.bed
```

Conversion of this two-section WIG input results in output with modified ID fields to denote their section association:

```
$ more foo.bed
_header 0       1       bar.1           track type=wiggle_0 name=foo description=foo
_header 1       2       bar.2           track type=wiggle_0 name=testfixed
_header 2       3       bar.2           fixedStep chrom=chrX start=100 step=10 span=5
chr1    147971108       147971158       bar.1-id-1    -0.590000
chr1    147971146       147971196       bar.1-id-2    0.120000
chr1    147971184       147971234       bar.1-id-3    0.110000
chr1    147971222       147971272       bar.1-id-4    -0.760000
chrX    99      104     bar.2-id-11   1.900000
chrX    109     114     bar.2-id-12   2.300000
chrX    119     124     bar.2-id-13   -0.100000
chrX    129     134     bar.2-id-14   1.100000
chrX    139     144     bar.2-id-15   4.100000
```

Note: Note the conversion from 1- to 0-based coordinate indexing, in the transition from WIG to BED. While BEDOPS supports 0- and 1-based coordinate indexing, the coordinate change made here is believed to be convenient for most end users.

In the case where the WIG data contain elements that have a start position of 0, the default use of `wig2bed` and `wig2starch` will exit early with an `EINVAL` error. Add the `--zero-indexed` option to denote that the WIG input is zero-indexed and re-run to convert without any coordinate shift.

Note: Multiple WIG sections in the input file are merged together by the default `wig2bed` behavior. When using the `--multisplit` option, each WIG section instead receives its own ID prefix.

Downloads

- Sample WIG dataset: `foo.wig`

2.7 Summary

These tables summarize BEDOPS utilities by option, file inputs and BED column requirements.

2.7.1 Set operation and statistical utilities

`bedextract`

- Efficiently extracts features from BED input.
- BEDOPS *bedextract* documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
`--list-chr`	Print every chromosome found in `input.bed`	1	1	3
`<chromosome>`	Retrieve all rows for specified chromosome, *e.g.* `bedextract chr8 input.bed`	1	1	3
`<query> <reference>`	Grab elements of `query` that overlap elements in reference. Same as `bedops -e -1 query reference`, except that this option fails when `query` contains fully-nested BED elements. May use `-` to indicate `stdin` for `reference` only.	2	2	3

bedmap

- Maps source signals from `map-file` onto qualified target regions from `ref-file`. Calculates an output for every `ref-file` element.
- BEDOPS *bedmap* documentation.

option	description
`--bases`	Reports the total number of bases from `map-file` that overlap the `ref-file` 's element.
`--bases-uniq`	Reports the number of distinct bases from `ref-file` 's element overlapped by elements in `ma`
`--bases-uniq-f`	Reports the fraction of distinct bases from `ref-file` 's element elements in `map-file`.
`--bp-ovr <int>`	Require `<int>` bases of overlap between elements of input files.
`--chrom <chromosome>`	Process data for given `<chromosome>` only.
`--count`	Reports the number of overlapping elements in `map-file`.
`--cv`	Reports the Coefficient of Variation: the result of `--stdev` divided by the result of `--mean`.
`--ec`	Error-check all input files (slower).
`--echo`	Echo each line from `ref-file`.
`--echo-map`	Reports the overlapping elements found in `map-file`.
`--echo-map-id`	Reports the IDs (4th column) from overlapping `map-file` elements.
`--echo-map-id-uniq`	List unique IDs from overlapping `map-file` elements.
`--echo-map-range`	Reports the genomic range of overlapping elements from `map-file`.
`--echo-map-score`	Reports the scores (5th column) from overlapping `map-file` elements.
`--echo-map-size`	Calculates difference between start and stop coordinates (or size) of each mapped element.
`--echo-overlap-size`	Calculates size of overlap between each mapped element and its reference element.
`--echo-ref-name`	Reports the first 3 fields of `ref-file` element in chrom:start-end format.
`--echo-ref-size`	Reports the length of the `ref-file` element.
`--faster`	**(Advanced)** Strong input assumptions are made. Review documents before use. Compatible wit
`--fraction-ref <val>`	The fraction of the element's size from `ref-file` that must overlap the element in `map-file`
`--fraction-map <val>`	The fraction of the element's size from `map-file` that must overlap the element in `ref-file`
`--fraction-both <val>`	Both `--fraction-ref <val>` and `--fraction-map <val>` must be true to qualify as
`--fraction-either <val>`	Both `--fraction-ref <val>` and `--fraction-map <val>` must be true to qualify as
`--exact`	Shorthand for `--fraction-both 1`. First three fields from `map-file` must be identical to
`--indicator`	Reports the presence of one or more overlapping elements in `map-file` as a binary value (`0` or
`--kth <val>`	Reports the value at the *k* th fraction. A generalized median-like calculation, where `--kth 0.`
`--mad <mult=1>`	Reports the 'median absolute deviation' of overlapping elements in `map-file`, multiplied by <
`--max`	Reports the highest score from overlapping elements in `map-file`.
`--max-element`	The lexicographically "smallest" element with the highest score from overlapping elements in `m`
`--max-element-rand`	A randomly-chosed element with the highest score from overlapping elements in `map-file`. If

Tabl

option	description
`--mean`	Reports the average score from overlapping elements in `map-file`.
`--median`	Reports the median score from overlapping elements in `map-file`.
`--min`	Reports the lowest score from overlapping elements in `map-file`.
`--min-element`	The lexicographically "smallest" element with the lowest score from overlapping elements in `ma`
`--min-element-rand`	A randomly-chosed element with the lowest score from overlapping elements in `map-file`. If
`--skip-unmapped`	Omits printing reference elements which do not associate with any mapped elements.
`--stdev`	Reports the square root of the result of `--variance`.
`--sum`	Reports the accumulated value from scores of overlapping elements in `map-file`.
`--sweep-all`	Reads through entire `map-file` dataset to avoid early termination that may cause SIGPIPE or
`--tmean <low> <hi>`	Reports the mean score from overlapping elements in `map-file`, after ignoring the bottom `<l`
`--variance`	Reports the variance of scores from overlapping elements in `map-file`.

bedops

- Offers set and multiset operations for files in BED format.
- BEDOPS *bedops* documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
`--chrom <chromosome>`	Process data for given `chromosome` only.	1	No imposed limit	3
`--complement, -c`	Reports the intervening intervals between the input coordinate segments.	1	No imposed limit	3
`--chop, -w`	Breaks up merged regions into fixed-size chunks, optionally anchored on start coordinates a fixed distance apart.	1	No imposed limit	3
`--difference, -d`	Reports the intervals found in the first file that are not present in any other input file.	2	No imposed limit	3
`--ec`	Error-check input files (slower).	1	No imposed limit	3
`--element-of, -e`	Reports rows from the first file that overlap, by a specified percentage or number of base pairs, the merged segments from all other input files.	2	No imposed limit	3
`--header`	Accept headers (VCF, GFF, SAM, BED, WIG) in any input file.	1	No imposed limit	3
`--intersect, -i`	Reports the intervals common to all input files.	2	No imposed limit	3
`--merge, -m`	Reports intervals from all input files, after merging overlapping and adjoining segments.	1	No imposed limit	3
`--not-element-of, -n`	Reports exactly everything that `--element-of` does not, given the same overlap criterion.	2	No imposed limit	3
`--partition, -p`	Reports all disjoint intervals from all input files. Overlapping segments are cut up into pieces at all segment boundaries.	1	No imposed limit	3
`--range L:R`	Add `L` bases to all start coordinates and `R` base to end coordinates. Either value may be positive or negative to grow or shrink regions, respectively. With the `-e` or `-n` operation, the first (reference) file is not padded, unlike all other files.	1	No imposed limit	3
`--range S`	Pad input file(s) coordinates symmetrically by `S` bases. This is shorthand for `--range -S:S`.	1	No imposed limit	3
`--symmdiff, -s`	Reports the intervals found in exactly one input file.	2	No imposed	3

closest-features

- For every element in `input-file`, find those elements in `query-file` nearest to its left and right edges.
- BEDOPS *closest-features* documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
(no option)	NA	2	2	3
`--chrom <chromosome>`	Process data for given `<chromosome>` only.	2	2	3
`--dist`	Output includes the signed distances between the `input-file` element and the closest elements in `query-file`.	2	2	3
`--ec`	Error-check all input files (slower).	2	2	3
`--no-overlaps`	Do not consider elements that overlap. Overlapping elements, otherwise, have highest precedence.	2	2	3
`--no-ref`	Do not echo elements from `input-file`.	2	2	3
`--closest`	Choose the nearest element from `query-file` only. Ties go to the leftmost closest element.	2	2	3

2.7.2 Sorting

sort-bed

- Sorts input BED file(s) into the order required by other utilities. Loads all input data into memory.
- BEDOPS *sort-bed* documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
(no option)	NA	1	1000	3
`--max-mem <val>`	`<val>` specifies the maximum memory usage for the *sort-bed* process, which is useful for very large BED inputs. For example, `--max-mem` may be `8G`, `8000M`, or `8000000000` to specify 8 GB of memory.	1	1000	3
`--unique`	Report unique elements (those which only occur once) in output.	1	1000	3
`--duplicates`	Report duplicate elements (those which occur 2+ times) in output.	1	1000	3

2.7.3 Compression and extraction

starch

- Lossless compression of any BED file.
- BEDOPS *starch* documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
(no option)	NA	1	1	3
`--bzip2` or `--gzip`	The internal compression method. The default `--bzip2` method favors storage efficiency, while `--gzip` favors compression and extraction time performance.	1	1	3
`--note="foo bar..."`	Append note to output archive metadata (optional).	1	1	3
`--report-progress`	Write progress to standard error stream for every N input elements.	1	1	3

unstarch

- Extraction of a `starch` archive or attributes.
- BEDOPS *unstarch* documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
(no option)	NA	1	1	NA
`--archive-type`	Show archive's compression type (either `bzip2` or `gzip`).	1	1	NA
`--archive-version`	Show archive version (at this time, either 1.x or 2.x).	1	1	NA
`--archive-timestamp`	Show archive creation timestamp (ISO 8601 format).	1	1	NA
`--bases <chromosome>`	Show total, non-unique base counts for optional `<chromosome>` (omitting `<chromosome>` shows total non-unique base count).	1	1	NA
`--bases-uniq <chromosome>`	Show unique base counts for optional `<chromosome>` (omitting `<chromosome>` shows total, unique base count).	1	1	NA
`<chromosome>`	Decompress information for a single `<chromosome>` only.	1	1	NA
`--duplicatesExist` or `--duplicatesExistAsString` with `<chromosome>`	Report if optional `<chromosome>` or chromosomes contain duplicate elements as 0/1 numbers or false/true strings	1	1	NA
`--elements <chromosome>`	Show element count for optional `<chromosome>` (omitting `<chromosome>` shows total element count).	1	1	NA
`--elements-max-string-length`	Show element maximum string length for optional `<chromosome>` (omitting `<chromosome>` shows maximum string length over all chromosomes).	1	1	NA
`--is-starch`	Test if the <starch-file> is a valid starch archive, returning 0/1 for a false/true result	1	1	NA
`--list` or `--list-json`	Print the metadata for a `starch` file, either in tabular form or with JSON formatting.	1	1	NA
`--list-chr` or `--list-chromosomes`	List all chromosomes in `starch` archive (similar to `bedextract --list-chr`).	1	1	NA
`--nestedsExist` or `--nestedsExistAsString` with `<chromosome>`	Report if optional `<chromosome>` or chromosomes contain nested elements as 0/1 numbers or false/true strings	1	1	NA
`--note`	Show descriptive note (if originally added to archive).	1	1	NA
`--signature` with `<chromosome>`	Show SHA-1 signature of specified chromosome (Base64-encoded) or all signatures if chromosome is not specified.	1	1	NA
`--verify-signature` with `<chromosome>`	Compare SHA-1 signature of specified chromosome with signature that is stored in the archive metadata, reporting error is mismatched.	1	1	NA

starchcat

- Merge multiple `starch` archive inputs into one `starch` archive output.
- BEDOPS *starchcat* documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
(no option)	NA	1	No imposed limit	NA
`--bzip2` or `--gzip`	The internal compression method. The default `--bzip2` method favors storage efficiency, while `--gzip` favors compression and extraction time performance.	1	No imposed limit	NA
`--note="foo bar..."`	Append note to output archive metadata (optional).	1	No imposed limit	NA
`--report-progress`	Write progress to standard error stream for every N input elements.	1	No imposed limit	NA

starchstrip

- Extract or filter a `starch` archive by one or more specified chromosome names.
- BEDOPS starchstrip documentation.

option	description	min. file inputs	max. file inputs	min. BED columns
(no option)	NA	1	No imposed limit	NA
`--include` or `--exclude` with <chromosomes>	Writes output with inclusion or exclusion of specified chromosome name records (comma-delimited string).	NA	No imposed limit	NA

2.8 Release

This document attempts to enumerate steps to get from a development branch to a final release, with all associated packages and documentation changes.

2.8.1 Preparation

Preparing a major, minor or maintenance release of BEDOPS from a development branch involves several steps, which we outline here:

1. Review the Github issues list
    a. Close out open documentation or feature issues, making necessary pushes to the current development branch.
    b. If any issues can't be closed out, rename the assigned version tag to the next anticipated release version (*e.g.*, *v2.4.37* to *v2p5p0*, etc.)
2. Pull the most recent commit for the development branch to a local folder on build hosts (Linux with sufficiently old kernel, current OS X, etc.).

a. Follow the *Installation (via source code)* documentation to build BEDOPS for the given platform.

   1) For Linux, we build a 64-bit version. It may help to use VirtualBox or a similar virtualization host to set up and run different (and consistent) versions of Linux build hosts.

   2) For Mac OS X, we currently build the Mac target with whatever the modern Xcode and current OS X release happens to be (currently, command-line tools that ship with Xcode 9 and OS X High Sierra/10.13). If things work correctly, build flags generate 64-bit binaries that should run on 10.10 and newer OS releases.

b. For all platforms, run test suites for various tools and conversion scripts; tests should pass on supported platforms. If not, add an Issue ticket, fix it, close it and start over with the build/test process.

c. If things work properly, make a bzip2-compressed tarball from the compiled binaries.

The naming scheme we currently use for Linux packages is as follows:

```
bedops_linux_x86_64-vX.Y.Z.tar.bz2 (64-bit)
```

Run `shasum -a 256` on the tarball to get its SHA256 hash (store this SHA256 hash in a file for later retrieval).

For the OS X Installer, use `productsign` per *OS X Installer* documentation to digitally sign the package. Compress the Installer with the Finder or *zip*:

```
BEDOPS.X.Y.Z.pkg.zip
```

The *X.Y.Z* scheme should follow the development branch name, *e.g.* 2.4.37, etc.

3. Collect tarballs and zipped Installer in one location for later addition with web browser, via BEDOPS Github web site.

2.8.2 Release

1. Merge BEDOPS development branch into master branch:

```
$ git checkout master
$ git pull origin master
$ git merge vXpYpZ
$ git push origin master
```

Ideally, whatever steps are used to merge the development branch into the master branch should preserve the overall commit history.

As before, the *X.Y.Z* scheme should follow the development branch name, *e.g.* 2.4.37, etc.

2. Add a new release via the Github site. Or click on the Draft a new release button from the Github Releases page.

Fill out the resulting form, as described below:

a. *Tag version* should be of the form *vX.Y.Z* (using the "semantic versioning" naming scheme triggers Github to set up useful and automatic package features).

Tags should be applied to the *master* branch, since we pushed the development branch up to the master branch.

b. *Release title* can be of the form *BEDOPS vX.Y.Z.*

c. *Describe this release* can be populated with the following Markdown-formatted boilerplate:

```
Downloads are available at the bottom of this page. Please read the [BEDOPS vX.Y.
↪Z revision history](http://bedops.readthedocs.io/en/latest/content/revision-
↪history.html#vX-Y-Z), which summarizes new features and fixes in this release.

------

### Linux
**bedops_linux_x86_64-vX.Y.Z.tar.bz2** (64-bit, SHA256: ``abcd1234``)
This package of BEDOPS vX.Y.Z binaries is for Linux 64-bit (glibc v2.17) hosts.␣
↪Those who require 32-bit or pre-2.17 glibc-compiled binaries will need to build␣
↪binaries from source code; please read [§2.2. Via source-code] (http://bedops.
↪readthedocs.io/en/latest/content/installation.html#via-source-code) of the␣
↪BEDOPS Installation document for details.

For installation instructions, please read [§2.1.1. Linux] (http://bedops.
↪readthedocs.io/en/latest/content/installation.html#linux) of the BEDOPS␣
↪Installation document.

------

### Mac OS X
**BEDOPS.X.Y.Z.pkg.zip**
This package of BEDOPS vX.Y.Z is a digitally-signed installer for 64-bit binaries␣
↪that run under OS X (10.10 - 10.13) on Intel-based Macs.

For installation instructions, please read [§2.1.2. Mac OS X] (http://bedops.
↪readthedocs.io/en/latest/content/installation.html#mac-os-x) of the BEDOPS␣
↪Installation document.
```

d. Attach per-platform binaries to this release by dragging each of them into the field underneath the description text. It can take a few moments for the web browser to upload each binary into the release page, so be patient. There should be at least two binary packages: one for Linux 64-bit, and one for Mac OS X.

e. Click the *Publish Release* button.

3. After at least 5-10 minutes from pushing the development branch to the master branch, check the BEDOPS documentation site to ensure that the "latest" or default documenation shown is for the new version.

   If not, take a look at the build page to manually trigger document rebuilds, or examine error logs, if necessary.

4. Update the Github bedops/bedops master README.md file to note the current version number, if necessary.

5. Push fixes to any documentation errors in the master branch.

Note: We should aim to fix typos and other errors as soon after a new release as possible, because then shortly afterwards we can simply pull a new development branch off the current state of the master branch with minimal commit losses.

Tip: If we push any subsequent changes to the `master` branch, it's not the end of the world. However, it is recommended that the version tag is pushed forwards to the latest commit:

```
$ git tag -f -a vX.Y.Z -m 'pushed current version tag forwards to latest commit'
...
$ git push -f --tags
...
```

This way, anyone who downloads source via GitHub will get the "freshest" code, with all the typo fixes and so forth.

6. Visit the BEDOPS documentation administration site to disable documentation for the development branch.

   Specifically, click on the versions tab to deactivate the old development branch. (Likewise, when adding a new development branch, add an active link here, so that edits to the documentation folder in the new development branch are available.)

7. Update a local fork of homebrew-science with details for the BEDOPS formula. Submit pull request to homebrew-science folks.

   a. After establishing a local fork, add the upstream remote so that you can fetch/pull updated formulas from Homebrew (if this is already done, this step can be skipped):

```
$ git remote add upstream git://github.com/homebrew/homebrew-science.git
```

   b. Fetch and pull data to the master branch from the upstream remote:

```
$ git checkout master
$ git fetch
$ git pull upstream master
...
```

   c. Make a branch of the master entitled *bedops-vXpYpZ* and check it out:

```
$ git branch bedops-vXpYpZ
$ git checkout bedops-vXpYpZ
```

   d. Edit changes to *bedops.rb* formula. Change the version number in the tarball download and remove the `sha1` line (you'll replace this later on).

   e. Test the new formula. Add the `--build-from-source` option to skip the per-platform bottle code:

```
$ brew install ./bedops.rb --build-from-source
```

   f. If the installation is successful, there will be a SHA1 validation code that you can copy and paste into the formula with the `sha1` header (see step *d*—basically, you are updating the line you removed in that step).

   g. Add, commit and push the updated formula to the *bedops-vXpYpZ* branch:

```
$ git add bedops.rb
$ git commit -am 'BEDOPS X.Y.Z'
$ git push origin bedops-vXpYpZ
```

   h. Visit the homebrew-science site and initiate a pull request from your local fork's newly pushed branch (there will be a big green button at the top of the GitHub site that asks you to start this pull request).

   i. Wait for success or failure; the homebrew-science people will indicate if there are any problems, usually within 48-72 hours.

8. Consider closing out or deleting the development branch, as well as setting up the next development branch.

2.8.3 Celebrate

At this point, we can email links to Linux packages to IT for updating the cluster BEDOPS module and make announcements on websites, mailing lists, etc.

2.9 Placeholder

OS X

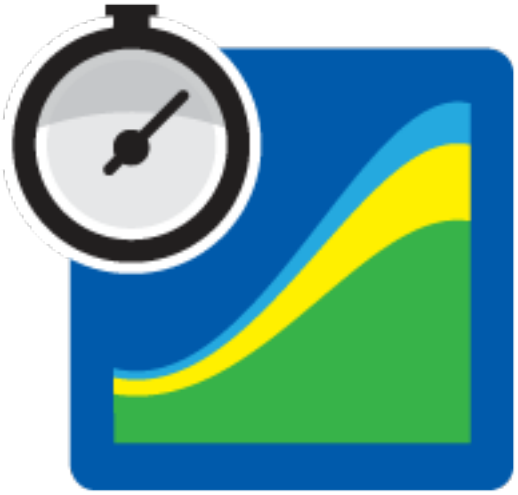
SOURCE

REFERENCE

SET OPERATIONS

STATISTICS

FILE MANAGEMENT



PERFORMANCE

OTHER

SUPPORT RESOURCES