Scalable analyses for big scRNA-seq data with Bioconductor

Aaron T. L. Lun[1]

[1]Cancer Research UK Cambridge Institute, Li Ka Shing Centre, Robinson Way, Cambridge CB2 0RE, United Kingdom

2019-01-09

1 Overview

Advances in single-cell RNA sequencing (scRNA-seq) technologies have increased the number of cells that can be assayed in routine experiments. For effective data analysis, the computational methods need to scale with the increasing size of scRNA-seq data sets. Scalability requires greater use of parallelization, out-of-memory representations and fast approximate algorithms to process data efficiently. Fortunately, this is easy to achieve within the Bioconductor ecosystem. This workflow will discuss how to tune the previous analysis pipelines for greater speed to handle large scRNA-seq data sets.

2 Out of memory representations

As we have previously discussed, the count matrix is the central structure around which our analyses are based. In the previous workflows, this has been held fully in memory as a dense `matrix` or as a sparse `dgCMatrix` . Howevever, in-memory representations may not be feasible for very large data sets, especially on machines with limited memory. For example, the 1.3 million brain cell data set from 10X Genomics (Zheng et al. 2017) would require over 100 GB of RAM to hold as a `matrix` and around 30 GB as a `dgCMatrix`. This makes it challenging to investigate the data on anything less than a high-performance computing system.

The obvious solution is to use a file-backed matrix representation where the data are held on disk and subsets are retrieved into memory as requested. While a number of implementations of file-backed matrices are available (e.g., *bigmemory (https://CRAN.R-project.org/package=bigmemory)*, *matter (https://bioconductor.org/packages/3.8/matter)*), we will be using the implementation from the *HDF5Array (https://bioconductor.org/packages/3.8/HDF5Array)* package. This uses the popular HDF5 format as the underlying data store, which provides a measure of standardization and portability across systems. We demonstrate with a subset of 20,000 cells from the 1.3 million brain cell data set, as provided by the *TENxBrainData (https://bioconductor.org/packages/3.8/TENxBrainData)* package[1].

[1] We could instead obtain the full-sized data set by using `TENxBrainData()`, but we will use the smaller data set here for demonstration purposes.

```
library(TENxBrainData)
sce <- TENxBrainData20k() # downloads once and caches it for future use.
sce
```

```
## class: SingleCellExperiment
## dim: 27998 20000
## metadata(0):
## assays(1): counts
## rownames: NULL
## rowData names(2): Ensembl Symbol
## colnames: NULL
## colData names(4): Barcode Sequence Library Mouse
## reducedDimNames(0):
## spikeNames(0):
```

Examination of the `SingleCellExperiment` object indicates that the count matrix is a `HDF5Matrix`. From a comparison of the memory usage, it is clear that this matrix object is simply a stub that points to the much larger HDF5 file that actually contains the data. This avoids the need for large RAM availability during analyses.

```
counts(sce)
```

```
## <27998 x 20000> HDF5Matrix object of type "integer":
##               [,1]     [,2]     [,3]     [,4] ... [,19997]
[,19998] [,19999]
##     [1,]        0        0        0        0   .        0
0        0
##     [2,]        0        0        0        0   .        0
0        0
##     [3,]        0        0        0        0   .        0
0        0
##     [4,]        0        0        0        0   .        0
0        0
##     [5,]        0        0        0        0   .        0
0        0
##      ...        .        .        .        .   .        .
.        .
## [27994,]        0        0        0        0   .        0
0        0
## [27995,]        0        0        0        1   .        0
2        0
## [27996,]        0        0        0        0   .        0
1        0
## [27997,]        0        0        0        0   .        0
0        0
## [27998,]        0        0        0        0   .        0
0        0
##          [,20000]
##     [1,]        0
##     [2,]        0
##     [3,]        0
##     [4,]        0
##     [5,]        0
##      ...        .
## [27994,]        0
## [27995,]        0
## [27996,]        0
## [27997,]        0
## [27998,]        0
```

```
object.size(counts(sce))
```

```
## [1] 2144 bytes
```

```
file.info(path(counts(sce)))$size
```

```
## [1] 76264332
```

Manipulation of the count matrix will generally result in the creation of a `DelayedArray` (from the *DelayedArray (https://bioconductor.org/packages/3.8/DelayedArray)* package). This stores delayed operations in the matrix object, to be executed when the modified matrix values are realized for use in calculations. The use of delayed operations avoids the need to write the modified values to a new file at every operation, which would unnecessarily require time-consuming disk I/O.

```
tmp <- counts(sce)
tmp <- log2(tmp + 1)
tmp
```

```
## <27998 x 20000> DelayedMatrix object of type "double":
##              [,1]     [,2]     [,3] ... [,19999] [,20000]
##     [1,]        0        0        0   .        0        0
##     [2,]        0        0        0   .        0        0
##     [3,]        0        0        0   .        0        0
##     [4,]        0        0        0   .        0        0
##     [5,]        0        0        0   .        0        0
##      ...        .        .        .   .        .        .
## [27994,]        0        0        0   .        0        0
## [27995,]        0        0        0   .        0        0
## [27996,]        0        0        0   .        0        0
## [27997,]        0        0        0   .        0        0
## [27998,]        0        0        0   .        0        0
```

Many functions described in the previous workflows are capable of accepting `HDF5Matrix` objects[2]. This is powered by the availability of common methods for all matrix representations (e.g., subsetting, combining, methods from *DelayedMatrixStats (https://bioconductor.org/packages/3.8/DelayedMatrixStats)*) as well as representation-agnostic C++ code using *beachmat (https://bioconductor.org/packages/3.8/beachmat)* (Lun, Pages, and Smith 2018). For example, we compute quality control (QC) metrics below with the same `calculateQCMetrics()` function that we used in the other workflows.

[2] If you find one that is not, please contact the maintainers.

```
library(scater)
sce <- calculateQCMetrics(sce, compact=TRUE) # compacting fo
r clean output.
sce$scater_qc
```

```
## DataFrame with 20000 rows and 2 columns
##       is_cell_control                            all
##             <logical>                    <DataFrame>
## 1               FALSE 1546:3.18949031369937:3060:...
## 2               FALSE  1694:3.2291697025391:3500:...
## 3               FALSE 1613:3.20790353038605:3092:...
## 4               FALSE 2050:3.31196566036837:4420:...
## 5               FALSE 1813:3.25863728272408:3771:...
## ...               ...                            ...
## 19996           FALSE 2050:3.31196566036837:4431:...
## 19997           FALSE 2704:3.43216726944259:6988:...
## 19998           FALSE 2988:3.47552591503928:8749:...
## 19999           FALSE 1711:3.23350376034113:3842:...
## 20000           FALSE  945:2.97589113640179:1775:...
```

Needless to say, data access from file-backed representations is slower than that from in-memory representations (assuming the latter is not moved into swap space). The time spent retrieving data from disk is an unavoidable cost of memory efficiency.

Comments from Aaron:

- By default, file locking is necessary for reading from HDF5 files via the *Rhdf5lib (https://bioconductor.org/packages/3.8/Rhdf5lib)* library, but this may be disabled on some file systems. Users can set the `HDF5_USE_FILE_LOCKING` environment variable to `FALSE` to avoid this requirement.

3 Parallelization

In many Bioconductor packages, different parallelization mechanisms are easily tested through the *BiocParallel (https://bioconductor.org/packages/3.8/BiocParallel)* framework. We construct a `BiocParallelParam` object that specifies the type of parallelization that we wish to use. For example, we might use forking[3] across 2 cores:

[3] Not available on Windows.

```
bpp <- MulticoreParam(2)
bpp
```

```
## class: MulticoreParam
##   bpisup: FALSE; bpnworkers: 2; bptasks: 0; bpjobname: BP
JOB
##   bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
##   bptimeout: 2592000; bpprogressbar: FALSE; bpexportgloba
ls: TRUE
##   bpRNGseed:
##   bplogdir: NA
##   bpresultdir: NA
##   cluster type: FORK
```

Another approach would be to distribute jobs across a network of computers:

```
bpp <- SnowParam(5)
bpp
```

```
## class: SnowParam
##   bpisup: FALSE; bpnworkers: 5; bptasks: 0; bpjobname: BP
JOB
##   bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
##   bptimeout: 2592000; bpprogressbar: FALSE; bpexportgloba
ls: TRUE
##   bpRNGseed:
##   bplogdir: NA
##   bpresultdir: NA
##   cluster type: SOCK
```

High-performance computing systems typically use job schedulers across a cluster of compute nodes. We can distribute jobs via the scheduler using the `BatchtoolsParam` class. The example below assumes a SLURM cluster, though the settings can be easily[4] configured for a particular system (see here (https://bioconductor.org/packages/3.8/BiocParallel/vignettes/BiocParallel_BatchtoolsParam.pdf) for details).

[4] In general. Some fiddling may be required, depending on the idiosyncrasies of the cluster set-up.

```
bpp <- BatchtoolsParam(10, cluster="slurm",
    resources=list(walltime=20000, memory=8000, ncpus=1))
```

Once we have defined the parallelization mechanism, we can pass the `BiocParallelParam` object to the function that we wish to run. This will instruct the function to run operations in parallel where it is allowed to (as defined by the developer). Different functions may parallelize operations across cells, or genes, or batches of data, depending on what is most appropriate. In the example below, we parallelize the QC calculations (across cells) using two cores:

```
alt <- calculateQCMetrics(sce, BPPARAM=MulticoreParam(2), co
mpact=TRUE)
```

This yields the same result as the single-core calculation, but faster.

```
all.equal(alt, sce)
```

```
## [1] TRUE
```

Comments from Aaron:

- Efficiently combining parallelization with file-backed matrix representations is likely to require systems that support parallel I/O.

4 Approximate nearest neighbours

searching

Identification of neighbouring cells in PC or expression space is a common procedure that is used in many functions, e.g., `buildSNNGraph()`, `doubletCells()`. The default is to favour accuracy over speed by using an exact nearest neighbour search, implemented with the k-means for k-nearest neighbours algorithm (Wang 2012). However, for large data sets, it may be preferable to use a faster approximate approach. The *BiocNeighbors (https://bioconductor.org/packages/3.8/BiocNeighbors)* framework makes it easy to switch between search options.

To demonstrate, we will use the PBMC data from the previous workflow (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/work-3-tenx.html):

```
sce.pbmc <- readRDS("pbmc_data.rds")
```

We had previously (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/work-3-tenx.html#clustering-with-graph-based-methods) generated a shared nearest neighbor graph with an exact neighbour search. We repeat this below using an approximate search, implemented using the Annoy (https://github.com/spotify/Annoy) algorithm. This involves constructing a `BiocNeighborParam` object to specify the search algorithm, and passing it to the `buildSNNGraph()` function.

```
library(scran)
library(BiocNeighbors)
snn.gr <- buildSNNGraph(sce.pbmc, BNPARAM=AnnoyParam(), use.
dimred="PCA")
```

The results from the exact and approximate searches are consistent with most clusters from the former re-appearing in the latter. This suggests that the inaccuracy from the approximation can be largely ignored. However, if the approximation was unacceptable, it would be simple to switch back to an exact algorithm by altering `BNPARAM`.

```
clusters <- igraph::cluster_walktrap(snn.gr)
table(Exact=sce.pbmc$Cluster, Approx=clusters$membership)
```

```
##      Approx
## Exact   1   2   3   4   5   6   7   8   9  10  11  12  13
##    1    0 839   0   1   0   2   0   7   1   0   0   0   0
##    2    0   0 586   5   0   0   0   0   0   0   0   0   0
##    3    0   0   3 541   0   0   0   0   0   0   0   0   0
##    4    0   0   0   0   0 200   0   0   0   0   0   0   0
##    5    0   0   0   0 534   0   0   0   0   0   0   0   0
##    6    0   0   0   0   0   0   8   0   0   0   0  38   0
##    7    0   0   0   0   0   0   0 120   0   0   0   0   0
##    8    0   0   0   0   0   0   0   0  46   0   0   0   0
##    9    0   0   8   0   0   0 790   0   0   0   0   0   0
##    10  39   0   1   0   0   0   0   0   0   0   0   0   0
##    11   0   0   0   0   0   0   1   0   0 135   0   0   0
##    12   0   0   0   0   0   6   0   0   0   0   0   0  23
##    13  17   0   0   0   0   0   0   0   0   0   0   0   0
##    14   0   0   0   0   0   0   0   0   0   0  86   0   0
```

Comments from Aaron:

- The neighbour search algorithms are interoperable with *BiocParallel (https://bioconductor.org/packages/3.8/BiocParallel)*, so it is straightforward to parallelize the search for greater speed.

5 Concluding remarks

All software packages used in this workflow are publicly available from the Comprehensive R Archive Network (https://cran.r-project.org (https://cran.r-project.org)) or the Bioconductor project (http://bioconductor.org (http://bioconductor.org)). The specific version numbers of the packages used are shown below, along with the version of the R installation.

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices uti
ls     datasets
## [8] methods   base
##
## other attached packages:
##  [1] BiocNeighbors_1.0.0
##  [2] TENxBrainData_1.2.0
##  [3] HDF5Array_1.10.1
##  [4] rhdf5_2.26.2
##  [5] readxl_1.2.0
##  [6] gdata_2.18.0
##  [7] R.utils_2.7.0
##  [8] R.oo_1.22.0
##  [9] R.methodsS3_1.7.1
## [10] scRNAseq_1.8.0
## [11] edgeR_3.24.3
## [12] Matrix_1.2-15
## [13] org.Hs.eg.db_3.7.0
## [14] EnsDb.Hsapiens.v86_2.99.0
## [15] ensembldb_2.6.3
## [16] AnnotationFilter_1.6.0
## [17] DropletUtils_1.2.2
## [18] pheatmap_1.0.12
## [19] cluster_2.0.7-1
## [20] dynamicTreeCut_1.63-1
## [21] limma_3.38.3
## [22] scran_1.10.2
## [23] scater_1.10.1
## [24] ggplot2_3.1.0
## [25] TxDb.Mmusculus.UCSC.mm10.ensGene_3.4.0
## [26] GenomicFeatures_1.34.1
## [27] org.Mm.eg.db_3.7.0
## [28] AnnotationDbi_1.44.0
## [29] SingleCellExperiment_1.4.1
## [30] SummarizedExperiment_1.12.0
## [31] DelayedArray_0.8.0
## [32] BiocParallel_1.16.5
## [33] matrixStats_0.54.0
## [34] Biobase_2.42.0
## [35] GenomicRanges_1.34.0
## [36] GenomeInfoDb_1.18.1
## [37] IRanges_2.16.0
## [38] S4Vectors_0.20.1
## [39] BiocGenerics_0.28.0
## [40] bindrcpp_0.2.2
## [41] BiocFileCache_1.6.0
## [42] dbplyr_1.2.2
## [43] knitr_1.21
## [44] BiocStyle_2.10.0
##
## loaded via a namespace (and not attached):
##   [1] tidyselect_0.2.5             RSQLite_2.1.1
##   [3] grid_3.5.2                   trimcluster_0.1-2.1
##   [5] Rtsne_0.15                   munsell_0.5.0
##   [7] destiny_2.12.0               statmod_1.4.30
##   [9] sROC_0.1-2                   withr_2.1.2
##  [11] colorspace_1.3-2             highr_0.7
##  [13] robustbase_0.93-3            vcd_1.4-4
##  [15] VIM_4.7.0                    TTR_0.23-4
##  [17] labeling_0.3                 GenomeInfoDbData_1.2.
0
##  [19] cvTools_0.3.2                bit64_0.9-7
##  [21] xfun_0.4                     ggthemes_4.0.1
##  [23] diptest_0.75-7               R6_2.3.0
##  [25] ggbeeswarm_0.6.0             robCompositions_2.0.9
##  [27] RcppEigen_0.3.3.5.0          locfit_1.5-9.1
##  [29] mvoutlier_2.0.9              flexmix_2.3-14
##  [31] bitops_1.0-6                 reshape_0.8.8
##  [33] assertthat_0.2.0             promises_1.0.1
##  [35] scales_1.0.0                 nnet_7.3-12
##  [37] beeswarm_0.2.3               gtable_0.2.0
##  [39] rlang_0.3.1                  scatterplot3d_0.3-41
##  [41] splines_3.5.2                rtracklayer_1.42.1
##  [43] lazyeval_0.2.1               BiocManager_1.30.4
##  [45] yaml_2.2.0                   reshape2_1.4.3
##  [47] abind_1.4-5                  httpuv_1.4.5.1
##  [49] tools_3.5.2                  bookdown_0.9
##  [51] zCompositions_1.1.2          RColorBrewer_1.1-2
##  [53] proxy_0.4-22                 Rcpp_1.0.0
##  [55] plyr_1.8.4                   progress_1.2.0
##  [57] zlibbioc_1.28.0              purrr_0.2.5
##  [59] RCurl_1.95-4.11              prettyunits_1.0.2
##  [61] viridis_0.5.1                cowplot_0.9.4
##  [63] zoo_1.8-4                    haven_2.0.0
##  [65] magrittr_1.5                 data.table_1.11.8
##  [67] openxlsx_4.1.0               lmtest_0.9-36
##  [69] truncnorm_1.0-8              mvtnorm_1.0-8
##  [71] ProtGenerics_1.14.0          xtable_1.8-3
##  [73] mime_0.6                     hms_0.4.2
##  [75] evaluate_0.12                smoother_1.1
##  [77] XML_3.98-1.16                rio_0.5.16
##  [79] mclust_5.4.2                 gridExtra_2.3
##  [81] compiler_3.5.2                biomaRt_2.38.0
##  [83] tibble_2.0.0                  KernSmooth_2.23-15
##  [85] crayon_1.3.4                  htmltools_0.3.6
##  [87] later_0.7.5                   pcaPP_1.9-73
##  [89] rrcov_1.4-7                   DBI_1.0.0
##  [91] ExperimentHub_1.8.0           MASS_7.3-51.1
##  [93] fpc_2.1-11.1                  rappdirs_0.3.1
##  [95] boot_1.3-20                   car_3.0-2
##  [97] sgeostat_1.0-27               bindr_0.1.1
##  [99] igraph_1.2.2                  forcats_0.3.0
## [101] pkgconfig_2.0.2               GenomicAlignments_1.1
8.1
## [103] foreign_0.8-71                laeken_0.4.6
## [105] sp_1.3-1                      vipor_0.4.5
## [107] XVector_0.22.0                NADA_1.6-1
## [109] stringr_1.3.1                 digest_0.6.18
## [111] pls_2.7-0                     Biostrings_2.50.2
## [113] rmarkdown_1.11                cellranger_1.1.0
## [115] DelayedMatrixStats_1.4.0      curl_3.2
## [117] kernlab_0.9-27                shiny_1.2.0
## [119] gtools_3.8.1                  Rsamtools_1.34.0
## [121] modeltools_0.2-22             Rhdf5lib_1.4.2
## [123] carData_3.0-2                 viridisLite_0.3.0
## [125] pillar_1.3.1                  lattice_0.20-38
## [127] GGally_1.4.0                  httr_1.4.0
## [129] DEoptimR_1.0-8                survival_2.43-3
## [131] interactiveDisplayBase_1.20.0 xts_0.11-2
## [133] glue_1.3.0                    zip_1.0.0
## [135] prabclus_2.2-6                bit_1.1-14
## [137] class_7.3-15                  stringi_1.2.4
## [139] blob_1.1.1                    AnnotationHub_2.14.2
## [141] memoise_1.1.0                 dplyr_0.7.8
## [143] irlba_2.3.2                   e1071_1.7-0
```

References

Lun, A. T. L., H. Pages, and M. L. Smith. 2018. “beachmat: A Bioconductor C++ API for accessing high-throughput biological data from a variety of R matrix types.” *PLoS Comput. Biol.* 14 (5):e1006135.

Wang, X. 2012. “A Fast Exact K-Nearest Neighbors Algorithm for High Dimensional Search Using K-Means Clustering and Triangle Inequality.” *Proc Int Jt Conf Neural Netw* 43 (6):2351–8.

Zheng, G. X., J. M. Terry, P. Belgrader, P. Ryvkin, Z. W. Bent, R. Wilson, S. B. Ziraldo, et al. 2017. “Massively parallel digital transcriptional profiling of single cells.” *Nat Commun* 8 (January):14049.