Challenges and Pragmatic Solutions to Statistical Analysis of High-throughput Genomic Data

Martin Morgan (mtmorgan@fhcrc.org)
Fred Hutchinson Cancer Research Center

January 4-5 2013

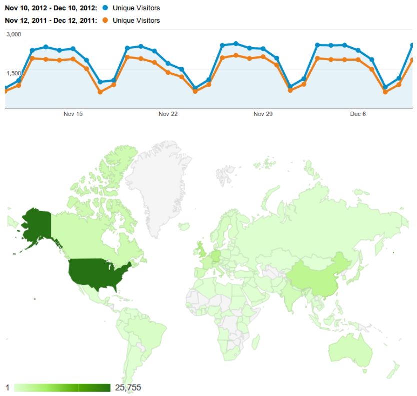
Abstract

The *R* / *Bioconductor* project[1] provides a proving ground for computational approaches to handling high-volume genomic data. Many investigators have primary interests and talent in domains other than computer science. Their research questions raise transient analytic needs that make it difficult to justify narrowly-focused investment in sophisticated computational methods or machinery. Very diverse computational environments make many solutions idiosyncratic. This leads us toward development of reusable infrastructure to support simple and standardized models of high-throughput computation, relying on opportunistic community standards, and offering consistently-configured computational environments for scalable evaluation.

[1] `http://bioconductor.org`

R / Bioconductor

- 610 packages for analysis and comprehension of high-throughput genomic data
- Statisticians, biologists, bioinformaticians in US, Europe, Asia, . . . ; mid-sized labs & researchers in academia, government, pharma
- Developed by advanced users, domain experts, core group



Google analytics, 1-month access, 10 December 2012

R / Bioconductor

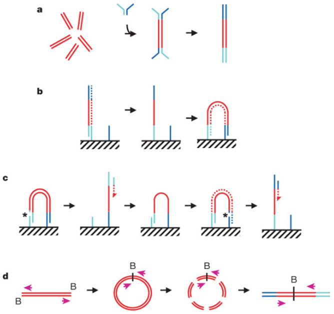
- 610 packages for analysis and comprehension of high-throughput genomic data
- Statisticians, biologists, bioinformaticians in US, Europe, Asia, . . . ; mid-sized labs & researchers in academia, government, pharma
- Developed by advanced users, domain experts, core group

Maintainers	Packages
340	1
68	2
16	3
9	4
3	5
5	6
1	7
1	20
1	44

R / Bioconductor

High-throughput genomic data

- Sequences: very large data summarized or filtered to modest size for advanced statistical analysis, e.g., *edgeR*, *DESeq*, *VariantTools*
- Variants: statistical association with phenotype; e.g., millions of SNPs $\times$ thousands of individuals, SNPs perhaps in combination, e.g., *snpStats*, *MatrixEQTL*
- Arrays: whole-genome scans with locally complex structure, e.g., *bumphunter*



Bentley et al., Nature 2008 456(7218):53-9

R / Bioconductor

High-throughput genomic data

- Sequences: very large data summarized or filtered to modest size for advanced statistical analysis, e.g., *edgeR*, *DESeq*, *VariantTools*
- Variants: statistical association with phenotype; e.g., millions of SNPs $\times$ thousands of individuals, SNPs perhaps in combination, e.g., *snpStats*, *MatrixEQTL*
- Arrays: whole-genome scans with locally complex structure, e.g., *bumphunter*

Differential expression

1. Align: third-party $\Rightarrow$ BAM files
2. Count: 'annotation', *GenomicRanges* `findOverlaps`; data reduction
3. Test: microarray-like

   $$\log \mu_{gi} = \mathbf{x}_i^T \beta_g + \log N_i$$

   Neg. binomial GLM

   Shared info. across experiment

R / Bioconductor

High-throughput genomic data

- Sequences: very large data summarized or filtered to modest size for advanced statistical analysis, e.g., *edgeR*, *DESeq*, *VariantTools*
- Variants: statistical association with phenotype; e.g., millions of SNPs $\times$ thousands of individuals, SNPs perhaps in combination, e.g., *snpStats*, *MatrixEQTL*
- Arrays: whole-genome scans with locally complex structure, e.g., *bumphunter*

Method	500k SNPs
Plink	583.3 days
Merlin	20.0 days
R/qtl	4.7 days
snpMatrix	5.1 days
Matrix eQTL	19.4 mins

Anecdote (old)

- `glm`, 100's / minute
- `glm.fit` & tricks, 1000's / minute
- Cluster: 500,000 / minute

R / Bioconductor

High-throughput genomic data

- Sequences: very large data summarized or filtered to modest size for advanced statistical analysis, e.g., *edgeR*, *DESeq*, *VariantTools*
- Variants: statistical association with phenotype; e.g., millions of SNPs $\times$ thousands of individuals, SNPs perhaps in combination, e.g., *snpStats*, *MatrixEQTL*
- Arrays: whole-genome scans with locally complex structure, e.g., *bumphunter*

Bump-hunting

$$Y_{ij} = \beta_0(l_j) + \beta_1(l_j)X_j + \varepsilon_{ij}$$

Subject i, location l_j, covariate X_j; baseline function $\beta_0(l_j)$, parameter of interest $\beta_1(l_j)$

Shared info. between nearby locations

Pragmatic approaches to big data

What is needed for big data analysis?

- Efficient, robust code
- Memory management
- Parallel evaluation
- Algorithms

Efficient, robust code

Experienced *R* programmers...

- Vectors, vs. element-wise iteration
  - `for` tempts users
- Pre-allocate & fill, vs. copy & append
  - `lapply` guides users
- Selective input
- Surprising gotchas, e.g., `unlist use.names=TRUE`, vs. `use.names=FALSE`
- Specialized packages & functions

Anecdotal (*Bioconductor* submission, R-help, StackOverflow[2], ...): a common shortcoming

[2] `http://stackoverflow.com`

Efficient, robust code

A common approach

- C code – directly or via add-ons like *Rcpp*

Robust

- Developers seem to want their code to work

	Used by
`src` directory	232
Rcpp	10
RUnit	78
testthat	7

Memory management

	Used by
SQL	43
ff, *bigmemory*	11
ncdf	3

- SQL used (appropriately) for relational data
- *R*-specific solutions require dedicated development without data re-use in other applications
- NetCDF (a standard) not widely used
  - 3rd party dependency
  - Experience of developers
  - Limitations of *R* interface

Memory management

	Used by
SQL	43
ff , *bigmemory*	11
ncdf	3

- Large vectors probably do not play well with using multiple cores (though what is large?)
- Instead: data slices, iteration, on-line algorithms; data containers, e.g., *IRanges*::*Rle*-class

Parallel evalution

	Used by
parallel	26
snow & c.	20
foreach & c.	11
rlecuyer, *setRNG*	2
Rmpi	1

- Strong adoption of base *R* packages (*parallel*)
  - Random numbers rarely handled properly
- MPI (a standard) not widely used
  - 3rd party dependency
  - Robust to user deployments
  - Error recovery
  - . . .

Algorithms

Used

- Manager / worker
- `lapply`-like
- Interactive

Ad hoc user interactions

Available

- `pvec`
- *snow*: subsets, `sendData` / `recvData` / ...
- *Rmpi*: rich MPI formulations
- Single instruction, multiple data (e.g., pbdR) and other models

Pragmatic *Bioconductor* solutions

- Data structures
- Standard packaging
- Iteration
- The cloud

Data structures

Use *de facto* standard data formats

- e.g., BAM, VCF files

Exploit column-oriented access

- e.g., *GRanges*-class: a single S4 instance
- More subtley: *IRangesList*-class a single S4 instance with partitioning
- Key operations, e.g., `findOverlaps`, efficiently implemented

Exploit sparsity

- e.g., *Rle*-class: effectively compress whole-genome 'coverage'
- Supports rich set of operations

Data structures

```
> gr
GRanges with 10 ranges and 2 metadata columns:
    seqnames    ranges strand |     score        GC
       <Rle> <IRanges>  <Rle> | <integer> <numeric>
  a     chr1  [ 1, 10]      - |         1         1
  b     chr2  [ 2, 10]      + |         2      0.88
  c     chr2  [ 3, 10]      + |         3      0.77
  d     chr2  [ 4, 10]      * |         4      0.66
  e     chr1  [ 5, 10]      * |         5      0.55
  f     chr1  [ 6, 10]      + |         6      0.44
  g     chr3  [ 7, 10]      + |         7      0.33
  ...
  ---
  seqlengths:
   chr1 chr2 chr3
   1000 2000 1500
```

Standard packaging: *BiocParallel*

Register parameterized back ends

- Sensible performance defaults
- Easy to switch between parallel & serial evaluation
- Scheduling of nested parallelism (to come)

Common signatures

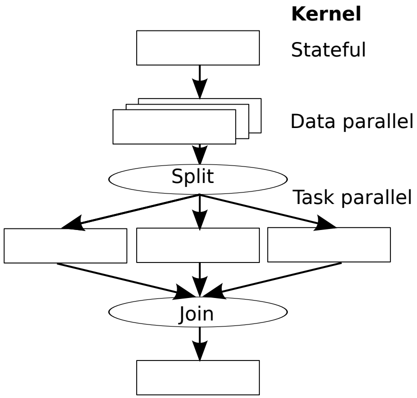
- `bplapply(X, FUN, ..., param)`, `bpvec(X, FUN, ..., param)`

Programming to contract, e.g., `bplapply`

- `X` must implement methods `length`, `[`, and `[[`
- Currently: `mclapply` requires `as.list`, which defeats the purpose of some high-volume containers

Iteration: *Streamer*

- *Producer* and *Consumer* kernels, assembled into streams
- `yield` output from a single chunk
- Requires on-line and other algorithms
- Formalism offers chance for code transformation / compilation



Streamer

```
p <- Seq(to=50, yieldSize=5) # Producer: 1:50

param <- MulticoreParam(size=5)
team <- Team(function(x) {
    Sys.sleep(1); mean(x)
}, param=param)
s <- Stream(p, team)

system.time({
    while(length(y <- yield(s)))
        print(y)
})  ## about 2 seconds
```

Streamer

```
dteam <-
    DAGTeam(A=FunctionConsumer(function(y) y),
            B=FunctionConsumer(function(A) -A),
            C=FunctionConsumer(function(A) 1 / A),
            D=FunctionConsumer(function(B, C) B + C))

plot(dteam)

strm <- Stream(Seq(to=10), dteam)
sapply(strm, c)
# [1]  0.00 -1.50 -2.67 -3.75 -4.80
# [6] -5.83 -6.86 -7.88 -8.89 -9.90
```

The cloud

- *Bioconductor* AMI, configured with, e.g., MPI support
- Helps address heterogeneity of user systems / administration
- Integration with *Galaxy* as a more ‘user friendly’ tool
- Unclear how this fits into academic / business funding models

Recap

Bioconductor

- Well-used
- Talented but not CS developers

Approaches to big data require. . .

- Efficient code, memory management, parallel evaluation

Pragmatic solutions

- Data structures, standard packaging, iteration, cloud

Acknowledgements

- Vince Carey (Brigham & Womens, Harvard), Wolfgang Huber (EBI), Rafael Irizzary (JHU), Robert Gentleman (Genentech)
- Hervé Pagès, Marc Carlson, Dan Tenenbaum, Valerie Obenchain, Paul Shannon.
- Michael Lawrence, Sean Davis, Kasper Hansen, James MacDondald
- NIH U41 HG004059