Crambled Vignette

Andy Lynch

12 January 2016

Overview

In this vignette we illustrate the Crambled R package. The functionality of Crambled was described in Lynch A. "Crambled: A Shiny application to enable intuitive resolution of conflicting cellularity estimates" *F1000Research* 2015, **4**:1407.

The library can be installed via the following commands:

```
install.packages("devtools")
devtools::install_github("dralynch/crambled/Package")
```

and loaded in the usual manner:

```
library("Crambled",warn.conflicts = FALSE, quietly = TRUE)
```

We also load the dependencies `Rsamtools` and `png`

```
suppressMessages(library("Rsamtools",warn.conflicts = FALSE, quietly = TRUE))
suppressMessages(library("png",warn.conflicts = FALSE, quietly = TRUE))
suppressMessages(library("knitr",warn.conflicts = FALSE, quietly = TRUE))
```

Crambled can be thought of as dividing into three parts:

1. The Crambled Shiny application
2. Code to generate plots for the crambled shiny application
3. Supporting data objects (including example plots and this vignette)

The code for generating the plots takes BAM files as input. Since BAM files are large, we do not include any examples, but the usage is straightforward. In the use case of having a tumour and matched normal pair of BAM files, the basic command is:

```
CrambledScan(normal="normal.bam",tumour="tumour.bam",title="TITLE")
```

While if one is dealing with a tumour cell line BAM file with no matched normal information, the command is:

```
CrambledScanCellline(normal="CELL_LINE.bam",title="TITLE")
```

In both cases the `title` argument provides not only the title displayed the resulting plot, but also the name of the png file in which it is stored.

Understanding the `CrambledScan` function

By default, `CrambledScan` expects BAM files to be aligned to the hg19 genome. The locations in the BAM file in which it will search are defined by the `CrambledSuggestSNPsHG19()` function. If using a different version of the genome, then this needs to be changed. The `CrambledSuggestSNPsHG19()` function returns the locations of more than 175,000 common polymorphisms in a GRanges format.

```
CrambledSuggestSNPsHG19()
```

```
## GRanges object with 177299 ranges and 0 metadata columns:
##              seqnames              ranges strand
##                 <Rle>           <IRanges>  <Rle>
##    rs2977608     chr1     [768253, 768253]      *
##   rs11240779     chr1     [808631, 808631]      *
##    rs4970382     chr1     [840753, 840753]      *
##    rs6679046     chr1     [850371, 850371]      *
##   rs13303369     chr1     [852875, 852875]      *
##          ...      ...                  ...    ...
##   rs12628844    chr22 [51144168, 51144168]      *
##    rs9616947    chr22 [51151631, 51151631]      *
##    rs8136930    chr22 [51158499, 51158499]      *
##    rs5770996    chr22 [51164115, 51164115]      *
##    rs2283675    chr22 [51179000, 51179000]      *
##   -------
##   seqinfo: 54 sequences from an unspecified genome; no seqlengths
```

Crambled will interrogate each of these locations in the two BAM files and return for each location a list with a member called "seq" which will be an array of base counts looking something like

	A	C	G	T	N
Normal	12	15	0	1	0
Tumour	30	10	3	0	0

which will be evaluated with the `CrambledScanInfo` function.

If we simulate this

```
x<-NULL
tmp<-array(c(12,15,0,1,0,30,10,3,0,0),dim=c(5,2,1))
rownames(tmp)<-c("A","C","G","T","N")
x[["seq"]]<-tmp
CrambledScanInfo(x)
```

```
## $myDepth
## [1] 40
##
## $AF
## [1] 0.25
```

we see that the `CrambledScanInfo` function establishes that the normal sample is heterozygous at this hypothesized site and then returns the depth (10+30 = 40) and minor allele fraction (10/40 = 0.25). Note

that bases other than those deemed truly to be present in the normal sample are ignored.

Had only one base been present in the normal sample, i.e. if the situation were

	A	C	G	T	N
Normal	1	26	0	1	0
Tumour	30	10	3	0	0

then the function returns `NA`.

```
x<-NULL
tmp<-array(c(1,26,0,1,0,30,10,3,0,0),dim=c(5,2,1))
rownames(tmp)<-c("A","C","G","T","N")
x[["seq"]]<-tmp
CrambledScanInfo(x)
```

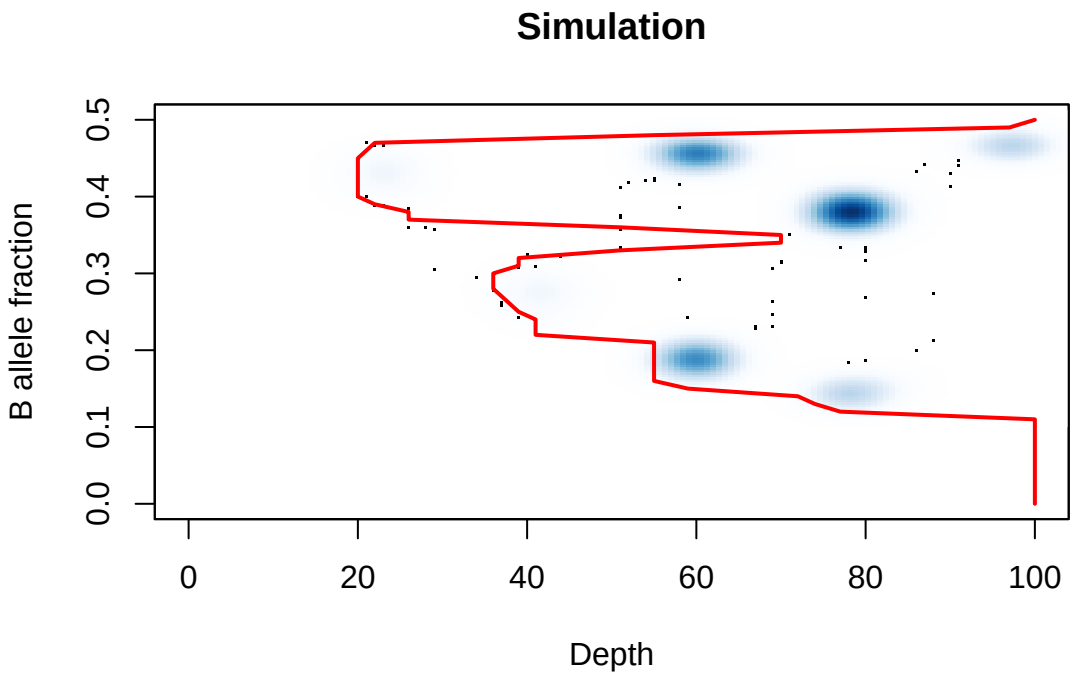
```
## $myDepth
## [1] NA
##
## $AF
## [1] NA
```

Having returned two vectors, one of depths and one of minor allele fractions, from locations that were heterozygous in the normal sample, a running average is applied to smooth out some of the variation due to low depths. These smoothed vectors are then sent to the `CrambledPlot` function to produce the plot that can by processed by the Crambled Shiny App.

To proceed, we simulate some representative vectors for a tumour with cellularity 0.62 where diploid regions are sequenced to a depth of 60.

```
SimCell<-0.62
SimSingleDepth<-30
SimDepths<-NULL
SimAFs<-NULL

numA<-rep(c(0,1,1,2,2,3,2),times=c(500,500,2000,2000,3000,1000,1000))
numB<-rep(c(0,0,1,0,1,0,2),c(500,500,2000,2000,3000,1000,1000))

DepthA<-(1+(numA-1)*SimCell)*SimSingleDepth
DepthB<-(1+(numB-1)*SimCell)*SimSingleDepth

SimA<-rpois(10000,DepthA)
SimB<-rpois(10000,DepthB)

SimDepths<-c(SimA+SimB)
SimAFs<-c(pmin(SimA,SimB)/(SimA+SimB))

SimDepths<-runmed(SimDepths,25)
SimAFs<-runmed(SimAFs,25)
```

and the command `CrambledPlot(depthvec=SimDepths,afvec=SimAFs,title="Simulation")` would produce a plot as follows:



We now write that file out so that it can be used later in the vignette.

```
simplotfile <- tempfile(fileext='.png')
png(simplotfile, width=600,height=400)
  smoothScatter(SimDepths,SimAFs,transformation = function(x){x^1},main="Simulation",xlab="Depth",ylab=
    afclass<-as.numeric(cut(SimAFs,seq(-0.005,0.505,0.01)))
    xbounds<-rep(NA,51)
    for(i in 1:51){
      xbounds[i]<-min(c(SimDepths[afclass==i],100))
    }
    lines(smooth(xbounds),seq(0,0.5,0.01),lwd=2,col="red")
dev.off()
```

```
## pdf
##   2
```

Understanding the `CrambledScanCellline` function

As with `CrambledScan`, `CrambledScanCellline` expects BAM files to be aligned to the hg19 genome, and locations to investigate are defined in the same manner.

```
CrambledSuggestSNPsHG19()
```

Crambled will interrogate each of these locations in the single BAM file and return for each location a list with a member called "seq" which will be an array of base counts looking something like

	A	C	G	T	N
Cellline	30	10	3	0	0

which will be evaluated with the `CrambledScanInfoCellline` function.

If we simulate this we see that the output is as before, except that this time there is no test for heterozygosity of the normal, so values are returned for all loci.

```
x<-NULL
tmp<-array(c(30,10,3,0,0),dim=c(5,1,1))
rownames(tmp)<-c("A","C","G","T","N")
x[["seq"]]<-tmp
CrambledScanInfoCellline(x)
```

```
## $myDepth
## [1] 40
##
## $AF
## [1] 0.25
```

Again, two vectors are returned, one of depths and one of minor allele fractions. However these are enriched for sites that were homozygous in the (unseen) germline. Because these will dominate the plot and, worse, will ruin the running averages, we separate out the sites showing no heterozygosity (both those that are germline homozygous and those that have suffered somatic loss of heterozygosity) from the clearly heterozygous sites.

To proceed, we simulate some representative vectors for a cell line with cellularity we presume to be 1 where diploid regions are sequenced to a depth of 60.

```
SimCell<-1
SimSingleDepth<-30
SimDepths<-NULL
SimAFs<-NULL

numA<-rep(c(1,1,2,2,3,2,2,3,4,2,3),times=c(1000,2000,3000,2000,3000,3000,3000,1000,3000,1000,3000))
numB<-rep(c(0,1,0,0,0,1,0,0,0,2,0),times=c(1000,2000,3000,2000,3000,3000,3000,1000,3000,1000,3000))

DepthA<-(1+(numA-1)*SimCell)*SimSingleDepth
DepthB<-(1+(numB-1)*SimCell)*SimSingleDepth

SimA<-rpois(25000,DepthA)
SimB<-rpois(25000,DepthB)

SimDepths<-c(SimA+SimB)
SimAFs<-c(pmin(SimA,SimB)/(SimA+SimB))
```

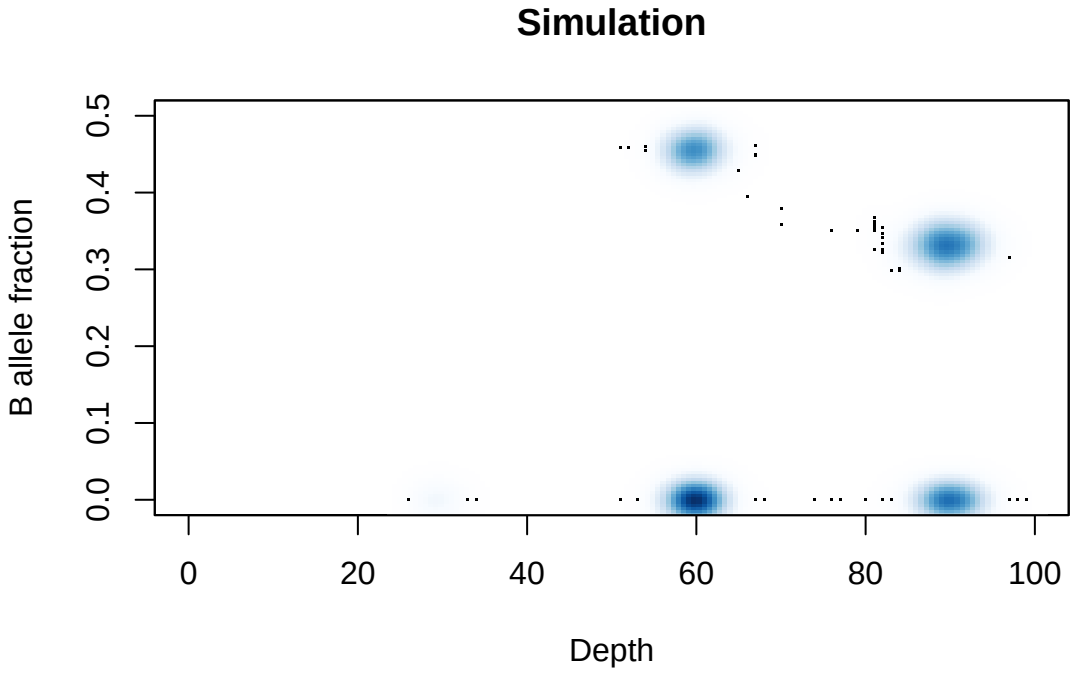
We now separate out values with an allele fraction of less than 0.1, and process the two sets of values separately

```
    SimDepths1<-runmed(SimDepths[which(SimAFs<0.1)],21)
    SimDepths2<-runmed(SimDepths[which(SimAFs>=0.1)],21)
    SimAFs1<-runmed(SimAFs[which(SimAFs<0.1)],21)
    SimAFs2<-runmed(SimAFs[which(SimAFs>=0.1)],21)
```

The potentially homozygous sites are then thinned

```
mypoints<-sample(length(SimDepths1),length(SimDepths2))
SimDepths<-c(SimDepths1[mypoints],SimDepths2)
SimAFs<-c(SimAFs1[mypoints],SimAFs2)
```

The vectors are then sent to the `CrambledPlot` function to produce the plot that can by processed by the Crambled Shiny App. The command `CrambledPlot(depthvec=SimDepths,afvec=SimAFs,title="Simulation")` would produce a plot as follows:



The Shiny App

The Shiny App is launched with the command `RunCrambledShinyApp`

```
RunCrambledShinyApp()
```

We now for a given input of depth and cellularity predict the locations of clonal copy number states in the depth vs. allele fraction plots.

```
    mycell<-0.62
    mydepth<-30

    Dmat<-matrix(ncol=8,nrow=8,NA)
    Amat<-matrix(ncol=8,nrow=8,NA)
    Bmat<-matrix(ncol=8,nrow=8,NA)

    for(i in 1:5){
      for(j in i:5){
        Dmat[i,j]<-2*mydepth*(1-mycell)+mycell*(i+j-2)*mydepth
        Amat[i,j]<-(1-mycell+mycell*(i-1))/( 2*(1-mycell)+(i+j-2)*mycell)

        mydens<-dbinom(0:round(Dmat[i,j]),round(Dmat[i,j]),Amat[i,j])
        myval<-(0:round(Dmat[i,j]))/round(Dmat[i,j])
        myval[myval>0.5]<-1-myval[myval>0.5]
```

```
        Bmat[i,j]<-sum(myval*mydens)

      }
    }

    ABlabels<-c("0",rep("",7),"A","AB",rep("",6),"AA","AAB","AABB",rep("",5),"AAA","AAAB","AAABB","AAAB
```

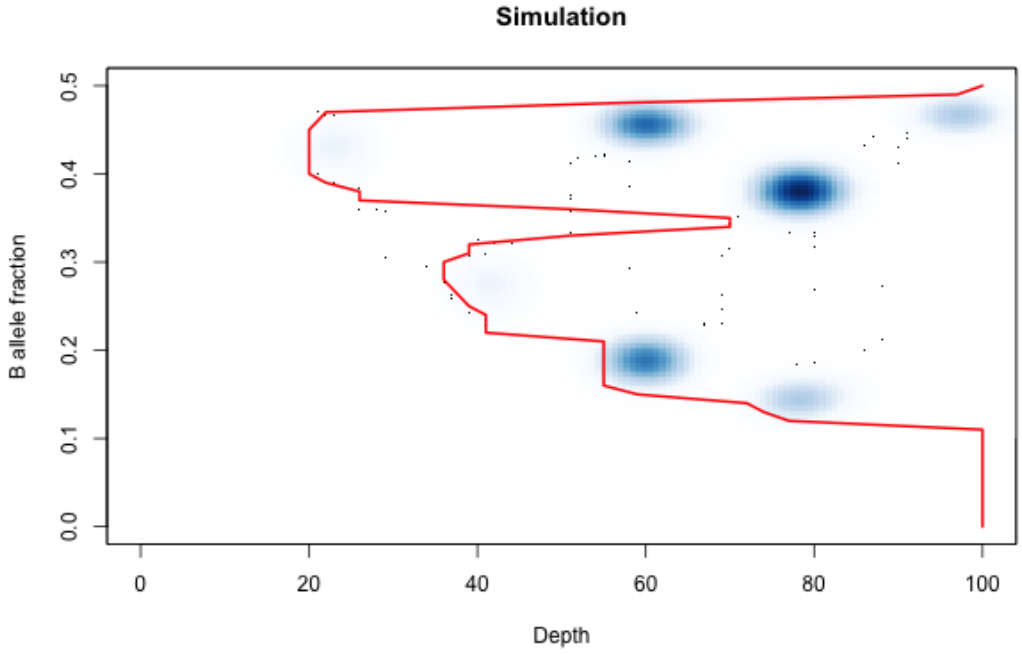
We generate a png file containing those values

```
    predfile <- tempfile(fileext='.png')
    png(predfile, width=600, height=400)
    plot(1,1,type="n",ylim=c(0,0.5),xlim=c(0,100),axes=F,xlab="",ylab="",main="")
    points(Dmat,Bmat,pch=21,col="black",bg="yellow")
    text(as.vector(Dmat)[1:37],as.vector(Bmat)[1:37],ABlabels,col="red",cex=0.75,pos=1)
dev.off()
```

```
## pdf
##   2
```

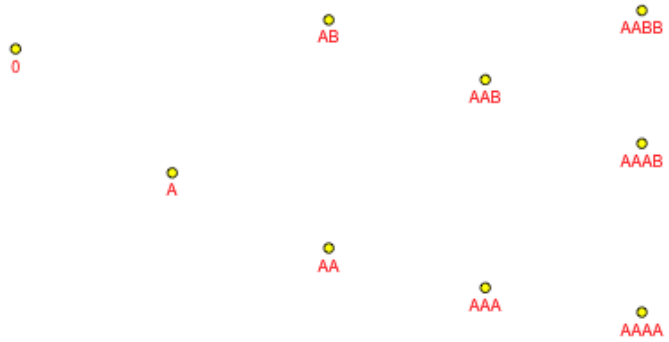
We now have two images that have been created. The first is from our simulated data:

```
underpng<-png::readPNG(simplotfile)
par(mar=c(0,0,0,0))
plot(1,1,type="n",ylim=c(0,0.5),xlim=c(0,100),axes=F,xlab="",ylab="",main="")
rasterImage(underpng,0,0,100,0.5)
```



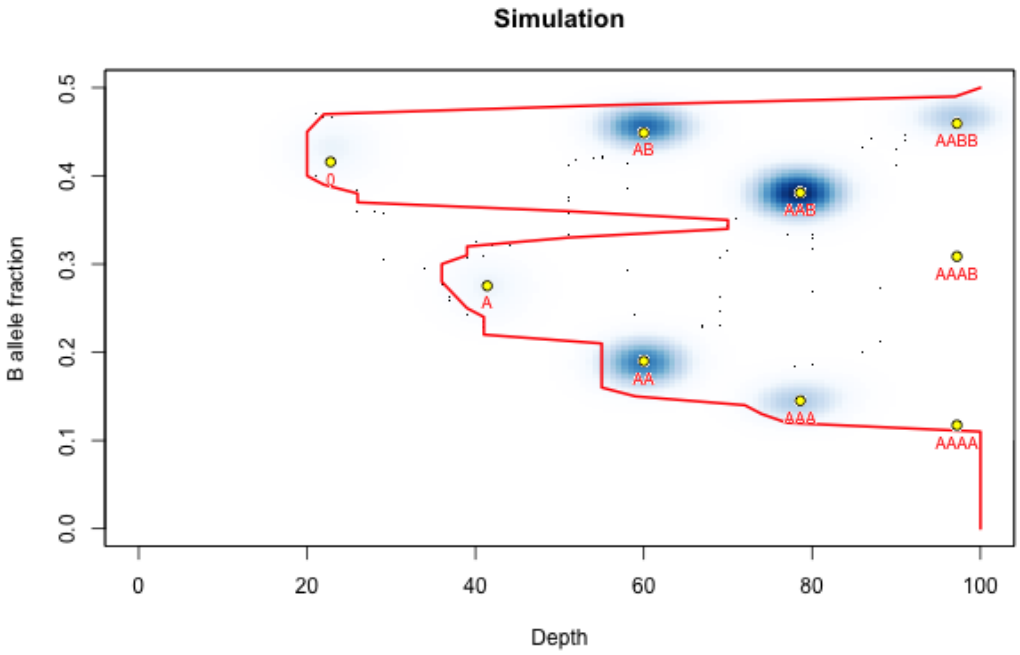
The second is from the predictions:

```
overpng<-png::readPNG(predfile)
par(mar=c(0,0,0,0))
plot(1,1,type="n",ylim=c(0,0.5),xlim=c(0,100),axes=F,xlab="",ylab="",main="")
rasterImage(overpng,0,0,100,0.5)
```



While the Crambled Shiny App combines the two images to allow comparison.

```
matte<-(overpng[,,1]+overpng[,,2]+overpng[,,3])<3
for(i in 1:3){
  tmpu<-underpng[,,i]
  tmpo<-overpng[,,i]
  tmpu[matte]<-tmpo[matte]
  underpng[,,i]<-tmpu
}
par(mar=c(0,0,0,0))
plot(1,1,type="n",ylim=c(0,0.5),xlim=c(0,100),axes=F,xlab="",ylab="",main="")
rasterImage(underpng,0,0,100,0.5)
```



Within the Crambled Shiny App, one can use sliders to vary the cellularity and depth parameters, and the display updates the predictions in the display.