

# CloneSeeker

Mark Zucker            Kevin R. Coombes

November, 2018

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Simulated Tumor Containing Multiple Clones</b>	<b>1</b>
2.1	Simulating Tumor Data . . . . .	3
<b>3</b>	<b>Seeking Clones</b>	<b>5</b>
3.1	Seeking Clones from Copy Number Data . . . . .	7
3.2	Sequencing Data . . . . .	8
3.3	Both Sequencing and SNP Array Data . . . . .	8

## 1 Introduction

Tumors often consist of multiple distinct subpopulations or clones. Information about the number of clones present in a tumor can be inferred using either mutation allele frequency data, from sequencing studies, or from copy number variants (CNVs), derived either from sequencing or from SNP array data. The **CloneSeeker** package can be applied to SNP array data, sequencing data, or both, from tumor cells from a cancer patient. **CloneSeeker** can determine the number of clones, the distribution of cells among clones, and the copy number variations and mutations (depending on the available data sources) that occur in each clone. The presence of multiple detectable clones is called “clonal heterogeneity” in the literature.

Clonal heterogeneity likely plays an important role in the clinical course of a cancer. It is possible, for example, that the tumor cells that will eventually become the refractory cancer after treatment are present as a minor subclone in the tumor early on.

First, we load the **CloneSeeker** package:

```
> library(CloneSeeker)
```

## 2 Simulated Tumor Containing Multiple Clones

In order to illustrate the algorithms, we are going to simulate data where we know the true structure. Specifically, we will simulate copy number and mutation data for a tumor with three clones. We start with an object that represents the Tumor at a somewhat abstract level.

```
> set.seed(21303) # for reproducibility
> simTumor <- Tumor(c(5, 3, 2), rounds = 100,
+                 nu = 10, pcnv = 0.8, norm.contam = FALSE)
```

The first argument to the `Tumor` constructor is a vector that specifies the relative proportions of cells belonging to each clone; the length of the vector determines the number of clones. These values are automatically converted to fractions that add up to one:

```
> simTumor@psi
```

```
An object of class "WeightVector"
Slot "psi":
[1] 0.5 0.3 0.2
```

The second argument, `rounds`, specifies the number of generations through which the tumor clones are evolved. The idea is that new abnormalities, either in the form of mutations or copy number variants (CNVs), are acquired at each evolutionary step from some parent cell. The parameter `nu` is the expected number of new mutations and the parameter `pcnv` is the probability of a new CNV at each step. The final parameter, `norm.contam`, is a logical indicator of whether the tumor sample is assumed to include a subset of cells that represent non-cancerous “normal contamination”.

The resulting simulated tumor contains descriptions of each individual clone. In the current implementation, these are stored as a list of clones.

```
> class(simTumor@clones)

[1] "list"

> length(simTumor@clones)

[1] 3
```

Individual clones contain descriptions of both CNVs and mutations.

```
> oneClone <- simTumor@clones[[1]]
> class(oneClone)

[1] "list"

> length(oneClone)

[1] 2

> names(oneClone)

[1] "cn" "seq"
```

The copy number data includes the chromosome, with start and end positions, the number of copies of the A and B alleles, an arbitrary “segment” identifier, and (as a residual from the simulated evolutionary history), a “parent” identifier.

```
> dim(oneClone$cn)
```

```
[1] 320 7
```

```
> summary(oneClone$cn)
```

chr		start		end		A	
Min.	: 1.000	Min.	: 1	Min.	: 512228	Min.	:1
1st Qu.:	4.000	1st Qu.:	41806795	1st Qu.:	58350286	1st Qu.:	1
Median	: 9.000	Median	:117639946	Median	:139678132	Median	:1
Mean	: 9.756	Mean	:114603107	Mean	:133296903	Mean	:1
3rd Qu.:	15.000	3rd Qu.:	181197780	3rd Qu.:	208210950	3rd Qu.:	1
Max.	:24.000	Max.	:248891168	Max.	:249250621	Max.	:1

  

B		seg		parent.index	
Min.	:0	Min.	: 1.00	Min.	:5
1st Qu.:	1	1st Qu.:	80.75	1st Qu.:	5
Median	:1	Median	:160.50	Median	:5
Mean	:1	Mean	:160.50	Mean	:5
3rd Qu.:	1	3rd Qu.:	240.25	3rd Qu.:	5
Max.	:2	Max.	:320.00	Max.	:5

The mutation data has a chromosomal location, arbitrary segment and mutation identifiers, the number of mutated and wild type copies for each mutation, and the affected allele.

```
> dim(oneClone$seq)
```

```
[1] 12 7
```

```
> oneClone$seq
```

	chr	start	seg	mut.id	mutated.copies	allele	normal.copies
1	1	30402443	6	26	1	B	1
2	1	62695072	10	27	1	B	1
3	2	35569893	32	28	1	B	1
4	4	106054079	81	911	1	B	1
5	4	167504306	88	912	1	B	1
6	5	152937009	106	913	1	B	1
7	8	20376073	147	914	1	A	1
8	14	86900273	232	29	1	B	1
9	14	93651071	232	30	1	B	1
10	21	12282334	291	31	1	A	1
11	23	55921166	306	915	1	B	1
12	24	45378979	318	916	1	B	1

## 2.1 Simulating Tumor Data

Now that we have the tumor in place, we can simulate data arising from a study of that tumor.

```

> simData <- generateTumorData(simTumor,
+                               snps.seq = 10000,
+                               snps.cgh = 600000,
+                               mu = 70,
+                               sigma.reads = 25,
+                               sigma0.lrr = 0.15,
+                               sigma0.baf = 0.03,
+                               density.sigma = 0.1)

```

For a description of the many parameters to the `generateTumorData` function, see the man page. The first two arguments are size parameters. The first, `snps.seq`, determines the number of *germline* variants to simulate; in the absence of separate copy number data, these are used to provide a crude estimate. The second, `snps.cgh`, represents the number of SNP locations on the simulated SNP chip from which copy number segments are derived. The remaining parameters control the simulated read depth and variability.

As with individual clones, the simulated data is structured as a list with separate data frames for the CNVs and mutations.

```

> class(simData)

[1] "list"

> length(simData)

[1] 2

> names(simData)

[1] "cn.data" "seq.data"

```

The simulated copy number data includes chromosomal locations along with estimated log R ratios (LRR), B allele frequencies (BAF), separate intensity values for the two parental alleles (X and Y), and the number of SNPs in each segment (**markers**).

```

> cnDat <- simData$cn.data
> dim(cnDat)

[1] 320 7

> summary(cnDat)

```

chr	seg	LRR	BAF
Min. : 1.000	Min. : 1.00	Min. : -0.1225427	Min. : 0.3335
1st Qu.: 4.000	1st Qu.: 80.75	1st Qu.: -0.0024157	1st Qu.: 0.4995
Median : 9.000	Median : 160.50	Median : 0.0000065	Median : 0.5000
Mean : 9.756	Mean : 160.50	Mean : -0.0006059	Mean : 0.5009
3rd Qu.: 15.000	3rd Qu.: 240.25	3rd Qu.: 0.0024774	3rd Qu.: 0.5005
Max. : 24.000	Max. : 320.00	Max. : 0.0949327	Max. : 0.6670
X	Y	markers	

Min.	:0.5031	Min.	:0.5083	Min.	: 852
1st Qu.	:0.9950	1st Qu.	:0.9941	1st Qu.	:1592
Median	:0.9999	Median	:1.0001	Median	:1884
Mean	:1.0012	Mean	:0.9970	Mean	:1875
3rd Qu.	:1.0057	3rd Qu.	:1.0054	3rd Qu.	:2107
Max.	:1.4959	Max.	:1.0459	Max.	:2971

The simulated sequencing data, in addition to chromosomal locations, has read counts for the number of reference alleles, alternate (meaning variant or mutated) alleles, total counts, the variant allele frequency (VAF), and a status indicator of whether the variant is believed to be germline or somatic.

```
> dim(simData$seq.data)
[1] 10071      8

> seqDat <- simData$seq.data
> somatic <- seqDat[seqDat$status=='somatic',]
> dim(seqDat)
[1] 10071      8

> summary(seqDat)

      chr      seg      mut.id      refCounts
Min.   : 1.000   Min.    : 1.0   Min.    : 1.0   Min.    : 22.00
1st Qu.: 4.000   1st Qu.: 83.0   1st Qu.: 85.5   1st Qu.: 61.00
Median : 9.000   Median :162.0   Median :159.0   Median : 70.00
Mean    : 9.864   Mean    :162.1   Mean    :280.2   Mean    : 70.29
3rd Qu.:15.000   3rd Qu.:242.0   3rd Qu.:504.5   3rd Qu.: 80.00
Max.    :24.000   Max.    :320.0   Max.    :916.0   Max.    :161.00
NA's    :10000

      varCounts      VAF      totalCounts      status
Min.    : 8.0   Min.    :0.05714   Min.    : 46   Length:10071
1st Qu.: 60.0   1st Qu.:0.47099   1st Qu.:123   Class :character
Median : 70.0   Median :0.50000   Median :140   Mode  :character
Mean    : 69.7   Mean    :0.49789   Mean    :140
3rd Qu.: 79.0   3rd Qu.:0.52996   3rd Qu.:157
Max.    :130.0   Max.    :0.74468   Max.    :233

> table(seqDat$status)

germline  somatic
10000      71
```

### 3 Seeking Clones

To run *CloneSeeker*, we will need a starting set of  $\psi$  vectors as inputs, where  $\psi$  records the fraction of cells belonging to each clone. For each  $\psi$  vector, the algorithm will compute the most

probable copy number state for each clone at each segment. The maximum posterior probability is computed for each input  $\psi$  vector, and these probabilities are used to resample new potential  $\psi$  vectors. We usually start by considering every possible decomposition of the tumor into five clones, where the fraction assigned to each clone is a multiple of  $1/20 = 0.05$ . We can generate this initial matrix of  $\psi$  vectors as follows:

```
> psis <- generateSimplex(20, 5)
> dim(psis)
```

```
[1] 192 5
```

```
> head(psis)
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,] 1.00 0.00 0.00 0.00 0.00
[2,] 0.95 0.05 0.00 0.00 0.00
[3,] 0.90 0.10 0.00 0.00 0.00
[4,] 0.90 0.05 0.05 0.00 0.00
[5,] 0.85 0.15 0.00 0.00 0.00
[6,] 0.85 0.10 0.05 0.00 0.00
```

```
> tail(psis)
```

```
      [,1] [,2] [,3] [,4] [,5]
[187,] 0.25 0.25 0.25 0.20 0.05
[188,] 0.25 0.25 0.25 0.15 0.10
[189,] 0.25 0.25 0.20 0.20 0.10
[190,] 0.25 0.25 0.20 0.15 0.15
[191,] 0.25 0.20 0.20 0.20 0.15
[192,] 0.20 0.20 0.20 0.20 0.20
```

For SNP array data, we also need, as input, a set of possible clonal segment copy number states. If none exists the function will automatically generate one. The version used here considers all possible copy number states from 0 to 5 copies, but it imposes a strong prior belief that two different clones cannot both gain and lose the same segment.

```
> cnmodels <- expand.grid(rep(list(0:5),5))
> include <- sapply(1:nrow(cnmodels), function(i) {
+   length(which(cnmodels[i,] >= 1))==5 | length(which(cnmodels[i,] <= 1)) == 5
+ })
> cnmodels <- cnmodels[include,]
```

Now we will define the other algorithm parameters:

```
> pars <- list(sigma0 = 1,      # SNP-wise standard deviation
+             ktheta = 0.3,    # geometric prior parameter on number of clones
+             theta = 0.9,     # geometric prior parameter on copy number changes
+             mtheta = 0.9,    # geometric prior parameter on point mutations)
```

```

+         alpha = 0.5,    # parameter for a symmetric Dirichlet prior on psi
+         thresh = 0.04, # smallest possible detectble clone
+         cutoff = 100,  # filter out copy number segments supported by fewer SNPs
+         Q = 100,       # number of new psi vectors resamples at each iteration
+         iters = 4)      # number of iterations

```

### 3.1 Seeking Clones from Copy Number Data

The `seekClones` function can estimate the clonal architecture from copy number data, or from mutation and variant data, or jointly from both kinds of data. In this section, we will run the algorithm using **only the copy number data**. To do that, we set the `varData` argument to `NULL`.

```

> resCN <- seekClones(cndata = cnDat, vardata = NULL,
+                   cnmodels = cnmodels, psiset = psis, pars = pars)

```

Here are the results of the “CNV only” analysis of this sample:

```

> resCN$psi

[1] 0.5 0.3 0.2 0.0 0.0

> simTumor@psi

```

```

An object of class "WeightVector"
Slot "psi":
[1] 0.5 0.3 0.2

```

In this case, `CloneSeeker` accurately estimates not only the number of clones but also the clonal fractions. Let’s look at the clonal copy number assignments as well:

```

> trueCN_Assignments <- t(sapply(1:nrow(resCN$filtered.data$cndata.filt),
+ function(i) {
+   index <- rownames(simTumor@clones[[1]]$cn) ==
+     rownames(resCN$filtered.data$cndata.filt)[i]
+   sapply(1:length(simTumor@clones),function(j){
+     simTumor@clones[[j]]$cn$A[index] + simTumor@clones[[j]]$cn$B[index]
+   })
+ }))
> inferredCN_Assignments <- (resCN$A+resCN$B)[,1:length(simTumor@clones)]
> colnames(inferredCN_Assignments) <- colnames(trueCN_Assignments) <-
+   paste("C", 1:3)
> data.frame(Truth = trueCN_Assignments,
+            Infer = inferredCN_Assignments)

```

	Truth.C.1	Truth.C.2	Truth.C.3	Infer.C.1	Infer.C.2	Infer.C.3
22	2	2	2	2	2	2
24	2	3	2	2	3	2

128	2	2	1	2	2	2
170	1	2	2	1	2	2
226	2	1	2	2	1	2
244	3	2	2	3	2	2
248	2	1	1	1	2	2
297	2	3	2	2	3	2
315	2	1	2	2	1	2

Although not perfect, the algorithm managed to correctly estimate most of the segment-wise allelic copy numbers of different clones.

### 3.2 Sequencing Data

Now, let's illustrate the use of `CloneSeeker` in analyzing mutation data (by which we mean variant data such as one would find in a `.vcf` file) to seek clones. This time, we run the `CloneSeeker` algorithm with the `cndata` argument set to `NULL`.

```
> resMut <- seekClones(cndata = NULL, vardata = seqDat,
+                      cnmodels = cnmodels, psiset = psis, pars = pars)
```

Here the results aren't as good; at least one of the actual clones has been split into separate pieces.

```
> resMut$psi
```

```
[1] 0.53443247 0.19437358 0.10946443 0.10387835 0.05785117
```

```
> simTumor@psi
```

```
An object of class "WeightVector"
```

```
Slot "psi":
```

```
[1] 0.5 0.3 0.2
```

### 3.3 Both Sequencing and SNP Array Data

Finally, we illustrate running `CloneSeeker` on a sample for which there is both SNP array and mutation data.

```
> resBoth <- seekClones(cndata = cnDat, vardata = somatic,
+                      cnmodels = cnmodels, psiset = psis, pars = pars)
```

And we can look at the inferred allocation of tumor fraction to clones:

```
> resBoth$psi
```

```
[1] 0.45 0.30 0.15 0.05 0.05
```

```
> simTumor@psi
```



```
An object of class "WeightVector"  
Slot "psi":  
[1] 0.5 0.3 0.2
```

Surprisingly, the results here are similar to the overaggressive results obtained using just the sequencing data rather than the simpler and correct results obtained when using just the copy number data.

In conclusion, CloneSeeker can be applied effectively to cases where one has SNP array data, (processed) sequencing data, or both.