Cancer Evolution 2

USN: 303039534

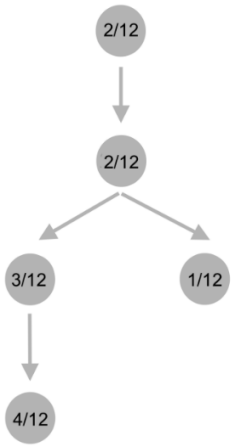
A

1

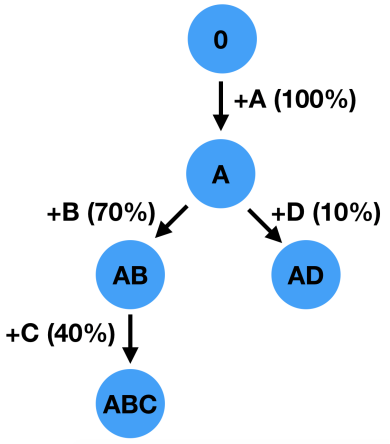
We label the aquired mutations and clones as illustrated in figure 1b. We then note from figure 1a that the healthy cell population represents $\pi_0 = 2/12$ of the total cell population. The cancer cell fraction for a given set of mutations i emerging in clone k is thus given by

$$\tau_i = \frac{N_i}{N_{cancer}} = \frac{12}{10}\tau_i^{cell} = \frac{12}{10}\sum_{I \in \text{de}(k)} \pi_I \tag{1}$$

Where τ_i^{cell} is the cell fraction of mutation i and π_I is the clonal frequency of clone I with all clonal frequencies given in figure 1a.



(a) Phylogenetic tree and clonal frequencies of the tumor given in the assignment.



(b) Phylogenetic tree with mutations and clones labelled. Cancer cell fractions are given in brackets next to the edge at which a given set of mutations is acquired.

Figure 1: Tumor phylogenetic trees illustrating clonal frequencies (left) and cancer cell fractions (right).

The cancer cell fractions are thus given by [100%, 70%, 40%, 10%] corresponding to cellular frequencies of [83.3%, 58.3%, 33.3%, 8.3%].

2

PyClone and Ccube were installed as specfied in their documentation and run with default parameters unless otherwise stated. All calculations were run on subliminal to make runtimes comparable.

In table 1 we report both the runtimes and inferred number of clusters of CCFs for both methods. We note that the runtime for Ccube depends on the maximum number of clusters considered (*numOfClusterPool*) and is super-linear in this parameter. We therefore report runtimes considering both numOfClusterPool=1:6 as in the vignette and numOfClusterPool=1:11 since the maximum number of possible clusters is 11 (see section 5).

We see that the runtime for PyClone is generally much higher than that for Ccube, even when allowing for up to 11 clusters in Ccube. This will be discussed further below. We also note that the runtime for PyClone increases by 76% when including copy number variation whereas it only increases by 50% when considering up to 11 clusters in Ccube. When only considering up to 6 mutations in Ccube, the runtime increases by 150% when including copy number variation, but the runtime is more than two orders of magnitude faster than PyClone in this case.

	PyClone 0%	PyClone 70%	Ccube 0% (6)	Ccube 70% (6)	Ccube 0% (11)	Ccube 70% (11)
runtime (s)	1071.04	1886.60	4.32	10.49	15.58	23.69
clusters	20	67	4	4	4	5
clusters (> 1)	4	6	4	4	4	5

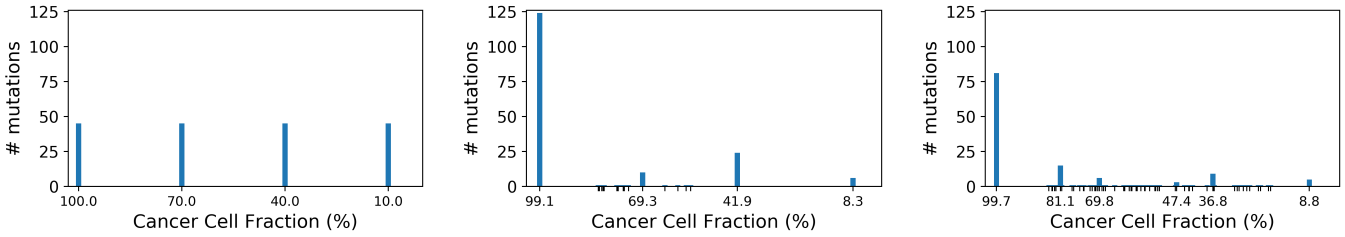
Table 1: Runtime and inferred cluster counts for PyClone and Ccube. "0%" indicates data with no copy number changes and "70%" indicates data with copy number changes at 70% of loci. The number in brackets for the Ccube calculations specifies the maximum number of clusters considered in the algorithm. The third line indicates the number of clusters assigned more than one mutation.

3

The number of inferred mutations for each calculation is also given in table 1. PyClone often found clusters of singletons which may arise primarily due to noise, and we therefore also report in the third row of the table the number of CCF clusters found for which the mutation count is greater than 1.

We see that when considering only these non-singleton clusters, both PyClone and Ccube correctly infer 4 CCF clusters. However, Ccube does so much more quickly and without the additional singleton clusters from PyClone. In addition, Ccube correctly identifies 4 clusters even in the presence of 70% copy number variation when allowing up to 6 clusters, although it identifies 5 clusters when allowing up to 11 clusters. In this case, PyClone incorrectly infers 6 clusters.

To further compare the inferred CCF clusters with the real CCF clusters, we plot in figure 2 the real and inferred cancer cell fractions in a format similar to the 'column summary' from assignment 1. Note that the bar heights are arbitrary for the reference data as we are not given information on how many mutations were simulated for each clone.

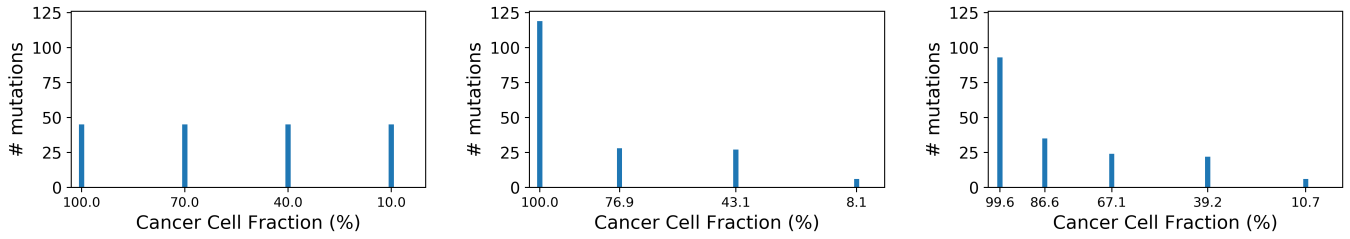


(a) Reference (b) No CNV (c) 70% CNV

Figure 2: CCF clusters for reference data (a) or PyClone inferences with no copy number variation (b) or copy number alterations at 70% of sites (c). x-axes indicate CCF of an inferred cluster of mutations, y-axes indicate number of mutations in a cluster. Note that the cluster sizes in (a) are arbitrary and only clusters of more than 1 mutation are annotated in (b), (c).

We see that for the data with no copy number variation, PyClone correctly finds four major clusters with cell fractions similar to those of the reference data. In addition, it finds a number of small clusters with only a single mutation. When including copy number variation at 70% of the sites, PyClone still finds major clusters at approximately 100%, 70%, 37% and 9%. However, it finds an additional major cluster at 81% which appears to have erronously been separated from clone A at 100% CCF. It also finds a small cluster of 2 mutations with a CCF of 47% which does not exist in the reference data. Performance thus deteriorates significantly both in terms of runtime and accuracy when including copy number variations.

We perform a similar set of analyses using Ccube with numOfClusterPool=1:11 and plot the results in figure 3



(a) Reference (b) No CNV (c) 70% CNV

Figure 3: CCF clusters for reference data (a) or Ccube inferences with no copy number variation (b) or copy number alterations at 70% of sites (c). x-axes indicate CCF of an inferred cluster of mutations, y-axes indicate number of mutations in a cluster. Note that the cluster sizes in (a) are arbitrary.

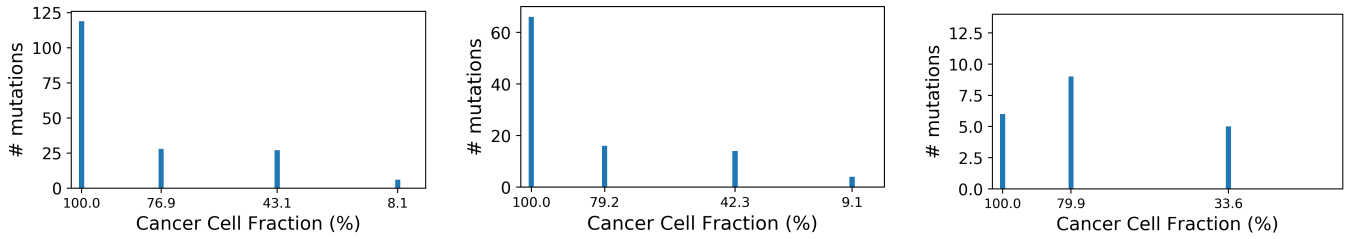
We again observe good performance in the case without copy number variations, and in contrast to PyClone Ccube converges to 4 distinct clusters with no singleton noise. However, the CCFs for mutations B and C are somewhat overestimated while that of D is underestimated. In contrast, PyClone yielded more accurate estimates of the underlying cancer cell fractions but less accurate clustering.

Ccube performs relatively well compared to PyClone when including copy number variation, although this method also breaks clone A into two separate clusters similar to PyClone, giving rise to an additional cluster at a cancer cell fraction of 87%.

4

We proceed to run a Ccube analysis on downsampled data containing either 100 or 20 sites rather than the original 180 sites. This provides less data for the algorithm and we thus expect worse performance.

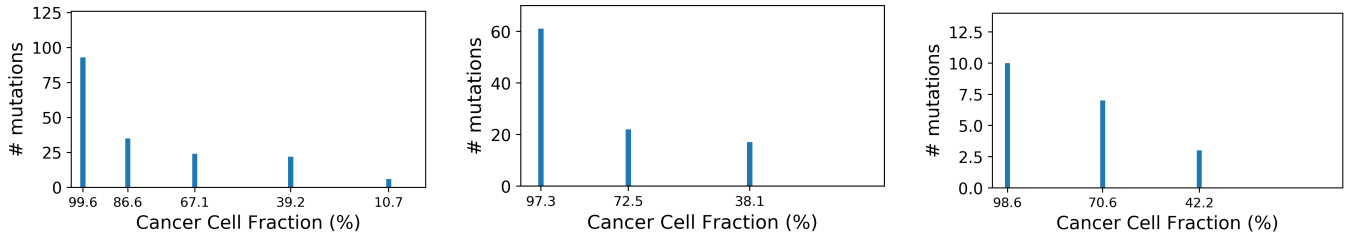
In figure 4 we compare the performance across the three datasets with no copy number variation. We see that performance is relatively robust with 4 clusters at approximately the same cellular fractions being inferred for both 180 and 100 sites. Only three clusters are identified after downsampling to 20 sites. The relative sizes of the clusters also differ between the three datasets, but this may be due to stochasticity in the subsampling rather than errors in the inference. We also note that the inferred CCFs become less consistent with those of the reference dataset following downsampling.



(a) 180 sites (b) 100 sites (c) 20 sites

Figure 4: CCF clusters inferred by Ccube for the orignal data (a) or downsampled datasets (b,c) in the case of no copy number alterations.

When considering the dataset with copy number alterations at 70% of sites, performance is less robust and we again converge to fewer clusters with increased downsampling. These clusters look qualitatively similar to those converged upon after downsampling the data without copy number variation.



(a) 180 sites (b) 100 sites (c) 20 sites

Figure 5: CCF clusters inferred by Ccube for the orignal data (a) or downsampled datasets (b,c) with copy number alterations at 70% of sites.

In summary, we thus conclude that Ccube generally outperforms PyClone both in terms of runtime and clustering. However, when ignoring singleton clusters, the CCFs inferred by PyClone tend to be quite accurate in the absence of copy number variation. We note that in general the problem of clustering becomes substantially harder when copy number alterations are introduced, and that results are less robust with downsampled data.

5

PyClone and Ccube are conceptually similar by virtue of both being stochastic methods that use infinite mixture modelling to infer cancer cell fractions from read data and copy number data by clustering genomic sites. In both methods, the tumor is considered to consist of three population at each site; a normal population, a tumor reference population, and a tumor variant population.

Both methods also use data generating models that draw VAFs from a distribution over possible mutational scenarios and CCFs, and then draw reads from a binomial distribution of the VAFs and total read counts. The log likelihood of possible CCF clustering scenarios are assessed and the optimum set of clusters returned as the most likely CCF values.

However, despite these underlying similarities there are also theoretical and methodological differences. For example, PyClone uses a uniform prior over different mutational patterns for generating mutational signatures with one of the possible priors being the Parental Copy Number Prior (PCN). The PCN is equivalent to our simulation of VAFs and read data detailed in sections 6 and 7 below. In contrast, Ccube assumes that the total number of alleles in the reference population is always equal to the total number of alleles in the variant population, i.e. that the copy number change is clonal. This allows the authors to derive a linear relationship between CCF and VAF

$$f = w\phi + \epsilon \tag{2}$$

$$w = \frac{t(m(1-\epsilon) - n_{tot_t}\epsilon)}{(1-t)n_{tot_n} + tn_{tot_t}} \tag{3}$$

Where ϕ is the CCF, f is the VAF, ϵ is the sequencing error, n_{tot} are total copy numbers, t is the tumor purity, and m is the number of mutated chromosomal copies.

This in turn allows Ccube to fit its model using an iterative optimization procedure where the posteriors of the CCFs (ϕ) are optimized and then clustered using a Student-t mixture model. However, this approach requires pre-specification of the numbers of clusters to be considered followed by model selection based on an estimated lower bound of the maximum likelihood (ELBO).

In contrast, PyClone uses a Bayesian non-parametric clustering method which does not require *a-priori* specification of the number of clusters. PyClone then uses a Markov Chain Monte Carlo (MCMC) simulation to estimate a posterior over possible CCFs. At each iteration of the MCMC, new values of ϕ are sampled for each mutation and each cluster of mutations using a Dirichlet Process parametized by a Gamma distribution. By default, this MCMC runs for 10,000 iterations to give an accurate posterior, and clustering is then performed based on a posterior similarity matrix quantifying the co-occurrence of each pair of mutations at each point in the MCMC chain.

However, this gives rise to an algorithm that scales as $\mathcal{O}(N^2M)$ where N is the number of mutations and M is the chain length. This in turn is the reason why the PyClone runtime is significantly longer than the Ccube runtime.

PyClone also allows for the incorporation of multiple samples in a single calculation, e.g. from different points in space and time, which can improve the inference. Additionally, PyClone was designed to work with high very coverage data (read depth $> 100x$), and in the original publication some of the example data was sequenced at 5,000x. We thus note that the somewhat poor performance of PyClone above may be mitigated by increased sequencing depth and multiple samples.

Another reason why more noise and singleton groups are observed for PyClone than for Ccube is that Ccube is explicitly designed to merge small clusters and clusters with similar CCF. Thus in Ccube, clusters with less than 1% of the total number of mutations are removed, and if two clusters differ by less than 10% CCF they are merged. This means that Ccube cannot generate more than at most 11 clusters, explaining the upper bound on the number of clusters in our calculations above.

We also note that both methods provide estimates of uncertainty for each mutational site; something we have not considered in the present report. However, it may also be interesting to compare how these uncertainty estimates compare between the two methods.

B

6

As in the lectures, we consider three different scenarios describing how copy number changes and mutations can co-occur at a single locus. In the following, we denote the total number of alleles at locus i by n and the minor allele count m. We assume that all CN changes (n, m) happen in a single step. Denoting the number of mutant alleles n_B, the three scenarios are

1. SNV occurs after CNA; $n_B = 1$
2. SNV occurs before CNA on the minor copy; $n_B = m$
3. SNV occurs before CNA on the major copy; $n_B = n - m$

Given n_B and n, m for each clone, we can then calculate n_A and the Variant Allele Frequency (f_B)

$$f_B = \frac{n_B}{n} = \frac{\sum_{k=0}^{K} \pi_k n_{kB}}{\sum_{k=0}^{K} \pi_k n_k} \tag{4}$$

In this section, we assume that all clones carrying an SNV share the same copy number at this locus and that all cancer cells without the SNV also share a potentially different copy number state. Normal cells are assumed to be diploid, and we assume that only one copy number change and one SNV occur at each site. This divides our tumor into three populations at each site. Denoting the allele counts of the total and mutant alleles $c = (n, n_B)$, these are

1. normal cells with $c^N = (2, 0)$
2. reference cancer cells with $c^R = (n^R, 0)$
3. variant cancer cells with $c^V = (n^V, n_B)$

Combining these approximations with our three possible scenarios for the emergence of a mutation gives us three possible count vectors for each cell population as specified below:

1. SNV occurs after CNA; $c^R = (n, 0)$ & $c^V = (n, 1)$
2. SNV occurs before CNA on the minor copy; $c^R = (2, 0)$ & $c^V = (n, m)$
3. SNV occurs before CNA on the major copy; $c^R = (2, 0)$ & $c^V = (n, n - m)$

$c^N = (2, 0)$ in all three cases.

In the following, we consider the clonal tree from assignment 1 (figure 6) with $\pi_0 = 0.2$. We assume all sites to be independent and thus choose n_B independently for each site.

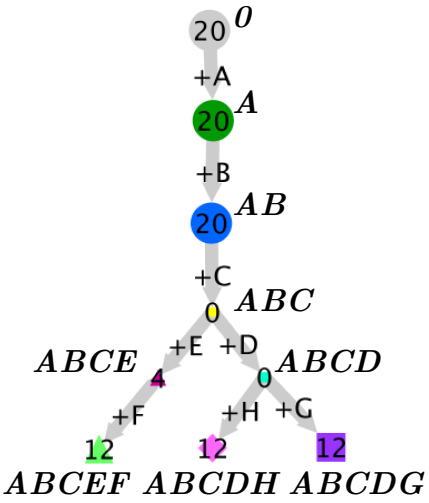


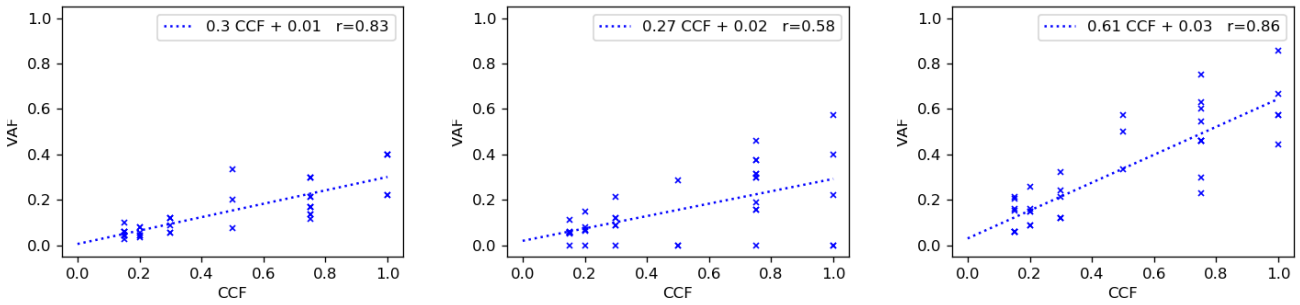
Figure 6: Clonal evolution tree from assignment 1. Nodes are labelled with their clonal frequency and genotypes.

In the Ccube vignette, the authors let the major copy number at each site be $n_{major} \in \{1, 2, 3, 4\}$ with probabilities $p = \{0.3, 0.3, 0.2, 0.2\}$ and they let the minor copy number be $n_{minor} \in \{0, 1, 2\}$ with probabilities $p = \{0.25, 0.5, 0.25\}$. Assuming that Yuan et al. have put some thought into this choice of copy number probabilities and acknowledging the fact that I know little about what would be a better choice, the same parameters have been used in the following.

This allows us to simulate for each of our 40 sites a major, minor and total copy number. This in turn allows us to find $c^N, c^R,$ & c^V conditional on the mutational scenario considered. Given these count vectors, we can now calculate the Variant Allele Frequency using equation 5. In the following, we assume the sequencing error rate to be neglible.

$$f_B = \frac{(1-\pi_0)\tau \cdot n_B}{\pi_0 \cdot 2 + (1-\pi_0)(1-\tau) \cdot n^R + (1-\pi_0)\tau \cdot n^V} \tag{5}$$

We use equation 5 to calculate VAFs for all 40 loci in each of the three mutational scenarios above after simulating copy numbers as described. We plot the resulting VAFs as a function of Cancer Cell Fraction (CCF) in figure 7.



(a) SNV after CNV (b) SNV in minor allele (c) SNV in major allele

Figure 7: Variant Allele Frequency as a function of CCF in the three different mutational scenarios discussed in the main text. Dotted lines and legends represent linear lines of best fit.

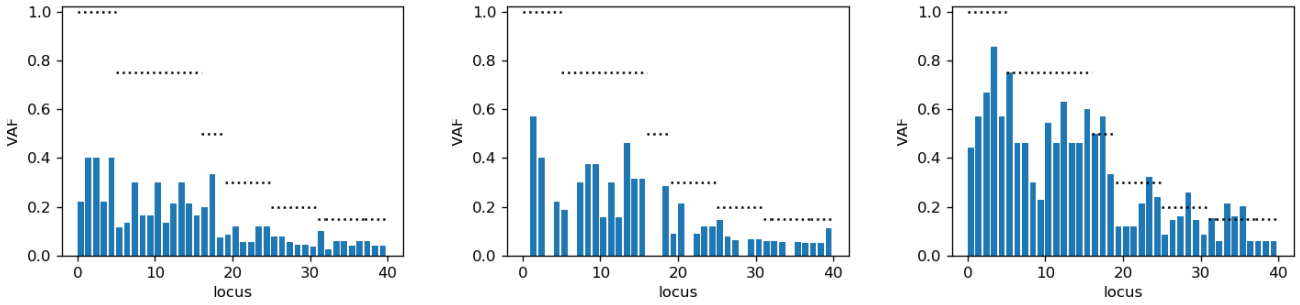
We see that in each case, VAF increases with CCF as expected since variant alleles can only be present in the population that has undergone an SNV, irrespective of major and minor copy numbers. For the the first and third scenario, relations between CCF and VAF are largely linear. We can explain this phenomenon from equation 5 by noting that the numerator is linear in τ and that the τ-dependent terms in the denominator are small when $\tau \to 0$. At larger τ, the τ-dependent terms cancel when $n^R = n^V$ as in the assumptions underlying the Ccube algorithm and scenario 1 above.

In contrast, there is more noise in the case of SNVs occurring in the minor allele, and much of this arises from cases where the VAF is 0. This is of course an un-realistic scenario as we cannot detect a mutation occurring in an allele

that no longer exists which will be discussed further below. In this case, the numerator is 0 and the VAF is no longer linear in CCF.

Finally, we note that we obtain much higher VAFs when mutations occur in the major allele than in either of the other two cases. n_B is always 1 when the SNV occurs after the CNV and $n_B \leq 2$ when the SNV occurs in the minor allele. The weighted average of minor allele frequencies in our model is also equal to 1 which explains why VAF as a function of CCF have similar slopes in the first and second scenarios. In contrast, the mean value of n_B when a mutation occurs in the major allele is 2.3 explaining why the slope in scenario 3 is approximately twice as high.

Finally, we also plot the same data as a bar plot showing the VAF for each individual locus in figure 8 and again note the high degree of variability even for loci with the same CCF, leading to significant VAF overlap between the clusters. This is a major reason why the clustering problem is hard.



(a) SNV after CNV (b) SNV in minor allele (c) SNV in major allele

Figure 8: VAF at each locus of the phylogenetic tree in figure 6. Horizontal dotted lines indicate the corresponding CCF at each locus.

7

To simulate real data for analysis by PyClone and Ccube, we use a combination of these three different scenarios since they all occur in vivo. We let it be random whether an SNV occurs before or after the copy number alteration. If the SNV occurs before the CNV and the minor allele count satisfies $m > 0$, we let it be random if the SNV occurred in the major or minor allele. However, if $m = 0$, the SNV must have occurred in the major allele for us to detect it, and we thus let it occur in the major allele with $p = 1$. This biases our data towards major allele mutations; an effect that is also expected in real data.

To model read data, we need to take into account the total copy number at each site as the total number of reads at a site r_i is expected to be proportional to the total copy number n_i at that site. Next-generation sequencing data is known to be overdispersed, and we therefore draw the total read count at site i r_i from a negative binomial distribution

$$r_i \sim \text{NegBin}(\lambda_i = 100 n_i, s = 100) \tag{6}$$

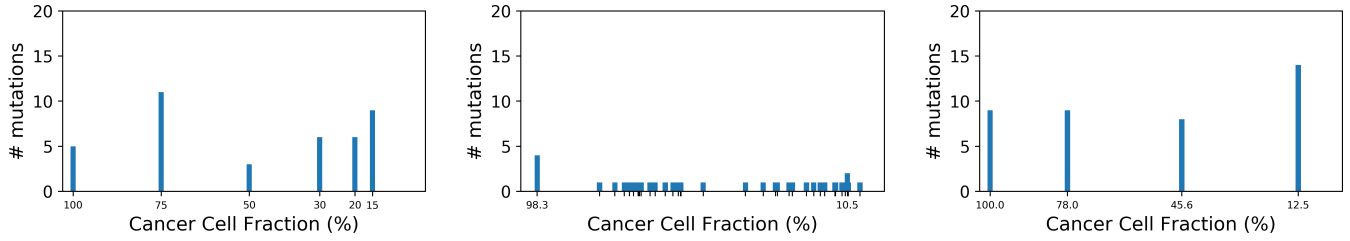
The mean used here is four times higher than that used in the Ccube vignette but still lower than the read-depth used for benchmarking in the original PyClone publication.

Given our read data r_i and VAF f_{Bi} at locus i, we can then draw our variant read count R_{Bi} from a binomial distribution

$$R_{Bi} \sim \text{Bin}(r_i, f_{Bi}) \tag{7}$$

Finally we can calculate the reference read count as $R_{Ai} = r_i - R_{Bi}$.

Together with the copy number information from above, this provides all of the data needed to run PyClone and Ccube on our simulated data with the resulting CCF profiles given in figure 9.



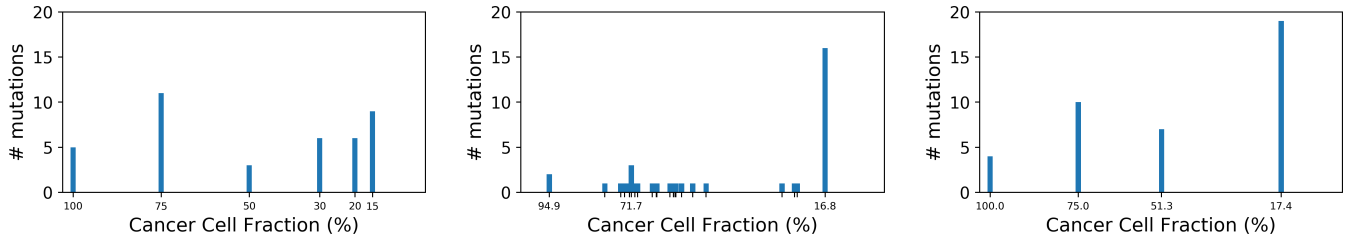
(a) (b) (c)

Figure 9: (a) CCF clusters for simulated data (100% CNVs, 100 reads per copy) and the corresponding inferred clusters by PyClone (b) and Ccube (c). x-axes indicate CCFs of the inferred clusters, y-axes indicate number of mutations in a cluster.

We see that the performance of both methods is significantly worse than in the example in section A, with Ccube underestimating the number of clusters and PyClone failing to identify more than 1-2 significant clusters. This drop in performance can be explained by the change in number of sites and number of clusters. In the present dataset, there are too few loci for each clone for PyClone to cluster them accurately, leading to predominantly singleton clusters.

This is similar to our considerations in section 4 where we downsample the original data, but with the clustering problem being even more difficult here since the increased number of clusters leads to more overlap between the distributions of VAFs. We also note that the data has three clusters at low CCF with very similar CCF values of 15%, 20% and 30%. This leads to merging of clusters in Ccube which by design is unable to resolve clusters separated by less than 10% CCF.

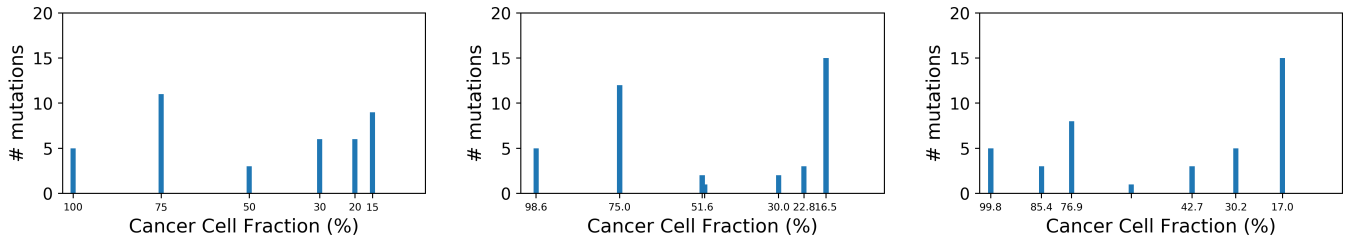
We find that the task becomes somewhat easier if we have fewer copy number variations, with both algorithms performing better if only 50% of the sites have simulated copy number changes as illustrated in figure 10. Of course Ccube still fails to resolve the low-CCF clusters; but the three high-CCF clusters are now reasonably accurate.



(a) (b) (c)

Figure 10: (a) CCF clusters for simulated data (50% CNVs, 100 reads per copy) and the corresponding inferred clusters by PyClone (b) and Ccube (c).

Even in the case of 50% CNV, performance by PyClone is still somewhat poor. However, as expected from section 5, we find that increasing the depth of sequencing by a factor of 10 dramatically increases the performance of PyClone while making less of a difference to Ccube (figure 11). In fact with a mean read depth of 1000 reads per copy, PyClone correctly identifies all six clusters even with 100% CNVs, albeit with somewhat erroneous cluster sizes at low CCFs.



(a) (b) (c)

Figure 11: (a) CCF clusters for simulated data (100% CNVs, 1000 reads per copy) and the corresponding inferred clusters by PyClone (b) and Ccube (c).

In summary, it thus seems that Ccube performs substantially better for low-coverage data with few clusters but PyClone appears to be competitive for higher-coverage data with more clusters. PyClone has the additional advantage that no prior cluster number must be specified and that the number of clusters is unbounded. PyClone is also designed for use with multiple samples; a feature we have not leveraged in the present report. We also note that for the present case with only 40 sites, the result of a Ccube analysis differs somewhat when repeating the analysis, but the above figures exemplify the general trends.

We could further investigate the variability and robustness of the methods by generating repeated datasets corresponding to additional samples from the tumors and using more data to infer clusters. However, that is considered beyond the scope of the present report.

8

We expand our clonal tree from figure 6 by two metastatic trees in figure 12. One of these is an 'early tree' where the metastatic tumor is seeded from from clone A, and the other is a 'late tree' where the tumor is seeded from clone $ABCE$.

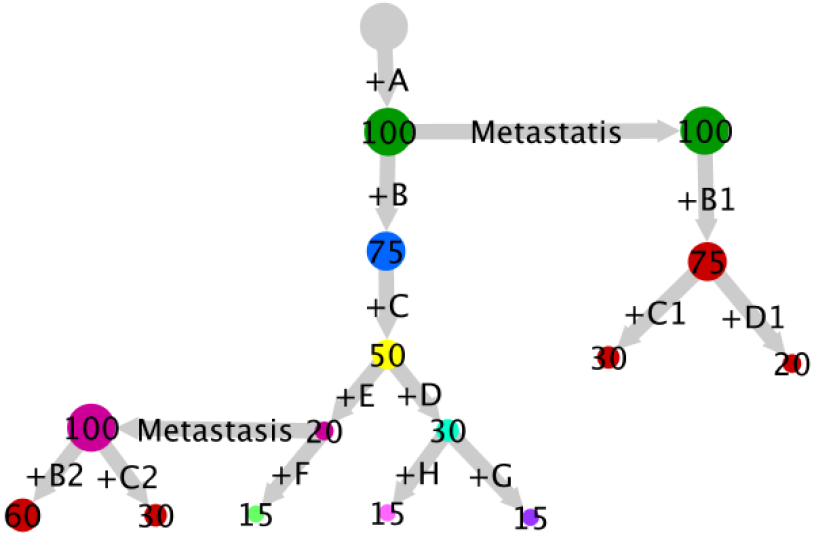
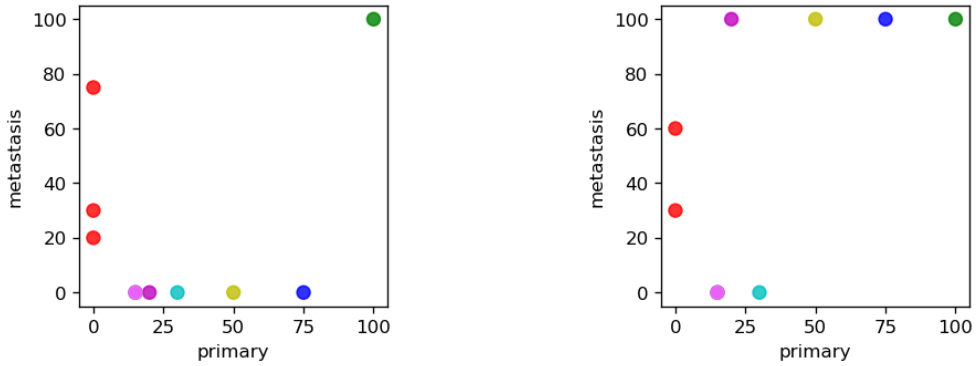


Figure 12: Clonal evolution tree extended by an early (right) and late (left) metastatic tree. Nodes in the original tree are coloured as in figure 6. All metastatic nodes are coloured red apart from the seeding clone which is coloured as in the base tree. Numbers on nodes indicate the CCF in that tumor of the most recently acquired mutation. Node size is a monotonically increasing function of this value.

In the early metastatic tumor (tumor 1), the only mutation present from the original tumor (tumor 0) is mutation A. All other mutations arised later in the phylogenetic tree of tumor 0, and since we assume that the same mutation does not arise twice these mutations will not be present in tumor 1; i.e. they have a CCF of 0% in tumor 1. Conversely, the mutations that arose in tumor 1 after metastasis will not be present in tumor 0 and thus have a CCF of 0% in tumor 0.

This is illsutrated in figure 13a where we see that the 7 later mutations in tumor 0 (coloured; three of these are overlapping) all have CCFs of 0% in tumor 1 while mutations B1, C1 and D1 arising in the metastatic tumor all have CCFs of 0% in the original tumor. When plotting the CCFs of the original vs metastatic tumor, an early metastasis is thus characterized by many mutations occurring on the primary axes with a CCF of 0% in one of the two tumors.



(a) Early metastasis

(b) Late metastasis

Figure 13: Plots showing the CCF in both the original tumor and an early (left) or late (right) metastatic tumor of mutations occurring in the original (coloured) or metastatic (red) tumor.

Conversely in the late metastatic tumor (tumor 2), all mutations present in the seeding cell $ABCE$ will have a CCF of 100% in the metastatic tumor (figure 13b). The late metastatic tumor is thus characterized by having multiple mutations which have CCFs of 100% in the metastatic tumor despite having less than 100% CCF in the original tumor. In the early metastasis in figure 13a, the only mutation with 100% CCF is the seeding mutation A. In the late metastatic tumor, the mutations arising after metastasis in either tumor 0 or tumor 2 will still have a CCF of 0% in the opposite tumor. In the case of polyclonal seeding, we can observe more complex CCF patterns with mid-range CCF values in both the primary and metastatic tumor, but that is not considered further in this report.

Appendix

This appendix includes all code used for the assignment

```
#function for plotting clusters as barplotts

import matplotlib.pyplot as plt
import numpy as np
import sys

def plot_data(cfs, sizes, sim = False, real=[10/10*100, 7/10*100, 4/10*100, 1/10*100],
                dirname='.'):
        args = np.argsort(-np.array(cfs))
        cfs = np.array(cfs)[args] #CCFs
        sizes = np.array(sizes)[args] #number of mutations per cluster

        #print some summary data
        print('cfs:', cfs)
        print('clusters:', len(cfs))
        print('sizes:', sizes)
        S = np.sum(sizes)
        print('total sites:', S)
        print('>1:', np.sum(sizes > 1.5))
        dirname = dirname+'/'

        if sim: fsize = 7.5
        else: fsize = 12

        ############plot inferred data
        ticks, va = [], []
        for i, f in enumerate(cfs): #need ticks for cellular fraction and CCF
                if sizes[i] > 1.5 or S < 30: ticks.append(str(np.round(f, 1)))
                else: ticks.append('')
                va.append(-0.03)
        fig = plt.figure(figsize = np.array([6,3])*0.8 )
        ax = plt.gca()
        plt.bar(cfs, sizes, width = 1.5)

        ax.set_xticks( cfs )
        ax.set_xticklabels( ticks, fontsize = fsize )
        plt.xlim(103,0)
        #ylims depend on the number of mutations considered
        if S > 150: smax = 126
        elif S > 50: smax = 70
        else: smax = 14
        if sim: smax = 20
        plt.ylim(0, smax)

        plt.yticks(FontSize = 12)
        #plt.title('Column summary', FontSize=fs)
        plt.xlabel('Cancer Cell Fraction (%)', FontSize=14)
        plt.ylabel('# mutations', FontSize=14)
        plt.savefig(dirname+'cols.png', bbox_inches = 'tight', dpi=360)
        plt.close()

        ############plot reference data
```

```
        if sim: #if working with simulated data, reference data is from assingment 1
                real = [100, 75, 50, 30, 20, 15]
                realsizes = [5, 11, 3, 6, 6, 9]
        else: realsizes = []
        ticks, va = [], []
        for i, f in enumerate(real): #need ticks for cellular fraction and CCF
                ticks.append(str(np.round(f, 1)))
                va.append(-0.03)
                if not sim: realsizes.append(180/4)
        fig = plt.figure(figsize = np.array([6,3])*0.8 )
        ax = plt.gca()
        plt.bar(real, realsizes, width = 1.5)

        ax.set_xticks( real )
        ax.set_xticklabels( ticks, fontsize = fsize )
        plt.xlim(103,0)
        plt.ylim(0, smax)
        plt.yticks(FontSize = 12)
        plt.xlabel('Cancer Cell Fraction (%)', FontSize=14)
        plt.ylabel('# mutations', FontSize=14)
        plt.savefig(dirname+'realcols.png', bbox_inches = 'tight', dpi=360)
        plt.close()



#script for plotting data from a pyclone calculation
import matplotlib.pyplot as plt
import numpy as np
import sys
from plot_data import plot_data

dirname = sys.argv[1]
real = [10/12*100, 7/12*100, 4/12*100, 1/12*100] #reference data from assignment 2
cfs = []
sizes = []
f = open(dirname+'/tables/cluster.tsv', 'r') #file with results
f.readline()
for line in f:
        split = line.split()
        cfs.append(float(split[3])*100) #extract CCF
        sizes.append(float(split[2])) #extract #mutations
f.close()

sim = False
if len(sys.argv) > 2:
        if sys.argv[2] == 'sim': sim = True #simulated data
plot_data(cfs, sizes, dirname=dirname, sim=sim) #plot result



#script for running a ccube calclation and plotting the result
require('ccube')
require('foreach')
require('dplyr')
library('reticulate')
```

```
args = commandArgs(trailingOnly=TRUE)
infile = args[1] #input file
outdir = args[2] #output directory
nclusts = as.numeric(args[3]) #max number of clusters to consider
sim = F
if (length(args)>3){
   if (args[4] == 'sim'){ #simulated data
      sim = T}
 }


if (sim){ #we read simulated data from a tsv file
dat = read.table(infile, sep = '\t', header=T)
}else{ dat = readRDS( infile ) } #we read assignemnt data from an RDS file

numOfClusterPool = 1:nclusts
numOfRepeat = 1

t = as.numeric(Sys.time()) #measure time of calculation
results <- RunCcubePipeline(ssm = dat,
                            numOfClusterPool = numOfClusterPool,
                            numOfRepeat = numOfRepeat,
                            runAnalysis = T,
                            runQC = T)
t = as.numeric(Sys.time())-t #store runtime
print(paste('runtime:', t))

summary = table(results$ssm$ccube_ccf_mean) #CCF and #mutations
cfs = as.numeric(names(summary))*100 #extract CCF data
sizes = as.numeric(summary) #extract #mutations

use_python('/local/data/public/kird/anaconda3/bin/python', required=T)
source_python('plot_data.py') #import plotting function
plot_data(cfs, sizes, dirname = outdir, sim=sim) #plot data



#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Code for generating read data corresponding to the tree in assignemnt 1
"""

import numpy as np
import matplotlib.pyplot as plt
from numpy.random import negative_binomial, binomial
from scipy.stats import linregress
import random

#'Type' specifies which type of mutation to simulate
Type = 'after'
#Type = 'major'
#Type = 'minor'
random_type = True #pick a type at random
save = False #save figures

#parameters for negative binomial distribution
```

```
pbin = 0.3333
nbin = 50
#print(np.mean(negative_binomial(nbin, pbin, 10000)))
#print(np.var(negative_binomial(nbin, pbin, 10000)))
#print(negative_binomial(nbin, pbin, 10))

pi0 = 0.2 #healthy cell fraction
pis = [0.2, 0.2, 0, 0, 0.04, 0.12, 0.12, 0.12]
taus = [1, 0.75, 0.50, 0.30, 0.20, 0.15, 0.15, 0.15] #CCFs

nmuts = [5, 11, 3, 6, 6, 1, 5, 3]
varcounts, refcounts, fs, ccfs = [], [], [], [] #store data

Nsnv = 40
maj_cn = [1,2,3,4] # major copy numbers
maj_ps = [0.30, 0.30, 0.2,0.2] #probabilities
min_cn = [0,1,2] # minor copy numbers
min_ps = [1/4, 1/2, 1/4] # probabilities

#simulate copy number states
tmp1 = np.random.choice(maj_cn, Nsnv, p=maj_ps, replace = True)
tmp2 = np.random.choice(min_cn, Nsnv, p=min_ps, replace = True)
tots = tmp1 + tmp2
profile = [tmp1, tmp2, tots]


Write = True #write to file for future use
if Write:
    with open('PyClone_simdeep.tsv', 'w') as f:
                f.write('mutation_id\t'+
                        'ref_counts\t'+
                        'var_counts\t'+
                        'normal_cn\t'+
                        'minor_cn\t'+
                        'major_cn\n')

    with open('Ccube_simdeep.tsv', 'w') as f:
                f.write('mutation_id\t'+
                        'ccf_true\t'+
                        'minor_cn\t'+
                        'major_cn\t'+
                        'total_cn\t'+
                        'purity\t'+
                        'normal_cn\t'+
                        'mult_true\t'+
                        'vaf\t'+
                        'total_counts\t'+
                        'var_counts\t'+
                        'ref_counts\n')

nsite = -1
fifty = False#True
deep = True
if deep: nbin = 10*nbin

for i, tau in enumerate(taus):
    for j in range(nmuts[i]): #generate read counts for each site
```

```
nsite += 1
if random_type:
    if tmp2[nsite] == 0:
        ps = [0.5, 0.5, 0] #cannot observe what's not there
    else:
        ps = [0.5, 0.25, 0.25]
    Type = np.random.choice(['after', 'major', 'minor'], p = ps )
if Type == 'after':
    nR = tots[nsite]
    nV = tots[nsite]
    nB = 1 #single allele
elif Type == 'major':
    nR = 2
    nV = tots[nsite]
    nB = tmp1[nsite] #minor mutation
elif Type == 'minor':
    nR = 2
    nV = tots[nsite]
    nB = tmp2[nsite] #major mutation
else: print('something wrong')

if fifty: #only seventy % CNVs
    if random.random() < 0.5:
        nR = 2
        nV = 2
        nB = 1

#VAF
p = (1-pi0)*tau*nB / (pi0*2 + (1-pi0)*(1-tau)*nR + (1-pi0)*tau*nV)
r = negative_binomial(tots[nsite]*nbin,pbin) #total reads;
R = binomial(r, p) #variant count
print(R)
refcount = r-R #reference allele
varcounts.append(R) #store data
refcounts.append(refcount)
fs.append(p) #variant allele frequency
ccfs.append(tau)

if Write:
    with open('PyClone_simdeep.tsv', 'a') as f:
        f.write(str(nsite)+'\t'+
                str(refcount)+'\t'+
                str(R)+'\t'+
                str(2)+'\t'+
                str(tmp2[nsite])+'\t'+
                str(tmp1[nsite])+'\n')

    with open('Ccube_simdeep.tsv', 'a') as f:
        f.write(str(nsite)+'\t'+
                str(tau)+'\t'+
                str(tmp2[nsite])+'\t'+
                str(tmp1[nsite])+'\t'+
                str(tots[nsite])+'\t'+
                str(0.80)+'\t'+
                str(2)+'\t'+
                str(nB)+'\t'+
                str(p)+'\t'+
```

```
                        str(r)+'\t'+
                        str(R)+'\t'+
                        str(refcount)+'\n')

#plot results
plt.figure(figsize = (4,3), dpi=120)
plt.bar(range(40), fs, align='edge')
plt.ylim(0,1.02)
plotmuts = np.cumsum([0]+nmuts)
for i in range(0, len(taus)):
    plt.plot([plotmuts[i], plotmuts[i+1]], [taus[i], taus[i]], 'k:')
plt.xlabel('locus')
plt.ylabel('VAF')
if save:
    plt.savefig(Type+'_bylocus.png')
plt.show()


plt.figure(figsize = (4,3), dpi=120)
s, i, r, p, std = linregress(ccfs, fs)
plt.plot([0, 1], np.array([0,1])*s+i, 'b:' )
plt.plot(ccfs, fs, 'bx', markersize=4)
plt.xlim(-0.05,1.05)
plt.ylim(-0.05,1.05)
plt.xlabel('CCF')
plt.ylabel('VAF')
plt.legend([str(np.round(s,2))+' CCF + '+str(np.round(i,2))+'   r='+str(np.round(r,2))])
if save:
    plt.savefig(Type+'_ccf_vaf.png')
plt.show()



#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Code for generating CCF plots for metastatic trees
"""

import matplotlib.pyplot as plt

#CCFs for reference tree
ccfs01 = [100, 75, 50, 30, 20, 15, 15, 15, 0, 0, 0 ]
ccfs02 = [100, 75, 50, 30, 20, 15, 15, 15, 0, 0 ]

#clonal colours
cols1 = ['g', 'b', 'y', 'c', 'm', '#00FF00', '#9933FF', '#FF66FF', 'r', 'r', 'r' ]
cols2 = ['g', 'b', 'y', 'c', 'm', '#00FF00', '#9933FF', '#FF66FF', 'r', 'r' ]

#CCFs for metastatic tumors
ccfs1 = [100, 0 , 0 , 0 , 0 , 0 , 0 , 0 , 75, 30, 20]
ccfs2 = [100, 100, 100, 0, 100, 0, 0, 0, 60, 30]

#tree 1
plt.figure(figsize = (3,3))
plt.scatter(ccfs01, ccfs1, color=cols1, alpha=0.8, s = 50)
plt.xlabel('primary')
```

```
plt.ylabel('metastasis')
plt.savefig('meta_early.png', dpi=120, bbox_inches = 'tight')
plt.show()

#tree 2
plt.figure(figsize = (3,3))
plt.scatter(ccfs02, ccfs2, color=cols2, alpha=0.8, s=50)
plt.xlabel('primary')
plt.ylabel('metastasis')
plt.savefig('meta_late.png', dpi=120, bbox_inches = 'tight')
plt.show()
```