‘Fragmentation duplicates’: Sweave

Andy Lynch and Mike Smith

March 29, 2016

Contents

1 Introduction: The LynchSmithEldridgeTavareFragDup Package

1.1 This Sweave

After offering guidance on using the package, this Sweave generates the tables and data-driven figures displayed in the paper ‘A method for the empirical correction of estimated PCR duplication rates, with applications’ by Lynch, Smith, Eldridge and Tavaré on behalf of the OCCAMS Consortium. It also provides greater information on some points of detail. It is dependent on the *Bioconductor* *BiocStyle* package.

This Sweave is part of the *R* package *FragmentationDuplicates* that provides functionality for the methods described in that paper.

1.2 The Data

The contents of the extdata folder in the package are:

```
> list.files(system.file("extdata", package="FragmentationDuplicates"))
```

```
 [1] "HetSNPDups.txt"              "HetSNPDupsByLane.txt"       "HetSNPDupsHighGC.txt"
 [4] "HetSNPDupsLowGC.txt"         "HetSNPTableSingleEnd.tsv"   "HetSNPTableTumourAABB.tsv"
 [7] "HetSNPTableTumourAB.tsv"     "HSTtest.tsv"                "ISDs"
[10] "masks"                       "Picard"                     "SNPstoextract.txt"
[13] "Tumour"                      "WeaverSuppTable1.txt"
```

’WeaverSuppTable1.txt’ reproduces data from the first Supplementary Table of Weaver et al. (2014) *Nature Genetics* **46**, 837-843, and describes the samples used to illustrate this manuscript.

The files ’HetSNPDups.txt’, ’HetSNPDupsByLane.txt’, ’HetSNPDupsHighGC.txt’, ’HetSNPDupsLowGC.txt’, ’HetSNPTableSingleEnd.tsv’, ’HetSNPTableTumourAABB.tsv’, ’HetSNPTableTumourAB.tsv’, ’HSTtest.tsv’ give tables of allele patterns from duplicate-fragment-sets for various BAM file and SNP combinations (the last file in that list being an artificial case) and are used through this vignette.

’SNPstoextract.txt’ gives details of the SNPs described in Section 5.

The contents of ’masks’ and ’Picard’ detail the numbers of reads and duplicates within defined regions of the genome as discussed in Section 6, while the contents of ’Tumour’ are used in Section 11. ’ISDs’ contains details of the insert size distributions for the samples we are considering in this study.

1.3 The Functions

The library defines several functions including the following key ones:

- `processBAMfordups`: a function to process a BAM file (or BAM files) and produce a table detailing the allele patterns of all observed duplicate fragment-sets. More details are given in the next section.
- `processduptable`: a function that processes the output of `processBAMfordups` and returns estimates of the duplicate rates broken down into PCR and fragmentation duplicates. More details are given in the next section.
- `genPL`: Generate the lists of all partitions of M objects.
- `genCoefs`: Generates the coefficients associated with partitions of objects.

1.4 Preparations

There are a few steps that we must take before we begin.

```
> library("xtable")
> library("Rsamtools")
> library("BSgenome.Hsapiens.UCSC.hg19")
> EDpath <- system.file("extdata", package="FragmentationDuplicates")
```

2 General use of the package

There are two primary functions provided to the user in this package. `processBAMfordups` takes an input BAM file and returns a table of allele patterns observed in sets of duplicate reads, while `processduptable` takes such a table and makes inference about the duplication rates.

2.1 `processBAMfordups`

This function takes as its arguments a list of BAM files to process, the path in which to find them, and a list of locations at which to look for heterozygous SNPs. The BAM files must be indexed and have had duplicate reads marked (by e.g. Picard). Other parameters specifying the format of the list of locations, and the criteria by which to accept a locus for use in the analysis can also be set.

The usage is as follows:

```
> duptable<-processBAMfordups(bamfilelist,path,snplist)
```

The output is a table, the first three columns of which are set as 1) The number of loci deemed to be heterozygous SNPs and therefore of use, 2) the number of well-mapped reads covering those loci and, 3) the number of duplicates at those loci in which allele patterns could be examined. Subsequent columns give counts for observed allele patterns (the pattern indicated by the column name), while the final column indicates the count of reads that had an inconsistent allele call at the relevant locus.

2.2 `processduptable`

The second key function takes the output of processBAMfordups and processes it to estimate the fraction of duplicates attributable to fragmentation.

```
> processduptable(duptable,partitionList,coefficientList)
```

This outputs a table with a) and estimate of the duplicate rate (but note there will probably be an estimate of this available to the user that is based on more data than are made use of here), b) the proportion of duplicates that are attributable to coincidental fragmentation, c) the resulting estimated PCR duplicate rate, and d) the resulting estimated fragmentation duplicate rate.

3 Methods

The manuscript that this package accompanies is concerned with separating the observed duplicate rate into a proportion due to PCR duplication of the same original molecule P_D, and the proportion due to identical fragmentation of two distinct molecules $F_D = 1 - P_D$. To do this we make use of sites of heterogenous SNPs where we can (at least some of the time) distinguish the two scenarios.

3.1 Case 1: A pair of duplicate fragments

Recall that if we have only observe duplicate fragments in pairs then, if we observe counts of N_{AA} fragment pairs reporting the same nucleotide, and N_{AB} reporting different nucleotides, the estimate of P_D is

$$P_D = 1 - 2 \times N_{AB}/N. \tag{1}$$

3.2 Case 2: More than two duplicate fragments

Recall that in this case we estimate P_D by maximizing the total log-likelihood

$$l(P_D) = \sum_M l_M(P_D). \tag{2}$$

where

$$l_M(P_D) = \sum_{k=1}^{Q_M} N(AP_k) \log \Pr(AP_k \mid P_D), \tag{3}$$

and

$$\Pr(AP_k \mid P_D) = \sum_i \Pr(AP_k \mid \text{PART}_i) \Pr(\text{PART}_i \mid P_D). \tag{4}$$

In these equations, M is the size of our set of duplicate fragments, PART_i is the i^{th} potential partition of the duplicate fragments into the original molecules, AP_k is the k^{th} observable allele pattern, and $N(AP_k)$ is the number of times that the allele pattern has been observed.

A worked example for M=4 is given in the main manuscript. An example for M=5 appears in Figure 1.

The probabilities $\Pr(\text{PART}_i \mid P_D)$ are all of the form $aP_D^bF_D^c$ where a, b and c are to be determined. b is clearly the number of PCR duplicates in the partition, and $c = M - b - 1$.

The calculation of a is not as obvious. Given a partition of the M fragments into N molecules that provide f_1,f_2,...,f_N fragments in turn ($f_i > 0 \forall i, \sum_i f_i = M$), we denote as ν_j the number of molecules contributing j fragments $1 \leq j \leq M$. The value of a is then calculated as $a = N!/\prod_j (\nu_j!) = \left(\sum_j \nu_j\right)!/\prod_j (\nu_j!)$.

To give a concrete example: When there are four fragments ($M = 4$) partitioned amongst three molecules ($N = 3$) such that $f_1 = 2$, $f_2 = 1$, and $f_3 = 1$, then $\nu_1 = 2$ and $\nu_2 = 1$ ($\nu_j = 0 \forall j > 2$). The value of a is then $3!/(2!1!) = 3$.

When M=2 the partition probabilities are not controversial, being P_D if a PCR duplicate is present and F_D if two molecules are present. These probabilities sum to 1 as required. In each case we can consider the addition of a third fragment (a PCR duplicate with probability P_D or a fragmentation duplicate with probability F_D). If the new fragment is a PCR duplicate, then it may be a duplicate of any of the existing molecules with equal probability and so the probabilities are shared accordingly. The recursive calculation of these values up to $M = 4$ is illustrated in Figure 2.

3.3 Proof that $a = \left(\sum_j \nu_j\right)!/\prod_j (\nu_j!)$

Given the recursive nature in which we have discussed the calculation of a, it is no surprise that the proof is by induction

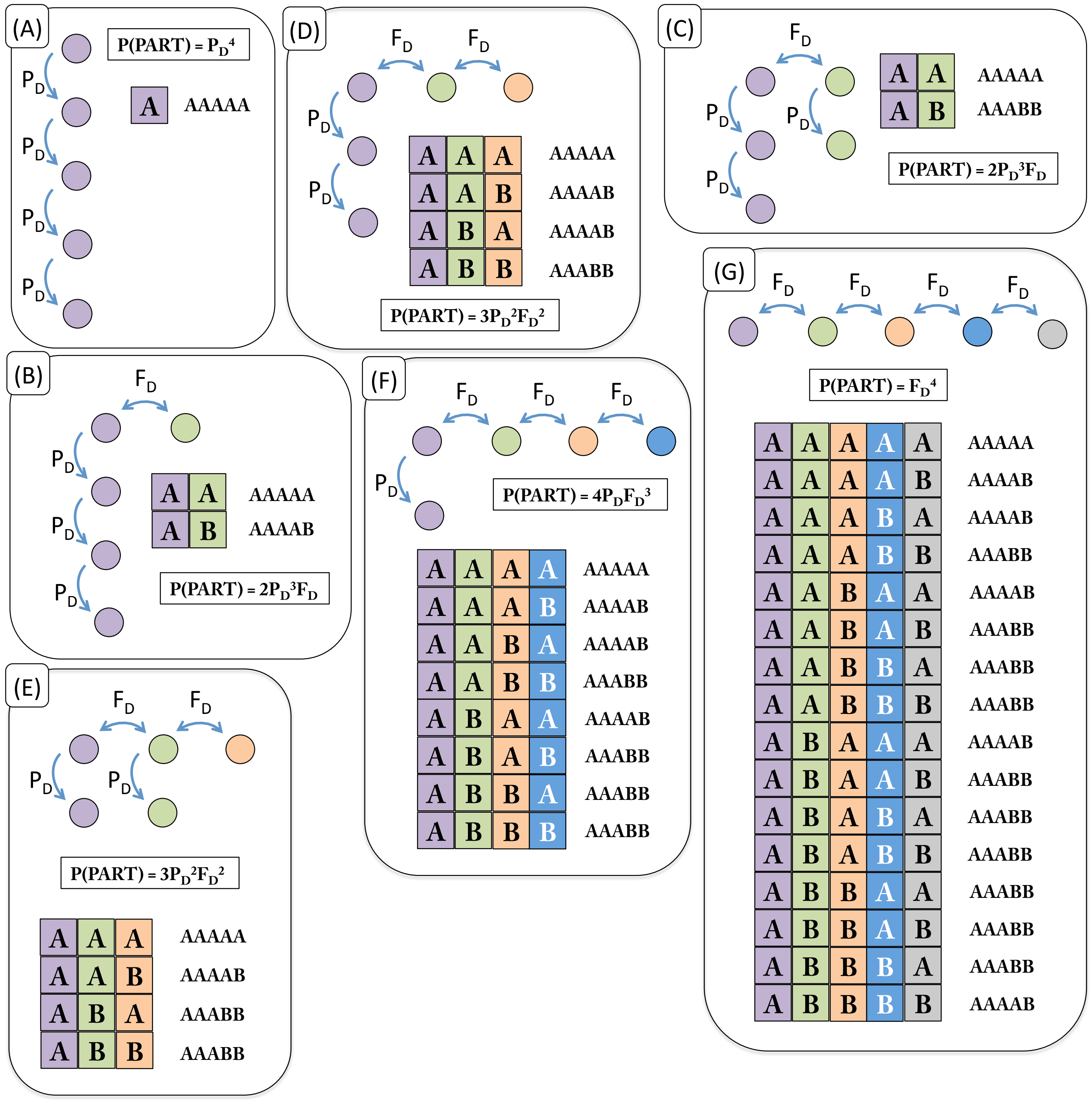

Figure 1: Worked example calculation of $\Pr(AP_k \mid P_D)$ for $M = 5$. There are seven identifiably distinct partitions of fragments into molecules. In panel A we have four PCR duplicates of a single fragment. In panel B there are two distinct duplicate fragments, one of which has three PCR duplicates, while panel C has two distinct duplicate fragments one of which has one PCR duplicate and one of which has two PCR duplicates. In panel D, there are three distinct duplicate fragments, one of which has two PCR duplicates and in panel E there are again three distinct fragments, two of which have a PCR duplicate. Panel F depicts the case where there are four distinct molecules, one of which has a PCR duplicate, and in panel G there are five distinct molecules with no PCR duplicates. The probabilities of the seven possible partitions $\Pr(\text{PART}_i \mid P_D)$ are shown, and for each partition the proportions in which the three possible observable allele patterns (AAAAA, AAAAB, AAABB) will arise are illustrated, allowing calculation of $\Pr(AP_k \mid P_D)$ For example $\Pr(\text{AAABB} \mid P_D) = P_D^3 F_D + \frac{3}{4} P_D^2 F_D^2 + \frac{3}{2} P_D^2 F_D^2 + 2 P_D F_D^3 + \frac{10}{16} F_D^4$

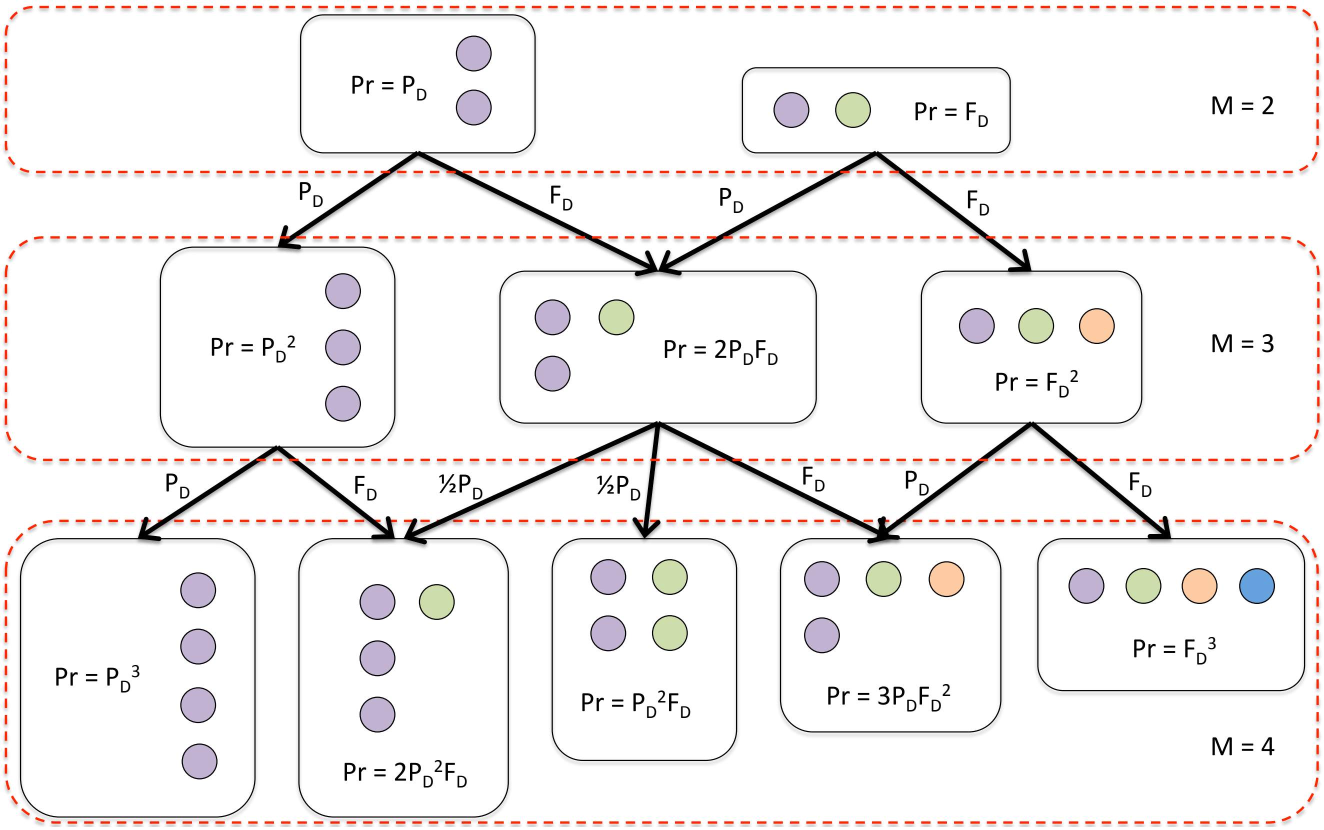

Figure 2: Recursive calculation of partition probabilities

3.3.1 The base case (M = 2)

If M = 2, we either have the left-hand case in Figure 2 where two fragments from the same starting molecule are observed (and so $\nu_2 = 1, \nu_j = 0 \forall j s.t. j \neq 2$), or we have the right hand case where one fragment is observed from each of two original molecules (and so $\nu_1 = 2, \nu_j = 0 \forall j s.t. j \neq 1$).

In the first case, $a = 1!/1! = 1$ and in the second $a = 2!/2! = 1$ as required.

3.3.2 The assumption ($M = G - 1, G > 2$)

We assume that for all partitions where $M = G - 1$, that the relationship

$$a = \left(\sum_j \nu_j\right)! / \prod_j (\nu_j!) \tag{5}$$

holds.

3.3.3 The inductive step ($M = G$)

We assume that our partition of G duplicates is represented by the vector $(\nu_1, \nu_2, \nu_3, ...)$. We view the set of G duplicates as having arisen from our having at some point had a set of $G - 1$ fragments and then sequencing one more. We must distinguish between the two cases: 1) where the new duplicate in the set is the first from a previously unseen molecule (only possible if $\nu_1 > 0$), and 2) where the new duplicate is a further PCR duplicate from a previously seen molecule.

Case 1: A new molecule If we have observed a new molecule with our Gth fragment then the previous set of $G-1$ duplicates must have been represented by the vector $(\nu_1 - 1, \nu_2, \nu_3, ...)$. Clearly this is only possible if $\nu_1 > 0$ and, since observing a new molecule in this situation will always result in our observed partition, the full coefficient is inherited from the previous set (there will of course be a factor of F_D as well).

Hence the contribution to a from this case is

$$\frac{\mathbb{I}_{\nu_1>0}\left(\left(\sum_j \nu_j\right)-1\right)!}{(\nu_1-1)!\prod_{j>1}(\nu_j!)} \tag{6}$$

where $\mathbb{I}$ is the indicator function

Case 2: A PCR duplicate from a previously observed molecule In this case, the previous set of $G-1$ duplicates must have been represented by the vector $(..., \nu_{k-1}+1, \nu_k - 1, ...)$ for some k such that $\nu_k > 0$ and $k > 1$.

The coefficient, a', associated with that vector is

$$a' = \frac{\mathbb{I}_{\nu_k>0}\left(\sum_j \nu_j\right)!}{(\nu_{k-1}+1)!\,(\nu_k-1)!\prod_{j\notin k,(k-1)}(\nu_j!)} \tag{7}$$

but a new PCR duplicate added to that set might create patterns other than the one in which we are interested, so only a portion of the coefficient makes a contribution to our estimate of a. It would only have led to our observed pattern if the PCR duplicate had been of a molecule of which there previously existed $k-1$ copies. The fraction of the coefficient, a', that contributes to our value of a (not withstanding a factor P_D) is therefore the proportion of molecules of which there were previously $k-1$ copies: $(\nu_{k-1}+1)\,/\left(\sum_j \nu_j\right)$.

The additive contribution to a for this value of k is therefore

$$\frac{(\nu_{k-1}+1)}{\sum_j \nu_j}\frac{\mathbb{I}_{\nu_k>0}\left(\sum_j \nu_j\right)!}{(\nu_{k-1}+1)!\,(\nu_k-1)!\prod_{j\notin k,(k-1)}(\nu_j!)} \tag{8}$$

and in total the contributions from this second case are

$$\sum_{k>1}\left(\frac{(\nu_{k-1}+1)}{\sum_j \nu_j}\frac{\mathbb{I}_{\nu_k>0}\left(\sum_j \nu_j\right)!}{(\nu_{k-1}+1)!\,(\nu_k-1)!\prod_{j\notin k,(k-1)}(\nu_j!)}\right) \tag{9}$$

Combining the two cases. If we combine the terms from the two cases as represented by expressions 6 and 9 then we get

$$a = \frac{\mathbb{I}_{\nu_1>0}\left(\left(\sum_j \nu_j\right)-1\right)!}{(\nu_1-1)!\prod_{j>1}(\nu_j!)} + \sum_{k>1}\left(\frac{(\nu_{k-1}+1)}{\sum_j \nu_j}\frac{\mathbb{I}_{\nu_k>0}\left(\sum_j \nu_j\right)!}{(\nu_{k-1}+1)!\,(\nu_k-1)!\prod_{j\notin k,(k-1)}(\nu_j!)}\right) \tag{10}$$

which we can simplify by removing the terms in the first fraction on the right hand side

$$a = \frac{\mathbb{I}_{\nu_1>0}\left(\left(\sum_j \nu_j\right)-1\right)!}{(\nu_1-1)!\prod_{j>1}(\nu_j!)} + \sum_{k>1}\left(\frac{\mathbb{I}_{\nu_k>0}\left(\left(\sum_j \nu_j\right)-1\right)!}{(\nu_{k-1})!\,(\nu_k-1)!\prod_{j\notin k,(k-1)}(\nu_j!)}\right) \tag{11}$$

and can be tidied to

$$a = \frac{\mathbb{I}_{\nu_1>0}\left(\left(\sum_j \nu_j\right)-1\right)!}{(\nu_1-1)!\prod_{j>1}(\nu_j!)} + \sum_{k>1}\left(\frac{\mathbb{I}_{\nu_k>0}\left(\left(\sum_j \nu_j\right)-1\right)!}{(\nu_k-1)!\prod_{j\neq k}(\nu_j!)}\right) \tag{12}$$

adjusting the products to be independent of 1 and k

$$a = \frac{\mathbb{I}_{\nu_1>0}\left(\left(\sum_j \nu_j\right)-1\right)!\;\nu_1!}{(\nu_1-1)!\prod_{j}(\nu_j!)} + \sum_{k>1}\left(\frac{\mathbb{I}_{\nu_k>0}\left(\left(\sum_j \nu_j\right)-1\right)!\;\nu_k!}{(\nu_k-1)!\prod_{j}(\nu_j!)}\right) \tag{13}$$

tidying up the other terms

$$a = \frac{\mathbb{I}_{\nu_1>0}\left(\left(\sum_j \nu_j\right)-1\right)!\;\nu_1}{\prod_{j}(\nu_j!)} + \sum_{k>1}\left(\frac{\mathbb{I}_{\nu_k>0}\left(\left(\sum_j \nu_j\right)-1\right)!\;\nu_k}{\prod_{j}(\nu_j!)}\right) \tag{14}$$

we can now combine everything into one sum over k

$$a = \sum_{k}\left(\frac{\mathbb{I}_{\nu_k>0}\left(\left(\sum_j \nu_j\right)-1\right)!\;\nu_k}{\prod_{j}(\nu_j!)}\right) \tag{15}$$

Moving the terms that are independent of k out of the sum

$$a = \frac{\left(\left(\sum_j \nu_j\right)-1\right)!}{\prod_{j}(\nu_j!)}\sum_{k}\left(\nu_k \mathbb{I}_{\nu_k>0}\right) \tag{16}$$

which equals

$$a = \frac{\left(\left(\sum_j \nu_j\right)-1\right)!}{\prod_{j}(\nu_j!)}\left(\sum_{j}\nu_j\right) \tag{17}$$

and so we get

$$a = \frac{\left(\sum_j \nu_j\right)!}{\prod_{j}(\nu_j!)} \tag{18}$$

which was to be shown.

3.4 Confirming that the two approaches match when M equals 2

Since $M = 2$, we only have two possible allele patterns, and we can write down $\Pr(AP_k|P_D)$ in a straightforward manner.

$$P(AA \mid P_D) = \frac{1}{2}(1 + P_D) \tag{19}$$
$$P(AB \mid P_D) = \frac{1}{2}(1 - P_D) \tag{20}$$

The log-likelihood is then

$$l(P_D) = n_{AA} \log(\frac{1}{2}(1 + P_D)) + n_{AB} \log(\frac{1}{2}(1 - P_D)), \tag{21}$$

and the first derivative is

$$\frac{dl}{dP_D} = \frac{n_{AA}}{(1 + P_D)} + \frac{n_{AB}}{(1 - P_D)}. \tag{22}$$

If we look to find the MLE of P_D,

$$0 = n_{AA}(1 - \hat{P}_D) + n_{AB}(1 + \hat{P}_D), \tag{23}$$

then we find it is equal to

$$\hat{P}_D = \frac{n_{AA} + n_{AB}}{N} = 1 - \frac{2n_{AB}}{N} \tag{24}$$

as required to match equation 1.

3.5 The estimate is (in some sense) well-behaved when M equals 3

The case when $M = 3$ is depicted in Figure 3. In this simple case, there are only two patterns of alleles that can be observed: 'AAA' and 'AAB'.

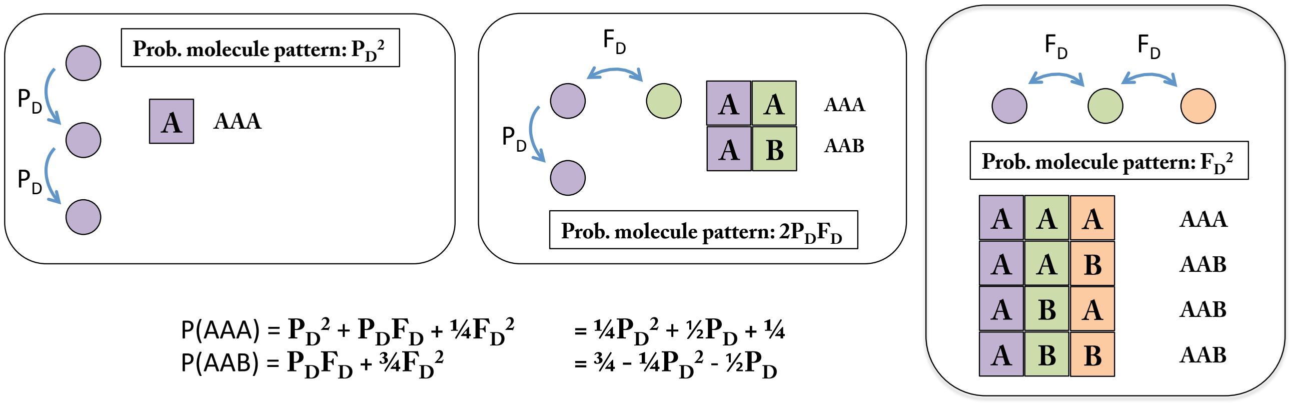
Prob. molecule pattern: P_D^2
P_D
P_D
A
AAA
F_D
P_D
A A AAA
A B AAB
Prob. molecule pattern: $2P_DF_D$
F_D
F_D
Prob. molecule pattern: F_D^2
A A A AAA
A A B AAB
A B A AAB
A B B AAB
$P(AAA) = P_D^2 + P_DF_D + ¼F_D^2 = ¼P_D^2 + ½P_D + ¼$
$P(AAB) = P_DF_D + ¾F_D^2 = ¾ - ¼P_D^2 - ½P_D$

Figure 3: Details of the case when $M = 3$.

The log-likelihood is then given as

$$l_3(P_D) = n_{AAA} \log(P(AAA \mid P_D)) + n_{AAB} \log(P(AAB \mid P_D)). \tag{25}$$

Since this is of the form

$$l = n \log f(P_D) + m \log(1 - f(P_D)), \tag{26}$$

we can see that the first derivative is of the form

$$nf'(P_D) - (n + m)f'(P_D)f(P_D) \tag{27}$$

So,

$$\begin{aligned}\frac{dl}{dP_D} =& n_{AAA}(\frac{1}{2}P_D + \frac{1}{2}) - \\ & (n_{AAA} + n_{AAB})(\frac{1}{2}P_D + \frac{1}{2})(\frac{1}{4}P_D^2 + \frac{1}{2}P_D + \frac{1}{4})\end{aligned} \tag{28}$$

and since P_D cannot equal -1, we have

$$0 = \hat{P}_D^2 + 2\hat{P}_D + 1 - \frac{4n_{AAA}}{n_{AAA} + n_{AAB}} \tag{29}$$

and

$$\hat{P}_D = -1 \pm 2\sqrt{\frac{n_{AAA}}{n_{AAA} + n_{AAB}}} \tag{30}$$

From Figure 3, we can see that for a large numbers of trios, $N = n_{AAA} + n_{AAB}$, n_{AAA} will be approximately equal to $N(\frac{1}{4}P_D^2 + \frac{1}{2}P_D + \frac{1}{4})$, which if substituted into the equation for $\hat{P}_D$ gives the estimate $\hat{P}_D = P_D$.

3.6 Inconsistency with a naive approach when M=3

Were we to adopt a naive approach, we might simply deal with cases where $M > 2$ by discarding all but 2 fragments in the set at random.

If $M = 3$ we would expect to see

$$n_{AA} = n_{AAA} + \frac{1}{3}n_{AAB} \tag{31}$$

$$n_{AB} = \frac{2}{3}n_{AAB} \tag{32}$$

and so the estimate of P_D will be

$$\hat{P}_D = 1 - \frac{4n_{AAB}}{3N} \tag{33}$$

which in general will not equal the estimate obtained from using all of the data (equation 30).

4 Simulation

4.1 Artificial data

We can demonstrate the performance of the method using four artificial data sets that implememnt some of the results just seen

	NoSNPs	NoReads	NoDups	AA	AB	AAA	AAB	NoN
Test1	1000.00	80000.00	4000.00	2700.00	300.00	0.00	0.00	0.00
Test2	1000.00	80000.00	4000.00	3800.00	200.00	0.00	0.00	0.00
Test3	1000.00	80000.00	4000.00	0.00	0.00	1728.00	147.00	0.00
Test4	1000.00	80000.00	4000.00	1777.00	98.00	0.00	0.00	0.00

Test 1 is an $M = 2$ case, engineered to have a fragmentation duplicate fraction of $2 \times 300/3000 = 0.2$.

Test 2 is an $M = 2$ case engineered to have a fragmentation duplicate fraction of $2 \times 200/4000 = 0.1$.

Test 3 is an $M = 3$ case engineered to have a fragmentation duplicate fraction of $1 - 2 \times \sqrt{\frac{1728}{1728+147}} = 0.08$.

Test 4 is the same case using a naive down-sampling to $M = 2$ (i.e. one third of AABs end up as AA, the rest as AB). Following the results of the previous section, we do not expect this to return the correct value.

We now read in this table. Then, to prepare for the probability calculations, we generate the list of sets of partitions possible with each value of M (the number of fragments in a duplicate set).

```
> HSTtest<-read.delim(paste(EDpath,"/HSTtest.tsv",sep="",collapse=""),header=T,as.is=T)
> PL<-genPL(10)
> CL<-genCoefs(PL)
> processduptable(HSTtest,PL,CL)
```

```
  ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
1               5        0.200           4.000       1.000
2               5        0.100           4.500       0.500
3               5        0.080           4.600       0.400
4               5        0.105           4.475       0.525
```

We see that after applying the `processduptable` to this artificial data set, the method produces the correct solution in each case.

4.2 Insert Size Distributions

To demonstrate that the approach works in a controlled scenario, we use data simulated from the following assumptions:

- Read start sites are uniformly distributed along a genome
- Insert size distributions are sampled from real data independently of the start sites
- PCR duplicate numbers for each individual read are IID Poisson distributed

The size selection step is crucial to the chances of seeing a fragmentation duplicate. Comparing two extreme insert size distributions from our example data set, we can see that (in the absence of any dependence on starting position etc.) the probability of two fragments sampled from the library being the same length differs by a factor of approximately 2.7 $(0.020$ vs $0.007)$.

```
> ISDnarrow<-read.delim(paste(EDpath,"/ISDs/SS6003110.insertsizedist.txt",sep="",collapse=""), as.is=T, hea
> ISDwide<-read.delim(paste(EDpath,"/ISDs/SS6003304.insertsizedist.txt",sep="",collapse=""), as.is=T, heade
> ISDnarrow<-ISDnarrow[-1000,]
> ISDwide<-ISDwide[-1000,]
> set.seed(31415927)
> table(sample(ISDnarrow[,1], 100000, prob=ISDnarrow[,2], replace=T) == sample(ISDnarrow[,1], 100000, prob

  FALSE    TRUE
0.97986 0.02014

> table(sample(ISDwide[,1], 100000, prob=ISDwide[,2], replace=T) == sample(ISDwide[,1], 100000, prob=ISDwi

  FALSE    TRUE
0.99255 0.00745
```

Details of the samples that give rise to these distributions are given in Section 6, but it should be noted that the starting tissues and consequently the nucleic acid preparation methods were different. Figure 4 illustrates the difference in distributions.

```
> plot(ISDnarrow[,1],ISDnarrow[,2]/sum(ISDnarrow[,2]),xlim=c(200,500),xlab="Insert size",ylab="Density",typ
> points(ISDwide[,1],ISDwide[,2]/sum(ISDwide[,2]),col="red",type="l",lwd=3)
> text(375,0.025,"SS6003110")
> text(400,0.01,"SS6003304",col="red")
```

If we simulate from these two distributions, we see the effect they have on the fragmentation duplicate proportion.

```
> set.seed(978342)
> HSTsim0<-simreads(depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,upperlength=1000,maxM=5)
> HSTsim0<-rbind(HSTsim0,simreads(depth=60,pcrdup=0.06,realISD=ISDwide,lowerlength=0,upperlength=1000,maxM=
> PDTsim0<-processduptable(HSTsim0,PL,CL)
> PDTsim0

     ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
[1,]        6.190174        0.048        5.893046   0.2971284
[2,]        6.195532        0.018        6.084013   0.1115196
```

We see that the proportion of fragmentation duplicates is substantially lower when using the wider insert size distribution. With the ratio of fragmentation driven duplicate rates also being approximately 2.7.

4.3 Demonstration that the method works

We simulate 1000 heterozygous SNPs, spaced at kilobase intervals. PCR duplication rates increase approximately linearly with depth, so we set them accordingly.

```
> set.seed(62536231)
> HSTsim<-simreads(genlength=1000000,depth=20,pcrdup=0.02,realISD=ISDnarrow,lowerlength=0,upperlength=1000
> HSTsim<-rbind(HSTsim,simreads(genlength=1000000,depth=40,pcrdup=0.04,realISD=ISDnarrow,lowerlength=0,uppe
> HSTsim<-rbind(HSTsim,simreads(genlength=1000000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,uppe
> HSTsim<-rbind(HSTsim,simreads(genlength=1000000,depth=80,pcrdup=0.08,realISD=ISDnarrow,lowerlength=0,uppe
> HSTsim<-rbind(HSTsim,simreads(genlength=1000000,depth=100,pcrdup=0.1,realISD=ISDnarrow,lowerlength=0,uppe
> PDTsim<-processduptable(HSTsim,PL,CL)
> PDTsim

     ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
[1,]        2.208724        0.055        2.087245   0.1214798
[2,]        4.250408        0.059        3.999634   0.2507741
[3,]        6.271941        0.045        5.989704   0.2822373
[4,]        8.365719        0.040        8.031091   0.3346288
[5,]       10.354593        0.041        9.930054   0.4245383

> plot(seq(20,100,20),PDTsim[,1]/seq(2,10,2),ylim=c(0.95,1.15),xlab="Simulated depth of sequencing",ylab="
> abline(h=1,lwd=3)
```

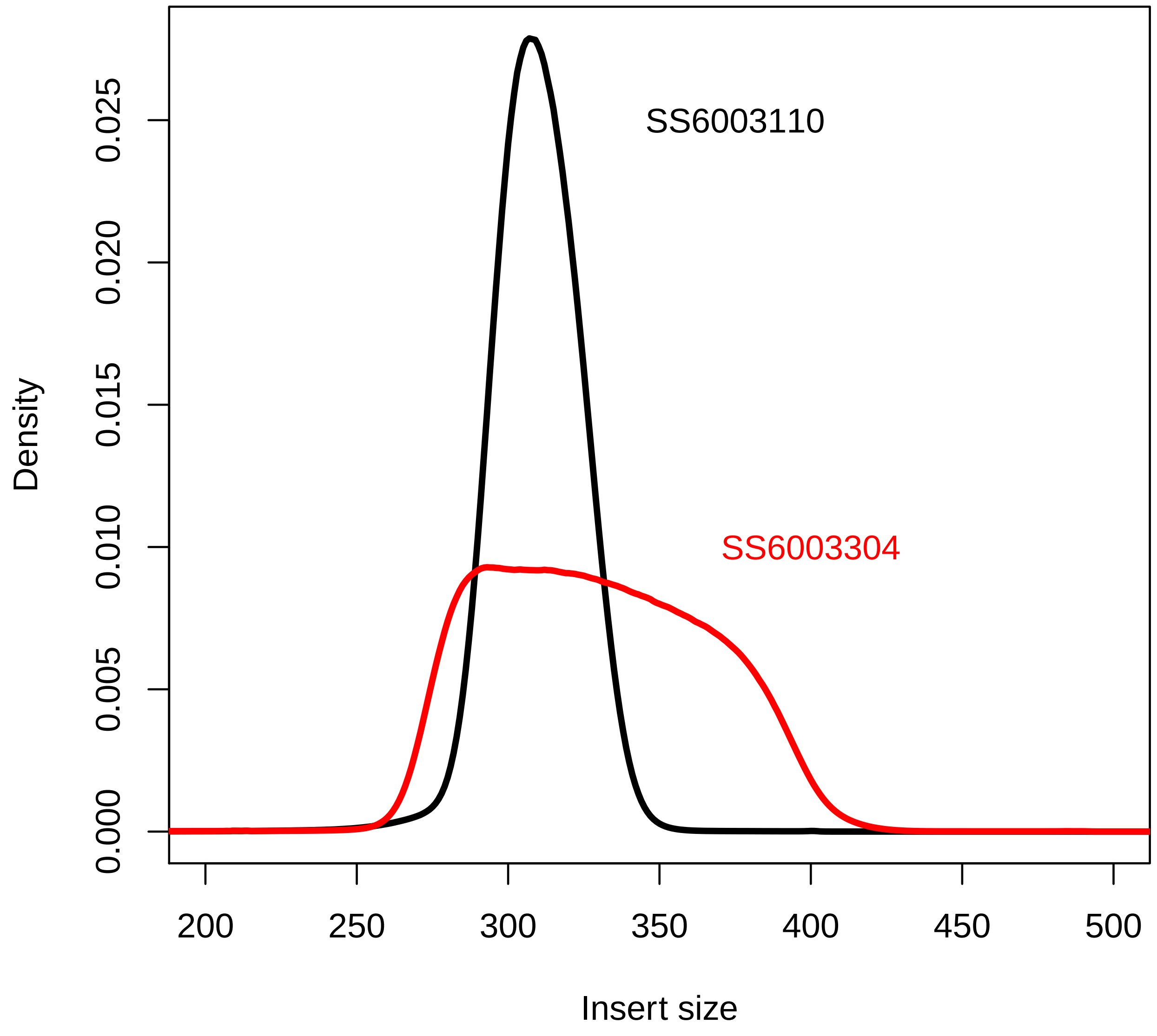

Figure 4: **ISDs.** Examples of extreme insert size distributions.

```
> points(seq(20,100,20),PDTsim[,3]/seq(2,10,2),col="red",pch=16)
> legend("topright",fill=c("black","red"),legend=c("Observed","Corrected"))
```

In figure 5, we see that, while there is naturally some stochasticity at low depth (where only 440 duplicate reads are observed), the corrected rate is notably closer (and indeed close) to the true value.

We won’t run the following code, as it takes too long for the sweave, but the greater depth emphasizes the improvement even more.

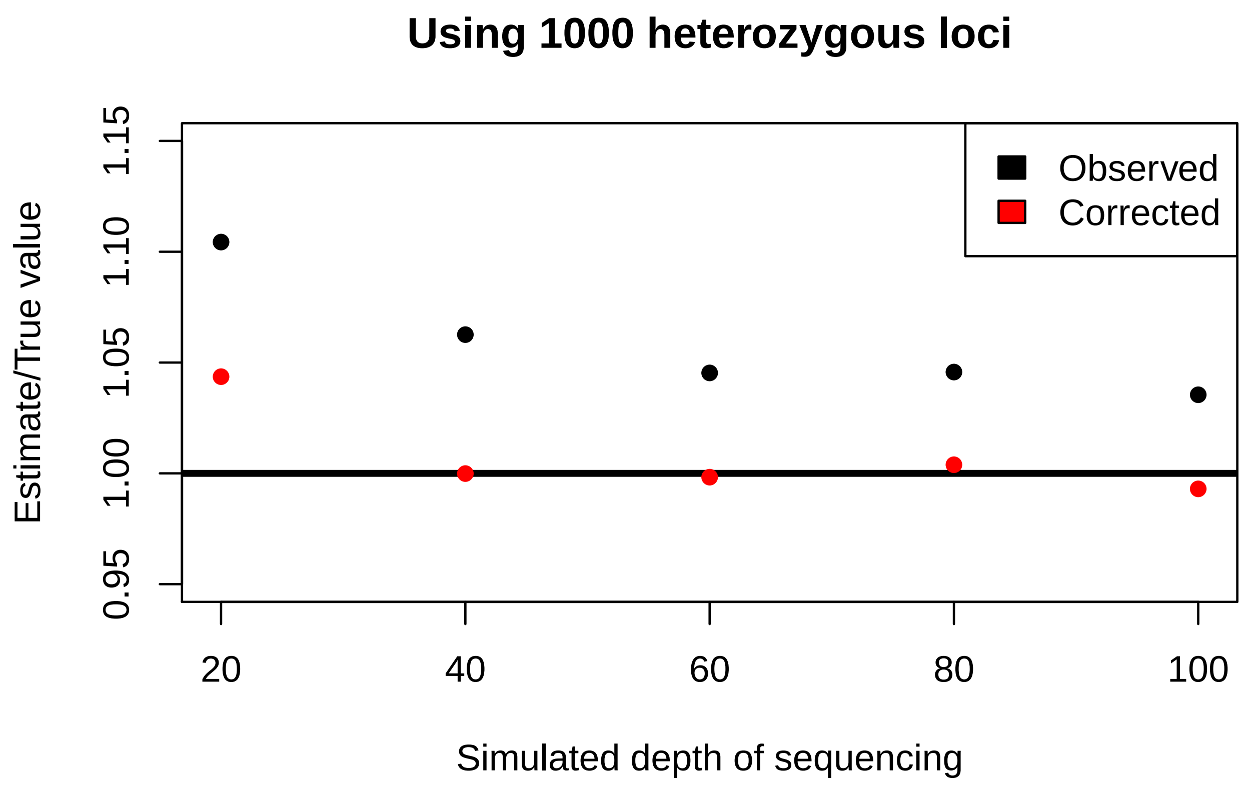

Figure 5: **Performance by depth** Using 1000 loci, the performance across a range of depths is plotted.

```
> set.seed(3502349)
> HSTsim2<-simreads(depth=200,pcrdup=0.2,realISD=ISDnarrow,lowerlength=0,upperlength=1000)
> #     NoSNPs NoReads NoDups    AA   AB   AAA AAB AAAA AAAB AABB AAAAA AAAAB AAABB AAAAAA AAAAAB AAAABB AAAA
> #[1,]    999  198406  40725 30693 323 3925 174  410   23   18    33     2     2      1      1      0
> processduptable(HSTsim2,PL,CL)
> #     ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
> #[1,]        20.52609        0.027        19.97189   0.5542045
```

4.4 How many SNPs are required?

We look now at the effect of changing the number of SNPs being used, for what are otherwise a typical set of parameters. We see that there is some volatility (especially at low values of SNPs), but that even at 500 SNPs the approach is useful. The estimate of the proportion of fragmentation duplicates is quite stable by the time 1000 SNPs are used.

```
> set.seed(5417583)
> HSTsim3<-simreads(genlength=100000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,upperlength=1000
> HSTsim3<-rbind(HSTsim3,simreads(genlength=200000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,up
> HSTsim3<-rbind(HSTsim3,simreads(genlength=300000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,up
> HSTsim3<-rbind(HSTsim3,simreads(genlength=400000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,up
> HSTsim3<-rbind(HSTsim3,simreads(genlength=500000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,up
> HSTsim3<-rbind(HSTsim3,simreads(genlength=750000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,up
> HSTsim3<-rbind(HSTsim3,simreads(genlength=1000000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,u
> HSTsim3<-rbind(HSTsim3,simreads(genlength=1500000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,u
> HSTsim3<-rbind(HSTsim3,simreads(genlength=2000000,depth=60,pcrdup=0.06,realISD=ISDnarrow,lowerlength=0,u
> PDTsim3<-processduptable(HSTsim3,PL,CL)
> PDTsim3
```

```
      ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
 [1,]        5.735913        0.053        5.431909   0.3040034
 [2,]        6.015038        0.031        5.828571   0.1864662
 [3,]        6.576102        0.036        6.339362   0.2367397
 [4,]        6.238972        0.047        5.945740   0.2932317
 [5,]        6.232533        0.053        5.902209   0.3303243
 [6,]        6.345920        0.053        6.009586   0.3363338
 [7,]        6.159110        0.050        5.851155   0.3079555
 [8,]        6.185923        0.043        5.919928   0.2659947
 [9,]        6.328509        0.041        6.069040   0.2594689
```

Plotting the performance of the method with differing numbers of SNPs we see that anything more than 500 heterozygous loci should suffice for a typical depth of sequencing (Figure 6).

```
> plot(HSTsim3[,1],PDTsim3[,1],ylim=c(5,7),,xlim=c(0,2000),pch=16,xlab="Number of Heterozygous SNPs used",
> points(HSTsim3[,1],PDTsim3[,3],pch=16,col="red",type="b",lwd=2)
> legend("topright",fill=c("black","red"),legend=c("Observed","Corrected"))
> abline(h=6,lwd=3)
```

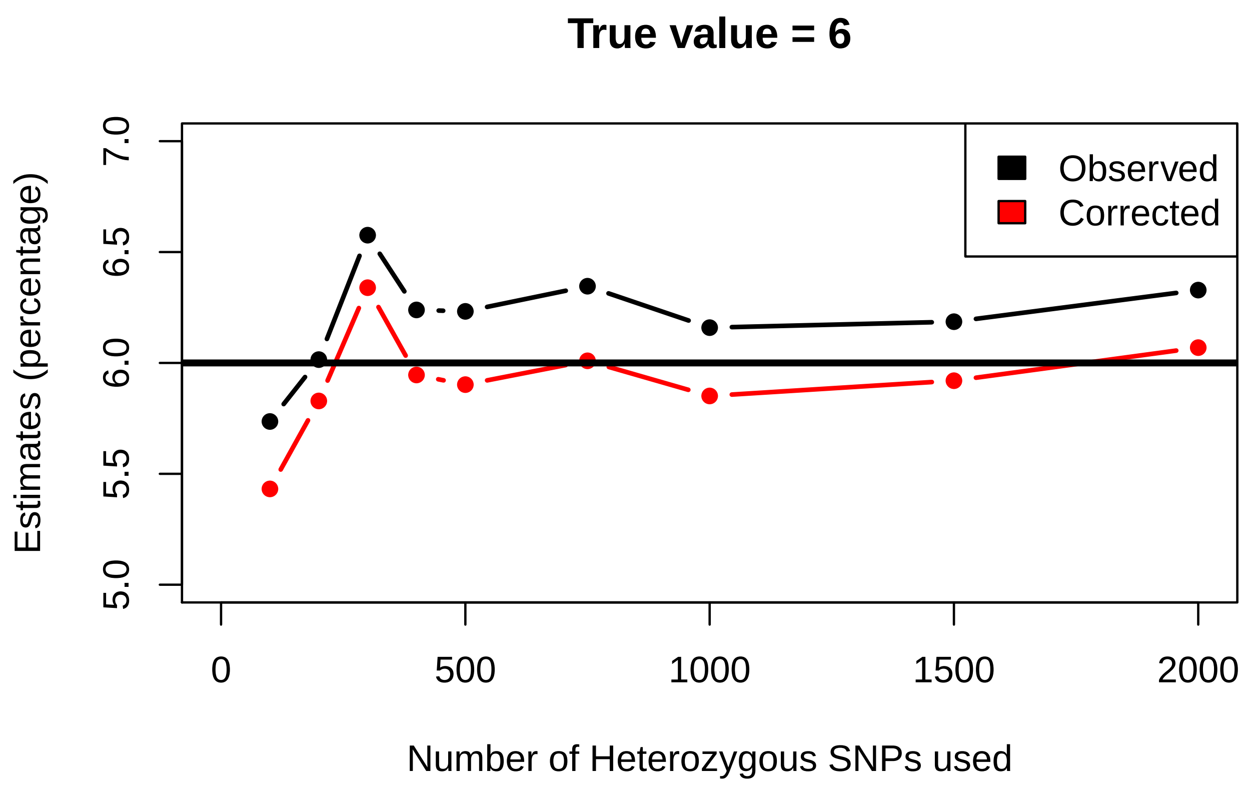

Figure 6: **SNP numbers** We see that at this (typical) depth, the performance of the method becomes stable once 400 SNPs have been used.

The suggestion from the previous two sections is that approximately 1500 duplicate reads are required (whether arising through high depth or a high number of heterozygoius loci).

5 SNPs to extract

5.1 Choosing the SNPs

We investigate $2,500$ common SNPs in anticipation of identifying $1,000$ heterozygous sites for each sample. In this manner we avoid the burden of having to define a bespoke set of sites for each case.

The SNPs were selected from UCSC’s snp138Common table, considering only those validated by the 1000 Genomes Project. We required the reported population minor allele frequency (MAF) of selected SNPs be > 0.49 based on $> 2,500$ observations and that the SNPs were located on one of the 24 ‘regular’ chromosomes. The $2,500$ SNPs were selected at random from all those meeting the criteria.

Despite these criteria, the true MAF can only be biased in one direction from the approximately 0.5 we nominally required, and our sequencing depth and requirements for calling heterozygous SNPs in the data mean that we will fail to identify up to ten percent of heterozygous sites because, by chance, their observed allele frequencies will be too extreme. Additionally, 35 of the randomly selected SNPs were from the sex chromosomes, and are ‘unlikely’ to be heterozygous in our predominantly male patients. In combination, these effects mean that we can anticipate approximately $1,000$ of the SNPs to be heterozygous in each patient.

By choosing a set of SNPs with high MAF, we can guarantee a good number of heterozygous observations in each sample, without the computational burden of defining a bespoke set of sites for each case. Extracting all reads mapping to these locations creates a BAM file that is approximately $10,000$ times smaller than the original and so allows for easy manipulation of large cohorts. We note that such a set of SNPs, and the BAM file they produce, are also useful for activities such as i) detecting sample contamination, ii) detecting sample mix ups, iii) detecting familial/ancestral relationships, and so have utility beyond our purposes in this study.

We now load in the set of SNPs and explore them.

```
> snplist<-read.delim(file.path(EDpath,"SNPstoextract.txt"),as.is=T,header=T)
```

5.2 Representativeness of the SNPs

5.2.1 Distribution across Chromosomes

We can see (Figure 7) that the SNPs are chosen from all chromosomes (Figure 7), and that we are not far from having numbers proportional to chromosome length.

```
> par(mar=c(4.6,4.1,1.6,1.6))
> plot(table(snplist$chrom)[c(1,12,16:22,2:11,13:15,23:24)],
+      ylab="number of SNPs",xlab="chromosome",axes=F)
> axis(2)
> axis(1,labels=c(1:22,"X","Y"),at=1:24,las=2)
> box()
```

5.2.2 GC content

We take the locations of the $2,500$ SNPs in our list and place a window of 500 bases around them, before counting the number of G and C bases in that window.

```
> genome <- Hsapiens
> GCNo<-rep(NA,2500)
> for(i in 1:2500){
+   GCNo[i]<-sum(unlist(strsplit(as.character(substr(genome[[snplist$chrom[i]]],snplist$chromEnd[i]-250,snp
+ }
```

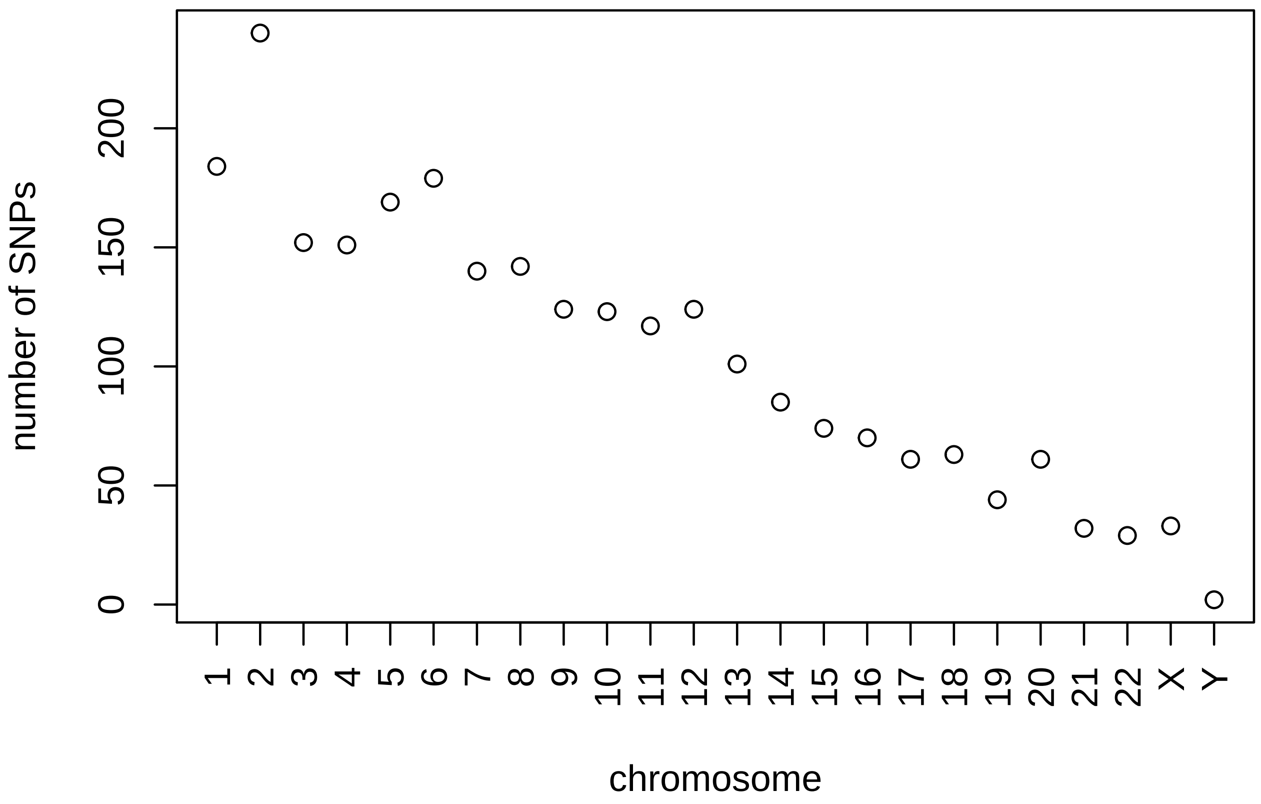

Figure 7: **SNP numbers.** The numbers of SNPs used from each chromosome.

For comparison, we take the human genome (chromosomes 1-22 and X), divide it up into bins of 500 and calculate the proportion of bases that are G or C in each bin.

```
> GCbinned<-NULL
> for(k in 1:23){
+     windowViews <- trim(Views(genome[[k]], start = seq(1, length(genome[[k]]), 500), width = 500))
+     letterFreq <- letterFrequency(windowViews, letters = c("A","C","G","T"))
+     seqGC <- rowSums(letterFreq[,2:3]) / rowSums(letterFreq)
+     GCbinned<-c(GCbinned,seqGC)
+ }
```

We see (Figure 8) that the distribution of GC in our selection is a good representation of the GC content of the genome (with departure only in the final percentile).

```
> par(mfrow=c(1,2))
> hist(GCbinned,freq=F,col=rgb(1,0.5,0.5,0.5),breaks=seq(0,1,0.02),main="Distribution of GC content of gen
> hist(GCNo/500,freq=F,col=rgb(0.5,0.5,1,0.5),breaks=seq(0,1,0.02),add=T)
> legend("topright",fill=c(rgb(1,0.5,0.5,0.5),rgb(0.5,0.5,1,0.5)),legend=c("Genome-wide","Our SNPs"))
> plot(quantile(GCbinned,probs=seq(0.01,.99,0.01),na.rm=T),quantile(GCNo/500,probs=seq(.01,.99,0.01),na.rm
+      pch=20,col="red",xlab="Genome wide 1st - 99th percentiles",
+      ylab="Our SNPs 1st - 99th percentiles",
+      main="Proportion GC Content\nKolmogorov-Smirnov p=0.08")
> for(i in 1:99){
+   GW<-quantile(GCbinned,probs=i/100,na.rm=T)
+   OS<-quantile(GCNo/500,probs=i/100,na.rm=T)
+ lines(c(GW,GW),c(0,OS),lwd=0.5)
+ lines(c(0,GW),c(OS,OS),lwd=0.5)
+ }
> points(quantile(GCbinned,probs=seq(0.01,.99,0.01),na.rm=T),quantile(GCNo/500,probs=seq(.01,.99,0.01),na.r
> abline(0,1,lwd=2)
```

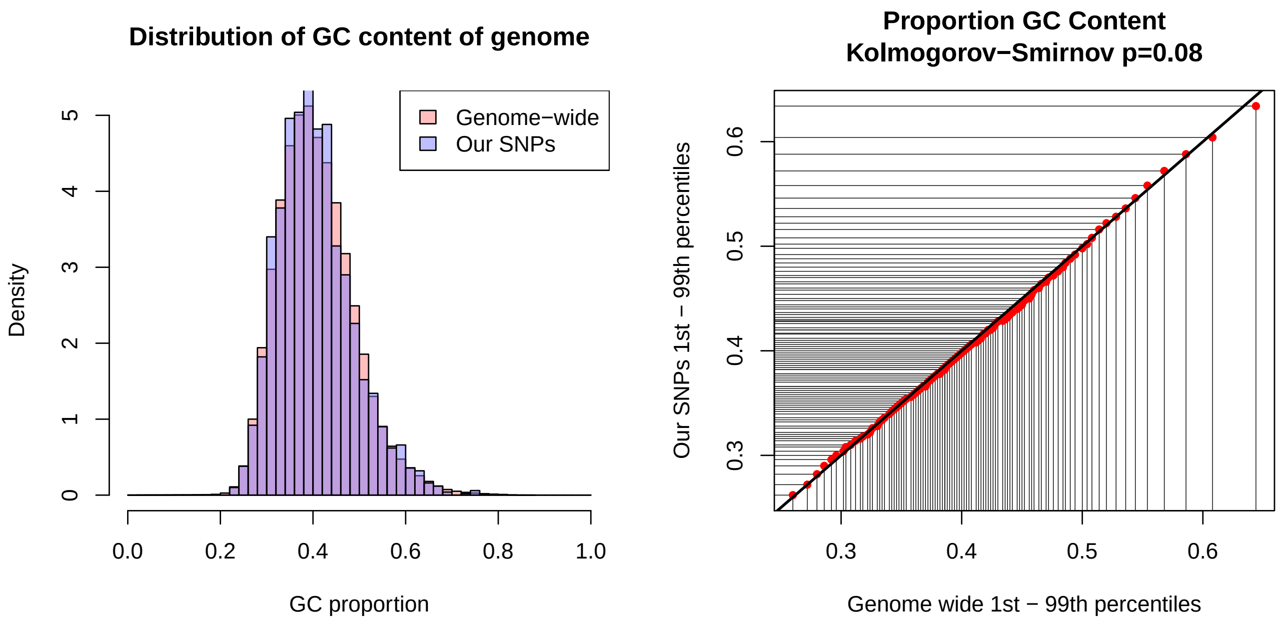

Figure 8: **SNP GC representativeness.** Illustrating that the distribution of GC in our selection is a good representation of the GC content of the genome.

Despite the power to detect deviation with this number of samples, the Kolmogorov-Smirnov test is not significant.

```
> ks.test(GCbinned,GCNo/500)

        Two-sample Kolmogorov-Smirnov test

data:  GCbinned and GCNo/500
D = 0.02552, p-value = 0.07717
alternative hypothesis: two-sided
```

6 Sample information

We are considering twenty-two 'normal' samples from patients forming part of the Oesophageal Adenocarcinoma ICGC study run by the OCCAMS Consortium. The full data for these samples are archived in the European Genome Archive [EGA:EGAD00001000704] and have featured in the paper Ordering of mutations in preinvasive disease stages of esophageal carcinogenesis.

Of the twenty-two samples, twelve were blood, and ten were oesophageal tissue. DNA from oesophageal tissue was extracted using the DNeasy kit (Qiagen) and from blood using the NucleonTM Genomic Extraction kit (Gen-Probe) (according to the manufacturer's instructions).

The sequencing was conducted under contract by Illumina, typically comprising five lanes of 100bp paired end sequencing. The depth of coverage ranges from 57x to 87x with a mean of 68x.

In addition, we use one tumour sample from the same study.

This study is a presentation of analysis methods not a presentation of the sequencing data, and raw data that could identify the patients is not provided. Sufficient data are provided to illustrate the methods presented. Further access is via the Project's Data Access Committee.

6.1 Picard

Picard was applied to the raw bam files, and from this output we extract data on the number of read pairs examined, the numbers of read pairs marked as duplicates, and the number of read pairs marked as optical duplicates. We keep the full results for incorporation into Table 1.

```
> unmaskedP<-read.delim(file.path(EDpath,"Picard", "samples.metrics.txt"),as.is=T)
> readPairsExamined <- readPairDuplicates <- OpticalPairDuplicates <- matrix(ncol=9, nrow=22, dimnames = li
> readPairsExamined[,1]<-unmaskedP[,3]
> readPairDuplicates[,1]<-unmaskedP[,6]
> OpticalPairDuplicates[,1]<-unmaskedP[,7]
```

6.1.1 Generate Table 1

We read in supplementary table 1 from Weaver et al, and extract the information of interest to us.

```
> WeaverSuppTable1<-read.delim(file.path(EDpath,"WeaverSuppTable1.txt"),sep="",header=T,as.is=T)
> WeaverSuppTable1$CaseID<-WeaverSuppTable1$CaseID-1
```

Next we collect the information for Table 1 of the main manuscript, and prepare it for insertion into the LaTeX document.

```
> Table1<-cbind(WeaverSuppTable1[,1],WeaverSuppTable1[,2],WeaverSuppTable1[,12],round(unmaskedP[,3]/100000
> Table1[Table1=="NormalOesophagus"]<-"NormalOes"
> colnames(Table1)<-c("ID","Sex","Tissue","Reads (10^9)","Dup. Rate (%)","Library Size (10^9)")
> xtab1<-xtable(Table1)
> write(print(xtab1,include.rownames = FALSE),file="Table1.txt")
```

We will define four groups based on sex and the origin of the tissue/DNA extraction kit used.

```
> table(WeaverSuppTable1[,2],WeaverSuppTable1[,12])

         Blood NormalOesophagus
  Female     2                2
  Male      10                8

> Group<-1+2*(WeaverSuppTable1[,2]=="Male")+(WeaverSuppTable1[,12]=="Blood")
> GroupN<-c("Female_Tissue","Female_Blood","Male_Tissue","Male_Blood")[Group]
```

ID	Sex	Tissue	Reads (10^9)	Dup. Rate (%)	Library Size (10^9)
3108	Female	Blood	1.15	6.15	9.03
3110	Male	Blood	1.16	7.86	7
3112	Male	Blood	1.29	6.94	8.86
3114	Male	Blood	0.88	5.05	8.46
3116	Male	NormalOes	0.93	11.92	3.59
3118	Male	NormalOes	1.16	14.49	3.61
3120	Male	Blood	1.11	4.01	13.55
3124	Male	Blood	0.99	8.07	5.81
3128	Male	Blood	0.97	4.8	9.8
3130	Male	Blood	1.03	8.34	5.83
3132	Male	Blood	1.02	6.03	8.15
3134	Male	Blood	1.1	4.38	12.21
3136	Male	Blood	0.99	5.76	8.3
3148	Female	Blood	1.03	3.73	13.55
3301	Male	NormalOes	0.89	5.92	7.27
3304	Male	NormalOes	1.01	8.37	5.74
3307	Male	NormalOes	1.04	6.85	7.27
3310	Female	NormalOes	1.03	10.71	4.46
3313	Female	NormalOes	1	5.22	9.31
3316	Male	NormalOes	0.97	5.99	7.86
3319	Male	NormalOes	1.05	8.24	6.01
3322	Male	NormalOes	0.93	11.34	3.8

6.1.2 Masks

The proportion of duplicate reads that are due to coincidental fragmentation is not constant along the genome due to changes in copy number and mapping accuracy. We attempt to restrict ourselves to regions where this proportion will be constant by masking areas that are likely to be problematic. The regions that we mask out of the genome for this study are defined in the '.bed' files in this package. For instance, the Centromeres are masked out by the 4th mask file. For further reference, we will record the total genome size, sizes of masked regions, and the remainder (residual).

```
> list.files(file.path(EDpath,"masks"))
```

```
[1] "Mask1-ChrX.bed"         "Mask2-ChrY.bed"         "Mask3-ChrM.bed"
[4] "Mask4-Centromeres.bed" "Mask5-Telomeres.bed"    "Mask6-lowdepth.bed"
[7] "Mask7-highdepth.bed"
```

```
> maskFiles <- list.files(file.path(EDpath,"masks"), full.names = TRUE)
> MASKS <- lapply(maskFiles, read.delim, header=FALSE, as.is=TRUE, skip=1)
> effectiveGenomeSize<-rep(0,9)
> effectiveGenomeSize[1]<-sum(as.numeric(MASKS[[5]][17:38,3]))
> +MASKS[[1]][1,3]+MASKS[[2]][1,3]+MASKS[[3]][1,3]
```

```
[1] 214676571
```

```
> for(i in 1:7){
+     effectiveGenomeSize[i+1] <- sum(MASKS[[i]][,3] - MASKS[[i]][,2])}
> effectiveGenomeSize[9]<-effectiveGenomeSize[1]-sum(effectiveGenomeSize[2:8])
```

Note that there is no real penalty for masking too much of the genome, as we have seen that only 1000 heterozygous SNPs are required, which suggests that less than 0.1% of the genome is required.

6.1.3 Picard Output

Picard was run on bam files defined by these masks (separately for each mask) and we read these data into the matrices created earlier, before creating Table 2 in the paper.

We only use the "READ_PAIRS_EXAMINED", "READ_PAIR_DUPLICATES", and "READ_PAIR_OPTICAL_DUPLICATES" values that Picard produces (as indeed does Picard for this problem). For simplicity, we create three tables to take these values and store in them the Picard output for the entire samples, and then also for Picard output generated from the masked regions.

```
> metfiles<-list.files(file.path(EDpath,"Picard"))
> metfiles<-grep("mask",metfiles,value=T)
> for(i in 1:7){
+   temp<-read.delim(file.path(EDpath,"Picard",metfiles[i]),as.is=T)
+   temp<-temp[-(15:19),]
+   readPairsExamined[,i+1]<-temp[,3]
+   readPairDuplicates[,i+1]<-temp[,6]
+   OpticalPairDuplicates[,i+1]<-temp[,7]
+ }
```

We now calculate the 'residual' counts by subtracting the masked regions from the total.

```
> readPairsExamined[,9]<-readPairsExamined[,1]-apply(readPairsExamined[,2:8],1,sum)
> readPairDuplicates[,9]<-readPairDuplicates[,1]-apply(readPairDuplicates[,2:8],1,sum)
> OpticalPairDuplicates[,9]<-OpticalPairDuplicates[,1]-apply(OpticalPairDuplicates[,2:8],1,sum)
```

6.1.4 Generate Table 2

Now we report the residual values and the numbers for the masked regions relative to the residual values. This is Table 2 of the main manuscript.

```
> reptable<-matrix(0,ncol=4,nrow=8)
> reptable[1,]<-round(100*sapply(split(((readPairDuplicates[,9]/readPairsExamined[,9])),Group),mean),2)
> for(i in 2:8){
+   reptable[i,]<-round(sapply(split((
+     (readPairDuplicates[,i]/readPairsExamined[,i])
+     /(readPairDuplicates[,9]/readPairsExamined[,9])),Group),mean),2)
+ }
> reptable[3,1:2]<-"-"
> rownames(reptable)<-c("Residual","X","Y","M","Centromeres","Telomeres","Low Cov","High Cov")
> colnames(reptable)<-c("Female Tissue","Female Blood","Male Tissue","Male Blood")
> write(print(xtable(reptable)),file="Table2.txt")
```

	Female Tissue	Female Blood	Male Tissue	Male Blood
Residual	6.82	3.48	7.8	4.43
X	1.23	1.43	1.09	1.2
Y	-	-	4.6	9.19
M	7.41	14.62	5.62	5.05
Centromeres	4.45	8.46	3.88	7.37
Telomeres	0.99	1.19	0.98	1.09
Low Cov	0.94	0.95	0.93	0.94
High Cov	1.6	2.76	1.44	2.45

7 SNP numbers in our samples

7.1 Processing the SNPs

The following code will not be evaluated, as the patients' data are not distributed with this sweave, but can be obtained from the European Genome Archive (as previously noted). The summary data which are distributed are loaded in after this section, but the code here shows how they were created.

Later, we are going to contrast the estimate obtained from SNPs in high-GC regions with that from those in low-GC regions. We prepare for this analysis now also.

```
> HetSNPTable<-processBAMfordups(bamfilelist=bamfilelist,path=path,snplist=snplist)
> write.table(HetSNPTable,file="HetSNPDups.txt",sep="\t")
> HetSNPTableLowGC<-processBAMfordups(bamfilelist=bamfilelist,path=path,snplist=snplist[GCNo<199,])
> HetSNPTableHighGC<-processBAMfordups(bamfilelist=bamfilelist,path=path,snplist=snplist[GCNo>=199,])
> write.table(HetSNPTableHighGC,file="HetSNPDupsHighGC.txt",sep="\t")
> write.table(HetSNPTableLowGC,file="HetSNPDupsLowGC.txt",sep="\t")
```

7.2 Exploration of SNP numbers

7.2.1 Preliminaries

We now load in the summary of the numbers of SNPs being investigated, and generate the various results given in the section "SNP numbers and duplicate numbers" in the main manuscript.

```
> HetSNPTable<-read.delim(file.path(EDpath,"HetSNPDups.txt"),as.is=T)
```

We define the duplicate rate as the number of duplicates identified within the unmasked regions of the genome divided by the number of read-pairs examined within those regions.

```
> ResDupR<-readPairDuplicates[,9]/readPairsExamined[,9]
```

We define the residual read depth as 200 (the number of bases sequenced per read-pair) multiplied by the number of read-pairs examined, divided by the length of the unmasked genome.

```
> ResDepth<-round(200*readPairsExamined[,9]/effectiveGenomeSize[9],2)
```

7.2.2 How many SNPs do we see?

"From the 2,500 sites considered, the median number of heterozygous SNPs identified per sample is 1,009 (range 942 to 1,093)."

```
> summary(HetSNPTable[,1])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  942.0   988.5  1009.0  1015.0  1034.0  1093.0 
```

"Average read depth (as measured by the number of reads mapping to the regions of the genome that we are not masking) is correlated with the number of heterozygous SNPs (correlation = 0.52, p = 0.014): a reflection on our stringent calling criterion."

```
> summary(lm(HetSNPTable[,1]~ResDepth+ResDupR))
```

```
Call:
lm(formula = HetSNPTable[, 1] ~ ResDepth + ResDupR)

Residuals:
    Min      1Q  Median      3Q     Max 
```

```
-64.061 -17.787   1.512  25.908  50.541

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  803.774     81.198   9.899  6.2e-09 ***
ResDepth       2.714      1.033   2.627   0.0166 *  
ResDupR       -1.191    262.094  -0.005   0.9964    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 34.61 on 19 degrees of freedom
Multiple R-squared:  0.2671,	Adjusted R-squared:   0.19 
F-statistic: 3.463 on 2 and 19 DF,  p-value: 0.0522
```

```
> cor.test(ResDepth,HetSNPTable[,1])
```

```
        Pearson's product-moment correlation

data:  ResDepth and HetSNPTable[, 1] 
t = 2.7001, df = 20, p-value = 0.01377
alternative hypothesis: true correlation is not equal to 0 
95 percent confidence interval:
 0.1217976 0.7705574 
sample estimates:
      cor 
0.5168646 
```

7.2.3 How many duplicate read pairs are we using?

“The number of sets of duplicate read fragments observed to overlie the heterozygous SNP sites varies from $1,334$ to $9,513$.”

```
> #summary(apply(HetSNPTable[,-1],1,sum))
> summary(HetSNPTable[,3])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1334    2483    3442    3768    4709    9513 
```

“The number of duplicate sets is unsurprisingly dependent on the duplicate rate ($p < 2x10^{-16}$) and the depth of coverage ($p = 3.3x10^{-5}$), but not directly on the number of heterozygous sites being considered (although this is correlated with the depth of coverage).”

```
> summary(lm(HetSNPTable[,3]~ResDupR+HetSNPTable[,1]+ResDepth))
```

```
Call:
lm(formula = HetSNPTable[, 3] ~ ResDupR + HetSNPTable[, 1] + 
    ResDepth)

Residuals:
    Min      1Q  Median      3Q     Max 
-867.83  -75.79   71.24  130.24  342.18 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)      -7342.603   1515.346  -4.845  0.00013 ***
ResDupR          67071.320   1971.187  34.026  < 2e-16 ***
HetSNPTable[, 1]     3.311      1.725   1.919  0.07096 .  
```

```
ResDepth            49.726      9.073   5.481 3.32e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 260.3 on 18 degrees of freedom
Multiple R-squared:  0.9859,        Adjusted R-squared:  0.9835
F-statistic: 418.2 on 3 and 18 DF,  p-value: < 2.2e-16
```

7.2.4 How do the duplicates read pairs break down into pairs, triples, etc.?

“Of the total of 82,903 sets of duplicate fragments identified among the 22 samples, ...”

```
> sum(HetSNPTable[,3])
```

```
[1] 82903
```

“...the vast majority (73,767 or 89%) are duplicate pairs,”

```
> sum(HetSNPTable[,4:5])
```

```
[1] 73767
```

```
> round(100*sum(HetSNPTable[,4:5])/sum(HetSNPTable[,3]))
```

```
[1] 89
```

“...8,262 (10%) are triples,...”

```
> sum(2*HetSNPTable[,6:7])
```

```
[1] 8262
```

```
> round(100*2*sum(HetSNPTable[,6:7])/sum(HetSNPTable[,3]))
```

```
[1] 10
```

“...771 (0.9%) are quartets,...”

```
> sum(3*HetSNPTable[,8:10])
```

```
[1] 771
```

```
> round(100*3*sum(HetSNPTable[,8:10])/sum(HetSNPTable[,3]))
```

```
[1] 1
```

“and the greatest number of fragments seen in a duplicate set is six (one instance).”

```
> sum(HetSNPTable[,14:17])
```

```
[1] 1
```

8 Duplicate rate estimates

8.1 Estimating the fragmentation-duplicate proportion

We now calculate the maximum likelihood estimates of the proportions of duplicates that are fragmentation duplicates.

```
> PL<-genPL(16)
> CL<-genCoefs(PL)
> FDres<-processduptable(HetSNPTable,PL,CL)
> FDres
```

```
              ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
SS6003108.bam        4.072517        0.086        3.722281   0.3502365
SS6003110.bam        5.577882        0.082        5.120496   0.4573864
SS6003112.bam        4.634298        0.098        4.180136   0.4541612
SS6003114.bam        2.602954        0.093        2.360879   0.2420747
SS6003116.bam        9.250972        0.036        8.917937   0.3330350
SS6003118.bam       12.130834        0.033       11.730516   0.4003175
SS6003120.bam        1.922691        0.198        1.541998   0.3806928
SS6003124.bam        5.917578        0.065        5.532936   0.3846426
SS6003128.bam        2.274395        0.160        1.910492   0.3639032
SS6003130.bam        6.170018        0.058        5.812157   0.3578610
SS6003132.bam        3.889274        0.072        3.609246   0.2800277
SS6003134.bam        2.071006        0.176        1.706509   0.3644970
SS6003136.bam        3.661727        0.094        3.317525   0.3442024
SS6003148.bam        1.853215        0.170        1.538169   0.3150466
SS6003301.bam        4.169853        0.042        3.994719   0.1751338
SS6003304.bam        6.814179        0.017        6.698338   0.1158410
SS6003307.bam        5.181666        0.039        4.979581   0.2020850
SS6003310.bam        8.962766        0.039        8.613218   0.3495479
SS6003313.bam        3.683735        0.060        3.462711   0.2210241
SS6003316.bam        4.271865        0.033        4.130893   0.1409715
SS6003319.bam        6.301027        0.098        5.683526   0.6175006
SS6003322.bam        9.855291        0.019        9.668041   0.1872505
```

8.2 The Picard estimates

From Picard, we generate two estimates of the duplication rate: the basic estimate as reported by Picard and a residual estimate generated only from regions of the genome that are not masked.

```
>   BasicEst<-(readPairDuplicates[,1]-OpticalPairDuplicates[,1])/
+   (readPairsExamined[,1]-OpticalPairDuplicates[,1])
>   ResidEst<-(readPairDuplicates[,9]-OpticalPairDuplicates[,9])/
+   (readPairsExamined[,9]-OpticalPairDuplicates[,9])
```

We now generate Figure 2 from the main manuscript (Figure 9 in this document).

```
>   par(mfrow=c(1,2))
> plot(c(0,0.15),c(0,0.15),type="n",xlab="Estimates from 2,500 SNPs",ylab="Picard Estimates",main="Uncorre
> for(i in 1:22){
+ lines(c(FDres[i,1]/100,FDres[i,1]/100),c(BasicEst[i],ResidEst[i]))
+ points(FDres[i,1]/100,BasicEst[i],pch=16,col="red")
+ points(FDres[i,1]/100,ResidEst[i],pch=16,col="black")
+ }
> abline(0,1)
> legend("bottomright",fill=c("red","black"),legend=c("Basic Estimate","Masked Estimate"))
> plot(100*BasicEst/FDres[,3],FDres[,1]/FDres[,3],xlim=c(0.9,2.6),ylim=c(0.9,1.3),pch=16,xlab="Picard Esti
> abline(v=1,lwd=2)
> abline(h=1,lwd=2)
```

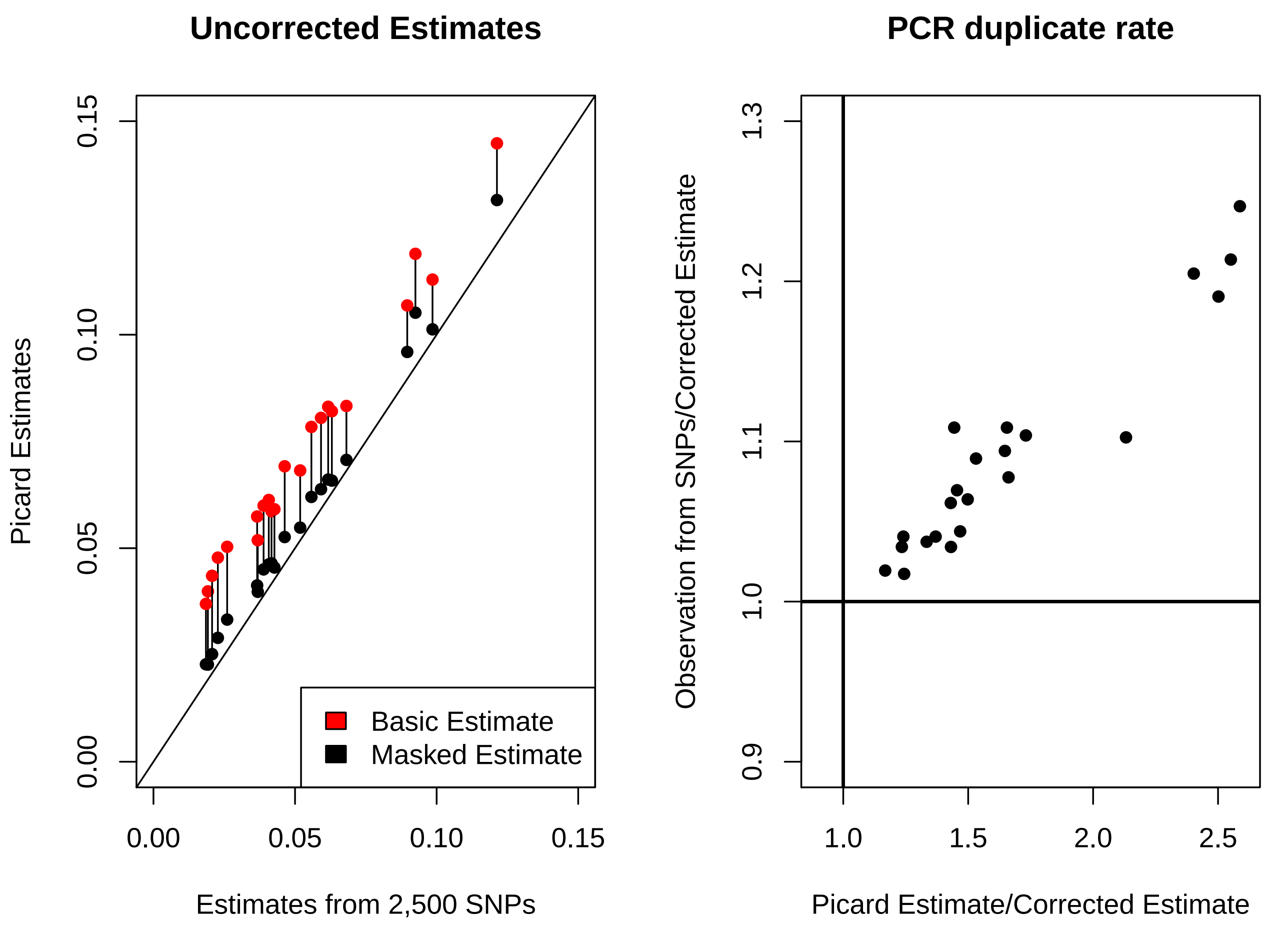

Figure 9: **The different estimates of duplication rate**

We see in Figure 9 that the masking of regions of the genome does a good job, but consistently fails to bring the estimate of the overall duplicate rate down to that observed from the 2,500 loci. Thus we use our observed duplicate rate from the SNPs as our baseline, and see that after correction it is reduced by up to one fifth. The original (standard) Picard estimate can overestimate the rate by a factor of 2.5.

8.3 Other properties

How many more reads are being marked as duplicates than we would want?

```
> readPairDuplicates[,1]-(FDres[,3]/100)*readPairsExamined[,1]/(1+FDres[,3]/100)
```

```
SS6003108 SS6003110 SS6003112 SS6003114 SS6003116 SS6003118 SS6003120 SS6003124 SS6003128
 29524695  34627752  37656006  24215186  34653344  46385765  27703407  28002007  28282071
SS6003130 SS6003132 SS6003134 SS6003136 SS6003148 SS6003301 SS6003304 SS6003307 SS6003310
 29212616  25963937  29626744  25286539  22808089  18449851  21152158  21947459  28546788
```

```
SS6003313 SS6003316 SS6003319 SS6003322
 18793636  19561578  29921162  23449747
```

What depth of coverage does this represent?

```
> round((readPairDuplicates[,1]-(FDres[,3]/100)*readPairsExamined[,1]/(1+FDres[,3]/100))*200/effectiveGenom
```

```
SS6003108 SS6003110 SS6003112 SS6003114 SS6003116 SS6003118 SS6003120 SS6003124 SS6003128
      2.0       2.4       2.6       1.7       2.4       3.2       1.9       1.9       2.0
SS6003130 SS6003132 SS6003134 SS6003136 SS6003148 SS6003301 SS6003304 SS6003307 SS6003310
      2.0       1.8       2.1       1.8       1.6       1.3       1.5       1.5       2.0
SS6003313 SS6003316 SS6003319 SS6003322
      1.3       1.4       2.1       1.6
```

9 Observations on consistency

9.1 Agreement of estimate from M equal to 2, with estimate from M greater than 2

We can separate out the estimate when $M = 2$ from that when $M > 2$. First of all we calculate the estimates only from the $M = 2$ data.

```
> HetSNPTableM2<-HetSNPTable
> HetSNPTableM2[,6:21]<-0
> FDvec2<-processduptable(HetSNPTableM2,PL,CL)
>
```

and then from the $M > 2$ data.

```
> HetSNPTableM3<-HetSNPTable
> HetSNPTableM3[,4:5]<-0
> FDvec3<-processduptable(HetSNPTableM3,PL,CL)
>
```

If we do this, we see that while there is some noise (to be expected as we have seen that there is a 19:1 ratio in terms of the numbers of fragment sets from which we draw our estimates) there is no evidence of bias (Figure 10).

```
> plot(FDvec2[,2],FDvec3[,2],xlab="estimates from sets of 2 duplicate fragments",
+      ylab="estimates from sets of >2 duplicate fragments")
> abline(0,1)
```

9.2 Agreement of estimate from High-GC SNPs, and Low-GC SNPs

We observe a median of $199/500$ Gs and Cs in our 2500 SNP regions and so divide up the SNPs into two sets: i) the 1242 SNP regions with GC content less than $199/500$, and ii) the 1258 SNP regions with GC content of $199/500$ or greater. We first load in the summary tables of SNP pattern counts.

```
> HetSNPTableHighGC<-read.delim(file.path(EDpath,"HetSNPDupsHighGC.txt"),as.is=T)
> HetSNPTableLowGC<-read.delim(file.path(EDpath,"HetSNPDupsLowGC.txt"),as.is=T)
```

We now process both tables using the methods developed for the main table.

```
> FDvecHigh<-processduptable(HetSNPTableHighGC,PL,CL)
> FDvecLow<-processduptable(HetSNPTableLowGC,PL,CL)
```

We note that there is good agreement, and no evidence of bias, in the estimate of the proportion of fragmentation duplicates when we compare the high-GC and low-GC values.

```
>   plot(FDvecHigh[,2],FDvecLow[,2],xlab="estimates from duplicates in 'high GC' regions",
+        ylab="estimates from duplicates in 'low GC' regions")
> abline(0,1)
```

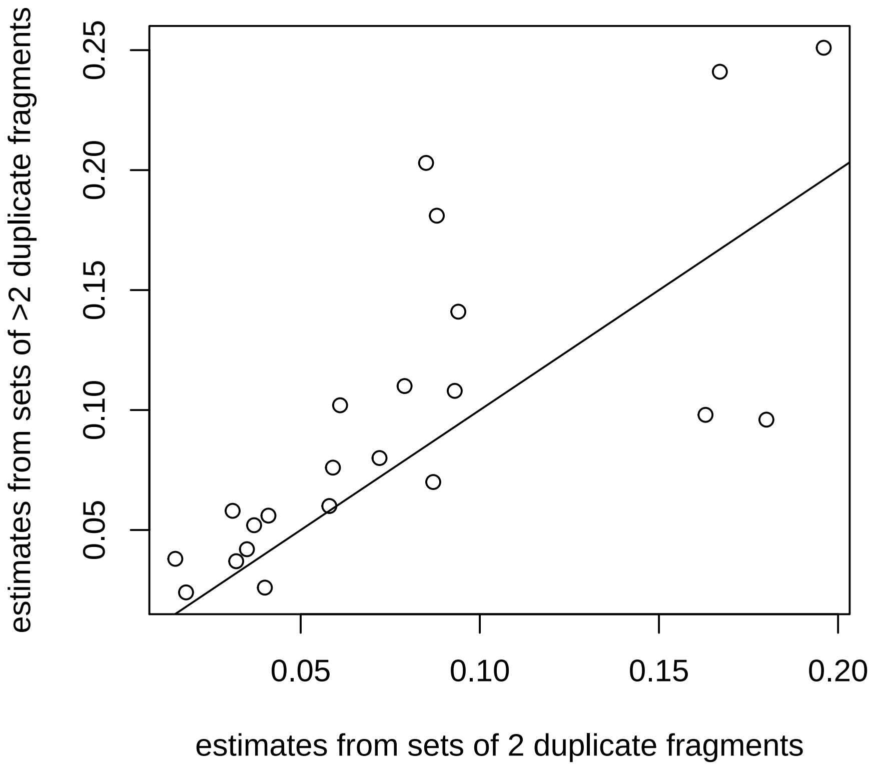

Figure 10: **Consistency of estimates from sets of 2 fragments and sets of more than 2 fragments.**

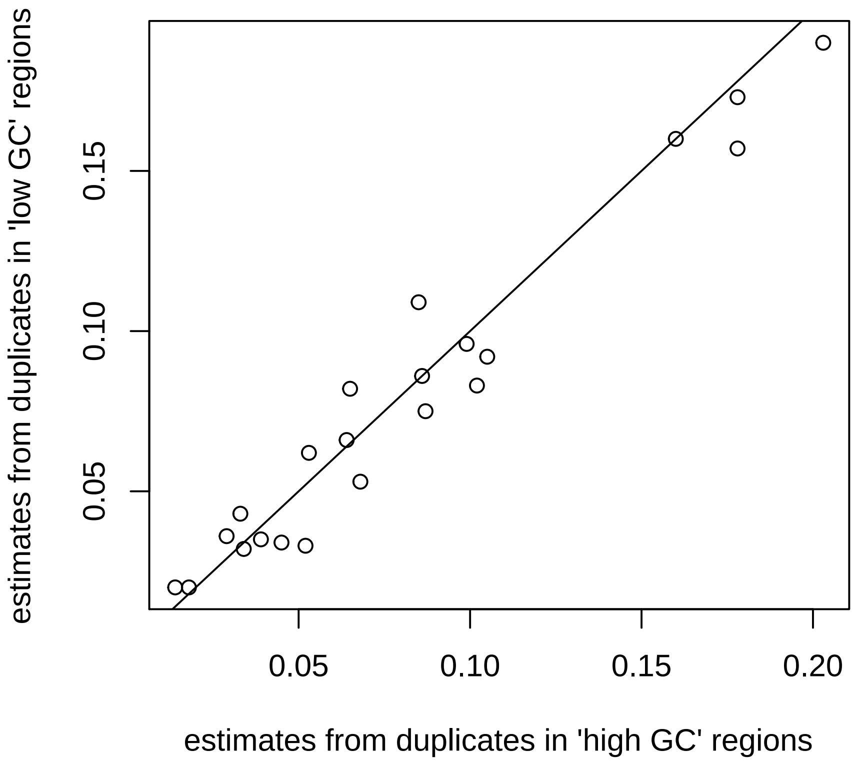

Figure 11: **GC effect:** Consistency of estimates from low and high GC regions.

10 Processing Single End sequencing

We can simulate having actually generated single end data by removing the connections between the two reads in a fragment. Although the true PCR duplicate rate is unchanged, the apparent duplicate rate is substantially higher as any two reads starting in the same location will appear to be duplicates (naturally a less-stringent criterion than having to start in the same location **and** have the same fragment length).

```
> HetSNPTableSE<-processBAMSE(bamfilelist=bamfilelist,path=path,snplist=snplist)
> write.table(HetSNPTableSE,file="HetSNPTableSingleEnd.tsv",sep="\t")
```

```
> HetSNPTableSE<-read.delim(file.path(EDpath,"HetSNPTableSingleEnd.tsv"),as.is=T,header=T)
> SEests<-processduptable(HetSNPTableSE,PL,CL,reso=1001)
> head(SEests)
```

```
              ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
SS6003108.bam        33.83172        0.881        4.025974    29.80574
SS6003110.bam        34.65467        0.858        4.920963    29.73371
SS6003112.bam        37.04922        0.898        3.779020    33.27020
SS6003114.bam        27.87598        0.929        1.979195    25.89679
SS6003116.bam        31.10318        0.724        8.584479    22.51870
SS6003118.bam        37.11522        0.686       11.654178    25.46104
```

A mean of 85% of the duplicates seen when treating the data as single end are attributable to fragmentation duplicates, and the correction places the duplicate estimates in remarkable agreement with the estimates derived treating the data as 'paired-end'.

```
> par(mfrow=c(1,2))
> plot(FDres[,1],SEests[,1],pch=16,xlab="Basic estimate from PE data (%age)",ylab="Basic estimate from SE d
> abline(0,1)
> plot(FDres[,3],SEests[,3],pch=16,xlab="Corrected estimate from PE data (%age)",ylab="Corrected estimate i
> abline(0,1)
```

```
> par(mfrow=c(1,2))
> plot(100*BasicEst/FDres[,3],FDres[,1]/FDres[,3],xlim=c(0.9,2.6),ylim=c(0.9,1.3),pch=16,xlab="Picard Estin
> abline(v=1,lwd=2)
> abline(h=1,lwd=2)
> plot(FDres[,3],SEests[,3],pch=16,xlab="Corrected estimate from PE data (%age)",ylab="Corrected estimate i
> abline(0,1)
```

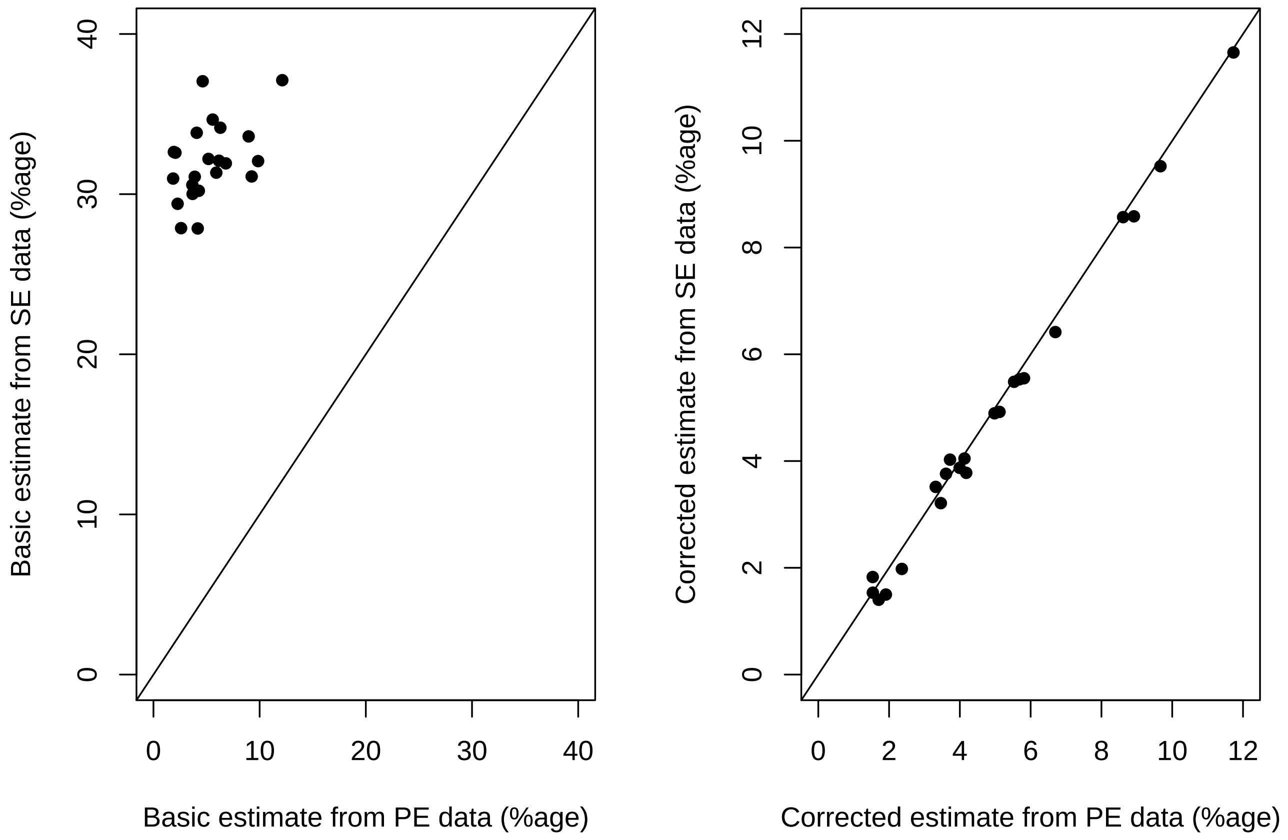

Figure 12: **Duplicate rates from paired- and single-end data**

11 Processing a Tumour sample

11.1 The Sample

We have, until now, been considering normal (diploid) samples. We now demonstrate the approach applied to a tumour sample: Case SS6003314 from the same paper and manuscript as the normal data. SS6003314 is a broadly tetraploid tumour with approximately 74% cellularity. It has some regions exhibiting sub-clonal copy number changes, but when we plot minor allele frequency against tumour depth 13, it notably has clear areas of diploid allelic-balance (AB), and tetraploid allelic-balance (AABB). Bed files defining these regions (`ABbed.bed` and `AABBbed.bed`) are included in the `extdata` folder of the LynchSmithEldridgeTavareFragDup package.

We load in Picard data for the entire library (duplicate rate: 5.2%), and for two sets of regions: i) The regions that are classified as AB (after removing masked regions - duplicate rate: 3.7%), and ii) The regions that are classified as AABB (after removing masked regions - duplicate rate: 3.8%). Note that this is a low duplication rate compared to those observed in libraries from normal tissue/blood. We attribute the change going from the entire library to the subsets as being due to the removal of masked regions; the AABB regions alone represent more than half of the sequencing library. We expect there to be a higher duplication rate in the AABB regions than in the AB region because of fragmentation duplicates. That it is such a small change suggests that the fragmentation duplicate rate will be small in this case.

```
> TumourPicard<-read.csv(file.path(EDpath,"Tumour","Picard.csv"),as.is=T)
> TumourPicard
```

```
         LIBRARY UNPAIRED_READS_EXAMINED READ_PAIRS_EXAMINED UNMAPPED_READS
1      SS6003314                10432535           994242462       61864339
2   SS6003314-AB                    1576              417850           1576
3 SS6003314-AABB                 2438478           588331041        2438895
  UNPAIRED_READ_DUPLICATES READ_PAIR_DUPLICATES READ_PAIR_OPTICAL_DUPLICATES
1                  7153353             51808523                       455107
2                      402                15495                          188
3                   872992             22418117                       270067
  PERCENT_DUPLICATION ESTIMATED_LIBRARY_SIZE
1            0.055415             9281651350
2            0.037493                5558009
3            0.038766             7609634846
```

```
> # and the duplicate rates
> round((TumourPicard[,6]-TumourPicard[,7])/(TumourPicard[,3]-TumourPicard[,7]),3)
```

```
[1] 0.052 0.037 0.038
```

11.2 The SNPs

Tumour samples require bespoke lists, so we restrict ourselves to sites that we know to be heterozygous. We can define two sets of heterozygous SNPs for the AB (985 SNPs), and AABB (a sample of $10,000$ SNPs) regions.

```
> snplistAABB<-read.delim(file.path(EDpath,"Tumour","usedupAABB.txt"),as.is=T,header=F)
> snplistAB<-read.delim(file.path(EDpath,"Tumour","usedupAB.txt"),as.is=T,header=F)
```

11.3 Counting the duplicates

As before, we are unable to distribute the raw data, which are archived in the European Genome Archive [EGA:EGAD00001000704]. The following code will generate the summary table in the manner we have come to expect.

```
> HetSNPTableAABB<-processBAMfordups("SS6003314.bam",path,snplistAABB,poscol=2)
> HetSNPTableAB<-processBAMfordups("SS6003314.bam",path,snplistAB,poscol=2)
```

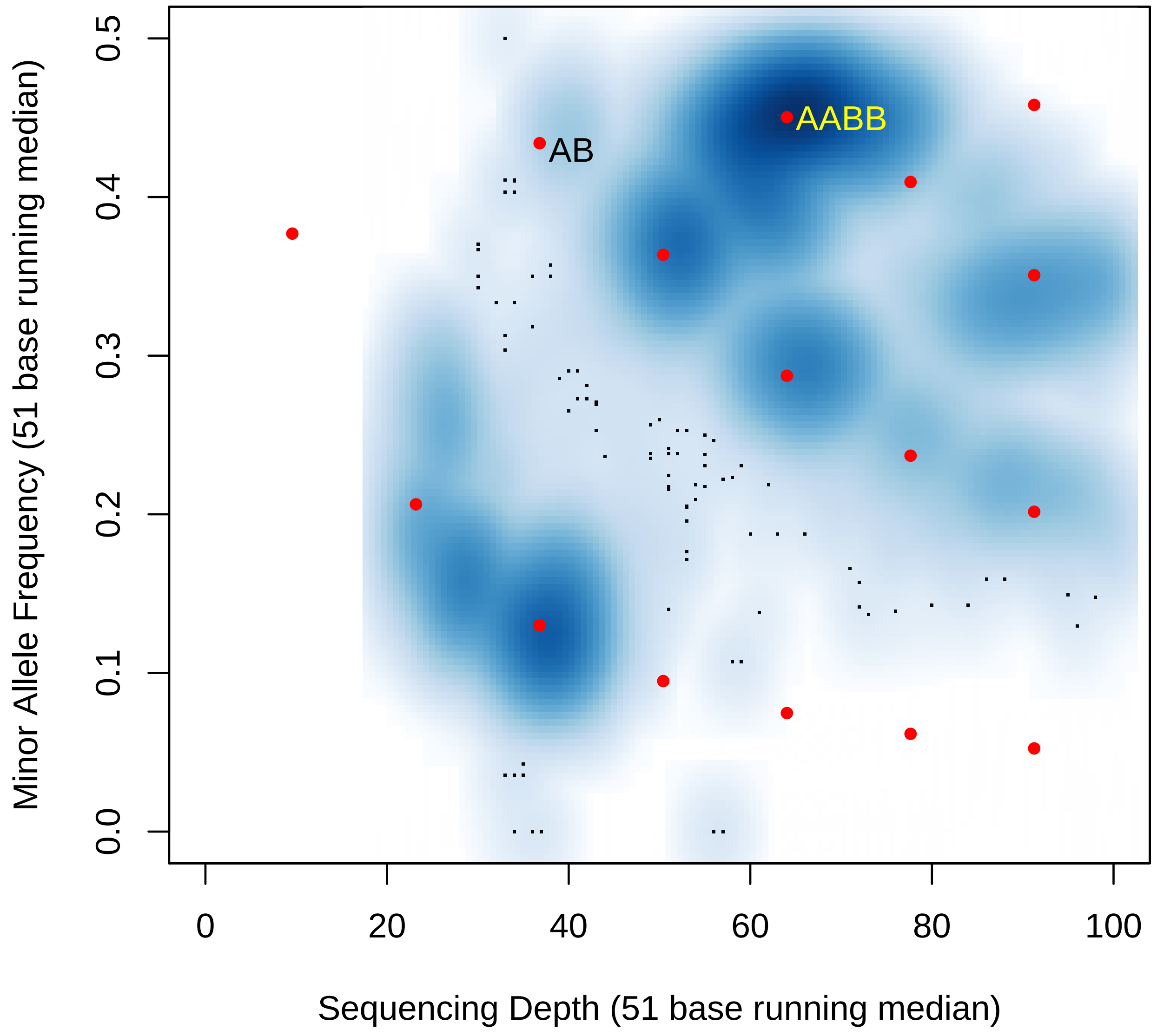

Figure 13: A plot of minor allele fraction against sequencing depth reveals regions of consistent copy number.

```
> write.table(HetSNPTableAABB,file="HetSNPTableTumourAABB.txt",sep="\t")
> write.table(HetSNPTableAB,file="HetSNPTableTumourAB.txt",sep="\t")
```

Assuming that the user hasn’t obtained the raw data, we now load in the precompiled summary table. Note that although we have pre-specified sites that are heterozygous, the filters in processBAMfordups have reduced the numbers to 8396 and 759.

```
> #HetSNPTableAABB<-read.delim(file.path(EDpath,"Tumour","HetSNPTableTumourAABB.txt"),as.is=T,header=T)
> #HetSNPTableAB<-read.delim(file.path(EDpath,"Tumour","HetSNPTableTumourAB.txt"),as.is=T,header=T)
> HetSNPTableAABB<-read.delim(file.path(EDpath,"HetSNPTableTumourAABB.tsv"),as.is=T,header=T)
> HetSNPTableAB<-read.delim(file.path(EDpath,"HetSNPTableTumourAB.tsv"),as.is=T,header=T)
> HetSNPTableTumour<-rbind(HetSNPTableAABB,HetSNPTableAB)
> rownames(HetSNPTableTumour)<-c("AABB","AB")
> rm(HetSNPTableAB,HetSNPTableAABB)
```

11.4 Estimating the fragmentation duplicate proportion

Processing this in the same manner as previous allele pattern counts...

```
> tumourresults<-processduptable(HetSNPTableTumour,PL,CL)
```

...we see that the fragmentation duplicate rate is higher in the AABB region than the AB region.

```
> tumourresults
```

```
     ObservedDupRate PropFragDups AdjustedDupRate FragDupRate
AABB        3.755750        0.047        3.579230  0.17652026
AB          3.683342        0.017        3.620726  0.06261682
```

The difference between the two estimates has close to halved. Given the noise inherrent in the estimate from the smaller AB region, the corrected estimates from the AABB and AB regions are remarkably consistent.

11.5 Extending to regions that are not in allelic balance

We have demonstrated the approach only in genomic regions of allelic balance, but in theory this could be extended to any region where both alleles are present in the library (which due to contamination from normal cells, will still allow for regions that exhibit loss of heterozygosity in the tumour).

The added complication comes in the form of $\Pr(AP_k \mid \text{PART}_i)$ (see section 3.2). Consider the case where we have two duplicate reads from the partition into two different molecules. Without loss of generailty let the alleles at the SNP of interest be C and T. Currently we only consider two possible allele patterns `AA` representing the posibilities CC and TT and `AB` representing CT and TC. Since the probability of each allele is $\frac{1}{2}$ we say that the probability of the `AA` pattern is also $\frac{1}{2} = 0.5^2 + 0.5^2$.

Now if the allelic proportions are in fact 0.25 and 0.75, then we have two options:

- We still consider only two allele patterns, but the probability of the AA pattern is now $0.25^2 + 0.75^2 = 0.625$
- We can confidently associate probabilities with specific alleles (e.g through phasing we may be confident that it is the C allele present in 75% of molecules) in which case we would now consider three allele patterns `CC`, `CT`, `TT` with probabilities 0.5625, 0.375 and 0.0625 respectively.

The second option is more powerful than the first, but would require a not insubstantial effort to ensure that the probabilities are matched to SNPs correctly. In both cases, since fragmentation duplicates are more inclined to share the same allele than when we have allelic balance, the analysis will be less powerful than when we have allelic balance. In studies of human tumours we have thus far always been able to identify regions of the genome in allelic balance, but we acknowledge that there is the potential need to go down this route of studying regions of allelic imbalance, and that indeed it may be a prerequisite for application of these methods to other organisms.

Note that for any sizeable segment of genome in the same state, we can empirically estimate the allelic fraction with enough precision that our methods will be useful, and so this does not cause problems.

12 Complexity estimates

If the library complexity is X, the number of read-pairs sequenced is N, and the duplicate rate is denoted by R_D, then our estimate of X is obtained by solving

$$R_D = \frac{X}{N} \exp\left(-\frac{N}{X}\right) + 1 - \frac{X}{N} \tag{34}$$

using numerical methods. See Lunter (http://www.newton.ac.uk/programmes/CGR/seminars/2010071310454.html) for a derivation.

12.1 Picard

Picard bases its estimate of complexity on a genome-wide estimate of duplicate rate that is adversely affected by fragmentation duplicates.

```
> CEPicard<-unmaskedP[,9]
> summary(CEPicard)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
3.590e+09 5.814e+09 7.568e+09 7.704e+09 8.988e+09 1.355e+10
```

12.2 Estimating complexity using a basic duplication rate estimate

We recalculate the complexity using the observed duplicate rate from the SNPs we interrogated. This corrects for the excessive influence of fragmentation duplicates in regions of high copy number/poor mappability, but not the general influence of fragmentation duplicates.

To obtain a starting point for the search, we use the closed form estimate based on a Maclaurin expansion $X = N/(2R_D)$.

```
> CEME<-rep(22,0)
> for(i in 1:22){
+   R<-FDres[i,1]/100
+   CEME[i]<-(readPairsExamined[i,1]-OpticalPairDuplicates[i,1])/(2*R)
+ }
```

Now we obtain our complexity estimates.

```
> CEobserved<-rep(22,0)
> for(i in 1:22){
+   R<-FDres[i,1]/100
+   N<-readPairsExamined[i,1]-OpticalPairDuplicates[i,1]
+   startX<-CEME[i]
+   CEobserved[i]<-optimize(libCompNewParam,R=R,N=N,interval=c(0,2*startX),maximum=F)$minimum
+ }
> summary(CEobserved)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
4.392e+09 7.950e+09 1.068e+10 1.259e+10 1.370e+10 2.853e+10
```

In general the values are higher reflecting the lower duplicate rate estimate.

12.3 Estimating complexity using a corrected duplication rate estimate

This is the best estimate we can produce.

```
> CEcorrected<-rep(22,0)
> for(i in 1:22){
+   R<-FDres[i,3]/100
+   N<-readPairsExamined[i,1]-OpticalPairDuplicates[i,1]
+   startX<-CEME[i]
+   CEcorrected[i]<-optimize(libCompNewParam,R=R,N=N,interval=c(0,2*startX),maximum=F)$minimum
+ }
> summary(CEcorrected)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
4.490e+09 8.521e+09 1.116e+10 1.420e+10 1.508e+10 3.567e+10 
```

In all cases the standard estimate from Picard is lower than our best estimate of the library complexity (with factors ranging from 1.21 to 2.87)

```
> summary(CEcorrected/CEPicard)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.182   1.403   1.502   1.697   1.743   2.631 
```

13 Adding additional lanes of sequencing

We now consider the effects of adding additional lanes of sequencing on estimates of complexity. First loading the Picard estimates, we see that the estimate of complexity increases with each lane.

```
> PicardLane<-read.delim(file.path(EDpath,"Picard", "SS6003301_lanes.metrics.txt"),as.is=T)
> PicardLane[,9]
```

```
[1] 4864861694 5876377366 6510297164 6933459855 7273503256
```

```
> round(PicardLane[,9]/PicardLane[5,9],2)
```

```
[1] 0.67 0.81 0.90 0.95 1.00
```

We now apply our methods. As before, we are unable to distribute the raw data, which are archived in the European Genome Archive [EGA:EGAD00001000704]. Upon obtaining the data, the following code would produce the required table.

```
> lanelist<-list(c("HSQ1004_100:1"),
+ c("HSQ1004_100:1","HSQ1004_100:2"),
+ c("HSQ1004_100:1","HSQ1004_100:2","HSQ1004_100:3"),
+ c("HSQ1004_100:1","HSQ1004_100:2","HSQ1004_100:3","HSQ1004_100:4"),
+ c("HSQ1004_100:1","HSQ1004_100:2","HSQ1004_100:3","HSQ1004_100:4","HSQ1004_97:8:"))
> HetSNPTableByLane<-processBAMbylane("SS6003301.bam",path,snplist,lanelist)
> write.table(HetSNPTableByLane,file="HetSNPDupsByLane.txt",sep="\t")
```

We now load and process the prepared table

```
> HetSNPTableByLane<-read.delim(file.path(EDpath,"HetSNPDupsByLane.txt"),as.is=T,header=T)
> EstimatesByLane<-processduptable(HetSNPTableByLane,PL,CL)
```

We generate a first order complexity estimate from which to begin our search.

```
> LaneCEME<-rep(0,5)
> for(i in 1:5){
+   R<-EstimatesByLane[i,1]/100
+   LaneCEME[i]<-(PicardLane[i,3]-PicardLane[i,7])/(2*R)
+ }
```

Now we generate complexity estimates using the observed duplicate rate from our heterozygous SNPs. Note that a) as expected from the previous section, the estimates are substantially greater, and b) the estimates are still monotonic but with a suggestion that they might be converging more quickly.

```
> LaneCEobserved<-rep(0,5)
> for(i in 1:5){
+   R<-EstimatesByLane[i,1]/100
+   N<-(PicardLane[i,3]-PicardLane[i,7])
+   startX<-LaneCEME[i]
+   LaneCEobserved[i]<-optimize(libCompNewParam,R=R,N=N,interval=c(0,2*startX),maximum=F)$minimum
+ }
> LaneCEobserved
```

```
[1]  7847379090  8965708035  9607168600  9946088681 10350952077
```

```
> round(LaneCEobserved/LaneCEobserved[5],2)
```

```
[1] 0.76 0.87 0.93 0.96 1.00
```

Now we generate complexity estimates using the corrected duplicate rate.

```
> LaneCEcorrected<-rep(0,5)
> for(i in 1:5){
```

```
+   R<-EstimatesByLane[i,3]/100
+   N<-(PicardLane[i,3]-PicardLane[i,7])
+   startX<-LaneCEME[i]
+   LaneCEcorrected[i]<-optimize(libCompNewParam,R=R,N=N,interval=c(0,2*startX),maximum=F)$minimum
+ }
> LaneCEcorrected
```

```
[1]  7975943131  9474139520 10100599958 10392584962 10817912779
```

```
> round(LaneCEcorrected/LaneCEcorrected[5],2)
```

```
[1] 0.74 0.88 0.93 0.96 1.00
```

```
> par(mfrow=c(1,2))
> plot(c(1,3),c(3,35),type="n",ylab="Library complexity estimate (billions)",xlab="",axes=F)
> axis(2)
> box()
> for(i in seq(0,40,2)){
+   rect(-1,i-1,5,i,border="grey95",col="grey95")
+ }
> for(i in 1:22){
+   tcol<-"black"
+   if(WeaverSuppTable1[i,12]=="Blood"){tcol<-"red"}
+   lines(1:3,c(CEPicard[i],CEobserved[i],CEcorrected[i])/(10^9),type="b",pch=16,col=tcol)
+ }
> axis(1,at=1:3,labels=c("Picard",
+                         "SNP observed",
+                         "Corrected"),las=3)
> plot(PicardLane[,9]/(10^9),pch=16,ylim=c(0,11),ylab="",xlab="Number of lanes",las=1,main="Complexity")
> points(LaneCEobserved/(10^9),pch=16,col="red")
> points(LaneCEcorrected/(10^9),pch=16,col="blue")
> legend("bottom",fill=c("black","red","blue"),legend=c("Picard","SNP\nobserved","SNP corrected"))
```

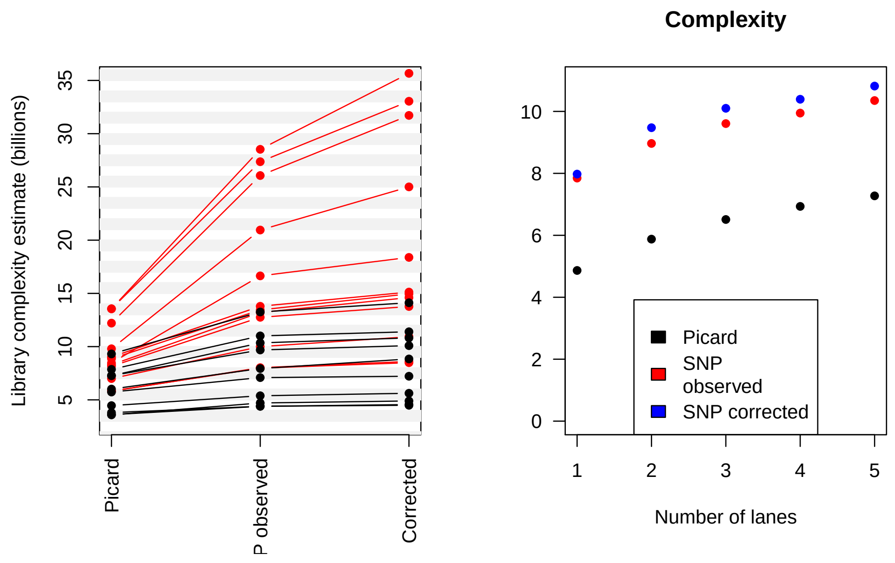

Figure 14: **Consistency of estimates from sets of 2 fragments and sets of more than 2 fragments.**

14 Mitochondria

The mitochondrial DNA poses an interesting problem as while it may exhibit heteroplasmy, it lacks sites that are heterozygous with predictable allele fractions, but at the same time often has such high copy number that the coincedental fragmentation duplicates are a large problem.

14.1 Simulation

We begin by simulating mitochondrial reads to determine the extent of the problem. We investigate various copy numbers, and use our two extreme insert size distribtuions. We also simulate single end sequencing, by halving the length of the reads and allowing no variation in insert sizes.

```
> set.seed(246810)
> NarrowMTres<-simMTreads(numcopies=100,realISD=ISDnarrow)
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=200,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=300,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=400,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=500,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=600,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=700,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=800,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=900,realISD=ISDnarrow))
> NarrowMTres<-rbind(NarrowMTres,simMTreads(numcopies=1000,realISD=ISDnarrow))
> WideMTres<-simMTreads(numcopies=100,realISD=ISDwide)
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=200,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=300,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=400,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=500,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=600,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=700,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=800,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=900,realISD=ISDwide))
> WideMTres<-rbind(WideMTres,simMTreads(numcopies=1000,realISD=ISDwide))
> SEMTres<-simMTreads(numcopies=100,ISmean=100,ISsd=0,readlength=50)
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=200,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=300,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=400,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=500,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=600,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=700,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=800,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=900,ISmean=100,ISsd=0,readlength=50))
> SEMTres<-rbind(SEMTres,simMTreads(numcopies=1000,ISmean=100,ISsd=0,readlength=50))
```

In Figure 15 we see that with a tight insert size distribution more than half of the reads can be marked as duplicates despite not being PCR duplicates. Moreover, comparing copy numbers between samples is hampered by the saturation effect that compresses the dynamic range, and particularly by differences between the insert size distributions, if one adopts the common practise of removing all 'duplicates'.

14.2 Picard

We revisit the Picard output from the mitochondrial mask discussed in the "Sample Information" section. In doing so we will create an object entitled `PicardMito` as described below.

```
> lf<-read.delim(file.path(EDpath,"Picard",metfiles[3]),as.is=T)
> lf<-lf[-(15:19),]
```

The Picard duplicate rate is inflated by up to 3% by the inclusion of‘unpaired’ reads in its calculation (these are virtually all marked as duplicates in the high-copy context of the mtDNA). However it indicates that on average a third of reads (and up to two-thirds of reads) are marked as duplicates, and since we have a good estimate of the true PCR duplicate rate we can see that the apparent rate in the mitochondria inflates this by a factor of up to $36\times$

```
> summary(lf[,8])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1446  0.1966  0.2608  0.3200  0.4042  0.6591
```

```
> summary(100*lf[,8]/FDres[,3])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.378   4.444   6.766   8.187   9.506  35.920
```

We now tabulate the depth of sequencing with and without removing duplicates, and also with our correction for the PCR duplicate rate

```
> MitoDepths<-cbind(lf[,3]*200+lf[,2]*100,lf[,3]*200+lf[,2]*100-lf[,6]*200-lf[,5]*100,(lf[,3]*200+lf[,2]*1
> MitoCNs<-MitoDepths/(ResDepth*(100-FDres[,3])/200)
>
```

Now we produce Figure 3 from the paper (Figure **??** in this document), combining complexity and mitochondrial copy number results.

```
> par(mfrow=c(1,2))
> plot(as.numeric(NarrowMTres[,8]),as.numeric(NarrowMTres[,9]),xlim=c(0,1000),ylim=c(0,1000),pch=16,type="
> points(as.numeric(WideMTres[,8]),as.numeric(WideMTres[,9]),col="red",pch=16,type="b")
> points(as.numeric(SEMTres[,8]),as.numeric(SEMTres[,9]),col="blue",pch=16,type="b")
> abline(0,1,lwd=2)
> legend("topleft",fill=c("red","black","blue"),legend=c("Wide ISD","Narrow ISD","Single end"))
> lines(c(-100,900),c(383,383),lty=3,col="black",lwd=2)
> lines(c(-100,500),c(397,397),lty=3,col="red",lwd=2)
> lines(c(900,900),c(-100,383),lty=3,col="black",lwd=2)
> lines(c(500,500),c(-100,397),lty=3,col="red",lwd=2)
> plot(c(1,3),c(60,800),type="n",ylab="MtDNA copy number",xlab="",axes=F,main="Our 22 Example Cases")
> axis(2)
> box()
> for(i in seq(0,800,200)){
+   rect(-1,i-100,5,i,border="grey95",col="grey95")
+ }
> for(i in 1:22){
+   tcol<-"black"
+   if(WeaverSuppTable1[i,12]=="Blood"){tcol<-"red"}
+   lines(1:3,(as.numeric(MitoCNs[i,c(1,3,2)])),type="b",pch=16,col=tcol)
+ }
> axis(1,at=1:3,labels=c("No\nduplicates\nremoved","Corrected\nestimate","All\nduplicates\nremoved"),las=3
```

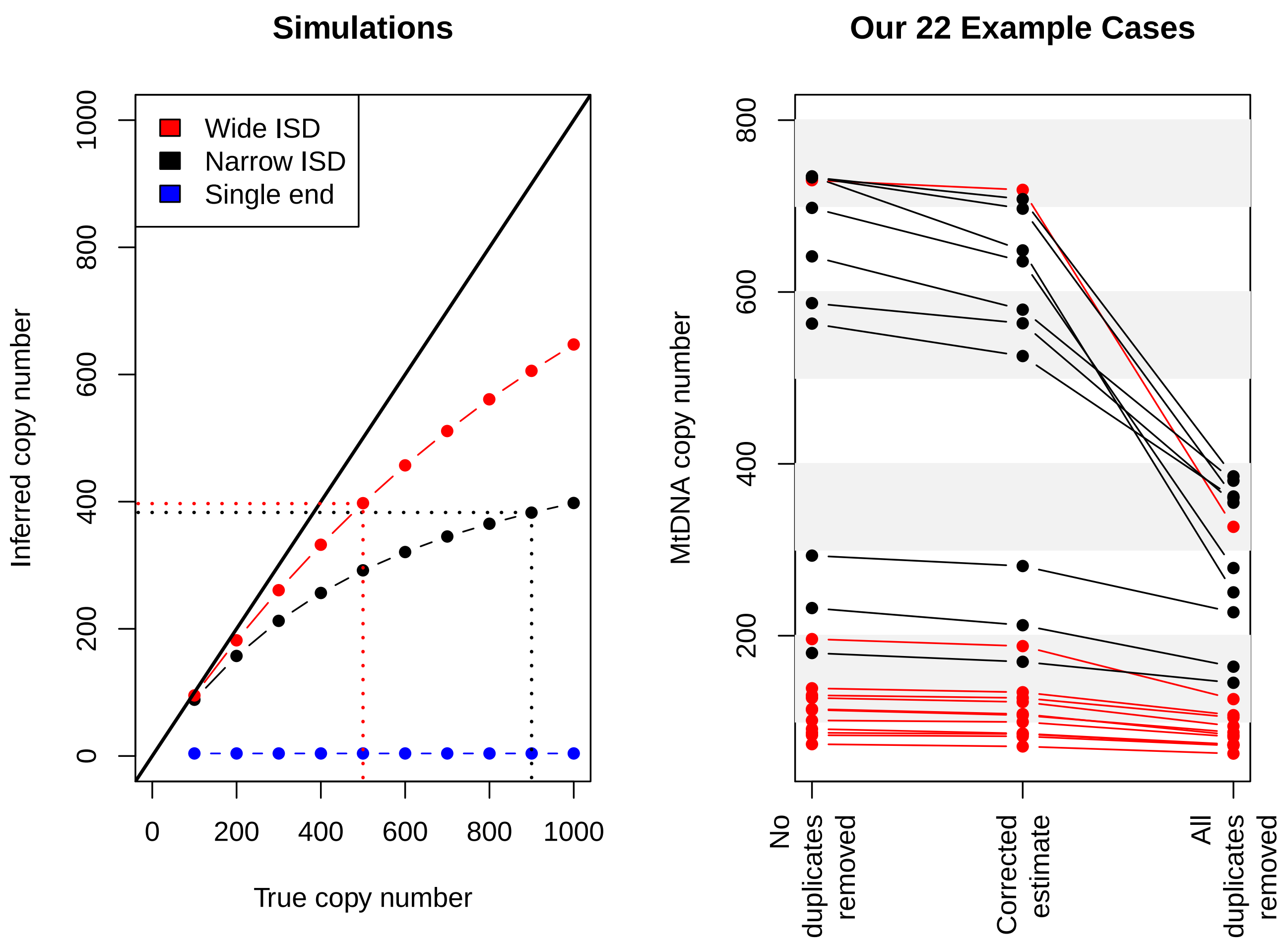

Figure 15: **mtDNA copy number results** Text text text

15 Session Info

For updates to this package/document, please visit www.compbio.group.cam.ac.uk.

```
> toLatex(sessionInfo())
```

- R version 3.2.3 (2015-12-10), `x86_64-apple-darwin13.4.0`
- Locale: `en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.16.1, Biostrings 2.38.4, BSgenome 1.38.0, BSgenome.Hsapiens.UCSC.hg19 1.4.0, FragmentationDuplicates 0.9, GenomeInfoDb 1.6.3, GenomicRanges 1.22.4, IRanges 2.4.8, Rsamtools 1.22.0, rtracklayer 1.30.3, S4Vectors 0.8.11, xtable 1.8-2, XVector 0.10.0
- Loaded via a namespace (and not attached): Biobase 2.30.0, BiocParallel 1.4.3, BiocStyle 1.8.0, bitops 1.0-6, futile.logger 1.4.1, futile.options 1.0.0, GenomicAlignments 1.6.3, lambda.r 1.1.7, RCurl 1.95-4.8, SummarizedExperiment 1.0.2, tools 3.2.3, XML 3.98-1.4, zlibbioc 1.16.0