CK.-C. Wong (ed.), Epistais: Methods and Protocols. Methods in Molecular Biology. Springer

Simulating Evolution in Asexual Populations with Epistasis

Ramon Diaz-Uriarte

Dept. Biochemistry, Universidad Autónoma de Madrid

Instituto de Investigaciones Biomédicas "Alberto Sols" (UAM-CSIC)

Madrid, Spain[*]

January 30, 2020

Abstract

I show how to use OncoSimulR, software for forward-time genetic simulations, to simulate evolution of asexual populations in the presence of epistatic interactions. This chapter emphasizes the specification of fitness and epistasis, both directly (i.e., specifying the effects of individual mutations and their epistatic interactions) and indirectly (using models for random fitness landscapes).

Keywords: simulation, epistasis, fitness landscape, evolution, mutation, fitness

[*] To whom correspondence should be addressed: ramon.diaz@iib.uam.es, rdiaz02@gmail.com, http://ligarto.org/rdiaz

1 Introduction

Here we illustrate the use of the BioConductor package OncoSimulR for simulating evolution of asexual populations with epistasis. OncoSimulR [11] implements forward-time genetic simulations in asexual populations, using biallelic loci. Fitness can be defined either directly (by specifying the fitness landscape, or the map between genotypes and fitness), as shown in sections 2.2.1 and 2.2.2, or by specifying epistatic interactions directly as shown in sections 2.2.4 to 2.2.7. Simulations use a continuous time model, and employ the state-of-the-art BNB algorithm of Mather et al. [23]. Some previous uses of OncoSimulR include the study of the sensitivity of cancer progression models to reciprocal sign epistasis [12], the predictability of cancer evolution [13, 19], and somatic mutation in plants [33].

2 Methods

Using OncoSimulR for the simulation of evolutionary processes involves:

1. Installing (if needed) and loading OncoSimulR.
2. Choosing the mapping between genotypes and fitness. It is at this stage that we specify epistasis.
3. Choosing the details of the evolutionary model, including growth models and the specifics of the mutation process.
4. Running simulations until pre-specified conditions are reached.

The second and third steps can be decided in any order, but they come necessarily (logically and chronologically) before the last. Since the focus of this book is on epistasis it is thus preferable to order the above steps as follows:

1. Installing (if needed) and loading OncoSimulR (section 2.1).
2. Specifying epistasis, which can be done either:
    - specifying epistasis indirectly, which includes possibly using models for random fitness landscapes (sections 2.2.1 and 2.2.2);

- specifying epistasis directly (sections 2.2.4 to 2.2.7).

3. Simulating evolution, which involves:

    - Choosing growth models (section 2.3.1).
    - Defining mutation rates and possible mutator genes (section 2.3.2).
    - Running simulations until pre-specified conditions are met (sections 2.3.3 to 2.3.7).

The next sections are structured following the above order. The code shown below illustrates the usage of the most important functionality, and we discuss the key options for specifying fitness and epistasis; not all options used are discussed, though (see the package vignette and function documentation for details).

2.1 Installing and loading OncoSimulR

If OncoSimulR is not installed, we must install it. Note that we are using the development version of OncoSimulR, from BioConductor 3.11, that runs under what will become R-4.0. Please refer to the specific instructions for installing R for your operating system (https://cran.r-project.org/navbar.html). Once we have installed R-4.0 we can install packages for BioConductor 3.11 (details about the release and development version of BioConductor are available from https://www.bioconductor.org/developers/how-to/useDevel/).

We are now ready to install OncoSimulR following https://www.bioconductor.org/packages/devel/bioc/html/OncoSimulR.html:

```r
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

# The following initializes usage of Bioc devel
BiocManager::install(version='devel')

BiocManager::install("OncoSimulR")
```

Installation of OncoSimulR only needs to be carried out once (more precisely, whenever the versions of OncoSimulR or of R change).

Once OncoSimulR is installed, in every R session where we want to use it, we must load the package. Here we load it and also verify its version (at least 2.17.1)

```
library(OncoSimulR)
packageVersion("OncoSimulR") ## should be >= 2.17.1

## [1] '2.17.1'
```

2.2 Specifying epistasis

We can either specify epistasis indirectly, as shown in sections 2.2.1 and 2.2.2, or directly, by specifying the fitness effects of mutations on genes (sections 2.2.3 to 2.2.7), thus having full control over the specification of epistatic interactions (see also **Note 1** and **Note 2**).

2.2.1 Specifying epistasis indirectly: explicit mapping from genotypes to fitness

We can specify the mapping between genotypes and fitness by explicitly indicating what the fitness of all possible genotypes (or all genotypes with non-zero fitness) is. Here, epistasis is not specified directly, but indirectly. For instance, we could specify a four genotype model with sign epistasis as:

```
## Create a two-column data frame with
## the mapping from genotypes to fitness
gf <- data.frame(Genotype = c("WT", "A", "B", "A, B"),
                 Fitness = c(1, 1.5, 0.5, 3),
                 stringsAsFactors = FALSE)


## Create a fitnessEffects object
## fitnessEffects objects are used as input for
## the simulations and also below to plot
## fitness landscapes and obtain epistasis
## statistics
fitness1 <- allFitnessEffects(genotFitness = gf)
```

We can plot the fitness of the four genotypes calling either plotFitnessLandscape or plot as follows (figure not shown —but see below, section 2.2.2, for a fitness landscape plot of a House of Cards model):

```
plotFitnessLandscape(fitness1)
```

And we can compute some epistasis statistics using the function Magellan_stats, that calls code provided by MAGELLAN [4] (more precisely, function fl_statistics). From the set of statistics provided by MAGELLAN, we only want the fraction of pairs of loci that have no epistasis, magnitude epistasis, sign epistasis, reciprocal sign epistasis, and γ, the correlation in fitness effects between genotypes that differ by one locus (averaged over the fitness landscape) (see details in [4] and [14]). Since we are only interested in some of the output provided by the Magellan_stats function, we write a simple wrapper to Magellan_stats (where use_log controls whether we take the log of the fitness values before computing the epistasis statistics):

```
epist_stats <- function(x, use_log = FALSE) {
    tmp <- Magellan_stats(x, use_log = use_log)[c("gamma",
                                                  "magn",
                                                  "sign",
                                                  "rsign")]
    tmp <- c(tmp, "none" = 1 - sum(tmp[c("magn", "sign",
                                         "rsign")]))
    tmp <- tmp[c(2:5, 1)]
    return(tmp)
    }
```

Now we can compute the epistasis statistics as:

```
epist_stats(fitness1, use_log = FALSE)

##  magn  sign rsign  none gamma
## 0.000 1.000 0.000 0.000 0.111
```

which shows that there is only sign epistasis in the model.

Computing the epistasis statistics on the log-transformed fitness data does not change the estimates of epistasis (it does change the estimate of γ, though), since sign epistasis is not affected by monotonic transformations:

```
epist_stats(fitness1, use_log = TRUE)

##   magn  sign rsign  none gamma
## 0.000 1.000 0.000 0.000 0.113
```

(Using `use_log = TRUE` is discussed with more detail below: see section 2.2.3.)

2.2.2 Specifying epistasis indirectly: using models for fitness landscapes

Alternatively, the mapping between genotypes and fitness can be done according to different random fitness landscapes models, which are characterized by different degrees of epistasis (see [4, 14, 35]). We will use function `rfitness`, that allows us to use the Rough Mount Fuji (RMF) model [1, 14, 28, 35], which includes as limit cases both a fully additive model [14, 35] and the House of Cards model [14, 22, 35], and the NK (or LK) model [14, 20, 21, 35]. It must be noted that the values returned by `rfitness` are to be interpreted as log-fitness values (and this is why, in the calls in this section, we call the function `epist_stats` without log-transforming the fitness values; see also section 2.2.3 for further discussion of using `use_log = TRUE` when calling `epist_stats`).

In the examples that follow, and for ease of representation, we will only use four loci. We will first use a fully additive and deterministic model. The additive model is sometimes also called a multiplicative model, as it becomes additive in the log-scale; the multiplicative effect on fitness that each mutation has does not depend on the state of other genes (see [4, 14]).

We can generate deterministic, additive models as special cases of Rough Mount Fuji models without noise [28, 35]. In the example below, the genotype with all genes mutated has maximum fitness (`reference = "max"`), and all genotypes have the same decrease in fitness per unit increase in Hamming distance from the genotype with maximum fitness (`c = 0.5`) (see also **Note 3**).

```
additive <- rfitness(4, c = 0.5, sd = 0, reference = "max")
epist_stats(additive)

##   magn   sign rsign   none gamma
##      0      0      0      1     1
```

As above, we could plot the fitness landscape calling either `plotFitnessLandscape` or `plot` as (figure not shown —but see below for a fitness landscape plot of a House of Cards model):

```
plot(additive)
```

The other extreme of the RMF model is the House of Cards model [14, 22, 35], that leads to maximally rugged fitness landscapes: in the House of Cards model the (log) fitness of each genotype is obtained, independently for each genotype, from some underlying distribution (normal, in this case):

```
## Set random number generator seed, for reproducibility
set.seed(5)
## HoC
hoc <- rfitness(4, c = 0, sd = 1)
epist_stats(hoc)

##   magn   sign rsign   none gamma
## 0.375 0.208 0.417 0.000 0.020
```

The plot of the fitness landscapes can be obtained as

```
plot(hoc)
```

and is shown in Figure 1.

The RMF model [1, 14, 28, 35] is a combination of an additive model (the “Mount Fuji” part, with the peak on the genotype of maximal fitness) and a House of Cards model (the random component). Thus, it has intermediate behavior in terms of epistasis:

```
set.seed(1)
rmf <- rfitness(4, c = 0.5, sd = 1)
## Fitness landscape plot not shown
## plot(rmf)
epist_stats(rmf)

##  magn  sign rsign  none gamma
## 0.333 0.500 0.167 0.000 0.512
```

Finally, among the models provided by OncoSimulR, the NK or LK model is Kauffman's model [4, 14, 20, 21, 35]: for a genotype of N loci, each locus has epistatic interactions with K other loci; the log-fitness of a genotype is the sum of the contributions of all its loci, where the contributions of each locus to the (log) fitness are random deviates from some probability distribution (here a uniform(0, 1)) that differ depending on the state of the given locus and all its K interacting loci. Thus, we go from minimal epistasis when $K = 1$ to maximal epistasis when $K = N - 1$ (which is equivalent to a House of Cards model [14]).

```
## Set random number generator seed, for reproducibility
set.seed(2)
## NK, K = 1
nk1 <- rfitness(4, K = 1, model = "NK")
## NK, K = 3
nk3 <- rfitness(4, K = 3, model = "NK")

## Fitness landscape plots not shown
## plot(nk1)
## plot(nk3)

epist_stats(nk1)

##  magn  sign rsign  none gamma
## 0.375 0.167 0.167 0.291 0.451

epist_stats(nk3)
```

```
##    magn    sign   rsign    none   gamma
##   0.125   0.375   0.500   0.000  -0.393
```

2.2.3 Specifying epistasis directly. Genes without interactions (no epistasis)

To better understand sections 2.2.4 to 2.2.7, we first provide a simple baseline example in which we specify a model without epistatic interactions:

```
## Four genes, with effects 0.05, -0.2, 0.1, 1.5
noInt <- allFitnessEffects(
    noIntGenes = c(A = 0.05, B = -0.2, C = 0.1, D = 1.5))

## Show the fitness of all genotypes
evalAllGenotypes(noInt, addwt = TRUE)

##       Genotype Fitness
## 1           WT  1.0000
## 2            A  1.0500
## 3            B  0.8000
## 4            C  1.1000
## 5            D  2.5000
## 6         A, B  0.8400
## 7         A, C  1.1550
## 8         A, D  2.6250
## 9         B, C  0.8800
## 10        B, D  2.0000
## 11        C, D  2.7500
## 12     A, B, C  0.9240
## 13     A, B, D  2.1000
## 14     A, C, D  2.8875
## 15     B, C, D  2.2000
## 16 A, B, C, D  2.3100
## Plot (not shown)
## plot(evalAllGenotypes(noInt, addwt = TRUE))
```

Computing epistasis statistics in these cases requires using the log of fitness, since we are using a multiplicative fitness specification: we specify the fitness effects of mutations using a multiplicative model —see also section 2.3.1—, $\prod(1 + s_i)$, where s_i is the fitness effect of gene or gene interaction i (thus we are explicitly modeling the effects of genes and gene interactions). Before we compute the epistasis specifications, however, we will check that fitness is what it should be under a multiplicative model for the contributions of genes (which is additive in the log scale):

```
all(evalAllGenotypes(noInt, addwt = TRUE)[, "Fitness"] ==
    c(1,
      1 + 0.05, ## A mutated
      1 - 0.2,  ## B mutated
      1 + 0.1,  ## ...
      1 + 1.5,  ## ...
      (1 + 0.05) * (1 - 0.2), ## A and B mutated
      (1 + 0.05) * (1 + 0.1), ## A and C mutated
      (1 + 0.05) * (1 + 1.5), ## A and D mutated
      (1 - 0.2)  * (1 + 0.1), ## B and C mutated
      (1 - 0.2)  * (1 + 1.5), ## B and D mutated
      (1 + 0.1)  * (1 + 1.5), ## C and D mutated
      (1 + 0.05) * (1 - 0.2) * (1 + 0.1), # A, B, C mutated
      (1 + 0.05) * (1 - 0.2) * (1 + 1.5), # A, B, D mutated
      (1 + 0.05) * (1 + 0.1) * (1 + 1.5), # A, C, D mutated
      (1 - 0.2)  * (1 + 0.1) * (1 + 1.5), # B, C, D mutated
      (1 + 0.05) * (1 - 0.2) * (1 + 0.1) * (1 + 1.5) # A, B,
                                                     ## C, D
                                                     ## mutated
      ))

## [1] TRUE
```

Now, compute epistasis statistics, first incorrectly setting `use_log = FALSE`:

```
## It incorrectly says there is magnitude epistasis
epist_stats(noInt, use_log = FALSE)

##  magn  sign rsign  none gamma
## 1.000 0.000 0.000 0.000 0.981
```

Now, set `use_log = TRUE`, to correctly compute the epistasis statistics, which shows there is no epistasis as all genes contribute additively in the log scale:

```
## Using logs shows there is no epistasis:
## all genes contribute additively
## in the log scale
epist_stats(noInt, use_log = TRUE)

##  magn  sign rsign  none gamma
##     0     0     0     1     1
```

2.2.4 Specifying epistasis directly. Two alternative specifications of epistasis

Suppose we want the effects of two genes and their interaction to be as shown in Table 1:

To make the example concrete, let $s_a = 0.2, s_b = 0.3, s_{ab} = 0.7$. We specify the above scenario as follows:

```
sa <- 0.2
sb <- 0.3
sab <- 0.7

e2 <- allFitnessEffects(epistasis =
                            c("A: -B" = sa,
                              "-A:B" = sb,
                              "A : B" = sab))
evalAllGenotypes(e2, addwt = TRUE)
```

```
##   Genotype Fitness
## 1       WT     1.0
## 2        A     1.2
## 3        B     1.3
## 4     A, B     1.7
```

Here we use a “-” to mean that we explicitly exclude a specific pattern; thus, “A:-B” is interpreted as “A mutated when B is not mutated”.

Alternatively, it is possible to specify the effects of genes and their interactions without using the “-”. This requires a different numerical value of the interaction, because now, as we are rewriting the interaction term as genotype “A mutated, B mutated”, the double mutant will incorporate the effects of “A mutated”, “B mutated”, and “both A and B mutated”. We can define a new s_2 that satisfies $(1 + s_{ab}) = (1 + s_a)(1 + s_b)(1 + s_2)$ so $(1 + s_2) = (1 + s_{ab})/((1 + s_a)(1 + s_b))$ and therefore we specify the model as:

```
s2 <- ((1 + sab)/((1 + sa) * (1 + sb))) - 1

e3 <- allFitnessEffects(epistasis =
                           c("A" = sa,
                             "B" = sb,
                             "A : B" = s2))
evalAllGenotypes(e3, addwt = TRUE)

##   Genotype Fitness
## 1       WT     1.0
## 2        A     1.2
## 3        B     1.3
## 4     A, B     1.7
```

For example, this is the way you would specify epistasis with FFPopsim [37]. Whether this specification or the previous one with “-” is simpler will depend on the model. For synthetic mortality and viability (see below, section 2.2.6 and 2.2.7), using “-” makes it simpler to map genotype tables to fitness effects.

Estimates of epistasis are the same regardless of how we specify the model (and note we continue using `use_log = TRUE` to obtain the epistasis statistics):

```
epist_stats(e2, use_log = TRUE)

##  magn  sign rsign  none gamma
##  1.00  0.00  0.00  0.00  0.95

epist_stats(e3, use_log = TRUE)

##  magn  sign rsign  none gamma
##  1.00  0.00  0.00  0.00  0.95
```

2.2.5 Two alternative specifications of epistasis: a three gene example

To further illustrate the two mechanisms for epistasis specification, here we show a more complex three-gene example. We want to use the epistatic interactions where there is no epistasis between genes A and C, but there is epistasis between genes A and B, and between genes B and C, as shown in Table 2:

We can specify that model, providing numerical values for each s, as follows:

```
sa <- 0.1
sb <- 0.15
sc <- 0.2
sab <- 0.3
sbc <- -0.25
sabc <- 0.4

sac <- (1 + sa) * (1 + sc) - 1

E3A <- allFitnessEffects(epistasis =
                            c("A:-B:-C" = sa,
                              "-A:B:-C" = sb,
                              "-A:-B:C" = sc,
                              "A:B:-C" = sab,
                              "-A:B:C" = sbc,
                              "A:-B:C" = sac,
                              "A : B : C" = sabc)
                                                )

evalAllGenotypes(E3A, addwt = TRUE)

##   Genotype Fitness
## 1       WT    1.00
## 2        A    1.10
## 3        B    1.15
## 4        C    1.20
## 5     A, B    1.30
## 6     A, C    1.32
## 7     B, C    0.75
## 8  A, B, C    1.40
```

We needed to pass the s_{ac} coefficient explicitly, even if that term is the product of the corresponding terms for the individual loci, because a full specification is required when using the "-".

We can use the alternative specification without "-", but we will need to perform some calculations to obtain some of the coefficients under this parameterization. To make it easier to tell the differences from the previous specification, I use capital "S" in what follows where the numerical values differ from the previous specification. Note that we can avoid specifying "A:C", as it just follows from the individual "A" and "C" terms, but we need to obtain new S_{ab}, S_{bc}, S_{abc}:

```
sa <- 0.1
sb <- 0.15
sc <- 0.2
sab <- 0.3
Sab <- ( (1 + sab)/((1 + sa) * (1 + sb))) - 1
```

```
Sbc <- ( (1 + sbc)/((1 + sb) * (1 + sc))) - 1
Sabc <- ( (1 + sabc)/
          ( (1 + sa) * (1 + sb) * (1 + sc) *
            (1 + Sab) * (1 + Sbc) ) ) - 1

E3B <- allFitnessEffects(epistasis =
                             c("A" = sa,
                               "B" = sb,
                               "C" = sc,
                               "A:B" = Sab,
                               "B:C" = Sbc,
                               ## "A:C" = sac, ## not needed now
                               "A : B : C" = Sabc)
                                                )
evalAllGenotypes(E3B, addwt = TRUE)

##   Genotype Fitness
## 1       WT    1.00
## 2        A    1.10
## 3        B    1.15
## 4        C    1.20
## 5     A, B    1.30
## 6     A, C    1.32
## 7     B, C    0.75
## 8  A, B, C    1.40
```

The actual fitness are the same:

```
all(evalAllGenotypes(E3A,  addwt = TRUE) ==
    evalAllGenotypes(E3B,  addwt = TRUE))

## [1] TRUE
```

Epistasis statistics are the same:

```
epist_stats(E3A, use_log = TRUE)

##  magn  sign rsign  none gamma
## 0.333 0.333 0.167 0.167 0.036

epist_stats(E3B, use_log = TRUE)

##  magn  sign rsign  none gamma
## 0.333 0.333 0.167 0.167 0.036
```

We can check the output from the above epistasis calculations using the graphical procedure for determining magnitude, sign, and reciprocal sign epistasis described in [7, 14]. The two cases with magnitude epistasis (frequency of $2/6$) correspond to the sets “abc”, “aBc”, “Abc”, “ABc” on the one hand and “Abc”, “AbC”, “ABc”, “ABC” on the other (where a capital letter denotes that the given locus is mutated, e.g., “AbC” denotes the genotype with both A and C mutated). The two cases with sign epistasis correspond to the two sets “abC”, “aBC”, “AbC”, “ABC” and “aBc”, “ABC”, “ABc”, “ABC”. The case with reciprocal sign epistasis (frequency of $1/6$) to the set “abc”, “aBc”, “abC”, “aBC”. Finally, the value for “none” (frequency of $1/6$) corresponds to the set of four genotypes “abc”, “Abc”, “abC”, and “AbC” .

2.2.6 Synthetic viability

Synthetic viability, where each individual mutant is lethal or has decreased fitness but the double mutant is viable, is just a case of reciprocal sign epistasis, but we illustrate it here separately with a minimal example. Suppose we want to model fitness as shown in Table 3:

We will set $s = 0.2$, and $s_a = s_b = -0.5$. Then, we can specify fitness as follows:

```
sa <- sb <- -0.5
s <- 0.2
sv <- allFitnessEffects(epistasis = c("-A : B" = sa,
                                      "A : -B" = sb,
                                      "A:B" = s))
evalAllGenotypes(sv, addwt = TRUE)

##   Genotype Fitness
## 1       WT     1.0
## 2        A     0.5
## 3        B     0.5
## 4     A, B     1.2

epist_stats(sv, use_log = TRUE)

##   magn   sign  rsign   none  gamma
##  0.000  0.000  1.000  0.000 -0.973
```

2.2.7 Synthetic sickness, synthetic lethality or synthetic mortality

Synthetic sickness and synthetic lethality or synthetic mortality are another case of reciprocal sign epistasis. Here, each single mutant leads to an increase in fitness but the double mutant leads to a decrease in fitness, including possible non-viability of the double mutant (synthetic lethality or synthetic mortality). A very simple case is shown in Table 4:

We will make $s_a = 0.1$, $s_b = 0.2$, $s_{ab} = -0.8$. We can specify it as:

```
sa <- 0.1
sb <- 0.2
sab <- -0.8
sm1 <- allFitnessEffects(epistasis = c("-A : B" = sb,
                             "A : -B" = sa,
                             "A:B" = sab))
evalAllGenotypes(sm1, addwt = TRUE)

##   Genotype Fitness
## 1       WT     1.0
## 2        A     1.1
## 3        B     1.2
## 4     A, B     0.2

epist_stats(sm1, use_log = TRUE)
```

```
##    magn    sign   rsign    none   gamma
##   0.000   0.000   1.000   0.000  -0.156
```

2.3 Simulating evolution

The main function for simulating evolution with OncoSimulR is `oncoSimulIndiv`. In addition, functions `oncoSimulPop` and `oncoSimulSample` allow us to run multiple simulations and in particular `oncoSimulPop` allows us to use multiple chores to parallelize execution.

Once epistasis or, equivalently, fitness of genotypes has been specified, as explained above (section 2.2), we can simulate evolution using any of the `oncoSimul*` functions. Before actually running simulations (section 2.3.3), though, we need to decide about the growth model and mutation rates of genes.

2.3.1 Growth models

OncoSimulR uses a continuous time model. The main choice for growth models is between a model with exponential growth or a model with carrying capacity (the model of McFarland et al [24–26]) (but see also **Note 4**). In both cases, when we specify the fitness effects of genes and gene interactions, and as shown above (sections 2.2.4 to 2.2.7), we evaluate fitness using the usual [2, 8, 18, 37] multiplicative model: fitness is $\prod(1 + s_i)$ where s_i is the fitness effect of gene (or gene interaction) i. In both models this fitness refers to the growth rate. The original model of McFarland et al. [26] has a slightly different parameterization, but you can go easily from one to the other (see below, "Birth rate parameterization in the model with carrying capacity"). If you specify fitness of genotypes directly (sections 2.2.1 and 2.2.2) then that is also taken as the birth rate of genotypes.

In the model with exponential growth we specify the growth rate, fixing death rate at 1 (it is possible to modify this if really needed, but there is rarely any need to do so). The model with carrying capacity follows the model of McFarland et al [24–26]: mutations affect the birth rate, with the death rate being density dependent (see below).

In OncoSimulR, we choose the exponential growth model setting `model = "Exp"` in the call to functions `oncoSimul*`. The model with carrying capacity is specified using `model = "McFL"`. Note that even if the `McFL` shows density dependence, there is no frequency-dependence of fitness in any of the models (but see **Note 5**).

Death rate in the model with carrying capacity. For death rate, we use the expression that McFarland et al. ([26], see p. 2911) use "(. . .) for large cancers (grown to 10^6 cells)": $D(N) = \log(1 + N/K)$ where K is the initial equilibrium population size. As the authors explain, for large N/K the above expression "(. . .) recapitulates Gompertzian dynamics observed experimentally for large tumors". By default, OncoSimulR uses a value of $K = initSize/(e^1 - 1)$ so that the starting population (which starts with population size $= initSize$) is at equilibrium.

Birth rate parameterization in the model with carrying capacity. For the birth rate, in the original model in McFarland et al. [26], the effects of drivers contribute to the numerator of the birth rate, and those of the (deleterious) passengers to the denominator as: $\frac{(1+s)^d}{(1+s_p)^p}$, where d and p are, respectively, the total number of drivers and passengers in a genotype, and the fitness effects of all drivers are the same (s) and that of all passengers the same (s_p). Note that, as written above, and as explicitly mentioned in McFarland et al. ([26] p. 2911, and [24], p. 9) "(. . .) s_p is the fitness disadvantage conferred by a passenger". In other words, the larger the s_p the more deleterious the passenger. As explained above, however, we use a multiplicative model $\prod(1 + s_i)$, where genes and their interactions can have arbitrary positive or negative effects (as we saw in sections 2.2.4 to 2.2.7). Thus, if one is given a model specified as in the parameterization $\frac{(1+s)^d}{(1+s_p)^p}$ one would simply need to rewrite the appropriate term for the s_p as $(1 + s_i) = -s_p/(1 + s_p)$, for the s_i in our parameterization.

2.3.2 Mutation rates, mutator genes

OncoSimulR can use both a common mutation rate for all loci as well as locus-specific mutation rates. Below we show two calls to `oncoSimulIndiv`, the first one, `rmf_common`, with a common mutation rate of $1e - 7$ for all loci, and the second, `rmf_loci_spec`, with different mutation rates for each locus. We reuse the RMF fitness specification from section 2.2.2.

```r
## We reuse the fitnessEffects object below
rmf_fe <- allFitnessEffects(genotFitness = rmf)


## simulation using a common mutation rate
rmf_common <- oncoSimulIndiv(rmf_fe, mu = 1e-7,
                             onlyCancer = FALSE)
```

```
## Vector of locus-specific mutation rates
mus <- c("A" = 1e-9, "B" = 1e-7,
         "C" = 2e-7, "D" = 5e-3)
## simulate with locus-specific mutation rates
rmf_loci_spec <- oncoSimulIndiv(rmf_fe,
                                mu = mus,
                                onlyCancer = FALSE)
```

It is also possible to specify mutator/antimutator genes (e.g. [15, 36]); these genes, when mutated, lead to an increase/decrease in the mutation rate across the genome. Mutator/antimutator genes must be a subset of the genes in the fitness effects (if you want to use mutator genes that have no direct fitness effects, give them a fitness effect of 0 —see examples in the documentation). In the example below, we specify that mutating gene "A" leads to an increase by a factor of fifty of the mutation rate. See also **Note 6** for numerical issues that can result from using multiple mutator genes.

```
mutg <- allMutatorEffects(noIntGenes = c("A" = 50))
rmf_loci_spec_mut <- oncoSimulIndiv(rmf_fe,
                                    muEF = mutg,
                                    mu = mus,
                                    onlyCancer = FALSE)
```

2.3.3 Running simulations until pre-specified conditions are met

In addition to the growth model, fitness effects, and possible mutator effects and locus-specific mutation rates, you need to decide:

- Where simulations will start from. This involves deciding the initial population size (argument `initSize`); sometimes you might want to start the simulations from a specific genotype (see **Note 7**).
- When will simulations stop: how long to run simulations, and whether or not to require simulations to reach a particular condition. This is covered below.

OncoSimulR provides very flexible ways to decide when to stop a simulation:

- Using option **`onlyCancer = TRUE`**.

  A simulation will be repeated until any one of the conditions below is met, if this happens before the simulation reaches `finalTime`. Because OncoSimulR was originally developed to simulate cancer evolution, this is often referred as "reaching cancer" but we can refer to it as "reach whatever interests me". These conditions are:

  - Total population size becomes larger than `detectionSize`.
  - A gene, gene combination, or genotype among those listed in `fixation` becomes fixed in the population (i.e., has a frequency of 1 or very close to 1); see sections 2.3.4, 2.3.5, and 2.3.6.
  - The tumor is detected according to a stochastic detection mechanism, where the probability of "detecting the tumor" increases with population size; see section 2.3.7.
  - The number of drivers in any one genotype becomes equal to, or larger than, `detectionDrivers` (this could also be used to stop the simulation as soon as a **specific genotype** is found, by using the genes that make that genotype as the drivers, but the mechanism in section 2.3.5 is generally simpler). This option is only reasonable in scenarios where we want to differentiate between driver and passenger genes, and it requires specifying driver genes; this will not be further discussed here.

  The simulations exit as soon as any of the exiting conditions is reached; therefore, if you only care about one condition, set the other conditions to `NA`.

- **`onlyCancer = FALSE`**.

  A simulation will run only once, and will exit as soon as any of the above conditions are met or as soon as the total population size becomes zero or we reach `finalTime`.

  Using `onlyCancer = FALSE` will often be the setting you want to use to examine general population genetics scenarios without focusing on possible sampling issues; set `finalTime` to the value you want and set `onlyCancer = FALSE`; in addition, set `detectionProb` to "NA" and `detectionDrivers` and `detectionSize` to "NA" or to huge numbers. This way you simply collect the simulation output at the end of the run, regardless of what happened with the population (it became extinct, it did not reach a large size, it did not accumulate drivers, etc).

Under the `onlyCancer = TRUE` case, if we reach `finalTime` (or the population size becomes zero) before any of the "reach cancer" conditions have been fulfilled, the simulation will be repeated again, within the limits given by the following parameters to the functions `oncoSimulIndiv` or `oncoSimulPop`: `max.wall.time`: the total wall time we allow an individual simulation to run; `max.num.tries`: the maximum number of times we allow a simulation to be repeated to reach cancer; if you use `oncoSimulSample`, `max.wall.time.total` and `max.num.tries.total`, similar to the previous two, but specific for function `oncoSimulSample`, are also of application. If the specified conditions for "reaching cancer" can not be met, no object with the population state (genotypes and population sizes) will be returned (i.e., simulations will abort without returning the population state, as no simulation has achieved the specified conditions).

2.3.4 Fixation of genes and gene combinations

Simulations will exit when any of the genes or gene combinations in the vector (or list) `fixation`, passed to the `oncoSimul*` functions reaches a frequency of 1, or very close to 1 (see section 2.3.6). The gene combinations might share genes (i.e., might have non-zero intersection). As explained above, if we want simulations to only exit when fixation of those genes/gene combinations is reached, we will set all other stopping conditions to `NA`. Note that if the stopping conditions can never be reached, simulations will eventually abort (e.g., when `max.wall.time` or `max.num.tries` are reached). Since we are running simulations until fixation of genes, the `Exp` model will rarely be appropriate here: the `McFL` model, that includes competition, is more appropriate.

The following code shows an example based in the model in Ochs and Desai [29]; the authors present a model like the one shown in Figure 2 (the numerical values are arbitrarily set by me). In this model $s_u > 0$, $s_v > s_u$, $s_i < 0$ and we can only arrive at v from i. Mutants "ui" and "uv" can never appear as their fitness is 0, or $-\infty$, so $s_{ui} = s_{uv} = -1$ (or $-\infty$).

We can specify fitness by specifying epistatic effects:

```
u <- 0.1
i <- -0.05
vi <- (1.2/0.95) - 1
ui <- uv <- -Inf

od2 <- allFitnessEffects(
    epistasis = c("u" = u,  "u:i" = ui,
                  "u:v" = uv, "i" = i,
                  "v:-i" = -Inf, "v:i" = vi))
evalAllGenotypes(od2, addwt = TRUE)

##   Genotype Fitness
## 1       WT    1.00
## 2        i    0.95
## 3        u    1.10
## 4        v    0.00
## 5     i, u    0.00
## 6     i, v    1.20
## 7     u, v    0.00
## 8  i, u, v    0.00
```

In p.2, section “Simulations” of Ochs and Desai [29], they explain that “Each simulated population was evolved until either the uphill genotype or valley-crossing genotype fixed.” To use the same procedure here, we specify that we want to end the simulation when either the “u” or the “v, i” genotypes have reached fixation, by passing those genotype combinations as the `fixation` argument (in this example using `fixation = c("u", "v")` would have been equivalent, since the “v” genotype by itself has fitness of 0). Fixation will be the one and only condition for ending the simulations, and thus we set arguments `detectionDrivers`, `finalTime`, `detectionSize` and `detectionProb` explicitly to `NA`. We want to run the simulations multiple times, so we use `oncoSimulPop` but we set the number of replicates to only 10 for the sake of speed in this example (a much larger number of replicates would be required for real cases):

```
initS <- 20
## We use only a small number of repetitions for the sake
## of speed.
od100 <- oncoSimulPop(10, od2,
                      fixation = c("u", "v, i"),
                      model = "McFL",
                      mu = 1e-4,
                      detectionDrivers = NA,
                      finalTime = NA,
                      detectionSize = NA,
                      detectionProb = NA,
                      onlyCancer = TRUE,
                      initSize = initS,
                      mc.cores = 2)
```

What is the frequency of each genotype among the simulations? (or, what is the frequency of fixation of each genotype?)

```
sampledGenotypes(samplePop(od100))

##
##  Subjects by Genes matrix of 10 subjects and 3 genes.

##   Genotype Freq
## 1     i, v    3
## 2        u    7
##
##  Shannon's diversity (entropy) of sampled genotypes: 0.6108643
```

Note the variability in time to reach fixation

```
head(summary(od100)[, c(1:3, 8:9)])

##   NumClones TotalPopSize LargestClone FinalTime NumIter
## 1         3           18           18 11759.300  470432
## 2         3           21           21 11860.700  474501
## 3         3           26           26  7941.800  317713
## 4         3           13           13  3675.175  147032
## 5         3           20           20   727.875   29120
## 6         2           18           18    47.450    1899
```

2.3.5 Fixation of genotypes

Suppose you are dealing with a five loci genotype and suppose that you want to stop the simulations only if genotypes “A”, “B, E”, or “A, B, C, D, E” reach fixation. You do not want to stop if, say, genotype “A, B, E” reaches fixation: the mechanism in section 2.3.4 would not be useful here. To specify genotypes, you prepend the genotype combinations with a "_,", and that tells OncoSimulR that you want fixation of **genotypes**, not just gene combinations.

The following example illustrates the differences between the two mechanisms:

```
## Create a simple fitness landscape
rl1 <- matrix(0, ncol = 6, nrow = 9)
colnames(rl1) <- c(LETTERS[1:5], "Fitness")
rl1[1, 6] <- 1
rl1[cbind((2:4), c(1:3))] <- 1
rl1[2, 6] <- 1.4
rl1[3, 6] <- 1.32
rl1[4, 6] <- 1.32
rl1[5, ] <- c(0, 1, 0, 0, 1, 1.5)
rl1[6, ] <- c(0, 0, 1, 1, 0, 1.54)
rl1[7, ] <- c(1, 0, 1, 1, 0, 1.65)
rl1[8, ] <- c(1, 1, 1, 1, 0, 1.75)
rl1[9, ] <- c(1, 1, 1, 1, 1, 1.85)
class(rl1) <- c("matrix", "genotype_fitness_matrix")
## plot(rl1) ## to see the fitness landscape
```

```r
## Gene combinations
local_max_g <- c("A", "B, E", "A, B, C, D, E")

## Specify the genotypes
local_max <- paste0("_,", local_max_g)
## show how it looks
local_max

## [1] "_,A"             "_,B, E"          "_,A, B, C, D, E"

fr1 <- allFitnessEffects(genotFitness = rl1)
initS <- 2000

######## Stop on gene combinations   #####
r1 <- oncoSimulPop(10,
                       fp = fr1,
                       model = "McFL",
                       initSize = initS,
                       mu = 1e-4,
                       detectionSize = NA,
                       sampleEvery = .03,
                       keepEvery = 1,
                       finalTime = 50000,
                       fixation = local_max_g,
                       detectionDrivers = NA,
                       detectionProb = NA,
                       onlyCancer = TRUE,
                       max.num.tries = 500,
                       max.wall.time = 20,
                       errorHitMaxTries = TRUE,
                       keepPhylog = FALSE,
                       mc.cores = 2)
```

```
sp1 <- samplePop(r1, "last", "singleCell")

##
##  Subjects by Genes matrix of 10 subjects and 5 genes.

## Show the frequency of final composition of the populations
sampledGenotypes(sp1)

##     Genotype Freq
## 1          A    6
## 2 A, B, C, D    2
## 3       B, E    2
##
##  Shannon's diversity (entropy) of sampled genotypes: 0.9502705

####### Stop on genotypes    ####

r2 <- oncoSimulPop(10,
                        fp = fr1,
                        model = "McFL",
                        initSize = initS,
                        mu = 1e-4,
                        detectionSize = NA,
                        sampleEvery = .03,
                        keepEvery = 1,
                        finalTime = 50000,
                        fixation = local_max,
                        detectionDrivers = NA,
                        detectionProb = NA,
                        onlyCancer = TRUE,
                        max.num.tries = 500,
                        max.wall.time = 20,
                        errorHitMaxTries = TRUE,
                        keepPhylog = FALSE,
                        mc.cores = 2)

 ## All final genotypes should be local maxima
 sp2 <- samplePop(r2, "last", "singleCell")

 ##
 ##  Subjects by Genes matrix of 10 subjects and 5 genes.

 ## Show the frequency of final composition of the populations
 sampledGenotypes(sp2)

 ##         Genotype Freq
 ## 1              A    4
 ## 2 A, B, C, D, E    2
 ## 3           B, E    4
 ##
 ##  Shannon's diversity (entropy) of sampled genotypes: 1.05492
```

2.3.6 Fixation: tolerance, number of periods, minimal size

When stopping simulations on fixation of genes, gene combinations, and genotypes, you need to consider three additional parameters: `fixation_tolerance`, `min_successive_fixation`, and `fixation_min_size`.

`fixation_tolerance`: fixation is considered to have happened if the genotype/gene combinations specified as genotypes/gene combinations for fixation have reached a frequency $> 1 - fixation\_tolerance$. (The default is 0, so we ask for genotypes/gene combinations with a frequency of 1, which might not be what you want with large mutation rates and complex fitness landscapes with genotypes of similar fitness.)

`min_successive_fixation`: during how many successive sampling periods the conditions of fixation need to be fulfilled before declaring fixation. These must be successive sampling periods without interruptions (i.e., a single period when the condition is not fulfilled will set the counter to 0). This can help to exclude short, transitional, local maxima that are quickly replaced by other genotypes. (The default is 50, but this is probably too small for "real life" usage).

`fixation_min_size`: you might only want to consider fixation to have happened if a minimal size has been reached; this can help eliminate local maxima that have fitness that is barely above that of the wild-type genotype. (The default is 0).

2.3.7 Stochastic detection mechanism

This process is controlled by the argument `detectionProb`. Under this process, we simulate stopping a simulation when a tumor is detected, where the probability of "tumor detection" increases with the total population size. The probability of detection is given by

$$P(N) = \begin{cases} 1 - e^{-c((N-B)/B)} & \text{if } N > B \\ 0 & \text{if } N \leq B \end{cases} \tag{1}$$

where $P(N)$ is the probability that a tumor with a population size N will be detected, and c (argument $cPDetect$ in the `oncoSimul*` functions) controls how fast $P(N)$ increases with increasing population size relative to a baseline, B ($PDBaseline$ in the `oncoSimul*` functions). This function is evaluated at regularly spaced times during the simulation, and the decision to exit the simulation is made by comparing $P(N)$ against a random uniform number. Using this exiting mechanism is probably only appropriate for modeling diseases such as cancer and will not be further discussed here. See the vignette and documentation for details and examples.

3 Output and data analysis

The output from the simulation functions `oncoSimulIndiv`, `oncoSimulPop`, and `oncoSimulSample` are lists (see details in the documentation). These lists contain, among other components, the state of the population (genotypes and number of cells) at the time of stopping the simulation and, for `oncoSimulIndiv` and `oncoSimulPop`, all other previous sampling times. What users do with the output will be completely dependent on the research question. Some of the questions that can be addressed with the output from OncoSimulR include:

- Effects of sign epistasis in the probability and time to cross fitness valleys. We have provided small examples in sections 2.3.4 and 2.3.5.

- The predictability of evolution in complex fitness landscapes, as shown in [13, 19].

- The effects of mutator/antimutator genes in reaching particular genotypes or population sizes.
- Whether we can recover restrictions in the order of accumulation of mutations with different types of epistatic relationships, as shown in [10, 12].

Simple examples illustrating those scenarios are provided in the vignette of the OncoSimulR package.

4 Notes: potential pitfalls and troubleshooting

1. It is also possible to specify epistasis using Directed Acyclic Graphs (DAGs) that represent order dependencies in the accumulation of mutations. This is equivalent to specifying sign epistasis [12], and it is used by "cancer progression models" such as Conjunctive Bayesian Networks (CBN) [16, 17, 27], oncogenetic trees (OT) [9, 34], CAncer PRogression Inference (CAPRI) [5, 31], or CAncer PRogression Extraction with Single Edges (CAPRESE) [30]. This usage, however, can only represent sign epistasis, or relaxations of sign epistasis; see the vignette of the OncoSimulR package for examples.

2. OncoSimulR also allows us to specify fitness not in terms of genes but in terms of modules. This can be useful in some scenarios as discussed in, for example, [6, 32]. Each module is a set of genes (and the intersection between modules is the empty set). Modules, then, play the role of a "union operation" over sets of genes. There is no major conceptual difference relative to what has been shown in this chapter, but one also needs to specify which genes belong to each module. This specification of fitness can be useful when using DAGs (as discussed in **Note 1**), but rarely in other scenarios.

3. It is also possible to use additive models where the contribution of each mutated allele i to the log-fitness is s_i, where s_i is a random deviate from a Normal distribution with user-specified mean and standard deviation, and the log-fitness of a genotype is the sum of the contributions of each mutated allele. This can be obtained using `model = "Additive"` in function `rfitness` with versions of OncoSimulR 2.17.7 and higher.

4. It is also possible to use an exponential growth model with birth rate fixed to 1, and where the fitness specification affects the death rate, a model inspired in [3]. Specification of

fitness effects via their effects on death rates, however, often leads to numerical issues (see documentation and vignette), and is not discussed in this paper.

5. A branch of OncoSimulR, `https://github.com/rdiaz02/OncoSimul/tree/freq-dep-fitness` , includes an implementation with frequency-dependent fitness. This implementation includes all the features mentioned here, but also allows users to make fitness depend on the frequency of other genotypes. We have not mentioned these features as they extend beyond the specification of epistasis and the software requires users to carry out manual installation of software, until a new R building toolchain becomes stable.

6. If we use several mutator genes with independent effects it is easy to run into computational problems. Suppose we specify five mutator genes, each with an effect of multiplying by 50 the mutation rate. The genotype with all those five genes mutated will have an increased mutation rate of $50^5 = 312500000$. If you set the mutation rate to the default of $1e - 6$ you will have a mutation rate of 312 which makes no sense (and leads to several numerical problems and an early warning from the software).

7. It is possible to start simulations from a specific genotype. This can be done using the `initMutant` argument to the `oncoSimul*` functions.

Acknowledgments

Four anonymous reviewers for comments that improved the ms.

Funding

Supported by BFU2015-67302-R (MINECO/FEDER, EU) to RDU.

References

[1] Aita, T., Uchiyama, H., Inaoka, T., Nakajima, M., Kokubo, T., Husimi, Y., 2000. Analysis of a local fitness landscape with a model of the rough Mt. Fuji-type landscape: Application to prolyl endopeptidase and thermolysin. *Biopolymers*, **54(1)**:64–79. doi: 10.1002/(SICI)1097-0282(200007)54:1<64::AID-BIP70>3.0.CO;2-R. URL `https://`

onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0282%28200007%2954%3A1%3C64%3A%3AAID-BIP70%3E3.0.CO%3B2-R.

[2] Beerenwinkel, N., Eriksson, N., Sturmfels, B., 2007. Conjunctive Bayesian networks. *Bernoulli*, **13(4)**:893–909. doi:10.3150/07-BEJ6133. URL http://projecteuclid.org/euclid.bj/1194625594.

[3] Bozic, I., Antal, T., Ohtsuki, H., Carter, H., Kim, D., Chen, S., Karchin, R., Kinzler, K. W., Vogelstein, B., Nowak, M. A., 2010. Accumulation of driver and passenger mutations during tumor progression. *Proceedings of the National Academy of Sciences of the United States of America*, **107**:18545–18550. doi:10.1073/pnas.1010978107. URL http://www.ncbi.nlm.nih.gov/pubmed/20876136.

[4] Brouillet, S., Annoni, H., Ferretti, L., Achaz, G., 2015. MAGELLAN: A tool to explore small fitness landscapes. *bioRxiv*, p. 031583. doi:10.1101/031583. URL http://biorxiv.org/content/early/2015/11/13/031583.

[5] Caravagna, G., Graudenzi, A., Ramazzotti, D., Sanz-Pamplona, R., Sano, L. D., Mauri, G., Moreno, V., Antoniotti, M., Mishra, B., 2016. Algorithmic methods to infer the evolutionary trajectories in cancer progression. *PNAS*, **113(28)**:E4025–E4034. doi:10.1073/pnas.1520213113. URL http://www.pnas.org/content/113/28/E4025.

[6] Constantinescu, S., Szczurek, E., Mohammadi, P., Rahnenführer, J., Beerenwinkel, N., 2015. TiMEx: A waiting time model for mutually exclusive cancer alterations. *Bioinformatics*. doi:10.1093/bioinformatics/btv400.

[7] Crona, K., Greene, D., Barlow, M., 2013. The peaks and geometry of fitness landscapes. *Journal of Theoretical Biology*, **317**:1–10. doi:10.1016/j.jtbi.2012.09.028. URL http://www.sciencedirect.com/science/article/pii/S0022519312005061.

[8] Datta, R. S., Gutteridge, A., Swanton, C., Maley, C. C., Graham, T. A., 2013. Modelling the evolution of genetic instability during tumour progression. *Evolutionary applications*, **6(1)**:20–33. doi:10.1111/eva.12024. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3567468&tool=pmcentrez&rendertype=abstract.

[9] Desper, R., Jiang, F., Kallioniemi, O. P., Moch, H., Papadimitriou, C. H., Schäffer, A. A., 1999. Inferring tree models for oncogenesis from comparative genome hybridization data. *J Comput Biol*, **6(1)**:37–51. URL http://view.ncbi.nlm.nih.gov/pubmed/10223663.

[10] Diaz-Uriarte, R., 2015. Identifying restrictions in the order of accumulation of mutations during tumor progression: Effects of passengers, evolutionary models, and sampling. *BMC Bioinformatics*, **16(41)**. doi:doi:10.1186/s12859-015-0466-7. URL http://www.biomedcentral.com/1471-2105/16/41/abstract.

[11] Diaz-Uriarte, R., 2017. OncoSimulR: Genetic simulation with arbitrary epistasis and mutator genes in asexual populations. *Bioinformatics*, **33(12)**:1898–1899. doi:10.1093/bioinformatics/btx077. URL https://academic.oup.com/bioinformatics/article/33/12/1898/2982052/OncoSimulR-genetic-simulation-with-arbitrary.

[12] Diaz-Uriarte, R., 2018. Cancer progression models and fitness landscapes: A many-to-many relationship. *Bioinformatics*, **34(5)**:836–844. doi:10.1093/bioinformatics/btx663. URL https://academic.oup.com/bioinformatics/article/34/5/836/4557185.

[13] Diaz-Uriarte, R., Vasallo, C., 2019. Every which way? On predicting tumor evolution using cancer progression models. *PLOS Computational Biology*, **15(8)**:e1007246. doi:10.1371/journal.pcbi.1007246. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007246.

[14] Ferretti, L., Schmiegelt, B., Weinreich, D., Yamauchi, A., Kobayashi, Y., Tajima, F., Achaz, G., 2016. Measuring epistasis in fitness landscapes: The correlation of fitness effects of mutations. *Journal of Theoretical Biology*, **396**:132–143. doi:10.1016/j.jtbi.2016.01.037. URL http://www.sciencedirect.com/science/article/pii/S0022519316000771.

[15] Gerrish, P. J., Colato, A., Perelson, A. S., Sniegowski, P. D., 2007. Complete genetic linkage can subvert natural selection. *Proc. Natl. Acad. Sci. U.S.A.*, **104(15)**:6266–6271. doi:10.1073/pnas.0607280104.

[16] Gerstung, M., Baudis, M., Moch, H., Beerenwinkel, N., 2009. Quantifying cancer progression with conjunctive Bayesian networks. *Bioinformatics*, **25(21)**:2809–2815. doi:10.1093/bioinformatics/btp505. URL http://dx.doi.org/10.1093/bioinformatics/btp505http://www.bsse.ethz.ch/cbg/software/ct-cbn.

[17] Gerstung, M., Eriksson, N., Lin, J., Vogelstein, B., Beerenwinkel, N., 2011. The Temporal Order of Genetic and Pathway Alterations in Tumorigenesis. *PLoS ONE*, **6(11)**:e27136. doi:10.1371/journal.pone.0027136. URL http://dx.plos.org/10.1371/journal.pone.0027136http://www.bsse.ethz.ch/cbg/software/ct-cbn.

[18] Gillespie, J. H., 1993. Substitution processes in molecular evolution. I. Uniform and clustered substitutions in a haploid model. *Genetics*, **134(3)**:971–981.

[19] Hosseini, S.-R., Diaz-Uriarte, R., Markowetz, F., Beerenwinkel, N., 2019. Estimating the predictability of cancer evolution. *Bioinformatics*, **35(14)**:i389–i397. doi:10.1093/bioinformatics/btz332. URL https://academic.oup.com/bioinformatics/article/35/14/i389/5529151.

[20] Kauffman, S. A., 1993. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, U.S.A., New York, 1 edition edition. ISBN 978-0-19-507951-7.

[21] Kauffman, S. A., Weinberger, E. D., 1989. The NK model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology*, **141(2)**:211–245. doi:10.1016/S0022-5193(89)80019-0. URL http://www.sciencedirect.com/science/article/pii/S0022519389800190.

[22] Kingman, J. F. C., 1978. A simple model for the balance between selection and mutation. *Journal of Applied Probability*, **15(1)**:1–12. doi:10.2307/3213231. URL https://www.cambridge.org/core/journals/journal-of-applied-probability/article/simple-model-for-the-balance-between-selection-and-mutation/26726A951C67C23ADC2240066887C1F1.

[23] Mather, W. H., Hasty, J., Tsimring, L. S., 2012. Fast stochastic algorithm for simulating evolutionary population dynamics. *Bioinformatics (Oxford, England)*, **28(9)**:1230–1238. doi:

10.1093/bioinformatics/bts130. URL http://www.ncbi.nlm.nih.gov/pubmed/22437850.

[24] McFarland, C., 2014. *The Role of Deleterious Passengers in Cancer.* Ph.D. thesis, Harvard University. URL http://nrs.harvard.edu/urn-3:HUL.InstRepos:13070047.

[25] McFarland, C., Mirny, L., Korolev, K. S., 2014. A tug-of-war between driver and passenger mutations in cancer and other adaptive processes. *Proc Natl Acad Sci U S A*, **111(42)**:15138–15143. doi:10.1101/003053. URL http://arxiv.org/pdf/1402.6354v1$backslash$nhttp://arxiv.org/abs/1402.6354.

[26] McFarland, C. D., Korolev, K. S., Kryukov, G. V., Sunyaev, S. R., Mirny, L. A., 2013. Impact of deleterious passenger mutations on cancer progression. *Proceedings of the National Academy of Sciences of the United States of America*, **110(8)**:2910–5. doi:10.1073/pnas.1213968110. URL http://www.ncbi.nlm.nih.gov/pubmed/23388632.

[27] Montazeri, H., Kuipers, J., Kouyos, R., Böni, J., Yerly, S., Klimkait, T., Aubert, V., Günthard, H. F., Beerenwinkel, N., Study, T. S. H. C., 2016. Large-scale inference of conjunctive Bayesian networks. *Bioinformatics*, **32(17)**:i727–i735. doi:10.1093/bioinformatics/btw459. URL http://bioinformatics.oxfordjournals.org/content/32/17/i727.

[28] Neidhart, J., Szendro, I. G., Krug, J., 2014. Adaptation in Tunably Rugged Fitness Landscapes: The Rough Mount Fuji Model. *Genetics*, **198(2)**:699–721. doi:10.1534/genetics.114.167668. URL http://www.genetics.org/content/198/2/699.

[29] Ochs, I. E., Desai, M. M., 2015. The competition between simple and complex evolutionary trajectories in asexual populations. *BMC Evolutionary Biology*, **15(1)**:1–9. doi:10.1186/s12862-015-0334-0. URL http://www.biomedcentral.com/1471-2148/15/55.

[30] Olde Loohuis, L., Caravagna, G., Graudenzi, A., Ramazzotti, D., Mauri, G., Antoniotti, M., Mishra, B., 2014. Inferring Tree Causal Models of Cancer Progression with Probability Raising. *PLOS ONE*, **9(10)**:e108358. doi:10.1371/journal.pone.

0108358. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0108358.

[31] Ramazzotti, D., Caravagna, G., Olde Loohuis, L., Graudenzi, A., Korsunsky, I., Mauri, G., Antoniotti, M., Mishra, B., 2015. CAPRI: Efficient inference of cancer progression models from cross-sectional data. *Bioinformatics*, **31(18)**:3016–3026. doi:10.1093/bioinformatics/btv296. URL https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btv296.

[32] Raphael, B. J., Vandin, F., 2015. Simultaneous Inference of Cancer Pathways and Tumor Progression from CrossSectional Mutation Data. *Journal of Computational Biology*, **22(00)**:250–264. doi:10.1089/cmb.2014.0161.

[33] Schoen, D. J., Schultz, S. T., 2019. Somatic Mutation and Evolution in Plants. *Annu. Rev. Ecol. Evol. Syst.*, **50(1)**:49–73. doi:10.1146/annurev-ecolsys-110218-024955. URL https://www.annualreviews.org/doi/10.1146/annurev-ecolsys-110218-024955.

[34] Szabo, A., Boucher, K. M., 2008. Oncogenetic trees. In Tan, W.-Y., Hanin, L., editors, *Handbook of Cancer Models with Applications*, pp. 1–24. World Scientific. URL http://www.worldscibooks.com/lifesci/6677.html.

[35] Szendro, I. G., Schenk, M. F., Franke, J., Krug, J., de Visser, J. A. G. M., 2013. Quantitative analyses of empirical fitness landscapes. *J. Stat. Mech.*, **2013(01)**:P01005. doi:10.1088/1742-5468/2013/01/P01005. URL http://stacks.iop.org/1742-5468/2013/i=01/a=P01005.

[36] Tomlinson, I. P., Novelli, M. R., Bodmer, W. F., 1996. The mutation rate and cancer. *Proc. Natl. Acad. Sci. U.S.A.*, **93(25)**:14800–14803.

[37] Zanini, F., Neher, R. A., 2012. FFPopSim: An efficient forward simulation package for the evolution of large populations. *Bioinformatics*, **28(24)**:3332–3333. doi:10.1093/bioinformatics/bts633. URL http://webdav.tuebingen.mpg.de/ffpopsim/http://bioinformatics.oxfordjournals.org/content/28/24/3332http://github.com/neherlab/ffpopsim/.

5 Figures

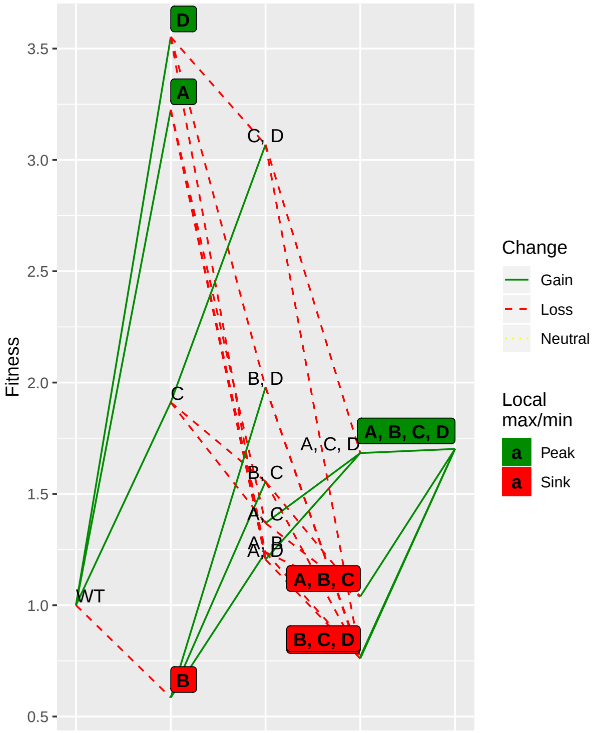

Figure 1: Plot of the fitness landscape for the House of Cards model in section 2.2.2.

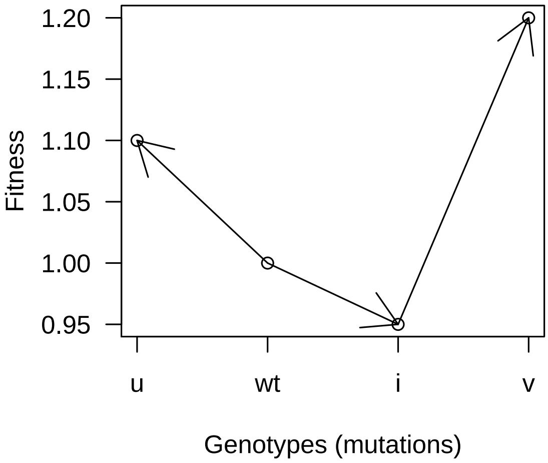

Figure 2: Model from Ochs and Desai [29]. Actual numerical fitness values arbitrarily set by me to conform to their figure. Redrawn from Figure 1.a in [29].

6 Tables

Table 1: Epistasis example for two genes. “wt”’ denotes wildtype and “M” denotes mutant.

A	B	Fitness
wt	wt	1
M	wt	$1 + s_a$
wt	M	$1 + s_b$
M	M	$1 + s_{ab}$

Table 2: A three-gene fitness specification with epistasis. “wt”’ denotes wildtype and “M” denotes mutant. Note that the mutant for exactly A and C has a fitness that is the product of the individual terms (so there is no epistasis in that case).

A	B	C	Fitness
wt	wt	wt	1
M	wt	wt	$1 + s_a$
wt	M	wt	$1 + s_b$
wt	wt	M	$1 + s_c$
M	M	wt	$1 + s_{ab}$
wt	M	M	$1 + s_{bc}$
M	wt	M	$(1 + s_a)(1 + s_c)$
M	M	M	$1 + s_{abc}$

Table 3: A simple synthetic viability example. “wt”’ denotes wildtype and “M” denotes mutant. $s > 0$, $s_a < 0$, $s_b < 0$.

A	B	Fitness
wt	wt	1
M	wt	$1 + s_a$
wt	M	$1 + s_b$
M	M	$1 + s$

Table 4: A simple synthetic sickness example. “wt”' denotes wildtype and “M” denotes mutant. $s_a > 0$, $s_b > 0$, $s_{ab} < 0$.

A	B	Fitness
wt	wt	1
M	wt	$1 + s_a$
wt	M	$1 + s_b$
M	M	$1 + s_{ab}$