An introduction to gaucho

Alex Murison
Alexander.Murison@icr.ac.uk

Christopher P Wardell
Christopher.Wardell@icr.ac.uk

October 17, 2016

This vignette serves as an introduction to the R package gaucho. It covers the basic use cases and usage of the package, explaining the input and output and contains several worked examples. If you use this package, please cite it[**gauchoR**] and/or a recent paper where it was used[**melchor**]. A BibTeX entry for LaTeX users is:

```
@Manual{gauchoR,
    title = {GAUCHO: Genetic Algorithms for Understanding Clonal Heterogeneity and Ordering},
    author = {A. Murison and C. P. Wardell},
    note = {R package},
}
```

Installation: The latest stable version can be installed from Bioconductor like so:

```
## Install
source("http://bioconductor.org/biocLite.R")
biocLite("gaucho")
## Load
library(gaucho)
```

The latest development version can be cloned from GitHub but must be built from source:

https://github.com/MurisonWardell/gaucho

Dependencies: Please note that gaucho depends on the following packages which are either available from CRAN or part of base R, but these should be automatically downloaded when Gaucho is installed using biocLite(): GA[**scrucca**], compiler[**baseR**], heatmap.plus[**heatmapplusR**], png[**pngR**], as well as graph[**graphR**].

Contents

1 Overview

1.1 Introduction

We work extensively with sequencing data, particularly next-generation sequencing data from tumour-normal pairs. The tumour genome contains many single nucleotide variants (SNVs) which are single-base differences between the tumour sample and the reference genome. These can be separated into two groups:

1. SNVs shared between tumour and normal samples. These are germline variants and were present before the tumour
2. SNVs only present in the tumour sample. These are somatic mutations

Tumours are known to be heterogeneous populations of related cells and in any given tumour sample there may be differing proportions of cells that contain certain SNVs. These proportions can be calculated for each SNV in turn using the number of supporting reads, the read depth and the copy number of that region. These proportions are termed the cancer cell fraction (CCF) and range between 0 (no cells contain the SNV in that sample) to 1 (all cells contain the SNV in that sample).

The SNV CCF values can be explained as a mixture of a number of related clones. We know that SNVs are heritable characteristics between clones and that new clones arise and go extinct over time.

As these mixtures are the product of evolutionary processes it seemed appropriate to use genetic algorithms to infer their ancestry and composition.

1.2 Genetic algorithms

Genetic algorithms (GAs) are stochastic heuristic optimisation algorithms. A population of potential solutions is generated and using a scoring function is compared to the observed data. Solutions that best explain the observed data are allowed to reproduce with one another. Reproduction allows both the recombination of solutions from different individuals and also the possibility of mutations.

GAs are particularly useful when a brute-force approach is undesirable or impractical, for example when the search space is very large. An important consideration is that the search space may contain multiple solutions that exist as local minima and these may be difficult to escape, even with a high mutation rate. **It is therefore imperative to run gaucho multiple times and select the best-scoring solution**. Further details are discussed in the Advanced gaucho usage and best practices section of the worked examples.

1.3 gaucho input

The main gaucho function (*gaucho()*) requires a data frame of observations as input. Each row represents a feature and each column represents a discrete sample separated by time or space. Every value must be a value between 0 and 1 and represents the proportion of individuals that contain that feature.

An example of such data would be single nucleotide variants (SNVs) detected using DNA sequencing in multiple tumour samples. Given the number of reference and non-reference reads as well as the copy number at the site, it is possible to infer the proportion of cells containing the SNV using the following equation:

$$p = min\left(\frac{r.CN}{(r+R)}\right) \tag{1}$$

Where:
p = proportion of cells containing the SNV
CN = copy number at the site
r = number of non-reference reads
R = number of reference reads

If a site were sequenced to 100x depth, with 25 non-reference reads, 75 reference reads and diploid copy number, the result would be min(1,25*2/(25+75)) = 0.5. Therefore, 50% of the cells in the sample contain the SNV.

1.4 How gaucho encodes individuals and solutions

Every individual in the population is encoded as a character string. Consider the following solution to an included data set. It contains three clones and three SNVs. The solution encodes a phylogeny, the proportion of each clone in each sample and the clone in which each SNV first occurred. Let there be s samples and k clones.
`0 1 2 5 0 0 3 7 0 0 1 4 1 2 3`
`0 1 2` : The first k characters encode the position in the phylogeny of each clone. In this example, the first element is the root (hence 0), the second is descended from the first and third is descended from the second. If the second and third clones were both descended from the first clone, the phylogeny would be `0 1 1`.
`5 0 0 3 7 0 0 1 4` : The next k*s characters encode the proportions. Each block of k characters encodes the proportions of each clone for the sth sample. Proportions are generated by dividing the numbers in the block by their sum. This example is explained below:

```
## First sample is reprented by first block of three digits
## First sample is 100% clone 1, 0% clone 2, 0% clone 3
c(5,0,0)/sum(c(5,0,0))

## [1] 1 0 0

## Second sample is reprented by second block of three digits
## Second sample is 30% clone 1, 70% clone 2, 0% clone 3
c(3,7,0)/sum(c(3,7,0))

## [1] 0.3 0.7 0.0

## Third sample is reprented by third block of three digits
## Third sample is 0% clone 1, 20% clone 2, 80% clone 3
c(0,1,4)/sum(c(0,1,4))

## [1] 0.0 0.2 0.8
```

`1 2 3` : The final k characters encodes the mutation matrix and the order of SNVs. In this example the first SNV occurred in clone 1 (and is inherited by its descendants), the second SNV occurred in clone 2 (and is inherited by its descendent) and SNV 3 occurred in clone 3. If all SNVs occurred in clone 1, this string would be `1 1 1`.

1.5 gaucho mutation function

The mutation function has been heavily modified in `gaucho` relative to the original in `GA`. If an individual in the population undergoes mutation there is an equal chance of it occurring in the phylogeny, proportion of clones or mutation matrix sections of the solution.

A mutation in the phylogeny section results in a new phylogeny being randomly assigned to the individual. A mutation in the proportion section results in between 1 to k increments or decrements to the assigned values. This enhanced level of mutation is required to help individuals escape from local minima. A mutation in the mutation matrix section results in a single value being randomly assigned to a new clone.

Also, note that crosses between parents are not permitted in the phylogeny section of the solution.

1.6 gaucho output

The `gaucho()` function returns an object of class `ga`. Although this object can be interrogated manually, the easiest way to look at the results is to use the gauchoReport() function. The default output will produce the following files in the current working directory (where k=number of clones, i=number of solution and n=total number of top-scoring solutions):

1. png image: kclones.solutioniofn.phylogeny.png: the inferred phylogeny of the solution
2. png image: kclones.solutioniofn.heatmap.png: a heatmap of the solution
3. png image: kclones.solutioniofn.proportions.png: a stacked barplot of the proportion of each clone that composes each sample
4. png image: kclones.fitnessconvergence.png: the fitness convergence plot. This can be used to ensure that gaucho is reaching convergence and being run for an appropriate number of generations
5. text file: kclones.complete.solutioniofn.txt: the solution encoded as a string
6. text file: kclones.phylogeny.solution.iofn.txt: the phylogeny of the solution encoded as a matrix. Note that these matrices are not directly compatible with the *graph* R package
7. text file: kclones.mutation.matrix.iofn.txt: a binary matrix showing the presence or absence of each SNV in each clone
8. text file: kclones.proportions.solution.iofn.txt: the proportion of each clone that composes each sample

Note that in the event of multiple solutions scoring equally well, files for up to the top five solutions will be produced.

```
## Write results to files
gauchoReport(gaucho_input_dataframe,gaucho_output_object)
```

2 Worked examples

A number of sample data sets are distributed with the gaucho package and are discussed in order of increasing complexity.

2.1 Example 1 - simple synthetic data

A very small and simple synthetic data set is included. To demonstrate that your gaucho installation is working, you can execute the following commands.

```
## Load library
library(gaucho)
```

```
## Loading required package:  compiler
## Loading required package:  GA
## Loading required package:  foreach
## Loading required package:  iterators
## Package 'GA' version 3.0.2
## Type 'citation("GA")' for citing this R package in publications.
## Loading required package:  graph
## Loading required package:  BiocGenerics
## Loading required package:  parallel
##
## Attaching package:  'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##    clusterExport, clusterMap, parApply, parCapply, parLapply,
##    parLapplyLB, parRapply, parSapply, parSapplyLB
## The following objects are masked from 'package:stats':
##
##    IQR, mad, xtabs
## The following objects are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##    as.data.frame, cbind, colnames, do.call, duplicated, eval,
##    evalq, get, grep, grepl, intersect, is.unsorted, lapply,
##    lengths, mapply, match, mget, order, paste, pmax, pmax.int,
##    pmin, pmin.int, rank, rbind, rownames, sapply, setdiff, sort,
##    table, tapply, union, unique, unsplit, which, which.max,
##    which.min
## Loading required package:  heatmap.plus
## Loading required package:  png
## Loading required package:  Rgraphviz
## Loading required package:  grid

## Load simple data set
gaucho_simple_data = read.table(file.path(system.file("extdata",package="gaucho"),"gaucho_simple_

## There  are three columns (time points T0, T1 and T2)
## and three rows (mutations M1, M2, M3)
gaucho_simple_data

##    T0  T1  T2
## M1  1 1.0 1.0
## M2  0 0.7 1.0
## M3  0 0.0 0.8
```

As we know the true number and relationship between the clones, we specify these and run the algorithm. It should converge at the minimum score of zero quite quickly and exit after 200 generations of coverging at the highest fitness value.

```
## Execute gaucho() function on the simple data set
simpleDataSolution=gaucho(gaucho_simple_data, number_of_clones=3, nroot=1,iterations=1000)
```

We can view the highest scoring solution(s) by accessing the appropriate slot in the returned object. For example:

```
## Access solution slot in returned object to show highest scoring solution(s)
simpleDataSolution@solution

     x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
[1,]  0  1  2  5  0  0  3  7  0   0   1   4   1   2   3
[2,]  0  1  2  7  0  0  3  7  0   0   1   4   1   2   3
[3,]  0  1  2  1  0  0  3  7  0   0   1   4   1   2   3
[4,]  0  1  2  6  0  0  3  7  0   0   1   4   1   2   3
[5,]  0  1  2  3  0  0  3  7  0   0   1   4   1   2   3
[6,]  0  1  2 11  0  0  3  7  0   0   1   4   1   2   3
[7,]  0  1  2 12  0  0  3  7  0   0   1   4   1   2   3
```

Note that it's possible for multiple solutions to have equally good fitness scores which may encode different phylogenies, mutation distributions or proportions. However, in this case all of the solutions are numerically identical and exist because of the degeneracy in the way that the clonal proportions are encoded.

We can now examine the best solution(s) using the gauchoReport() function.

```
## Produce plots for the phylogeny, heatmap and proportions in turn
gauchoReport(gaucho_simple_data,simpleDataSolution,outType="phylogeny")
gauchoReport(gaucho_simple_data,simpleDataSolution,outType="heatmap")
gauchoReport(gaucho_simple_data,simpleDataSolution,outType="proportion")

## Create output files representing the solution(s) in the current working directory
gauchoReport(gaucho_simple_data,simpleDataSolution,outType="complete")

## In case you want to know the current working directory, it can be reported using this functio
getwd()
```

You should see images like those below. The colours used for clones are consistent between the phylogeny, the sidebar of the heatmap and the proportion barplot.

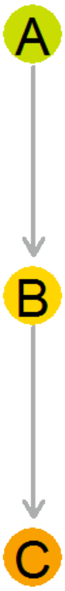

Figure 1: The phylogeny of the optimum solution to the simple data set. There is a single root (clone A) which gives rise to clone B which gives rise to clone C.

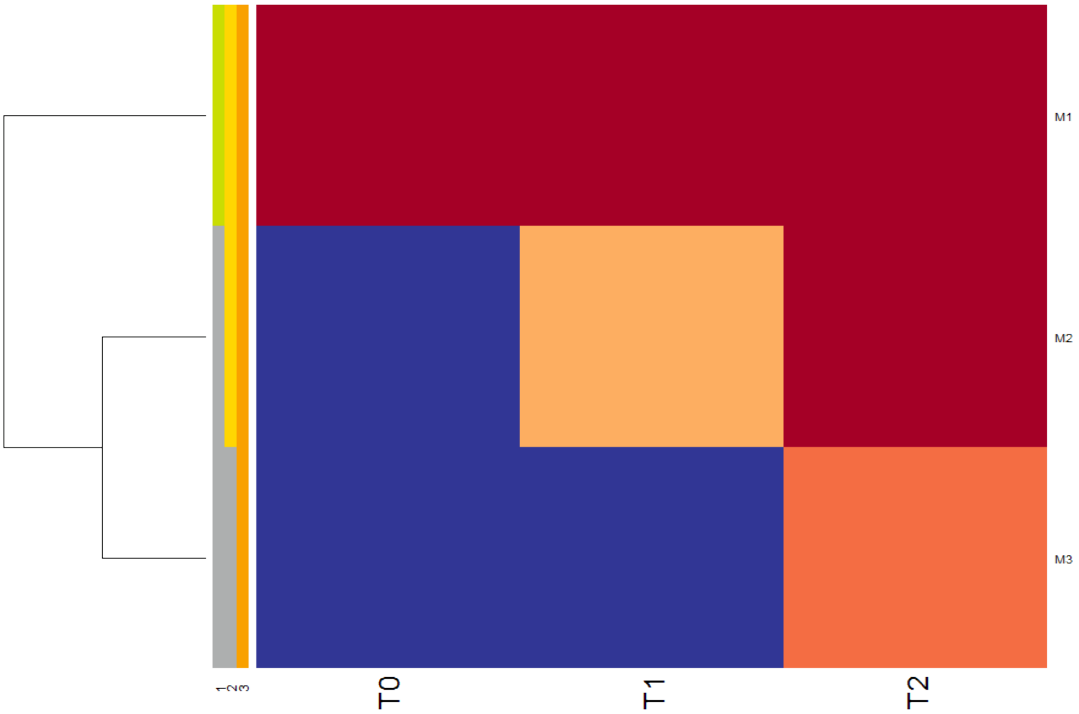

Figure 2: The input data has been clustered using the heatmap.plus package. The colour scale goes from blue (low values) through yellow to red (high values). The coloured sidebar shows which clones contain which mutations. For example, M1 is contained by all clones, whereas M3 is contained only by clone C.

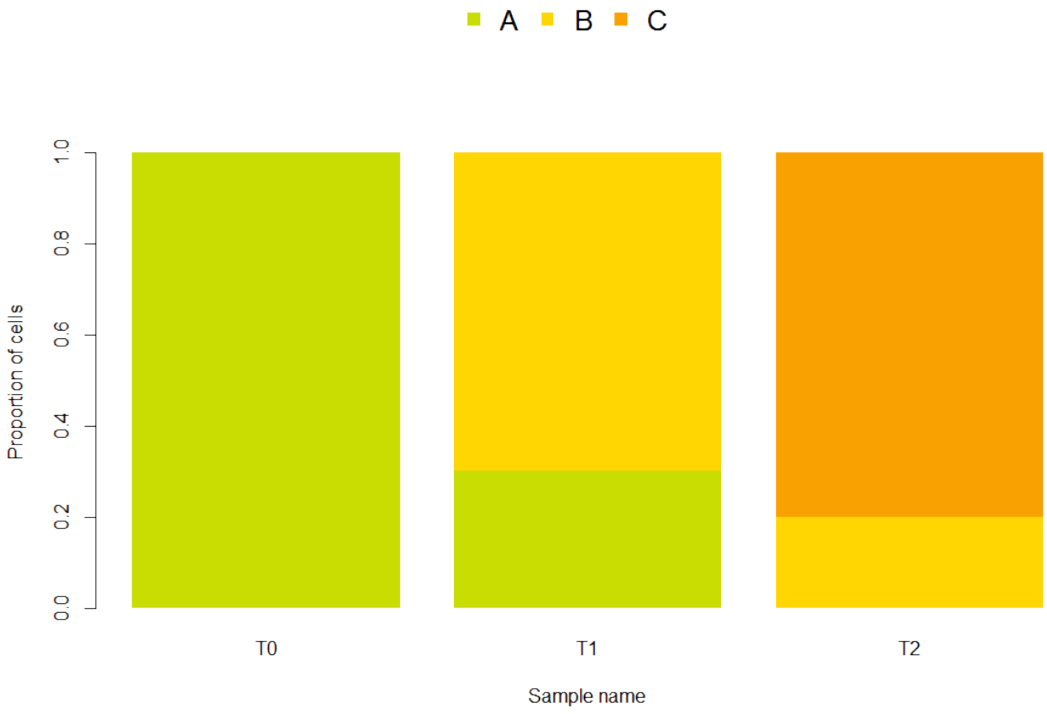

Figure 3: The proportions of each clone in each sample of the simple data set. Sample T0 is composed entirely of clone A, whereas sample T1 is an unequal mixture of clones A and B, while sample T2 is an unequal mixture of clones B and C.

2.2 Example 2 - synthetic data containing a hidden clone

The second synthetic data set demonstrates the ability of gaucho to infer intermediate clones that are not explicitly detected in the input data and therefore ”hidden”.

```
## Load library
library(gaucho)

## Load hidden data set
gaucho_hidden_data = read.table(file.path(system.file("extdata",package="gaucho"),"gaucho_hidden_

## There  are three columns (time points T0, T1 and T2)
## and five rows (mutations M1, M2, M3, M4 and M5)
gaucho_hidden_data

##    T0  T1  T2
## M1  1 1.0 1.0
## M2  0 0.5 1.0
## M3  0 0.3 0.7
## M4  0 0.2 0.3
## M5  0 0.0 0.6
## Execute gaucho() function on the hidden data set
hiddenDataSolution=gaucho(gaucho_hidden_data, number_of_clones=5,
                        nroot=1,iterations=3000)

## Access solution slot in returned object to show highest scoring solution(s)
## This solution's score is -0.02500000
hiddenDataSolution@solution

     x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20 x21
[1,]  0  1  2  2  4 29  0  0  0   0   8   0   3   5   0   0   0   6   2  12   1
    x22 x23 x24 x25
[1,]  2   4   3   5
```

```
## Produce plots for the phylogeny, heatmap and proportions in turn
gauchoReport(gaucho_hidden_data,hiddenDataSolution,outType="phylogeny")
gauchoReport(gaucho_hidden_data,hiddenDataSolution,outType="heatmap")
gauchoReport(gaucho_hidden_data,hiddenDataSolution,outType="proportion")

## Create output files representing the solution(s) in the current working directory
gauchoReport(gaucho_simple_data,simpleDataSolution,outType="complete")
```

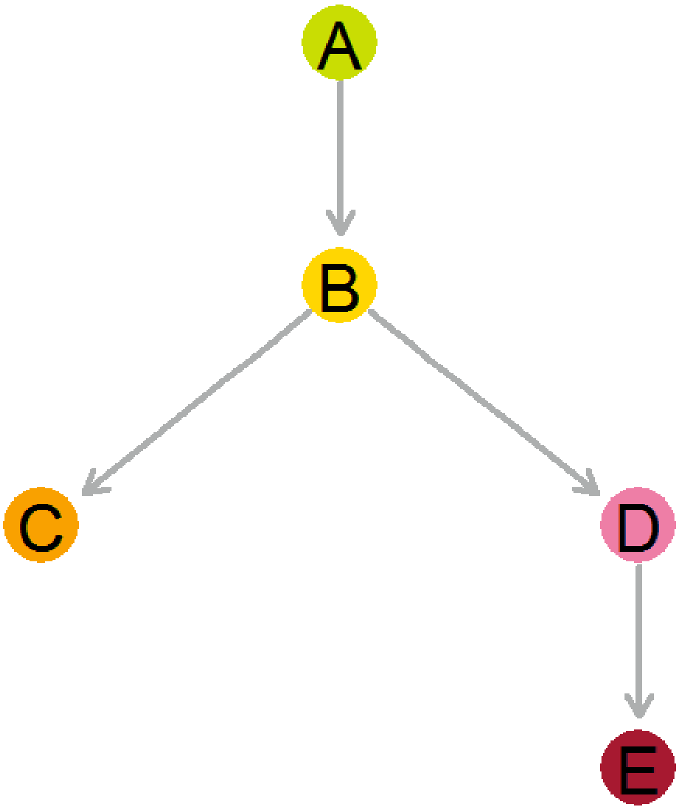

Figure 4: The phylogeny of the optimum solution to the hidden data set. Clone B is both the child of clone A and the parent of all other clones.

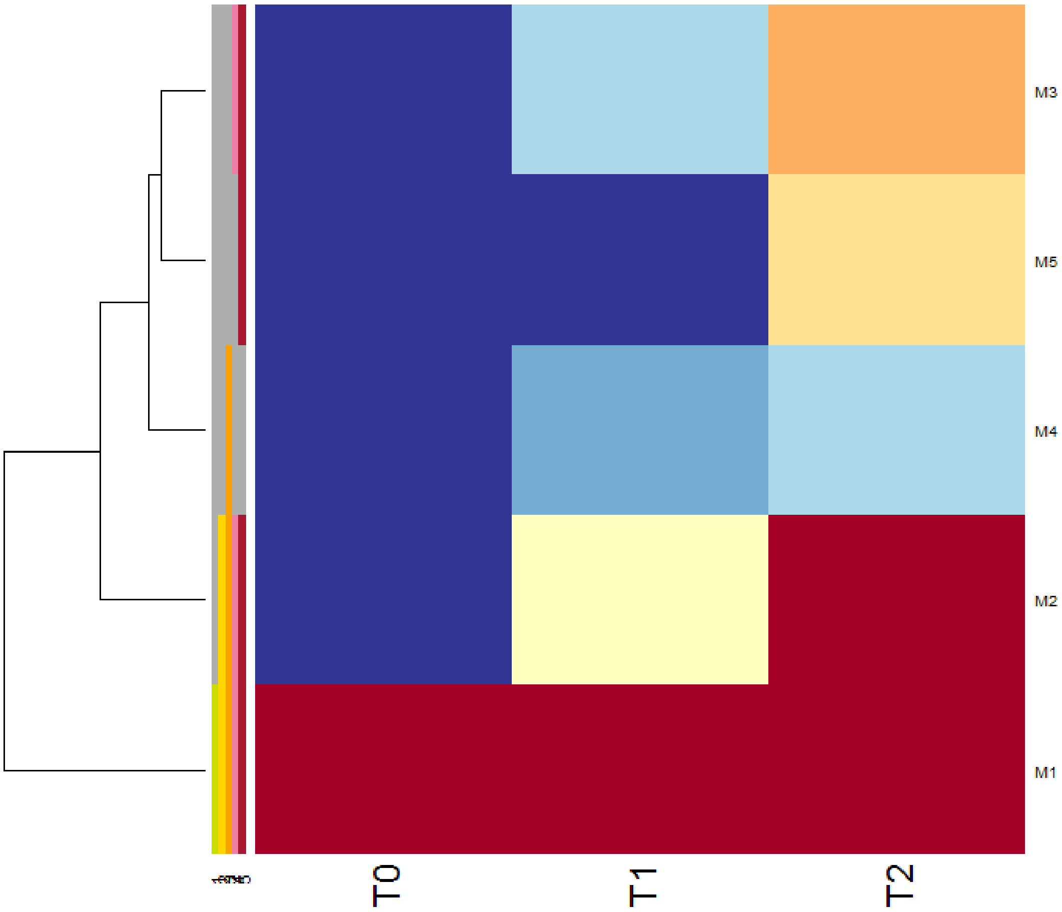

Figure 5: Heatmap of the hidden data showing the mutation content of the clones.

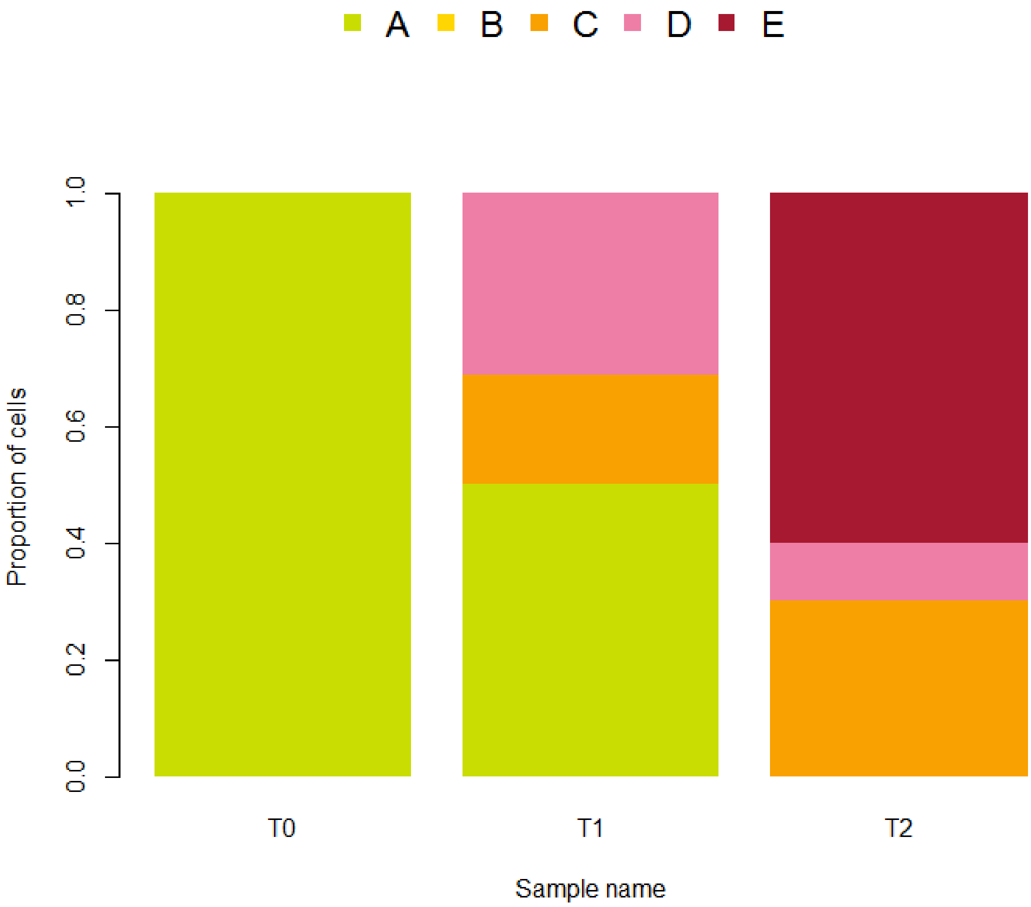

Figure 6: The proportions of each clone in each sample of the hidden data set. Although clone B is not present in any of the samples its existance and ancestry have been correctly inferred.

2.3 Example 3 - complex synthetic data and noisy synthetic data

These synthetic data are close to the sort of observations produced by a typical tumour-normal paired exome sequencing experiment. There are four time points (T0 to T3) and 90 mutations (gene1 to gene90). Each of the values represents the proportion of cells that contain that mutation at that time point - the cancer cell fraction (CCF). We also include a synthetic data set with noise added, as real-world data is never perfect.

Note that this example considers a single run of the algorithm, but real use requires multiple runs. See the Advanced gaucho usage and best practices section for details and sample code.

```
## Load library
library(gaucho)

## Load synthetic data set and the same data with added jitter
gaucho_synth_data = read.table(file.path(system.file("extdata",package="gaucho"),"gaucho_synth_da
gaucho_synth_data_jittered = read.table(file.path(system.file("extdata",package="gaucho"),"gaucho

## Execute gaucho() function on the synthetic data set - we know that
## there are 6 clones
s=gaucho(gaucho_synth_data,number_of_clones=6,iterations=3000)

## Access solution slot in returned object to show highest scoring solution(s)
## The optimum solution's score is -2.22E-15, which is a rounding error from zero
s@solution

## Create output files representing the solution(s)
## in the current working directory
gauchoReport(gaucho_synth_data,s)
```

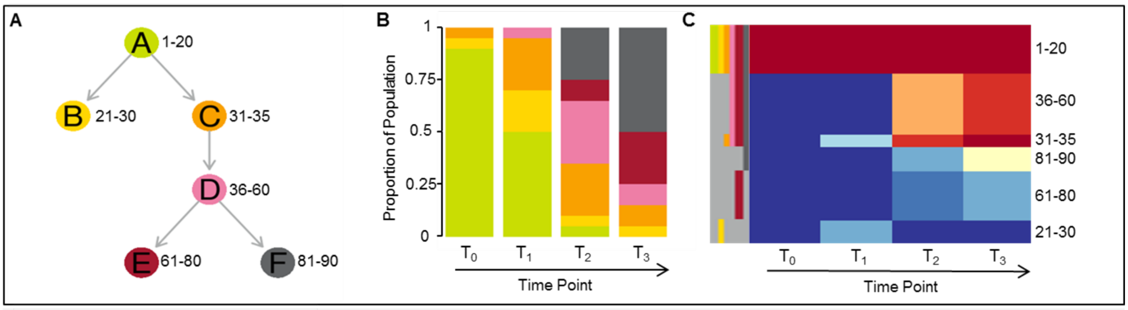

Figure 7: The optimum solution for the synthetic data set. A shows the relationship between the six clones with the mutations that are gained in each clone labelled. B shows the proportions of each sample that is composed by each clone. C shows hierarchical clustering of the input data, with the corresponding clones for each mutation as a coloured side bar.

2.4 Example 4 - real experimental data with contamination

The final included data set and is important for several reasons:

1. It's experimental, biological data from a published source [**pmid23873039**]
2. The publication suggests an optimum (”correct”) solution
3. The input data is contaminated, so we can test the `contamination` parameter of `gaucho()`
4. The data is functionally identical to the sort of mutation data we see in tumour/normal paired sequencing experiments

```
## Load library
library(gaucho)

## Load yeast data set
BYB1_G07_pruned = read.table(file.path(system.file("extdata",package="gaucho"),"BYB1_G07_pruned.t

## Execute gaucho() function on the yeast data set
## The paper claims that there are 6 clones with multiple roots
yDataSolution=gaucho(BYB1_G07_pruned, number_of_clones=6, contamination=1, iterations=3000)

## Access solution slot in returned object to show highest scoring solution(s)
yDataSolution@solution
```

```
## Create output files representing the solution(s) in the current working directory
gauchoReport(BYB1_G07_pruned,yDataSolution)
```

As expected, solutions with 6 clones (as suggested by the publication) score the highest as a group. However, we found a subtly different solution with 7 clones that scored even higher. Therefore, the output of that solution is presented below.

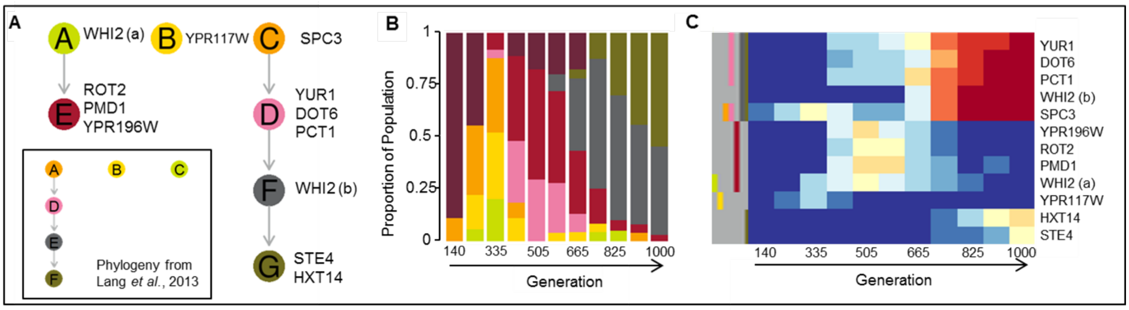

Figure 8: A high-scoring solution solution for the yeast data set composed of 7 clones. A shows the reconstructed phylogeny with each clone labelled with its associated mutations; inset is the 6 clone phylogeny described by Kernighan et al. B shows the proportions of each sample that is composed by each clone. Note that clone H, which is not present in either the phylogeny or heatmap, represents ”contamination”. In this context, the contamination is yeast cells that contain none of the mutations that were measured. C shows hierarchical clustering of the input data, with the corresponding clones for each mutation as a coloured side bar.

2.5 Advanced gaucho usage and best practices

We have covered the basic usage of gaucho using small data sets with simple phylogenies, larger data sets with complex phylogenies and finally we recapitulated the results found through other methods in a published data set. Here we explain more complex use of gaucho and provide sample code for running your own analyses.

2.5.1 Finding an initial estimate of the number of clones

All gaucho requires to run is some input and the number of clones. How can we supply the number of clones to gaucho without knowing it first?

Fortunately, gaucho itself provides a method to estimate the number of clones. Although there are other methods we plan to implement in gaucho in a later release, the best method currently is run gaucho a number of times with a number of different clone numbers and select the value that gives the lowest scores.

Ideally, you should run gaucho as many times as possible over as large a range of clone numbers as possible, but in practice we have found 10 data points to be more than enough to find a suitable estimate. Note that each run does not need to be as thorough as when looking for the final solution; a few hundred to a thousand iterations per run should be sufficient to indicate the result.

```
## Load library
library(gaucho)

## Load simple data set
gaucho_synth_data = read.table(file.path(system.file("extdata",package="gaucho"),"gaucho_synth_da
```

```r
## We know that there are 6 clones in this data set.
## Let's loop from number_of_clones=4 to number_of_clones=8 to illustrate this
## Also, assume we know nothing about phylogeny, so leave nroot as the default value

## Assign parameters and create an empty data frame to hold the results
clonerange=4:8
n=5
results=matrix(NA,nrow=iterations,ncol=length(clonerange))
colnames(results)=paste0(rep("clone",length(clonerange)),clonerange)

## Execute gaucho n times for each number of clones
for(c in clonerange){
  for(thisn in 1:n){
    message(paste("Iteration",thisn,"using",c,"clones"))
    s=gaucho(gaucho_synth_data, number_of_clones=c,iterations=1000)
    ## Assign best fitness value to an element in the matrix
    results[thisn,which(c==clonerange)]=s@fitnessValue
  }
}

## Plot the results
boxplot(results)
```

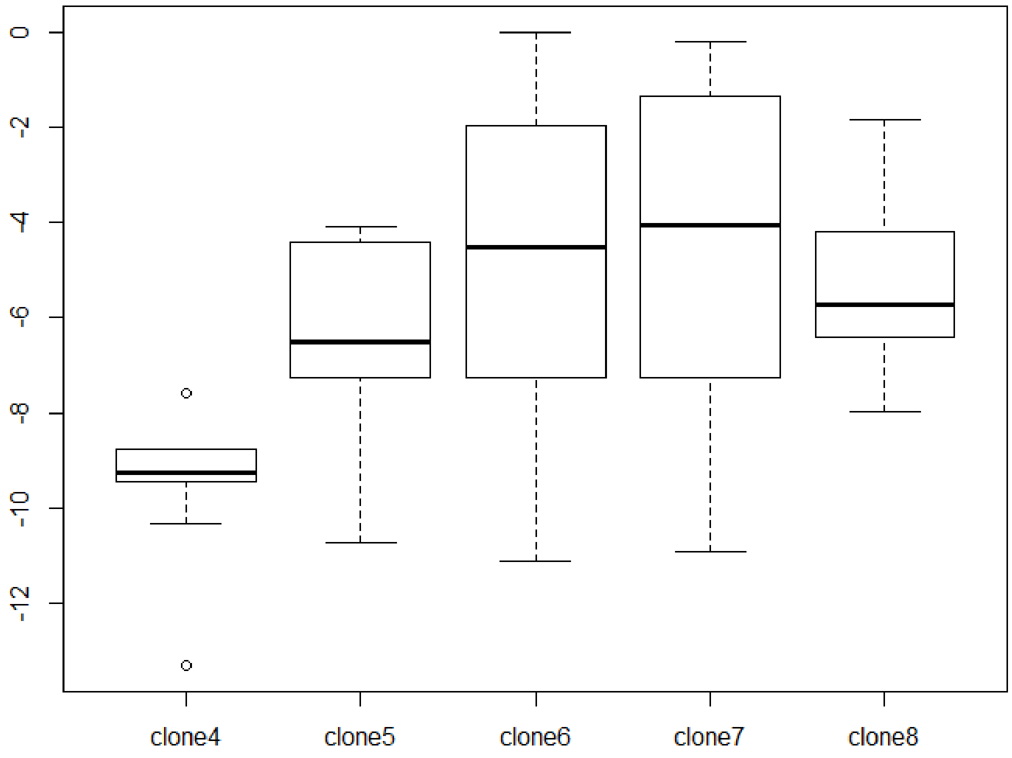

Figure 9: Lowest fitness scores for different numbers of clones in the synthetic data. Although 7 clones gives a lower median score across 10 runs, 6 clones acheived the lowest single score. Therefore, 6 and 7 clones should be explored further.

2.5.2 Executing gaucho multiple times

Now that the number of clones has been established to most likely be 6 or 7, a larger number of longer runs should be executed. There are a number of ways to address this, but in this example a simple loop executes multiple runs of gaucho for a set number of clones and adds the resulting object to a list. When the run is complete (or is manually ended using ctrl-C), the user may interrogate the list to find the best solution.

```
## Load library
library(gaucho)

## Load simple data set
gaucho_synth_data = read.table(file.path(system.file("extdata",package="gaucho"),"gaucho_synth_da

## Execute 20 runs
n=20

## Create an empty list to hold all results
```

```
gauchoResults = list()

## Execute gaucho n times, pushing each resulting object into a list
for(thisn in 1:n){
  message(paste("Iteration",thisn))
  syntheticDataSolution=gaucho(gaucho_synthetic_data, number_of_clones=6, iterations=10000)
  gauchoResults[thisn]=syntheticDataSolution
}

## Find lowest fitness values of all runs
bestScore = max(sapply(gauchoResults, function(x) x@fitnessValue))

## Which run was it?
which(sapply(gauchoResults, function(x) x@fitnessValue)==bestScore)
```

Multicore, parallel solutions that run multiple gaucho instances concurrently will be added in a future release of gaucho.

3 Session Info

```
sessionInfo()

## R version 3.3.1 (2016-06-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.1 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
##  [1] grid      parallel  compiler  stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
## [1] gaucho_1.10.0       Rgraphviz_2.18.0    png_0.1-7
## [4] heatmap.plus_1.3    graph_1.52.0        BiocGenerics_0.20.0
## [7] GA_3.0.2            iterators_1.0.8     foreach_1.4.3
##
## loaded via a namespace (and not attached):
##  [1] codetools_0.2-15 stats4_3.3.1     formatR_1.4      magrittr_1.5
##  [5] evaluate_0.10    highr_0.6        stringi_1.1.2    tools_3.3.1
##  [9] stringr_1.1.0    knitr_1.14
```