Tutorial 5: Constrained Shuffling of Genomic Contacts to Form a Control Set

Problem Statement

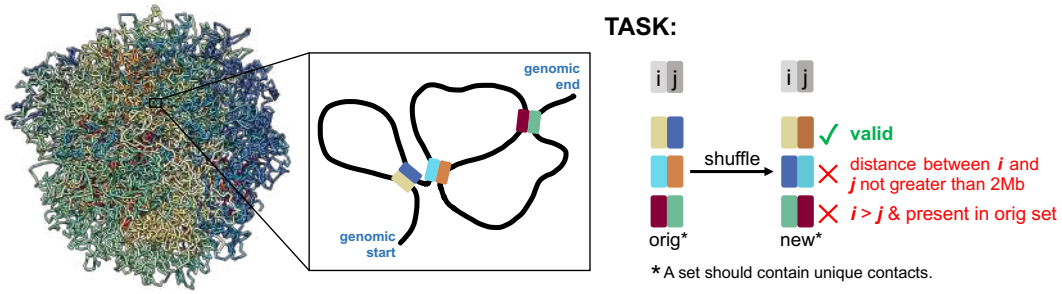
Here, we shall consider a problem from the field of 3D genome organisation, to exemplify how ROptimus can optimise a specific constrained shuffling task given a certain set of conditions.

Mammalian genomes are packaged into a complex three-dimensional (3D) structure inside the nucleus, bringing DNA regions linearly near or far from each other into contacts.

Here, we take a set of 734 pairs of i and j 40-kilobase DNA regions that are known to be in contact inside a cell. Binning the DNA (e.g., a single chromosome) into 40-kb regions, each region is represented as a single integer that is equal to its end position divided by the length of the region, which is 40 kb. For instance, the 1st region, with start and end positions at the 1^{st} and 40000^{th} nucleotides, respectively, is denoted as 1 (40000^{th} base /40000 bases = 1). This simplifies the notation for a contact between two regions to a pair of positive integers as shown below:

```
##      i   j
## 1   96 151
## 2   96 174
## 3  322 374
## 4  309 392
## 5  321 392
## 6  373 460
```

where $i \in Z^+$ and $j \in Z^+$.



Take note that our set of contacts comply with the following criteria.

1. The set only contains **unique, long-range contacts**, with i and j regions always separated by greater than 2 megabases (Mb). In terms of the integer identifiers, the threshold is 50 ($2Mb/40kb = 50$), such that $(j - i) > 50$.
2. i is set to be always less than j. This avoids duplicate contacts in the set definition, such as

```
##        i   j
## [1,] 181 273
## [2,] 273 181
```

3. A region can take part in more than one contact (as in region 96 in rows 1 and 2 in the table above), given that the contacts it forms comply with the two aforementioned criteria.

The task herein is to take the DNA regions from the original set (shown above) and to shuffle them to produce a new set of contacts. This is useful for a research on genomic 3D contacts requiring good quality of negative controls that still have the constraints that the original true data possess. The challenge here is to maximise

the number of new control contacts that we can devise and that are valid, meaning that they A) still satisfy the aforementioned 3 criteria, and B) are not present in the original set.

Defining ROptimus Inputs

We can solve this highly constrained shuffling problem using ROptimus by doing the following preparations. Keeping the original order of i's in the loaded real set of contacts (referred hereafter as `IJ_ORIG`), we can randomly shuffle all j's in the set once, without replacement. The output will be the object `K` to be supplied to `Optimus()` for the optimisation.

```
K <- IJ_ORIG
set.seed(840)
# Shuffle all j's once
K[,"j"] <- sample(x=K[,"j"], size=nrow(K), replace=FALSE)
```

The `m()` function should then operate on the object `K`, which, to reiterate, is derived from the original set of contacts with all j's shuffled once (see above). It can also accept a supplementary object `DATA` that holds data necessary for the model calculations and outcome comparison. Output of `m()` is the observable object `O` that is the percentage of contacts not satisfying the aforementioned criteria or are already in the original set. To get `O`, `m()` will require a) the original set of contacts (`IJ_ORIG`), b) number of contacts, and c) gap threshold set between contacting regions in terms of positive integer (50). These are first stored in the `DATA` object then passed to `m()`.

```
DATA <- list(IJ_ORIG=IJ_ORIG,
             gaplimit=50,
             numContacts=nrow(IJ_ORIG))

m <- function(K, DATA){
  # Total number of erroneous contacts in the new set
  errCont <- sum(
    # Contacts not satisfying the threshold value of the gap between their two regions
    ( (K[,"j"]-K[,"i"]) <= DATA$gaplimit ) |
    # Contacts also present in the original set
    ( K[,"j"]==DATA$IJ_ORIG[,"j"]        ) |
    # Duplicated contacts in the new set
    ( duplicated(K)                      )
  )
  # Percentage of erroneous contacts
  O <- (errCont/DATA$numContacts)*100
  return(O)
}
```

In the `u()` function, the error percentage `O` will be directly used as the pseudo energy `E`, since we do want to minimise this value. Meanwhile, the quality `Q` for the given snapshot of `K`, which we want to increase, can be the negative of `O`.

```
u <- function(O, DATA){
  RESULT <- NULL
  RESULT$Q <- -O
  RESULT$E <- O
  return(RESULT)
}
```

Finally, `r()`, which defines how object `K` will be altered at every step, takes two j's and then swaps them, without changing the order of their partner i's.

```
r <- function(K){
  K.new <- K
  # Indices of two randomly chosen j's to swap
  new.ind <- sample(x=1:length(K.new[,"j"]), size=2, replace=FALSE)
  # Swap the j's
  K.new[new.ind,"j"] <- K.new[rev(new.ind),"j"]
  return(K.new)
}
```

Acceptance Ratio Simulated Annealing ROptimus Run

We first use the Acceptance Ratio Simulated Annealing scheme in ROptimus to solve the problem. As in **Tutorials 2** and **4**, `Optimus()` is set to do 200 000 steps, 2 annealing cycles, and is to produce 4 independent replicas of the same simulation.

```
Optimus(NCPU=4, OPTNAME="IJ.NEW.OPTI.SA",
        NUMITER=200000, CYCLES=2, DUMP.FREQ=100000, LONG=FALSE,
        OPT.TYPE="SA",
        K.INITIAL=K, rDEF=r, mDEF=m, uDEF=u, DATA=DATA)
```

Note that if additional data is required by `m()` or `u()`, that object should be assigned to `DATA` argument of `Optimus()`, otherwise `DATA` defaults to `NULL`.

The 4 new sets obtained as optimums from each of the 4 replicates of independent simulated annealing runs have the following error percentages:

Table 19: 4-core Acceptance Ratio Simulated Annealing results from ROptimus run.

Set	Erroneous Contact %
1	11.444
2	10.899
3	12.262
4	11.717

Acceptance Ratio Replica Exchange ROptimus Run

Now, we can use the Acceptance Ratio Replica Exchange mode of ROptimus using 12 cores and the same number of optimisation steps (200 000) as in the Acceptance Ratio Simulated Annealing ROptimus run. Similar to **Tutorials 2** to **4**, `STATWINDOW` is set to 50 and the acceptance ratios are as follows:

```
ACCRATIO <- c(90, 82, 74, 66, 58, 50, 42, 34, 26, 18, 10, 2)

Optimus(NCPU=12, OPTNAME="IJ.NEW.OPTI.RE",
        NUMITER=200000, STATWINDOW=50, DUMP.FREQ=100000, LONG=FALSE,
        OPT.TYPE="RE", ACCRATIO=ACCRATIO,
        K.INITIAL=K, rDEF=r, mDEF=m, uDEF=u, DATA=DATA)
```

Set	Replica Acceptance Ratio	Erroneous Contact %

Table 20: 12-core Acceptance Ratio Replica Exchange results from ROptimus run.

Set	Replica Acceptance Ratio	Erroneous Contact %
1	90	34.060
2	82	29.155
3	74	24.523
4	66	18.937
5	58	18.937
6	50	17.847
7	42	18.256
8	34	18.392
9	26	18.529
10	18	18.529
11	10	18.529
12	2	18.529

Notably, in this setup, none of the 12 new sets of contacts from the replica exchange run are better than the sets produced by the Acceptance Ratio Simulated Annealing mode of ROptimus.

Summary

We have added yet another one to the diverse applications of ROptimus, this time by performing a constrained shuffling of pairs of positive integers that, in this case, represent contacts between DNA regions, to form a new set of control contacts. Additionally, ROptimus showed to be a platform, where certain restrictions or conditions can easily be incorporated to a given shuffling task, which usually is required for specific research objectives. The table below shows the % of erroneous contacts of the best outputs from the acceptance ratio annealing and replica exchange ROptimus runs, with the former producing the better set.

Table 21: Summary of solutions.

Method	Erroneous Contact %
ROptimus (AR Simulated Annealing)	10.899
ROptimus (AR Replica Exchange)	17.847