Contextualizing large scale signalling networks from expression footprints with CARNIVAL

Enio Gjerga, Matteo Spatuzzi, Olga Ivanova,

2021-10-20

Introduction

While gene expression profiling is commonly used to gain an overview of cellular processes, the identification of upstream processes that drive expression changes remains a challenge. To address this issue, we introduce CARNIVAL (Liu, Trairatphisan, Gjerga et al 2019), a causal network contextualization tool which derives network architectures from gene expression footprints. CARNIVAL (CAusal Reasoning pipeline for Network identification using Integer VALue programming) integrates different sources of prior knowledge including signed and directed protein-protein interactions, transcription factor targets, and pathway signatures.

Pipeline

CARNIVAL refines a quantitative objective function for ILP problem by incorporating TF and pathway activities on a continuous scale. In addition, the CARNIVAL framework allows us to contextualize the network with or without known targets of perturbations. The implementation is separated into two pipelines which will be referred henceforth as Standard CARNIVAL StdCARNIVAL (with known perturbation targets as an input) and Inverse CARNIVAL InvCARNIVAL (without information on targets of perturbation). The differential gene expression is used to infer transcription factor (TF) activities with DoRothEA, which are subsequently discretized in order to formulate ILPconstraints. As a result, CARNIVAL derives a family of highest scoring networks which best explain the inferred TF activities. Continuous pathway and TF activities can be additionally considered in the objective function.

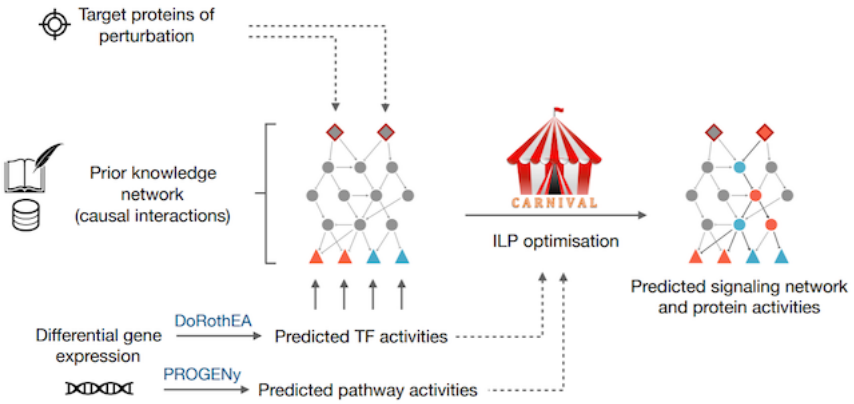

Figure 1: Figure 1: CARNIVAL pipeline

ILP solvers

CARNIVAL is an extension of the previously implemented Causal Reasoning method from Melas et al.. The network inference process is swiftly performed with an Integer Linear Programming (ILP) formulation of causal

reasoning using four solvers: the R-CRAN lpSolve free software used for solving linear problems; the open-source mixed integer programming solver Cbc (Coin-or branch and cut); the CPLEX optimizer from IBM which can be obtained for free through the Academic Initiative; or Gurobi which also can be obtained for free through an academic licence. To perform the analysis with cplex or cbc, the users will then need to store the binary cbc or cplex executables on any directory they wish. The binary files of cbc can be found by first downloading one of the optimization suites provided here: https://www.coin-or.org/download/binary/OptimizationSuite/, unzip the download and from there save the cbc executable (which can be found on the bin directory) file on any of the direcotries they wish of their machines. As for the cplex, the executable file can be obtained after registration on the ILOG CPLEX Optimization Studio Similar like before, users will have to find the cplex executable binary file and save on a directory of their own wish or keep them on their default installation paths. The path to interactive version of CPLEX is differed based on the operating system. The default installation path for each OS is as follows:

For Mac OS:

```
~/Applications/IBM/ILOG/CPLEX_Studio129/cplex/bin/x86-64_osx/cplex
```

For Linux:

```
/opt/ibm/ILOG/CPLEX_Studio129/cplex/bin/x86-64_linux/cplex
```

For Windows:

```
C:/Program Files/IBM/ILOG/CPLEX_Studio129/cplex/bin/x64_win64/cplex.exe
```

Note that the version of CPLEX has to be changed accordingly (the latest current version is CPLEX-Studio129).

To install Gurobi, executable can be downloaded from Gurobi downloads page. Similarly to cplex, the executable location will differ depending on the operating system.

The lpSolve solver can be used after downloading and installing the lpSolve R-package. This solver only works for smaller examples and it can give only one optimal solution. For larger real-case examples, the users can use cbc, cplex or gurobi solvers.

Prerequisites

Besides the above mentioned solvers, users need also to install the following R-package dependencies: readr; igraph; dplyr; lpSolve

In order to visualize the automatically generated CARNIVAL networks, users will also need to download and install the Graph Visualization software graphviz.

Running CARNIVAL

The CARNIVAL package provides the use with five functions to run the Carnival pipeline in a flexible way. runVanillaCarnival is used to run the standard carnival pipeline with one function, while runInverseCarnival runs inverse Carnival (no input). The generateLPFileCarnival and runCarnivalFromLp functions work in tandem and allow the user for example to obtain the LPFile without running the whole pipeline or create an LPFile from a different source and run the Carnival pipeline with it. These functions also contain multiple parameters to tailor the

In the CARNIVAL package, built-in examples are available as the test cases as follows:

1. A small toy example where the perturbations are known (vanialla CARNIVAL flavour)
2. A small toy example where the perturbations are not known (inverse CARNIVAL flavour)
3. A small toy example to run with two functions (vanilla CARNIVAL flavour)

The Data of these toy examples looks as follows:

- Two input Nodes I1 and I2 connected to the nodes N1 and N2 respectively
- N1 and N2 are connected to both measured Nodes M1 and M2
- All connections beside I2 to N2 are activatory

Toy Example - 1

Toy example for the CARNIVAL standard pipeline:

```
library(CARNIVAL)
load(file = system.file("toy_perturbations_ex1.RData",
                        package = "CARNIVAL"))
load(file = system.file("toy_measurements_ex1.RData",
                        package = "CARNIVAL"))
load(file = system.file("toy_network_ex1.RData",
                        package = "CARNIVAL"))

carnivalOptions <- defaultLpSolveCarnivalOptions()

# Output dir
dir.create("ToyExample1", showWarnings = FALSE)
carnivalOptions$outputFolder <- "ToyExample1"

# lpSolve
resultsLpSolve <- runVanillaCarnival( perturbations = toy_perturbations_ex1,
                                      measurements = toy_measurements_ex1,
                                      priorKnowledgeNetwork = toy_network_ex1,
                                      carnivalOptions = carnivalOptions)
```

which generates the following output:

```
print(resultsLpSolve)
```

```
## $weightedSIF
##   Node1 Sign Node2 Weight
## 1    I1    1    N1    100
## 2    N1    1    M1    100
## 3    N1    1    M2    100
## 4    I2   -1    N2    100
##
## $nodesAttributes
##   Node ZeroAct UpAct DownAct AvgAct nodesType
## 1   I1       0   100       0    100         P
## 2   N1       0   100       0    100
## 3   I2       0   100       0    100         P
## 4   N2     100     0       0      0
## 5   M1       0   100       0    100         M
## 6   M2       0   100       0    100         M
##
## $sifAll
## $sifAll[[1]]
##   Node1 Sign Node2
## 1    I1    1    N1
## 2    N1    1    M1
## 3    N1    1    M2
## 4    I2   -1    N2
##
```

```
##
## $attributesAll
## $attributesAll[[1]]
##   Nodes Activity
## 1    I1        1
## 2    N1        1
## 3    I2        1
## 4    M1        1
## 5    M2        1
```

The output from CARNIVAL can be saved in `Rds` format like so:

```
saveRDS(resultsLpSolve, "ToyExample1/network_solution.Rds")
```

Toy Example - 2

Toy example for the CARNIVAL inverted pipeline:

```
load(file = system.file("toy_measurements_ex2.RData",
                        package="CARNIVAL"))
load(file = system.file("toy_network_ex2.RData",
                        package="CARNIVAL"))

carnivalOptions <- defaultLpSolveCarnivalOptions()

# Output dir
dir.create("ToyExample2", showWarnings = FALSE)
carnivalOptions$outputFolder <- "ToyExample2"

# lpSolve
toy_network_ex2 <- as.data.frame(toy_network_ex2)
colnames(toy_network_ex2) <- c("source", "interaction", "target")
toy_measurements_ex2 <- c("M1" = 1, "M2" = 1, "M3" = 1)

inverseCarnivalResults <- runInverseCarnival( measurements = c(toy_measurements_ex2),
                                              priorKnowledgeNetwork = toy_network_ex2,
                                              carnivalOptions = carnivalOptions)
```

which generates the following output:

```
print(inverseCarnivalResults)
```

```
## $weightedSIF
##           Node1 Sign Node2 Weight
## 1  Perturbation    1    I2    100
## 2  Perturbation    1    I3    100
## 3            I2    1    N1    100
## 4            I2    1    N2    100
## 5            N1    1    M1    100
## 6            N1    1    M2    100
## 7            N2    1    M2    100
## 8            N2    1    M3    100
## 9  Perturbation    1    I1    100
## 10 Perturbation   -1    I1    100
## 11 Perturbation   -1    I2    100
## 12 Perturbation   -1    I3    100
```

```
##
## $nodesAttributes
##           Node ZeroAct UpAct DownAct AvgAct nodesType
## 1           I1     100     0       0      0
## 2           I2       0   100       0    100
## 3           I3     100     0       0      0
## 4           N1       0   100       0    100
## 5           N2       0   100       0    100
## 6 Perturbation       0   100       0    100         P
## 7           M1       0   100       0    100         M
## 8           M2       0   100       0    100         M
## 9           M3       0   100       0    100         M
##
## $sifAll
## $sifAll[[1]]
##           Node1 Sign Node2
## 2            I2    1    N1
## 3            I2    1    N2
## 5            N1    1    M1
## 6            N1    1    M2
## 7            N2    1    M2
## 8            N2    1    M3
## 9  Perturbation    1    I1
## 10 Perturbation    1    I2
## 11 Perturbation    1    I3
## 12 Perturbation   -1    I1
## 13 Perturbation   -1    I2
## 14 Perturbation   -1    I3
##
##
## $attributesAll
## $attributesAll[[1]]
##          Nodes Activity
## 1           I2        1
## 2           N1        1
## 3           N2        1
## 4 Perturbation        1
## 5           M1        1
## 6           M2        1
## 7           M3        1
```

Toy Example - 3

Toy example for the CARNIVAL standard pipeline with generateLPFileCarnival and runCarnivalFromLp:

```
load(file = system.file("toy_perturbations_ex1.RData",
                        package = "CARNIVAL"))
load(file = system.file("toy_measurements_ex1.RData",
                        package = "CARNIVAL"))
load(file = system.file("toy_network_ex1.RData",
                        package = "CARNIVAL"))

carnivalOptions <- defaultLpSolveCarnivalOptions()
```

```
# Output dir
dir.create("ToyExample3", showWarnings = FALSE)
carnivalOptions$outputFolder <- "ToyExample3"

# lpSolve
generateLpFileCarnival(perturbations = toy_perturbations_ex1,
                       measurements = toy_measurements_ex1,
                       priorKnowledgeNetwork = toy_network_ex1,
                       carnivalOptions = carnivalOptions)
```

```
##                                                 lpFile
##        "ToyExample3//lpFile_t16_37_30d20_10_2021n50.lp"
##                                           parsedDataFile
## "ToyExample3//parsedData_t16_37_30d20_10_2021n50.RData"
```

which writes two files, an Lp File and a parsed Data file. The next function takes these files as inputs and runs the vanilla (standard) CARNIVAL pipeline with them.

```
lpFilename <- "toy_files_vignettes/lpFile_t18_01_53d28_04_2021n4.lp"
parsedDataFile <- "toy_files_vignettes/parsedData_t18_01_53d28_04_2021n4.RData"
resultsFromLp <- runFromLpCarnival(lpFile = lpFilename,
                                   parsedDataFile = parsedDataFile,
                                   carnivalOptions = defaultLpSolveCarnivalOptions())
```

which generates the following output:

```
resultsFromLp
```

```
## $weightedSIF
##   Node1 Sign Node2 Weight
## 1    I1    1    N1    100
## 2    N1    1    M1    100
## 3    N1    1    M2    100
## 4    I2   -1    N2    100
##
## $nodesAttributes
##   Node ZeroAct UpAct DownAct AvgAct nodesType
## 1   I1       0   100       0    100         P
## 2   N1       0   100       0    100
## 3   I2       0   100       0    100         P
## 4   N2     100     0       0      0
## 5   M1       0   100       0    100         M
## 6   M2       0   100       0    100         M
##
## $sifAll
## $sifAll[[1]]
##   Node1 Sign Node2
## 1    I1    1    N1
## 2    N1    1    M1
## 3    N1    1    M2
## 4    I2   -1    N2
##
##
## $attributesAll
## $attributesAll[[1]]
##   Nodes Activity
```

```
## 1    I1        1
## 2    N1        1
## 3    I2        1
## 4    M1        1
## 5    M2        1
```

Gurobi remote services

When using Gurobi solver, it is possibe to distribute the optimisation amongst multiple nodes of a computing cluster using the Gurobi remote services.

Gurobi remote services need to be set up by a system administrator of a cluster.

To use multiple cluster nodes when using gurobi in CARNIVAL, add the following options to `carnivalOptions` parameter:

```
distributed=TRUE,
WorkerPassword="<password>",
```

where `<password>` is given by cluster system administrator. The number of nodes will need to be specified through workload a workload manager, such as slurm.

```
sessionInfo()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-conda-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS/LAPACK: /home/bartosz/.miniconda3/envs/test_carnival/lib/libopenblasp-r0.3.17.so
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_GB.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] CARNIVAL_2.2.0 knitr_1.35
##
## loaded via a namespace (and not attached):
##  [1] igraph_1.2.6      magrittr_2.0.1    hms_1.1.1         R6_2.5.1
##  [5] rlang_0.4.11      fansi_0.4.2       stringr_1.4.0     tools_4.1.0
##  [9] lpSolve_5.6.15    xfun_0.24         utf8_1.2.2        cli_3.0.1
## [13] htmltools_0.5.1.1 ellipsis_0.3.2    yaml_2.2.1        digest_0.6.27
## [17] tibble_3.1.3      lifecycle_1.0.1   crayon_1.4.1      readr_1.4.0
## [21] vctrs_0.3.8       glue_1.4.2        evaluate_0.14     rmarkdown_2.11
## [25] stringi_1.7.3     compiler_4.1.0    pillar_1.6.4      pkgconfig_2.0.3
```