The R-scripts below are used for our analysis

1 Checking for power law distribution

```r
install.packages("poweRlaw")
library("poweRlaw")
readNetwork <- function(nwFileName, nwName) {
  inFile <- paste(nwFileName,sep="")
  return(read.csv(inFile, stringsAsFactors=FALSE))
}

simplify <- function(genes, rules){
  srules <- list()
  for (r  in 1:length(rules)) {
    rule <- gsub(" ", "", rules[r])
    rule <- gsub("&", ",",rule, fixed=TRUE)
    rule <- gsub("|", ",",rule, fixed=TRUE)
    rule <- gsub("(", "", rule, fixed=TRUE)
    rule <- gsub(")", "", rule, fixed=TRUE)
    rule <- gsub("!","", rule, fixed=TRUE)
    srules[r] <- list(which(genes %in% strsplit(rule,",")[[1]]))
  }
  return (srules)
}


countDegree <- function(network, name, noOfSims = 500,
    noOfThreads=8 ) {
  ccNw <- readNetwork(network, name)
  ccGenes <- ccNw[,1]
  ccRules <- ccNw[,2]
  sr <- simplify(ccGenes,ccRules)
  degs <- matrix(0,ncol=6, nrow=length(ccGenes), dimnames=list(
    ccGenes,list("out","inp","loop","total","Z (out)", "Z (total)
    ")))
  degs[,1] <- sapply(1:length(ccGenes),function(g) { length(sr[[
    g]]) })
  degs[,2] <- c(tabulate(unlist(sr)), rep(0,length(ccGenes)-max(
    unlist(sr))))
  degs[,3] <- sapply(1:length(ccGenes), function(g) { if (g %in%
     sr [[g]]) {return(1)} else {return(0)} } )
  degs[,1] <- degs[,1] - degs[,3]
  degs[,2] <- degs[,2] - degs[,3]
  degs[,4] <-  degs[,1] + degs[,2] + degs[,3]
  degMean <- mean(degs[,1])
  degSd <- sd(degs[,1])
  degs[,5] <- round((degs[,1] - degMean)/degSd,2)
  degMean <- mean(degs[,4])
  degSd <- sd(degs[,4])
  degs[,6] <- round((degs[,4] - degMean)/degSd,2)
  degtable <- degs
  degtable[degtable==0] <- NA
  m <- degs[,4]
  m <- m[m>0]
  m_pl <- displ$new(m)
  est <- estimate_xmin(m_pl)
  if(is.na(est$xmin)) {
    print("failed to set model parameters")
    pvalue <- "failed to set model parameters"
  } else {
    m_pl$setXmin(est)
    bt_pl <- bootstrap_p(m_pl, no_of_sims=noOfSims, threads=
    noOfThreads)
    pvalue <- bt_pl$p
  }
  return(list(bootstrap=bt_pl, degrees=degs))
}

network <- "Siegle2018.txt"
Zscore <- countDegree(network,network, noOfSims=100, noOfThreads
    =4)
Zscore
```

1.1 Results checking for power law distribution

```
$bootstrap
$p
[1] 0.93
$gof
[1] 0.05216334
```

```
$bootstraps
             gof                  xmin   pars               ntail
1            0.136942044995854    3      2.77369850967599   23
2            0.0448839682230041   7      11.2828388422027   9
3            0.161472167614597    3      2.54573985782762   21
4            0.090091448377601    4      4.00674699489996   17
5            0.178212507092397    3      2.56062655377231   21
6            0.103441728693891    3      2.96709120038193   23
7            0.194266611303647    3      2.5242843342734    20
8            0.1295607248718      4      4.47589240554146   16
9            0.1889976782448      3      2.69782012261854   23
10           0.117775958084985    7      6.68703215810997   9
11           0.0961654168950838   7      9.11649342014046   9
12           0.0962333646653397   3      3.15127847955675   22
13           0.11603816037344     8      8.73197598085696   6
14           0.189634854521846    4      3.42340722478621   17
15           0.0877570138763647   3      3.0970282872859    22
16           0.096042222906737    7      8.0589917607067    6
17           0.111301491350799    3      3.02926442684047   21
18           0.0476886487590654   4      5.02597054563359   15
19           0.124359486772465    4      3.90649316310233   17
20           0.0904281313383013   3      2.99291113155928   23
21           0.0846194602177959   4      4.83262754860118   11
22           0.171941921798433    3      2.79246372425374   22
23           0.0218682575410785   7      11.2204737304356   12
24           0.0465335214373354   7      10.6324188034946   8
25           0.0577875324023283   7      8.8147303669821    9
26           0.123630095225934    3      3.03983498305515   22
27           0.0977003221908667   4      3.66516075164814   17
28           0.0837971432809811   3      3.29447955373761   22
29           0.140124408875909    3      3.0511654843753    21
30           0.103988652515343    4      4.00967674092546   16
31           0.197542251090064    3      2.49268225147018   21
32           0.0885614907166987   7      8.76610350338565   7
33           0.0929187923515679   7      8.72239224557615   9
34           0.216688295226077    5      4.16396517712618   15
35           0.0856554181483967   3      3.08093578019419   22
36           0.083963081231796    3      3.3879807733751    21
37           0.136858403656694    3      2.57547614683141   22
38           0.110526123178566    4      3.93112587704033   18
39           0.081854456463847    7      8.50510403448387   7
40           0.150034641983164    3      2.6728853980334    22
41           0.0491453263403123   7      9.95245372259776   7
42           0.0724963010557631   4      4.02425435816921   15
43           0.0983414421372162   5      4.6852343229975    9
```

	gof	xmin	pars	ntail
44	0.115007762547291	3	2.81298990126627	21
45	0.13704479524806	3	2.91769592919336	22
46	0.148270898190632	3	2.66346619232764	22
47	0.135634052701317	4	4.65702820324682	15
48	0.125880799094738	3	2.93400612038704	22
49	0.126960365486805	3	2.86082800985222	22
50	0.3120114034084	2	1.84536289661792	23
51	0.14535803698718	3	2.89950394671102	22
52	0.192839133656966	3	2.4222086604436	22
53	0.0521606219582301	7	9.18631473567897	6
54	0.12053740416553	7	7.78937362091118	7
55	0.147370723920584	3	2.49742222225757	23
56	0.130469410466006	3	2.88033238455757	22
57	0.0521637640737111	7	9.18641949189525	6
58	0.125295579434874	3	2.78583209331276	22
59	0.041835946550092	7	11.7812783838926	10
60	0.119816381729715	7	6.31017847786434	11
61	0.169576704539143	3	2.65081695452721	21
62	0.102656900888864	7	8.60774016010095	9
63	0.100842935189801	4	4.7994009830004	15
64	0.213855318340983	3	2.73597075333549	22
65	0.140275350836933	3	2.79220738703189	23
66	0.0993133583780017	4	4.33048913413711	13
67	0.237297313553374	3	2.51052991336611	23
68	0.14397413262365	3	2.67957817141125	21
69	0.162534550205235	4	4.07593301268817	19
70	0.0767249710424621	7	9.99728480583154	9
71	0.0926842126762615	7	7.35259888987725	4
72	0.106140977631904	4	3.95527549105766	16
73	0.154952346819929	4	3.52760500274225	19
74	0.0711937607742632	3	3.03842058145965	21
75	0.137534967681493	3	2.81930157559197	22
76	0.106086923449147	7	7.95461250554254	7
77	0.0885227326156095	4	3.76388692440703	16
78	0.0822429902239181	7	9.40961387584867	8
79	0.191893588956594	3	2.71179846449867	21
80	0.0818549510014596	7	8.50511424348007	7
81	0.107114574116007	7	5.64144568968663	8
82	0.0926831741044603	7	7.35258067597884	4
83	0.107963260014716	3	2.83664997983981	21
84	0.106086857274663	7	7.95460862500374	7
85	0.139758626978447	3	2.61842615916193	22
86	0.0784894241633813	4	4.46117459292847	15
87	0.0638990867667695	7	8.32797923134336	5

```
     gof                   xmin  pars               ntail
88   0.120536377526133     7     7.78934996678828   7
89   0.148163480304126     4     3.54231198442264   17
90   0.0909176194490049    3     3.41506563096534   22
91   0.0978410708389754    4     4.25643129874371   17
92   0.263176253913127     3     2.37831007370035   21
93   0.17263435865817      3     2.61446807860418   23
94   0.117734074718171     4     3.92392770793886   14
95   0.165658946106819     3     3.01915809336431   22
96   0.124415505883316     3     3.15155276837951   21
97   0.0521666011389047    7     9.18651460406775   6
98   0.172332711259803     3     2.612991432889     23
99   0.089172779596194     3     3.56693354605009   22
100  0.101949569547948     3     3.19180473041733   21
```

```
$sim_time
[1] 0.06135431
$seed
NULL
$package_version
[1] '0.70.2'
$distance
[1] "ks"
attr(,"class")
[1] "bs_p_xmin"
```

	out	inp	loop	total	Z (out)	Z (total)
Wnt	0	6	1	7	-1,89	1,19
axin	2	1	0	3	-0,25	-0,87
GSK3b	3	4	0	7	0,57	1,19
DC	2	1	0	3	-0,25	-0,87
bcatenin	1	2	0	3	-1,07	-0,87
TCF	3	0	0	3	0,57	-0,87
FoxO	2	1	0	3	-0,25	-0,87
Rho	5	4	0	9	2,21	2,22
Rac	4	3	0	7	1,39	1,19
MEKK1	2	1	0	3	-0,25	-0,87
JNK	2	2	0	4	-0,25	-0,36
PKC	3	2	0	5	0,57	0,16
IGF	0	2	1	3	-1,89	-0,87
IRS	3	1	0	4	0,57	-0,36
PI3K	3	4	0	7	0,57	1,19
Akt	2	4	0	6	-0,25	0,67
TSC2	3	2	0	5	0,57	0,16
mTORC1	1	1	0	2	-1,07	-1,39
S6K	2	2	0	4	-0,25	-0,36
Ras	2	2	0	4	-0,25	-0,36
Raf	3	1	0	4	0,57	-0,36
ERK	1	3	0	4	-1,07	-0,36
mTORC2	4	4	0	8	1,39	1,7

2 Removing superfluous out nodes

```
input  : Boolean network $F = (F_1, \dots, F_n)$
output: Reduced network
1 $\mathsf{In} \leftarrow \{x_i \mid F_i(x) = x_i\}$
2 while $\exists x_l \in F, x_{i_1}, \dots, x_{i_k} \in \mathsf{In} : F_l(x) = f(x_{i_1}, \dots, x_{i_k})$ do
3     remove $F_l$ from ruleset $F$
4     substitute all occurences of $x_l$ in $F$ by $f(x_{i_1}, \dots, x_{i_k})$
5 end
6 while $\exists x_l \in F : x_l$ is not contained in any rule $F_i$ do
7     remove $F_l$ from ruleset $F$
8 end
```

```r
readNetwork <- function(nwFileName, nwName) {
  inFile <- paste(nwFileName,sep="")
  return(read.csv(inFile, stringsAsFactors=FALSE))
}

simplify <- function(genes, rules){
  srules <- list()
  for (r  in 1:length(rules)) {
    rule <- gsub(" ", "", rules[r])
    rule <- gsub("&", ",",rule, fixed=TRUE)
    rule <- gsub("|", ",",rule, fixed=TRUE)
    rule <- gsub("(", "", rule, fixed=TRUE)
    rule <- gsub(")", "", rule, fixed=TRUE)
    rule <- gsub("!","", rule, fixed=TRUE)
    srules[r] <- list(which(genes %in% strsplit(rule,",")[[1]]))
  }
  return (srules)
}

removeOutnodes <- function(network, name) {
  ccNw <- readNetwork(network, name)
  ccGenes <- ccNw[,1]
  print(paste("Genes: ", paste(ccGenes,collapse=", ")))
  ccRules <- ccNw[,2]
  ccInputs <- simplify(ccGenes, ccRules)
  sr <- simplify(ccGenes,ccRules)
  names(sr) <- ccGenes
  tsr <- table(unlist(sr))
  n <- length(sr)
  while (length(tsr) < n) {
    n <- length(tsr)
    x <- 1:length(sr)
    y <- x[-as.integer(names(tsr))]
    print(y)
    sr[y] <- NA
    tsr <- table(unlist(sr))
  }

  srem <- ccNw[sapply(sr,function(s) { sum(is.na(s))==0 }),]
  writeNetwork(paste("reduced_",network, sep=""), srem)
  return(srem)
}

writeNetwork <- function(nwFileName, network) {
  outFile <- paste(nwFileName,sep="")
  return(write.csv(network,outFile,quote=FALSE, row.names=FALSE)
    )
}

nwFile <- "Siegle2018.txt"
srem <- removeOutnodes(nwFile,"")
print(srem)
```

2.1 Results Removing superfluous out nodes

```
[1] "Genes: Wnt, axin, GSK3b, DC, bcatenin, TCF, FoxO, Rho, Rac, MEKK1, JNK, PKC,
    IGF, IRS, PI3K, Akt, TSC2, mTORC1, S6K, Ras, Raf, ERK, mTORC2"
```

```
[1] 6
[1] 6 7
[1] 5 6 7
[1] 4 5 6 7
[1] 2 4 5 6 7
```

	targets	factors
1	Wnt	Wnt
3	GSK3b	!(Wnt | ERK | Akt)
8	Rho	(Wnt | PI3K | mTORC2) & !(Rac | PKC)
9	Rac	(Wnt | PI3K | mTORC2) & !Rho
10	MEKK1	Rac | Rho
11	JNK	MEKK1 | Rac
12	PKC	Rho | Wnt | mTORC2
13	IGF	IGF
14	IRS	IGF & !(S6K & JNK)
15	PI3K	(IRS | Ras) & !Rho
16	Akt	PI3K | mTORC2
17	TSC2	!(Akt | ERK) | GSK3b
18	mTORC1	!TSC2
19	S6K	mTORC1 | GSK3b
20	Ras	IGF | Wnt
21	Raf	(Ras | PKC) & !Akt
22	ERK	Raf
23	mTORC2	!(S6K | GSK3b) & (PI3K | TSC2)

3 Determine implicant set

Select increasing number of input genes until evaluation leads to a full state for all truth assignements to the selected input genes.

3.1 Heuristic approach

```
input  : Boolean network $F = (F_1, \dots, F_n)$
output: A minimal implicant set
G ← $\emptyset$
repeat
    w ←nodeWeight($F$,$G$)
    g ← $x_i$, where $w(x_i) = \max(w)$
    G ←G $\cup$ $\{g\}$
    observables ←observableStates($F$,$G$)
until observables;
return $G$
nodeWeight($F$,$G$)
    for all nodes $x_i$ in $G$ do
        remove rule $F_i$ and all occurences of $x_i$ from $F$
    end
    for all nodes $x_i$, where $F_i$ depends only on $x_i$ do
        w ($x_i$) ← $n$                                  // max weight
    end
    for all nodes $x_i$, where $F_i$ depends only on $x_j$, $i \neq j$ do
        w ($x_i$) ← 0                                   // min weight
    end
    for all nodes $x_i$, where $w(x_i)$ is not already set do
        w ($x_i$) ← the number of $j$ such that $F_j$ depends on $x_i$
    end
    return w
```

Nodes are selected with decreasing node weight.

```
install.packages("combinat")
library("combinat")

readNetwork <- function(nwFileName, nwName) {
  inFile <- paste(nwFileName,sep="")
  return(read.csv(inFile, stringsAsFactors=FALSE))
}

simplify <- function(genes, rules){
  srules <- list()
  for (r  in 1:length(rules)) {
    rule <- gsub(" ", "", rules[r])
    rule <- gsub("&", ",",rule, fixed=TRUE)
    rule <- gsub("|", ",",rule, fixed=TRUE)
    rule <- gsub("(", "", rule, fixed=TRUE)
    rule <- gsub(")", "", rule, fixed=TRUE)
    rule <- gsub("!","", rule, fixed=TRUE)
    srules[r] <- list(which(genes %in% strsplit(rule,",")[[1]]))
  }
  return (srules)
}

initcircuit <- function(genes, stateValues){
  nextState <- list()
  for (i in 1:length(stateValues)){
    nextState[genes[i]] <- stateValues[i]
  }
  return (nextState)
}

sufficientEvalcircuit <- function(genes, rules, inputs, states)
   {
  nextState <- list()
  for (i in 1:length(genes)) {
    assign(genes[i], states[[i]])
  }
  for (i in 1:length(genes)) {
    ruleInputs <- inputs[[i]]
    ruleInputsState <- states[ruleInputs]
    if (min(as.integer(ruleInputsState))>=0) {
      nextState[genes[i]] <- eval(parse(text=rules[i]))
    } else {
      x <- which(ruleInputsState < 0)
      k <- length(x)
      bin <- as.integer(intToBits(0))
      ruleInputsState[x] <- bin[1:k]
      evalGene <- evalGeneCircuit(genes, ruleInputs, rules[i],
   ruleInputsState)
      nextState[genes[i]] <- evalGene
      for (tt in 1:(2**k-1)) {
        bin <- as.integer(intToBits(tt))
        ruleInputsState[x] <- bin[1:k]
        if (evalGene != evalGeneCircuit(genes, ruleInputs, rules
    [i], ruleInputsState)) {
          nextState[genes[i]] <- states[genes[i]]
          break
        }
      }
    }
  }
  return (nextState)
}

evalGeneCircuit <- function(genes, geneInputs, rule, geneStates)
     {
  for (i in 1:length(geneInputs)) {
    gi <- genes[geneInputs[i]]
    assign(gi, as.logical(geneStates[i]))
  }
  nextState <-  eval(parse(text=rule))
  return (nextState)
}

nextNodeId <- function(init=FALSE) {
  if (init) {
    nodeId <<- 1
  } else {
    nodeId <<- nodeId + 1
  }
  return (nodeId)
}

testGeneset <- function(ccGenes,ccRules,ccInputs,inputGenes,
    verbose=TRUE) {
  determined <- TRUE
  geneMapping <- list()
  if (verbose)
    print(paste("Selected: ", paste(ccGenes[inputGenes],collapse
    =", ")))
  k <- length(inputGenes)
  lastbin <- rep(-1,k)
  for (finiteStates in 0:(2**k-1)) {
    bin <- rev(as.integer(intToBits(finiteStates))[1:k])
    if (identical(bin[1:length(lastbin)],lastbin))
      next
    firstL <- 0
    for (firstl in 1:k) {
      firstL <- firstl
```

```r
      ccState <- initcircuit(ccGenes, rep(-1,length=length(ccGenes)))
      ccState[inputGenes[1:firstL]] <- bin[1:firstL]
      booleanState <- FALSE
      statetransitions <- list()
      for (nextStep in 1:20) {
        statetransitions[nextStep] <- list(unname(unlist(ccState)))
        ccState <- sufficientEvalcircuit(ccGenes,ccRules, ccInputs, ccState)
        if (min(as.integer(ccState)) >= 0) {
          booleanState <- TRUE
          break
        }
      }
      if (booleanState) {
        statetransitions[nextStep+1] <- list(unname(unlist(ccState)))
        break
      }
    }
    if (booleanState) {
      lastbin <- bin[1:firstL]
    } else {
      determined=FALSE
      return (NA)
    }
    gs <- matrix(unlist(statetransitions),
                 ncol = length(ccGenes), byrow = TRUE,  dimnames=list(list(),ccGenes))
    if (verbose) {
      initState <- initcircuit(ccGenes, rep(-1,length=length(ccGenes)))
      initState[inputGenes[1:firstL]] <- bin[1:firstL]
      print(unlist(initState[inputGenes[1:firstL]]))
      print(gs)
    }
    geneMapping[paste(c(lastbin,rep(-1,k-length(lastbin))), collapse=",")] <- list(gs)
  }
  return (geneMapping)
}

removeSingle <- function(ct) {
  ct <- sapply(ct,unique)
  num <- which(sapply(1:length(ct), function(ind){length(ct[[ind]])==1 && ct[[ind]]>0 && ct[[ind]]!=ind})==TRUE)
  while (length(num)>0) {
    ind <- num[1]
    pind <- paste("^", ind, "$", sep="")
    print(paste("gene ", ind, " propagates gene ", ct[ind]))
    ct <- sapply(ct,function(rule) {as.integer(gsub(pind, ct[[
   ind]], rule)) })
    ct[[ind]] <- c(-nextNodeId())
    ct <- sapply(ct,unique)
    num <- which(sapply(1:length(ct), function(ind){length(ct[[
   ind]])==1 && ct[[ind]]>0 && ct[[ind]]!=ind})==TRUE)
  }
  return(ct)
}

selectGenesSet <- function(network, name, k=20, all=FALSE,
    verbose=FALSE) {
  nextNodeId(TRUE)
  ccNw <- readNetwork(network, name)
  ccGenes <- ccNw[,1]
  k <- min (k,length(ccGenes))
  print(paste("Network: ", name))
  print(paste("Genes: ", paste(ccGenes,collapse=", ")))
  ccRules <- ccNw[,2]
  ccInputs <- simplify(ccGenes, ccRules)
  ct <- ccInputs
  res <- rep(0,k)
  ctGenes <- c()
  for (i in 1:k) {
    ct <- removeSingle(ct)
    st <- 0
    num <- which(sapply(1:length(ct), function(ind){length(ct[[
   ind]])==1 && ct[[ind]]==ind})==TRUE)
    if (length(num)>0) {
      st <- num[[1]]
      print(paste("input gene: ", st))
    } else {
      xt <- table(unlist(ct))
      print(xt)
      st <- names(which(xt==max(xt)))[1]
      print(paste("max weight node: ", st))
    }
    if (st <1)
      break
    ct[as.integer(st)] <- c(-nextNodeId())
    ct <- sapply(ct,function(cn) {cn[cn!=st]})
    res[i] <- st
    ctGenes <- c(ctGenes,as.integer(st))
    selectedSet <- testGeneset(ccGenes,ccRules,ccInputs,ctGenes,
     verbose)
    if (!all & length(selectedSet[!is.na(selectedSet)]))
      break
  }
  res <- res[res>0]
  names(res) <- ccGenes[as.integer(res)]
  return (as.integer(res))
}

solutionSet <- list()

hasSolution <- function(solution, init=FALSE) {
  if (init) {
    solutionSet <<- list()
    return (FALSE)
  } else {
    newSol <- paste(sort(solution), collapse=",")
    if (is.null(solutionSet[[newSol]])) {
      return (FALSE)
    } else {
      return (TRUE)
    }
  }
}

addSolution <- function(solution, value) {
  newSol <- paste(sort(solution), collapse=",")
  solutionSet[newSol] <<- value
}

isSolution <- function(solution) {
  newSol <- paste(sort(solution), collapse=",")
  return (solutionSet[newSol])
}

recursive.selectGenesSet <- function(network, name, k=20, all=
    FALSE) {
  hasSolution(c(),init=TRUE)
  ccNw <- readNetwork(network, name)
  ccGenes <- ccNw[,1]
  k <- min (k,length(ccGenes))
  print(paste("Genes: ", paste(ccGenes,collapse=", ")))
  ccRules <- ccNw[,2]
  ccInputs <- simplify(ccGenes, ccRules)
  ct <- ccInputs
  ctGenes <- c()
  return (recSelectGeneset(ccGenes,ccRules,ccInputs,ct,ctGenes,k
    ,all))
}

recSelectGeneset <- function(ccGenes,ccRules,ccInputs,ct,ctGenes
    ,k, all) {
```

```r
  if (k < 1) {
    print("k<1")
    print(unlist(ctGenes))
    return(NA)
  }
  sts <- which(sapply(1:length(ct), function(ind){length(ct[[ind
    ]])==1 && ct[[ind]]==ind})==TRUE)
  if (length(sts)==0) {
    xts <- table(unlist(ct))
    sts <- as.integer(names(which(xts==max(xts))))
  }
  sts <- sts[sts>0]
  if (length(sts)==0) {
    print("no gene found")
    return (NA)
  }
  ctGs <- sapply (sts, function(st, ct, ctGenes, k) {
    ct[st] <- c(-k)
    ct <- sapply(ct,function(cn) {cn[cn!=st]})
    ctGenes <- c(ctGenes,as.integer(st))
    selectedSet <- c()
    if (hasSolution(ctGenes)) {
      print("has solution")
      print(ctGenes)
      return (NA)
    } else {
      print("add solution")
      print(ctGenes)
      addSolution(ctGenes,TRUE)
    }
    selectedSet <- testGeneset(ccGenes,ccRules,ccInputs,ctGenes,
     verbose=FALSE)
    if (!all & length(selectedSet[!is.na(selectedSet)])>0) {
      return(ctGenes)
    } else {
      return (recSelectGeneset(ccGenes,ccRules,ccInputs,ct,
    ctGenes,k-1,all))
    }
  }, ct, ctGenes, k)
  return(ctGs)
}

nwFile <- "Siegle2018.txt"
selectedSet <- selectGenesSet(nwFile, nwFile, k=20)
print(selectedSet)
```

3.1.1 Results Heuristic approach

```
[1] "Network: Siegle2018.txt"
[1] "Genes: Wnt, axin, GSK3b, DC, bcatenin, TCF, FoxO, Rho, Rac, MEKK1, JNK, PKC,
    IGF, IRS, PI3K, Akt, TSC2, mTORC1, S6K, Ras, Raf, ERK, mTORC2"
[1] "gene 5 propagates gene 4"
[1] "gene 18 propagates gene 17"
[1] "gene 22 propagates gene 21"
[1] "input gene 1"
[1] "gene 2 propagates gene 21"
[1] "gene 20 propagates gene 13"
[1] "input gene: 13"
```

```
-8  -7  -6  -5  -4  -3  -2  3  4  7  8  9  10  11  12  14  15  16  17  19  21  23
1   1   1   1   1   1   1   4  2  1  4  3  1   2   2   1   4   4   2   2   3   4
```

```
[1] "max weight node: 3"
[1] "gene 4 propagates gene 21"
[1] "gene 19 propagates gene 17"
```

```
-11  -10  -9  -8  -7  -6  -5  -4  -3  -2  7  8  9  10  11  12  14  15  16  17  21
1    1    1   1   1   1   1   1   1   1   1  4  3  1   2   2   1   4   3   2   3
```

```
[1] "max weight node: 8"
[1] "gene 10 propagates gene 9"
[1] "gene 11 propagates gene 9"
[1] "gene 12 propagates gene 23"
[1] "gene 15 propagates gene 14"
```

```
-16  -15  -14  -13  -12  -11  -10  -9  -8  -7  -6  -5  -4  -3  -2  7  9  14  16  17
1    1    1    1    1    1    1    1   1   1   1   1   1   1   1   1  2  3   3   2
```

```
[1] "max weight node: 14"
[1] "gene 9 propagates gene 23"
[1] "gene 16 propagates gene 23"
[1] "gene 21 propagates gene 23"
[1] "gene 7 propagates gene 23"
[1] "gene 6 propagates gene 23"
[1] "gene 17 propagates gene 23"
[1] "input gene: 23"
[1] 1 13 3 8 14 23
```

3.2 Exhaustive approach

```
input  : Boolean network $F = (F_1, \ldots, F_n)$
output: Minimal implicant sets

success ← false
implicants ← $\emptyset$
for increasing values of $k$ do
    for all possible selections $G = \{x_{i_1}, \ldots, x_{i_k}\}$ of $k$ nodes do
        observables ←observableStates($F$,$G$)
        if observables then
            success ← true
            implicants ←implicants ∪ $\{G\}$
        end
    end
    if success then
        return implicants
    end
end

observableStates($F$,$G$)
    for every assignment $a$ to the $k$ nodes of $G$ do
        let the assignment $a$ be undefined for all remaining n-k nodes
        evaluate $F$ until an attractor $A_a$ ist found
        if $A_a$ is undefined on some node then
            return false
        end
    end
    return true
```

```
testNetwork <- function(network, name, verbose=FALSE,kmin = 1,
    kmax=6) {
    ccNw <- readNetwork(network, name)
    ccGenes <- ccNw[,1]
    print(paste("Genes: ", paste(ccGenes,collapse=", ")))
    ccRules <- ccNw[,2]
    ccInputs <- simplify(ccGenes, ccRules)
    for (k in kmin:kmax) {
       print(paste("k:", k))
       genesets <- combn(1:length(ccGenes),k, simplify=FALSE)
       selectedSets <- sapply(genesets, function(geneset) {
          geneset <- unlist(geneset)
          a <- paste(ccGenes[geneset],collapse=", ")
          l <- list(testGeneset(ccGenes,ccRules,ccInputs,geneset
    , verbose))
          names(l) <- a
          return(l)
        })
        selectedSets <- selectedSets[!is.na(selectedSets)]
        if (verbose) {
          print(selectedSets)
        }
        if (length(selectedSets)>0)
          break
    }
    return(selectedSets)
  }

nwFile <- "Siegle2018.txt"
selectedSet <- selecteGenesSet(nwFile, nwFile, k=10)
combSet <- testNetwork(nwFile,nwFile)
print(names(combSet))
```

3.2.1 Results Exhaustive approach

```
[1] "Genes: Wnt, axin, GSK3b, DC, bcatenin, TCF, FoxO, Rho, Rac, MEKK1, JNK, PKC,
    IGF, IRS, PI3K, Akt, TSC2, mTORC1, S6K, Ras, Raf, ERK, mTORC2"
[1] "k: 1"
[1] "k: 2"
[1] "k: 3"
[1] "k: 4"
[1] "Wnt, Rho, IGF, mTORC2"
```

4 Intervention effects

4.1 Comparison attractors of implicant set and exhaustive attractor search

```r
library("BoolNet")
boolNetwork <- loadNetwork("Siegle2018.txt" ,symbolic=T)
boolNetwork
InterventionNode<- c("IGF")
InterventionAttrsKo <- function(boolNetwork, InterventionNode) {

  id <- which(boolNetwork$genes == InterventionNode)

  sptAttrs <- getAttractors(boolNetwork, method="sat.exhaustive"
    )
  pertKo<-fixGenes(boolNetwork, InterventionNode, 0)
  pertKoAttrs <- getAttractors(pertKo,method="sat.exhaustive")
  pertOe<-fixGenes(boolNetwork, InterventionNode, 1)
  pertOeAttrs <- getAttractors(pertOe,method="sat.exhaustive")
  print("Intervention Attrs Ko: ")
  print(sapply(pertKoAttrs$attractors,nrow))
  print("Intervention Attrs Oe: ")
  print(sapply(pertOeAttrs$attractors,nrow))
  print("Exh Attrs")
  print(sapply(sptAttrs$attractors,nrow))
  missingAttrs <- list()
  print("Missing Attrs")
  inKO <- sapply(sptAttrs$attractors, function(attr) any(sapply(
    pertKoAttrs$attractors, function(koAttr) identical(attr[,-id
    ],koAttr[,-id]))))
  inOE <- sapply(sptAttrs$attractors, function(attr) any(sapply(
    pertOeAttrs$attractors, function(oeAttr) identical(attr[,-id
    ],oeAttr[,-id]))))
  missing <- !(inKO | inOE)
  print(missing)
  return (sptAttrs$attractors[missing])
}

InterventionAttrsKo(boolNetwork, InterventionNode)
```

4.1.1 Results of the comparison attractors of implicant set and exhaustive attractor search

```
[1] "Intervention Attrs Ko: "
[1] 1 1
[1] "Intervention Attrs Oe: "
[1] 3 1 1
[1] "Exh Attrs"
[1] 3 1 1 1 1
[1] "Missing Attrs"
[1] FALSE FALSE FALSE FALSE FALSE
list()
```

4.2 Identical exhaustive and knockout attractors

```
pertKo<-fixGenes(boolNetwork, InterventionNode, 0)
pertKoAttrs <- getAttractors(pertKo,method="sat.exhaustive")
pertKoAttrs$attractors


InterventionAttrsKo <- function(boolNetwork, InterventionNode) {

  id <- which(boolNetwork$genes == InterventionNode)

  sptAttrs <- getAttractors(boolNetwork, method="sat.exhaustive"
    )
  pertKo<-fixGenes(boolNetwork, InterventionNode, 0)
  pertKoAttrs <- getAttractors(pertKo,method="sat.exhaustive")
  pertOe<-fixGenes(boolNetwork, InterventionNode, 1)
  pertOeAttrs <- getAttractors(pertOe,method="sat.exhaustive")
  print("Intervention Attrs Ko: ")
  print(sapply(pertKoAttrs$attractors,nrow))
  print("Intervention Attrs Oe: ")
  print(sapply(pertOeAttrs$attractors,nrow))
  print("Exh Attrs")
  print(sapply(sptAttrs$attractors,nrow))
  missingAttrs <- list()
  print("Missing Attrs")
  inExh <- sapply(pertKoAttrs$attractors, function(attr) any(
    sapply(sptAttrs$attractors, function(sptAttr) identical(attr
    [,-id],sptAttr[,-id]))))
  missing <- !(inExh)
  print(missing)
  return (pertKoAttrs$attractors[missing])
}
```

```
InterventionAttrsKo(boolNetwork, InterventionNode)
```

4.2.1 Results identical exhaustive and knockout attractors

```
[1] "Intervention Attrs Ko: "
[1] 1 1
[1] "Intervention Attrs Oe: "
[1] 3 1 1
[1] "Exh Attrs"
[1] 3 1 1 1 1
[1] "Missing Attrs"
[1] FALSE FALSE
list()
```

4.3 Identical exhaustive and overexpression attractors

```
pertOe<-fixGenes(boolNetwork, InterventionNode, 1)
pertOeAttrs <- getAttractors(pertOe,method="sat.exhaustive")
pertOeAttrs$attractors


InterventionAttrsKo <- function(boolNetwork, InterventionNode) {

  id <- which(boolNetwork$genes == InterventionNode)

  sptAttrs <- getAttractors(boolNetwork, method="sat.exhaustive"
    )
  pertKo<-fixGenes(boolNetwork, InterventionNode, 0)
  pertKoAttrs <- getAttractors(pertKo,method="sat.exhaustive")
  pertOe<-fixGenes(boolNetwork, InterventionNode, 1)
  pertOeAttrs <- getAttractors(pertOe,method="sat.exhaustive")
  print("Intervention Attrs Ko: ")
  print(sapply(pertKoAttrs$attractors,nrow))
  print("Intervention Attrs Oe: ")
  print(sapply(pertOeAttrs$attractors,nrow))
  print("Exh Attrs")
  print(sapply(sptAttrs$attractors,nrow))
  missingAttrs <- list()
  print("Missing Attrs")
  inExh <- sapply(pertOeAttrs$attractors, function(attr) any(
    sapply(sptAttrs$attractors, function(sptAttr) identical(attr
    [,-id],sptAttr[,-id]))))
  missing <- !(inExh)
  print(missing)
  return (pertOeAttrs$attractors[missing])
}
```

```
InterventionAttrsKo(boolNetwork, InterventionNode)
```

4.3.1 Results identical exhaustive and overexpression attractors

```
[1] "Intervention Attrs Ko: "
[1] 1 1
[1] "Intervention Attrs Oe: "
[1] 3 1 1
[1] "Exh Attrs"
[1] 3 1 1 1 1
[1] "Missing Attrs"
[1] FALSE FALSE FALSE
list()
```

4.4 Identical attractors after overexpression in the attractor set of knockout experiments

```
InterventionAttrsKo <- function(boolNetwork, InterventionNode) {

  id <- which(boolNetwork$genes == InterventionNode)

  sptAttrs <- getAttractors(boolNetwork, method="sat.exhaustive"
    )
  pertKo<-fixGenes(boolNetwork, InterventionNode, 0)
  pertKoAttrs <- getAttractors(pertKo,method="sat.exhaustive")
  pertOe<-fixGenes(boolNetwork, InterventionNode, 1)
  pertOeAttrs <- getAttractors(pertOe,method="sat.exhaustive")
  print("Intervention Attrs Ko: ")
  print(sapply(pertKoAttrs$attractors,nrow))
  print("Intervention Attrs Oe: ")
  print(sapply(pertOeAttrs$attractors,nrow))
  print("Exh Attrs")
  print(sapply(sptAttrs$attractors,nrow))
  missingAttrs <- list()
  print("Missing Attrs")
  inKO <- sapply(pertOeAttrs$attractors, function(attr) any(
    sapply(pertKoAttrs$attractors, function(KoAttr) identical(
    attr[,-id], KoAttr[,-id]))))
  missing <- !(inKO)
  print(missing)
  return (pertOeAttrs$attractors[missing])
}

InterventionAttrsKo(boolNetwork, InterventionNode)
```

4.4.1 Results identical attractors after overexpression in the attractor set of knockout experiments

```
[1] "Intervention Attrs Ko: "
[1] 1 1
[1] "Intervention Attrs Oe: "
[1] 3 1 1
[1] "Exh Attrs"
[1] 3 1 1 1 1
[1] "Missing Attrs"
[1] TRUE FALSE TRUE
[[1]]
```

```
   Wnt  axin  GSK3b   DC   bcatenin  TCF  FoxO  Rho     Rac  MEKK1
1  0    1     0       0    0         1    0     0       0    0
2  0    1     1       0    1         0    0     1       1    0
3  0    1     0       1    1         1    0     0       0    1
   JNK  PKC   IGF     IRS  PI3K      Akt  TSC2  mTORC1  S6K  Ras
1  1    0     1       0    1         0    0     0       0    1
2  0    0     1       1    1         1    1     1       0    1
3  1    1     1       1    0         1    1     0       1    1
   Raf  ERK   mTORC2
1  0    0     0
2  1    0     1
3  0    1     0
```

```
   Wnt  axin  GSK3b   DC   bcatenin  TCF  FoxO  Rho     Rac  MEKK1
1  0    1     0       0    1         1    0     0       1    1
   JNK  PKC   IGF     IRS  PI3K      Akt  TSC2  mTORC1  S6K  Ras
1  1    0     1       0    1         1    0     1       1    1
   Raf  ERK   mTORC2
1  0    0     0
```

4.5 Identical attractors after knockout in the attractor set of overexpression experiments

```
InterventionAttrsKo <- function(boolNetwork, InterventionNode) {

  id <- which(boolNetwork$genes == InterventionNode)
```

```r
  sptAttrs <- getAttractors(boolNetwork, method="sat.exhaustive")
  pertKo<-fixGenes(boolNetwork, InterventionNode, 0)
  pertKoAttrs <- getAttractors(pertKo,method="sat.exhaustive")
  pertOe<-fixGenes(boolNetwork, InterventionNode, 1)
  pertOeAttrs <- getAttractors(pertOe,method="sat.exhaustive")
  print("Intervention Attrs Ko: ")
  print(sapply(pertKoAttrs$attractors,nrow))
  print("Intervention Attrs Oe: ")
  print(sapply(pertOeAttrs$attractors,nrow))
  print("Exh Attrs")
  print(sapply(sptAttrs$attractors,nrow))
  missingAttrs <- list()
  print("Missing Attrs")
  inOE <- sapply(pertKoAttrs$attractors, function(attr) any(sapply(pertOeAttrs$attractors, function(oeAttr) identical(attr[,-id],oeAttr[,-id]))))
  missing <- !(inOE)
  print(missing)
  return (pertKoAttrs$attractors[missing])
}

InterventionAttrsKo(boolNetwork, InterventionNode)
```

4.5.1 Results identical attractors after overexpression and knock-out

```
[1] "Intervention Attrs Ko: "
[1] 1 1
[1] "Intervention Attrs Oe: "
[1] 3 1 1
[1] "Exh Attrs"
[1] 3 1 1 1 1
[1] "Missing Attrs"
[1] FALSE TRUE
[[1]]
```

```
     Wnt   axin   GSK3b    DC    bcatenin   TCF   FoxO   Rho      Rac   MEKK1
  1  0     1      1        1     0          0     0      0        0     0
     JNK   PKC    IGF      IRS   PI3K       Akt   TSC2   mTORC1   S6K   Ras
  1  0     0      0        0     0          0     1      0        1     0
     Raf   ERK    mTORC2
  1  0     0      0
```