Metrics to Assess Algorithm Performance

Mixed graph markings:

0 Nothing (no edge exists)

1 Undetermined

2 Arrowhead

3 Tail

Note: The $*$ mark used in these rules represents a wildcard that denotes any of the three marks. If this symbol appears in the consequent of a rule, then the remark remains the same as it was in the antecedent.

```r
library(LocalFCI)

## Loading required package:  bnlearn

data("asiaDAG")
data("asiadf")
asiadf <- as.matrix(asiadf)
```

Introduction

For all metrics, we are comparing the adjacency matrix of an estimated graph to a graph regarded as the "ground truth." In our simulations, we regard the subgraph of the CPDAG as the ground truth we will measure our estimates against. The subgraph contains all of the target nodes and the union of their neighborhoods.

`allMetrics`

In this section, we will check the validity of the constituent functions producing the metrics returned by this function. The implementation for all of these may be found in `metrics.cpp`.

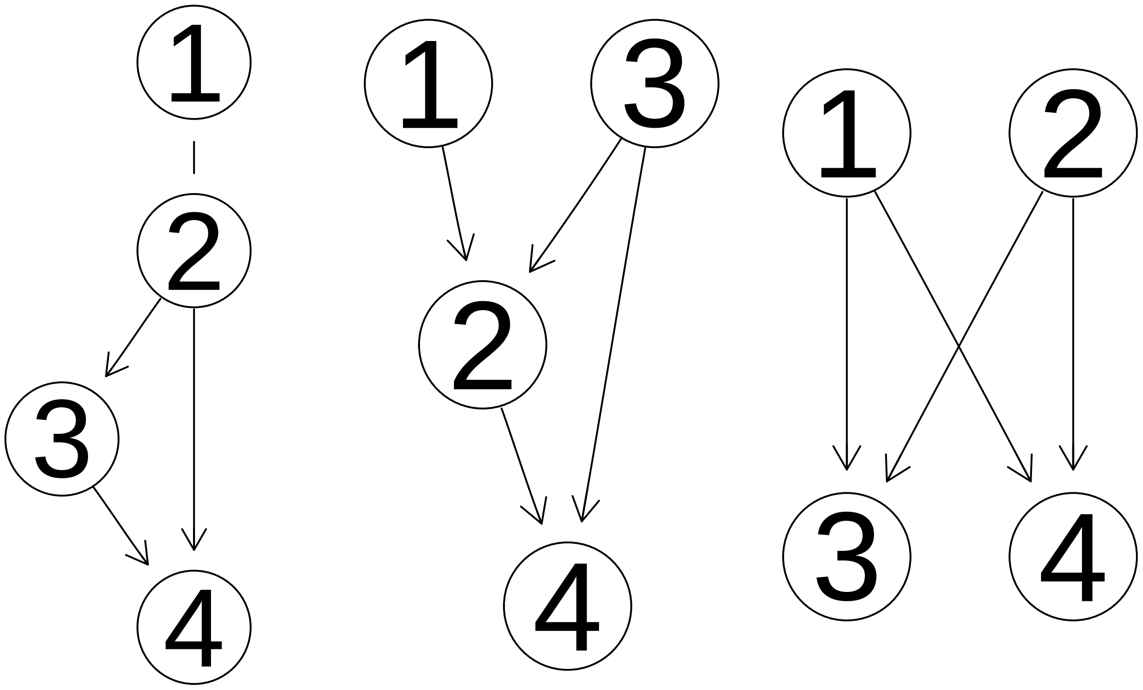

Figure 1: From left to right, we have the true graph, the graph with the perfect skeleton, and the graph with an incorrect skeleton.

Skeleton Estimation Metrics

```r
true_amat <- matrix(c(0,1,0,0,
                      1,0,1,1,
                      0,0,0,1,
                      0,0,0,0),byrow = TRUE,nrow = 4)

perfect_skel <- matrix(c(0,1,0,0,
                         0,0,0,1,
                         0,1,0,1,
                         0,0,0,0),byrow = TRUE,nrow = 4)

false_skel <- matrix(c(0,0,1,1,
                       0,0,1,1,
                       0,0,0,0,
                       0,0,0,0),byrow = TRUE,nrow = 4)
```

```r
compareSkeletons(false_skel,true_amat,targets = c(1,3))

## $skel_fp
## [1] 2
```

```
##
## $skel_fn
## [1] 2
##
## $skel_correct
## [1] 2

compareSkeletons(false_skel,true_amat,targets = 3)

## $skel_fp
## [1] 0
##
## $skel_fn
## [1] 1
##
## $skel_correct
## [1] 2

compareSkeletons(perfect_skel,true_amat,targets = 3)

## $skel_fp
## [1] 0
##
## $skel_fn
## [1] 0
##
## $skel_correct
## [1] 3

compareSkeletons(perfect_skel,true_amat,targets = c(1,3))

## $skel_fp
## [1] 0
##
## $skel_fn
## [1] 0
##
## $skel_correct
## [1] 4
```

Note the differences in the values for each call are due to the differences in the provided target node. The false skeleton correctly has edges $(2, 3)$ and $(2, 4)$. It is missing edges $(3, 4)$ and $(1, 2)$. It adds edges $(1, 3)$ and $(1, 4)$.
In our implementation, we only count edges from nodes that are found in the same target neighborhood in the ground truth. In this way, we won't wrong penalize any remaining edges

from our algorithm that link separate neighborhoods.

V-Structure Estimation Metrics

In this implementation, we again only consider those v-structures found within the same target neighborhood in the ground truth graph, so we will not penalize any v-structures found between target neighborhoods. Also note that this implementation only recognizes notation for a typical graph without additional edge marks.

We use the same graphs shown above.

```
compareVStructures(false_skel,true_amat,targets = c(0,3),TRUE)

## Checking: 1, 2, and 3
## Estimated Graph:
## No unshielded triple
##
## True Graph:
## No unshielded triple
## $missing
## [1] 0
##
## $added
## [1] 0
##
## $correct
## [1] 0

compareVStructures(false_skel,true_amat,targets = 3,TRUE)

## Checking: 1, 2, and 3
## Estimated Graph:
## No unshielded triple
##
## True Graph:
## No unshielded triple
## $missing
## [1] 0
##
## $added
## [1] 0
##
## $correct
## [1] 0

compareVStructures(perfect_skel,true_amat,targets = 1,TRUE)
```

```
## Checking: 0, 1, and 2
## Estimated Graph:
## 0 -> 1 <- 2
## True Graph:
## v-structure *not* present in other graph
## True Graph:
## No unshielded triple
##
## Checking: 0, 1, and 3
## Estimated Graph:
## No unshielded triple
##
## True Graph:
## No unshielded triple
##
## Checking: 0, 2, and 3
## Estimated Graph:
## No unshielded triple
##
## True Graph:
## No unshielded triple
##
## Checking: 1, 2, and 3
## Estimated Graph:
## No unshielded triple
##
## True Graph:
## No unshielded triple
## $missing
## [1] 0
##
## $added
## [1] 1
##
## $correct
## [1] 0
```

```
compareVStructures(false_skel,true_amat,targets = 1,TRUE)
```

```
## Checking: 0, 1, and 2
## Estimated Graph:
## 0 -> 2 <- 1
## True Graph:
## v-structure *not* present in other graph
## True Graph:
```

```
## No unshielded triple
##
## Checking: 0, 1, and 3
## Estimated Graph:
## 0 -> 3 <- 1
## True Graph:
## v-structure *not* present in other graph
## True Graph:
## No unshielded triple
##
## Checking: 0, 2, and 3
## Estimated Graph:
## No unshielded triple
##
## True Graph:
## No unshielded triple
##
## Checking: 1, 2, and 3
## Estimated Graph:
## No unshielded triple
##
## True Graph:
## No unshielded triple
## $missing
## [1] 0
##
## $added
## [1] 2
##
## $correct
## [1] 0
```

Parent Recovery Metrics

One of the primary goals of this algorithm is to properly identify parents of target nodes so that we can estimate the causal effects on downstream nodes.
In our accounting of target parent recovery, we identify true positives, false negatives, and false positives. In addition, we have another category called "potential" parents. These nodes could be

- An identified parent in the estimated graph where there is an undirected edge in the ground truth
- An undirected edge in the estimated graph where there is a parental relationship identified in the ground truth

- Undirected edges in both the estimated graph and the true graph

```
parentRecoveryAccuracy(perfect_skel,true_amat,targets = 3)

## $missing
## [1] 0
##
## $added
## [1] 0
##
## $correct
## [1] 2
##
## $potential
## [1] 0

parentRecoveryAccuracy(false_skel,true_amat,targets = 3)

## $missing
## [1] 1
##
## $added
## [1] 1
##
## $correct
## [1] 1
##
## $potential
## [1] 0

parentRecoveryAccuracy(false_skel,true_amat,targets = c(0,3))

## $missing
## [1] 1
##
## $added
## [1] 1
##
## $correct
## [1] 1
##
## $potential
## [1] 0
```

Next, we add an undirected edge in the graph with the false skeleton and observe the changes this makes in the reported metrics on parent recovery.

```
# Adding a potential
false_skel[3,4] <- 1
false_skel[4,3] <- 1
parentRecoveryAccuracy(false_skel,true_amat,targets = 3)

## $missing
## [1] 1
##
## $added
## [1] 1
##
## $correct
## [1] 1
##
## $potential
## [1] 1

# Using 1-index numbering:
# Missing 3 -> 4, correctly has 2->3 and 2->4, has 3-4 as undirected edge,
# added 1->4 and 1->3
parentRecoveryAccuracy(false_skel,true_amat,targets = c(2,3))

## $missing
## [1] 1
##
## $added
## [1] 2
##
## $correct
## [1] 2
##
## $potential
## [1] 1

parentRecoveryAccuracy(false_skel,true_amat,targets = c(1,2,3))

## $missing
## [1] 1
##
## $added
## [1] 2
##
## $correct
## [1] 2
##
## $potential
## [1] 1
```

Change the edge to a bi-directed edge using the values from mixed graphs to observe the changes once again.

```r
false_skel[3,4] <- 2
false_skel[4,3] <- 2
parentRecoveryAccuracy(false_skel,true_amat,targets = 3)

## $missing
## [1] 1
##
## $added
## [1] 1
##
## $correct
## [1] 1
##
## $potential
## [1] 0
```

Inter-Neighborhood Edge Metrics

One of the advantages of using the local FCI algorithm is that we are able to retain and orient some of the edges between neighborhoods. These metrics intend to demonstrate the value of this algorithm in identifying these relationships.

Algorithm 1 Inter-Neighborhood Edge Metrics

```
1: Input: Estimated graph adj. matrix est, Ground truth graph adj. matrix ref
2: Find pairs of nodes $(i, j)$ that are in different target neighborhoods
3: if Nodes $i$ and $j$ are not connected in the estimated graph then
4:     if Node $j$ (or $i$) is an ancestor of node $i$ ($j$), and the path from $j$ to $i$ ($i$ to $j$) is unmediated by
       another considered node then
5:         Increment the number of missing ancestral relationships
6:     end if
7: else
8:     if The edge is oriented with a proper unmediated ancestral relationship then
9:         Increment the number of true ancestors
10:    else if The edge is undirected or bidirected then
11:        Increment the number of those missing orientation
12:    else if The edge reverses the ancestral relationship and the path is unmediated then
13:        Increment the number with reversed orientations
14:    else if There is a directed edge present without an ancestral relationship then
15:        Increment the number of false positive arrowheads
16:    else
17:        Increment the number of false positive connections
18:    end if
19: end if
```

Overall F1 Score

- A true positive (TP) is when the orientation exactly matches in both graphs
- A true negative is when there is no edge in both graphs
- A false positive is when there is an edge in the estimated graph but no edge in the true graph
- A false negative is whenever there is an edge in the true graph which does not exactly match the edge in the estimated graph

$$F_1 = \frac{2 \times TP}{2 \times TP + FP + FN}$$