A Guide to the unifDAG Package for R
(unifDAG version: 1.0.2)

Markus Kalisch

April 4, 2019

Contents

1 Introduction

For some applications, such as obtaining good starting points for Markov Chain methods, it is desirable to sample a DAG uniformly from the space of all (labelled) DAGs given p nodes. While it is trivial to sample labelled *undirected* graphs with p nodes uniformly, sampling DAGs uniformly is much more difficult.

Suppose, for example, we want to sample uniformly from the labelled DAGs on two nodes A and B. There are three such DAGs:

- $G_1: \ A \to B$
- $G_2: \ A \leftarrow B$
- $G_3: \ A \quad B$ (i.e., without an edge between A and B).

With uniform sampling each of these graphs should be sampled with probability $\frac{1}{3}$, i.e. $P(G_1) = P(G_2) = P(G_3) = \frac{1}{3}$.

One naive approach could be first sampling a labelled undirected graph on p nodes uniformly and then sampling uniformly from all DAGs on the sampled undirected graph (i.e. choosing a random permutation of the labels and thus fixing an order).

Let us illustrate this using the previous example. On two nodes A and B, there are two undirected graphs possible: $A \quad B$ and $A - B$. If we sample $A \quad B$ nothing needs to be oriented an we end up with G_3. If we sample $A - B$, we choose a random order of the nodes A and B and thus would obtain $G_1 : A \to B$ and $G_2 : A \leftarrow B$ each with conditional proability 0.5.

It is easy to see that with this approach $P(G_1) = 0.25$, $P(G_2) = 0.25$ and $P(G_3) = 0.5$.

Thus, this approach does not lead to a uniform distribution on the space of DAGs with p nodes but overrepresents sparse DAGs. In particular, the empty DAG with p nodes is much more likely than a particular complete DAG on p nodes: While the empty and the complete undirected graph are equally likely, all permutations of the empty undirected graph will lead to the empty DAG, while every permutation of the complete undirected graph will lead to a different DAG.

2 Enumeration methods

This problem was solved in [1] by relating each DAG to a sequence of outpoints (nodes with no incoming edges) and then to a composition of integers. The package `pcalg` implements this solution in two ways: Function `unifDAG()` performs the exact procedure using precomputed enumeration tables and is feasible for DAGs with up to 100 nodes. The function `unifDAG.approx()` uses the exact procedure for DAGs up to a specified number of nodes (option `n.exact`). For larger numbers of nodes an approximation is used instead. The accuracy of the approximation is based on the option `n.exact` and will be within the uniformity limits of a 32 (64) bit integer sampler when set to `n.exact=20` (`n.exact=40`). Thus, for practical purposes these approximations are indistiguishable from the exact solution but are much faster to compute. Both functions can optionally generate edge weights.

In the following example we first sample a DAG (`dag1`) uniformly from the space of all DAGs with $p = 10$ nodes using the exact method. Then, we sample a DAG (`dag2`) uniformly from the space of all DAGs with $p = 150$ nodes using the approximate method. The option `n.exact=40` is used, so that the sampling procedure will match the exact sampling procedure on a 64-bit integer sampler. In both cases, the edge weights are sampled independently from $Uniform(0, s)$.

```
myWgtFun <- function(m, lB, uB) {
    runif(m, lB, uB)
}
set.seed(123)
dag1 <- unifDAG(n = 10, weighted = TRUE, wFUN = list(myWgtFun, 0,
    2))
dag2 <- unifDAG.approx(n = 150, n.exact = 40, weighted = TRUE, wFUN = list(myWgtFun,
```

```
    0, 2))
```

Returning to our original example with two nodes, we see that function `unifDAG()` indeed samples G_1, G_2 and G_3 with roughly equal frequencies.

```
cnt <- c(0, 0, 0)  ## count occurances of G1, G2 and G3
set.seed(123)
for (i in 1:100) {
    g <- unifDAG(n = 2, weighted = FALSE)
    m <- as(g, "matrix")  ## adjacency matrix
    if ((m[2, 1] == 0) & (m[1, 2] == 0)) {
        cnt[3] <- cnt[3] + 1  ## G3
    } else if (m[2, 1] == 0) {
        cnt[1] <- cnt[1] + 1  ## G1
    } else {
        cnt[2] <- cnt[2] + 1  ## G2
    }
}
cnt

## [1] 31 41 28
```

References

[1] Jack Kuipers and Giusi Moffa. Uniform random generation of large acyclic digraphs. *Statistics and Computing*, 25(2):227–242, 2015.