# Generative DAGs as an Interface Into Probabilistic Programming with the R Package causact

**Adam J. Fleischhacker**[1] **and Thi Hong Nhung Nguyen**[2]

**1** Adam Fleischhacker, JP Morgan Chase Faculty Fellow, University of Delaware, Newark, DE 19716 **2** Institute for Financial Services Analytics, University of Delaware, Newark, DE 19716

**DOI:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:** Open Journals

**Reviewers:**

- @openjournals

**Submitted:** 01 January 1970
**Published:** unpublished



## Summary

The causact package provides R functions for visualizing and running inference on generative directed acyclic graphs (DAGs). Once a generative DAG is created, the package automates Bayesian inference via the greta package (Golding, 2019) and TensorFlow (Dillon et al., 2017). The package eliminates the need for three separate versions of a model: 1) the narrative describing the problem, 2) the statistical model representing the problem, and 3) the code enabling inference written in a probabilistic programming language. Instead, causact users create one unified model, a generative DAG, using a visual representation.

## Statement of Need

Bayesian data analysis mixes data with domain knowledge to quantify uncertainty in unknown outcomes. Its beautifully-simple theoretical underpinnings are deployed in three main steps (Gelman et al., 2013):

- **Modelling:** Joint probability distributions are specified to encode domain knowledge about potential data generating processes.
- **Conditioning:** Bayes rule is used to reallocate plausibility among the potential data generating processes to be consistent with both the encoded domain knowledge and the observed data. The conditioned model is known as the posterior distribution.
- **Validation:** Evidence is collected to see whether the specified model as well as the computational implementation of the model and conditioning process are to be trusted or not.

Algorithmic advances in the *conditioning* step of Bayesian data analysis have given rise to a new class of programming languages called probabilistic programming languages (PPLs). Practical and complex statistical models which are analytically intractable can now be solved computationally using inference algorithms. In particular, Markov Chain Monte Carlo (MCMC) algorithms (Congdon, 2010; Gelfand & Smith, 1990; Gilks & Roberts, 1996) handle arbitrarily large and complex models via highly effective sampling processes that quickly detect high-probability areas of the underlying distribution Kruschke (2014).

The causact package, presented in this paper, focuses on solving a three-language problem that occurs during Bayesian data analysis. First, there is the language of the domain expert which we refer to as the *narrative* of how data is generated. Second, there is the language of *math* where a statistical model, amenable to inference, is written. Lastly, there is the language of *code*, where a PPL language supports computational inference from a well-defined statistical model. The existence of these three languages creates friction as diverse stakeholders collaborate to yield insight from data; often mistakes get made in both communicating and translating between the three languages. Prior to causact, any agreed upon narrative of a data-generating process must ultimately be modelled in code using an error-prone process where model misspecification, variable indexing errors, prior distribution omissions, and other mismatches between desired model and coded model go easily unnoticed.

To unify inference-problem narratives, the statistical models representing those narratives, and the code implementing the statistical models, `causact` introduces a modified visualization of *directed acyclic graphs* (DAGs), called the *generative DAG*, to serve as a more intuitive and collaborative interface into probabilistic programming languages and to ensure faithful abstractions of real-world data generating processes.

# Modelling with Generative DAGs

Generative DAGs pursue two simultaneous goals. One goal is to capture the narrative by building a conceptual understanding of the data generating process that lends itself to statistical modelling. And two, gather all the mathematical elements needed for specifying a complete Bayesian model of the data generating process. Both of these goals will be satisfied by iteratively assessing the narrative and the narrative's translation into rigorous mathematics using `causact` functions.

Capturing the narrative in code uses some core `causact` functions like `dag_create()`, `dag_node()`, `dag_edge()`, and `dag_plate()` with the chaining operator `%>%` used to build a DAG from the individual elements. `dag_render()` or `dag_greta()` are then used to visualize the DAG or run inference on the DAG, respectively. The simplicity with which generative DAGs are constructed belies the complexity of models which can be supported. For example, multi-level or hierarchical models are easily constructed as shown here in code for constructing and visualizing an oft-cited Bayesian example known as eight schools (Sturtz et al., 2005) whose data is included in `causact` (`causact::schoolsDF`). The example is a study of coaching effects on test scores where students from eight schools were put into coached and uncoached groups.

```
R> graph = dag_create() %>%
+    dag_node(”Treatment Effect”,”y”,
+             rhs = normal(theta, sigma),
+             data = causact::schoolsDF$y) %>%
+    dag_node(”Std Error of Effect Estimates”,”sigma”,
+             data = causact::schoolsDF$sigma,
+             child = ”y”) %>%
+    dag_node(”Exp. Treatment Effect”,”theta”,
+             child = ”y”,
+             rhs = avgEffect + schoolEffect) %>%
+    dag_node(”Population Treatment Effect”,”avgEffect”,
+             child = ”theta”,
+             rhs = normal(0,30)) %>%
+    dag_node(”School Level Effects”,”schoolEffect”,
+             rhs = normal(0,30),
+             child = ”theta”) %>%
+    dag_plate(”Observation”,”i”,nodeLabels = c(”sigma”,”y”,”theta”)) %>%
+    dag_plate(”School Name”,”school”,
+              nodeLabels = ”schoolEffect”,
+              data = causact::schoolsDF$schoolName,
+              addDataNode = TRUE)
R> graph %>% dag_render()
```

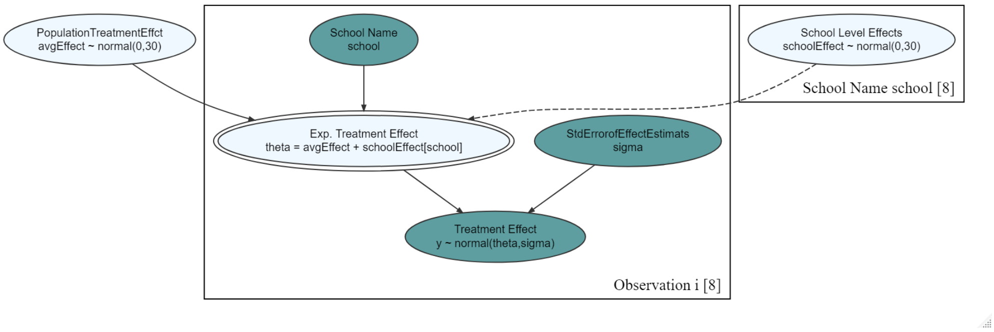


**Figure 1:** A generative DAG of the eight schools model.

Figure 2 replicates Figure 1 without math for less intimidating discussions with domain experts about the model using the `shortLabel = TRUE` argument (shown below). `causact` does not require a complete model specification prior to rendering the DAG, hence, `causact` facilitates qualitative collaboration on the model design between less technical domain experts and the model builder.

```
R> graph %>% dag_render(shortLabel = TRUE)
```

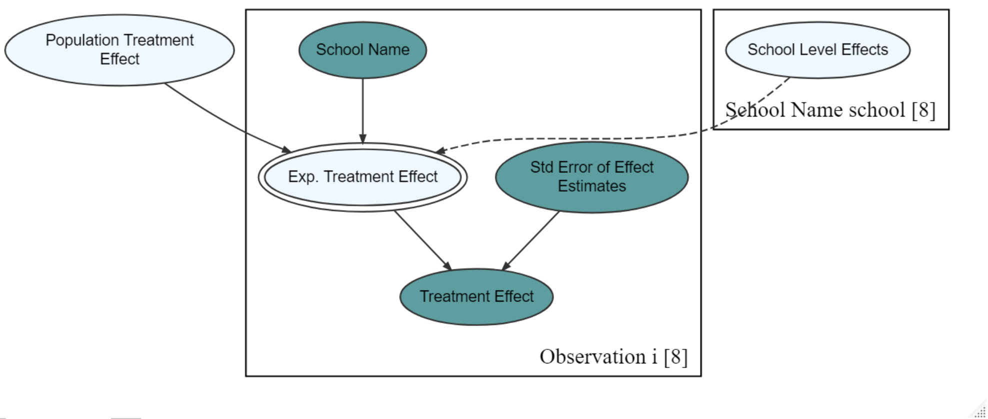


**Figure 2:** Hiding mathematical details to facilitate collaborations with domain experts.

All visualizations, including Figure 1 and Figure 2, are created via `causact`'s calls to the `DiagrammeR` package (Iannone, 2020). The `dag_diagrammer()` function can convert a `causact_graph` to a `dgr_graph` (the main object when using `DiagrammeR`) for further customizing of a visualization using the `DiagrammeR` package.

Sampling from the posterior of the eight schools model (Figure 1) does not require a user to write PPL code, but rather a user will simply pass the generative DAG object to `dag_greta()` and then inspect the data frame of posterior draws:

```
R> library(greta) ## greta uses TensorFlow to get sample
R> drawsDF = graph %>% dag_greta()
R> drawsDF
```

```
# A tibble: 4,000 x 9
   avgEffect schoolEffect_Sc~ schoolEffect_Sc~ schoolEffect_Sc~
       <dbl>            <dbl>            <dbl>            <dbl>
 1     0.102             40.1             3.59            -4.51
```

```
 2      4.59          23.6          4.43          -26.3
 3     -0.451         18.5         24.3            16.5
 4     18.9            8.07       -26.3           -28.6
 5     17.3           -5.83        -4.25          -26.2
 6      1.97          42.7          2.25           12.6
 7     12.7          -11.2          5.31          -16.5
 8      9.11         -17.4          9.09          -12.7
 9     -3.74          71.5          1.82           23.6
10     -2.43          48.2        -13.3             2.89
# ... with 3,990 more rows, and 5 more variables:
#   schoolEffect_School4 <dbl>, schoolEffect_School5 <dbl>,
#   schoolEffect_School6 <dbl>, schoolEffect_School7 <dbl>,
#   schoolEffect_School8 <dbl>
```

Behind the scenes, causact creates the model's code equivalent using the greta PPL, but this is typically hidden from the user. However, for debugging or further customizing a model, the greta code can be printed to the screen without executing it by setting the mcmc argument to FALSE:

```
R> graph %>% dag_greta(mcmc=FALSE)
```

```
sigma <- as_data(causact::schoolsDF$sigma)    #DATA
y <- as_data(causact::schoolsDF$y)            #DATA
school     <- as.factor(causact::schoolsDF$schoolName)   #DIM
school_dim <- length(unique(school))   #DIM
schoolEffect <- normal(mean = 0, sd = 30, dim = school_dim) #PRIOR
avgEffect    <- normal(mean = 0, sd = 30)                   #PRIOR
theta  <- avgEffect + schoolEffect[school]    #OPERATION
distribution(y) <- normal(mean = theta, sd = sigma)   #LIKELIHOOD
gretaModel  <- model(avgEffect,schoolEffect)   #MODEL
meaningfulLabels(graph)
draws       <- mcmc(gretaModel)               #POSTERIOR
drawsDF     <- replaceLabels(draws) %>% as.matrix() %>%
                dplyr::as_tibble()            #POSTERIOR
tidyDrawsDF <- drawsDF %>% addPriorGroups()  #POSTERIOR
```

The produced greta code is shown in the above code snippet. The code can be difficult to digest for some and exemplifies the advantages of working visually using causact. The above code is also challenging to write without error or misinterpretation. Indexing is particularly tricky in PPL's with indexing based on meaningless numbers (e.g. 1,2,3,...). To facilitate quicker interpretation causact abbreviates posterior parameters using human-interpretable names.

The output of dag_greta() is in the form of a data frame of draws from the joint posterior. To facilitate a quick look into posterior estimates, the dagp_plot() function creates a simple visual of 90% credible intervals. It is the only core function that does not take a graph as its first argument. By grouping all parameters that share the same prior distribution and leveraging the meaningful parameter names constructed using dag_greta(), it allows for quick comparisons of parameter values.

```
R> drawsDF %>% dagp_plot()
```

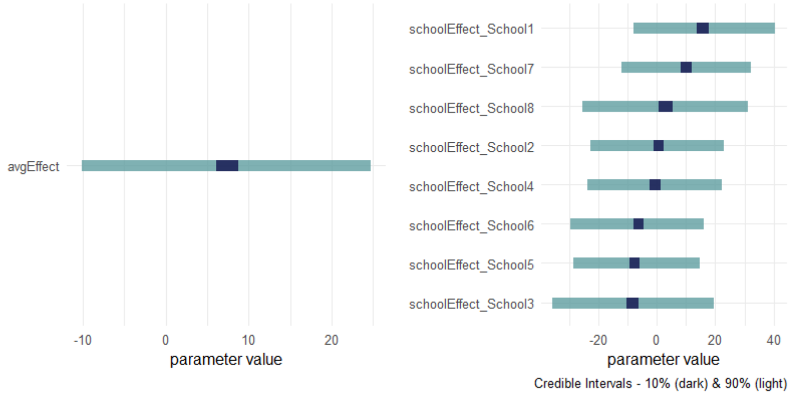


**Figure 3:** Credible intervals for the nine parameters of the eight schools model.

The code above makes the plot in Figure 3. For further posterior plotting, users would make their own plots using `ggplot2` (Wickham, 2016), `ggdist` (Kay, 2020), or similar. For further model validation, including MCMC diagnostics, the user would use a package like `bayesplot` (Gabry et al., 2019) or `shinystan` (Gabry, 2018). For users who prefer to work with an `mcmc` object, they can extract the `draws` object after running the generated `greta` code from `dag_greta(mcmc=FALSE)` or find the object in the `cacheEnv` environment after running `dag_greta(mcmc=FALSE)` using `get("draws",envir = causact:::cacheEnv)`.

# Comparison to Other Packages

By focusing on generative DAG creation as opposed to PPL code, `causact` liberates users from the need to learn complicated probabilistic programming languages. As such, it is similar in spirit to any package whose goal is to make Bayesian inference accessible without learning a PPL. In this vein, causact is similar to `brms` (Bürkner, 2017), `rstanarm` (Goodrich et al., 2020), and `rethinking` (McElreath, 2020a) - three `R` packages which leverage `Stan` (Stan Development Team, 2021) for Bayesian statistical inference with MCMC sampling. Like the `rethinking` package which is tightly integrated with a textbook (McElreath, 2020b), a large motivation for developing `causact` was to make learning Bayesian inference easier. The package serves a central role in a textbook titled *A Business Analyst's Introduction to Business Analytics: Intro to Bayesian Business Analytics in the R Ecosystem.* (Fleischhacker, 2020).

# Conclusion

The causact modelling syntax is flexible and encourages modellers to make bespoke models. The long-term plan for the `causact` package is to promote a Bayesian workflow that philosophically mimics the Principled Bayesian Workflow outlined by Betancourt (2020). The structure of a generative DAG is sure to be much more transparent and interpretable than most other modern machine learning workflows; this is especially true when models are made accessible to those without statistical or coding expertise. For this reason, generative DAGs can help facilitate effective communication between modelers and domain users both during the designing process of the models and when explaining the results returned by the models.

# Acknowledgements

The `Stan` Development team has been inspirational for this work and has formed a wonderful Bayesian inference community around their powerful language. Additionally, the books of Kruschke (2014) and McElreath (2020b) are tremendous resources for learning Bayesian data analysis and their pedagogy is aspirational. This work would not be possible without the greta dev team and special thanks to Nick Golding and Nick Tierney. Lastly, thanks to the University of Delaware students, MBAs and PhDs, who have contributed time, code, testing, and enthusiasm for this project from its beginning.

# References


Betancourt, M. (2020). *Towards a principled Bayesian workflow*. https://betanalpha.github.io/assets/case_studies/principled_bayesian_workflow.html.

Bürkner, P.-C. (2017). Brms: An r package for Bayesian multilevel models using stan. *Journal of Statistical Software*, *80*(1), 1–28.

Congdon, P. D. (2010). *Applied Bayesian hierarchical methods*. CRC Press.

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., & Saurous, R. A. (2017). Tensorflow distributions. *arXiv Preprint arXiv:1711.10604*.

Fleischhacker, A. (2020). *A business analyst's introduction to business analytics: Intro to Bayesian business analytics in the r ecosystem*. Independently Published. ISBN: 9798667128175

Gabry, J. (2018). *Shinystan: Interactive visual and numerical diagnostics and posterior analysis for Bayesian models*. https://CRAN.R-project.org/package=shinystan

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, *182*(2), 389–402.

Gelfand, A. E., & Smith, A. F. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, *85*(410), 398–409.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis*. CRC press.

Gilks, W. R., & Roberts, G. O. (1996). Strategies for improving MCMC. *Markov Chain Monte Carlo in Practice*, *6*, 89–114.

Golding, N. (2019). Greta: Simple and scalable statistical modelling in r. *Journal of Open Source Software*, *4*(40), 1601.

Goodrich, B., Gabry, J., Ali, I., & Brilleman, S. (2020). *Rstanarm: Bayesian applied regression modeling via Stan*. https://mc-stan.org/rstanarm

Iannone, R. (2020). *DiagrammeR: Graph/network visualization*. https://github.com/rich-iannone/DiagrammeR

Kay, M. (2020). *ggdist: Visualizations of distributions and uncertainty*. https://doi.org/10.5281/zenodo.3879620

Kruschke, J. (2014). *Doing Bayesian data analysis: A tutorial with r, JAGS, and stan*.

McElreath, R. (2020a). *Rethinking: Statistical rethinking book package*.

McElreath, R. (2020b). *Statistical rethinking: A Bayesian course with examples in r and stan*. CRC press.

Neal, R. M. (1993). *Probabilistic inference using markov chain monte carlo methods*. Department of Computer Science, University of Toronto Toronto, Ontario, Canada.

Pfeffer, A. (2016). *Practical probabilistic programming*. Manning Publ.

Stan Development Team. (2021). *Stan Modeling Language User's Guide and Reference Manual, Version 2.26*. http://mc-stan.org/

Sturtz, S., Ligges, U., & Gelman, A. (2005). R2WinBUGS: A package for running WinBUGS from r. *Journal of Statistical Software, Articles*, *12*(3), 1–16. https://doi.org/10.18637/jss.v012.i03

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. ISBN: 978-3-319-24277-4