Review Article

Amy J. Pitts* and Charlotte R. Fowler

Comparison of open-source software for producing directed acyclic graphs

https://doi.org/10.1515/jci-2023-0031
received May 19, 2023; accepted October 26, 2023

Abstract: Many software packages have been developed to assist researchers in drawing directed acyclic graphs (DAGs), each with unique functionality and usability. We examine five of the most common software to generate DAGs: Ti*k*Z, DAGitty, ggdag, dagR, and igraph. For each package, we provide a general description of its background, analysis and visualization capabilities, and user-friendliness. In addition, in order to compare packages, we produce two DAGs in each software, the first featuring a simple confounding structure and the second with a more complex structure with three confounders and a mediator. We provide recommendations for when to use each software depending on the user's needs.

Keywords: directed acyclic graphs, TikZ, DAGitty, ggdag, dagR, igraph

MSC 2020: 92B15, 62D20, 97Kxx, 68R10

1 Introduction

Research describing how to establish causal relationships has become of increased interest in many disciplines [1–6], especially in cases where a randomized controlled experiment is not feasible. One key tool to visualize hypothesized causal relationships, identify where biases may arise, and inform how to address them is a directed acyclic graph (DAG) [1,2,5]. These graphs provide a display of the connections between exposure, outcome, and other relevant variables. DAGs are employed across disciplines including epidemiology [7–9], sociology [10–12], education [13–15], and economics [16–18]. DAGs consist of nodes and edges, where the nodes represent variables and the edges convey direct causal effects by displaying an arrow leaving from the cause and pointing toward the effect. Importantly, a graph qualifies as a DAG if no variable is an ancestor of itself, meaning no cycles occur in the graph, and each edge is pointed in a single direction [19]. For a DAG to be considered causal, it is required to include all variables that are common causes of any two existing variables in the graph [1].

Developers have introduced software to produce DAGs across a variety of platforms. DAGitty, dagR, ggdag, igraph, pcalg, and bnlearn are open-source packages in R offering a range of plotting and analysis capabilities [20–25]. DAGitty offers both a browser-based platform and an R package for creating, editing, and analyzing causal diagrams [20]. Ggdag extends the plotting functionality of DAGitty and is tidyverse and ggplot compatible [21]. dagR focuses on analysis and data simulation capabilities and provides a framework to draw, manipulate, and evaluate DAGs [23]. The R package igraph is designed for network analysis and is especially capable of handling large graphical systems [22]. pcalg not only centers around causal structure learning and causal inference discovery but also has some visualization features [24,26]. The bnlearn package also focuses on causal discovery through Bayesian network structure learning and parameter learning [25]. In Python,

*** Corresponding author: Amy J. Pitts,** Department of Biostatistics, Columbia University, New York, NY 10027, United States, e-mail: ajp2257@cumc.columbia.edu
Charlotte R. Fowler: Department of Biostatistics, Columbia University, New York, NY 10027, United States

three prominent libraries for causal graphs are causal-learn, causal discovery toolbox, and gCastle [27–29]. All three software packages are focused on their algorithms for causal discovery but have some limited DAG plotting capabilities. In the document preparation system LaTeX, the graphing library Ti*k*Z is commonly used to draw DAGs [30]. Quiver, a web-based application, allows users to quickly draw a DAG through click and drag motions and has the functionality to export created DAGs to LaTeX code [31]. Causal fusion is a similar web-based application; however, access to this resource requires an approved account [32]. Tetrad, a free downloadable tool with over 30 years of history, creates, simulates, estimates, tests, and predicts causal and statistical models using DAGs [33]. This tool's functionality is very similar to pcalg and bnlearn. With this wide array of DAG drawing software, knowing which option is the most appropriate and easily implementable is a challenge. In 2006, Haughton et al. provided a comparison article [34] that reviewed three statistical methods to illustrate DAGs (MIM, Tetrad, and WinMine). Since their publication, statistical methods to illustrate DAGs have changed; indeed, Tetrad is the only software compared by Haughton et al. that is still maintained.

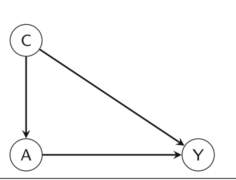
While numerous methods clearly exist for designing, analyzing, and visualizing a DAG using software, there is no centralized resource comparing modern methods or providing recommendations. DAGs are a commonly used visual tool in publications; indeed, Tennant et al. analyzed a collection of 234 articles published between 1999 and 2017 that mentioned concepts related to DAGs and found that two-thirds of the articles made at least one DAG available [35]. However, Tennant et al. noted that such DAGs ranged drastically in size, quality, and notation [35]. This is likely in part due to the lack of a comprehensive description of the available software for drawing DAGs. Here, we provide a guide that highlights, compares, and demonstrates how to employ each DAG software.

In our review, we include open-source software that provides a manual of all features or published documentation and can be implemented directly in popular programming languages among statisticians, such as R or LaTeX. We additionally choose to restrict our review to packages focused on visualizing and producing DAGs, and thus exclude pcalg, bnlearn, causal-learn, causal discovery toolbox, and gCastle because their main purpose is causal discovery and analysis. This restricts our scope to five resources: Ti*k*Z, DAGitty, ggdag, dagR, and igraph. We evaluate the software across three categories: visual design, analysis capability, and utility. For visual design, we assess whether the software can include curved edges, subscripts, and math notations, the default visual settings, design customization capabilities, and the ability to allow for auto-generated and user-specified node placements. Next, to evaluate analysis capability, we check for the presence of an exposure, outcome, and covariate framework; the ability to identify ancestor/descendant relationships, conditional independencies, and minimally sufficient adjustment sets; and the capacity to simulate data. Finally, we evaluate the utility of the five packages by comparing the resources available, our experience of the learning curve, and the required base software. In Section 2, we describe each software across evaluation criteria and compare performance between methods. In the discussion, we provide general recommendations specific to the user's needs and future directions for DAG software.

2 DAG producing methods

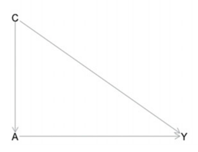
To compare the five software packages, we create the same two DAGs using each program. In Figure 1, we implement an identical simple confounding causal structure in each software and compare the output. In Figure 2, we draw a more complex causal relationship, including one mediator and three confounders. For Figures 1 and 2, we include below the plots the code used to produce each DAG. We create the DAGs using DAGitty (web version 3.0, CRAN version 0.3-1), ggdag (version 0.2.10), dagR (version 1.2.1), and igraph (version 1.5.0.1) with R-4.2.0 [36] and the Ti*k*Z graphs in LaTeX 2022 with Ti*k*Z (version 3.1.10). In Table 1, we summarize the capabilities of each software based on its visual design, analysis capability, and utility. We produce the graphs from the four R packages by saving the output as a PNG file, while the Ti*k*Z graphs are produced in LaTeX by compiling them to a PDF. We note, however, that all five methods are compatible with R Markdown and Quarto. In Figures 1 and 2, we attempt to maintain uniform styling across methods, with circles around

TikZ



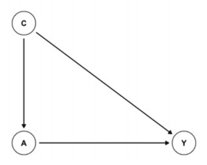
```
\usepackage{tikz}
\tikzset{> = stealth}

\begin{document}
\begin{tikzpicture}
% nodes
\node[circle,draw] (C) at (0,2) {C};
\node[circle,draw] (A) at (0,0) {A};
\node[circle,draw] (Y) at (3,0) {Y};
% edges
\draw[->, thick]
(C) edge (A)
(C) edge (Y)
(A) edge (Y);
\end{tikzpicture}
\end{document}
```

DAGitty



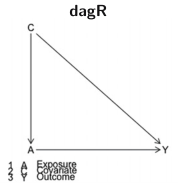
```
library(dagitty)
dag1 <- dagitty(
  "dag {
   A [exposure]
   Y [outcome]
   A -> Y
   C -> A
   C -> Y}" )
coordinates(dag1) <-  list(
  x = c(C = 0, A = 0, Y = 1),
  y = c(C = 0, A = 1, Y = 1)
)
plot(dag1, lwd=2)
```

ggdag



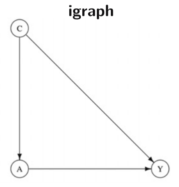
```
library(ggdag)
dag1 <- dagify(
    A ~ C,
    Y ~ A + C,
    exposure = "A",
    outcome = "Y",
    # location for each node
    coords = list(
      x = c(Y = 4, A = 2, C = 2),
      y = c(Y = 2, A = 2, C = 3)
    )
)
dag1 %>%
  ggplot(aes(x = x, y = y,
    xend = xend, yend = yend)) +
  geom_dag_text(color = "black") +
  geom_dag_edges() +
  geom_dag_point(shape = 1) +
  theme_dag()
```

dagR



```
library(dagR)
dag1 <- dag.init(
  covs = c(1), #1 per cov
  arcs = c(1,0, # C to A
           1,-1, # C to Y
           0,-1),# A to Y
  assocs = c(0),#directed arcs
  xgap = 0.04,
  ygap = 0.05,
  len = 0.1,
  x.name = "Exposure",
  cov.names = c("Covariate"),
  y.name = "Outcome",
  symbols = c("A", "C", "Y"),
  noxy = T
  )

#x, y-axis positions
dag1$x <- c(0, 0, 1)
dag1$y <- c(0, 1, 0)
plot1 <- dag.draw(
  dag1, noxy = T, legend = T)
```

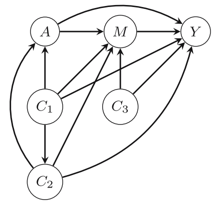
igraph



```
library(igraph)
dag1 <- make_graph(
  edges = c("C","A", #C to A
            "C","Y", #C to Y
            "A","Y"),#A to Y
  directed = TRUE)
plot(dag1,
  layout =
    matrix(c(0, 0, 1,
             1, 0, 0),
      nrow = 3, ncol = 2),
  vertex.size = 25,
  vertex.color = "white",
  edge.color = "black",
  vertex.label.color = "black",
  edge.arrow.size = 0.6)
```

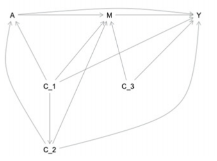
Figure 1: Simple confounding DAG example for each software. "A" denotes the exposure, "Y" the outcome, and "C" the confounder. The code used to generate each DAG is included below the relative plot.

TikZ



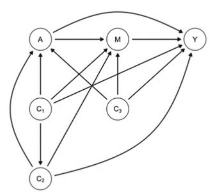
```
\usepackage{tikz}
\tikzset{> = stealth}

\begin{document}
\begin{tikzpicture}
% nodes
\node[circle,draw](C1) at (0,2){$C_1$};
\node[circle,draw](C2) at (0,0){$C_2$};
\node[circle,draw](C3) at (2,2){$C_3$};
\node[circle,draw](A) at (0,4){$A$};
\node[circle,draw](Y) at (4,4){$Y$};
\node[circle,draw](M) at (2,4){$M$};
% edges
\draw[->, thick]
(C1) edge (A)
(C1) edge (M)
(C1) edge (Y)
(C1) edge (C2)
(C2) edge [bend left=40] (A)
(C2) edge (M)
(C2) edge [bend right=30] (Y)
(C3) edge (M)
(C3) edge (Y)
(A) edge [bend left=30] (Y)
(A) edge (M)
(M) edge (Y);
\end{tikzpicture}
\end{document}
```

DAGitty



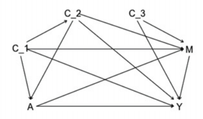
```
library(dagitty)
dag2 <- dagitty('dag {
  A [exposure,pos="0.250,0.150"]
  C_3 [pos="1.025,0.600"]
  C_2 [pos="0.500,1.000"]
  C_1 [pos="0.500,0.600"]
  M [pos="0.900,0.150"]
  Y [outcome,pos="1.500,0.150"]
  A -> M
  A -> Y [pos="1.000,0.075"]
  C_1 -> C_2
  C_3 -> M
  C_3 -> Y
  C_2 -> A [pos="0.150,0.500"]
  C_2 -> M
  C_2 -> Y [pos="1.500,0.700"]
  C_1 -> A
  C_1 -> M
  C_1 -> Y
  M -> Y
}')

plot(dag2)
```

ggdag



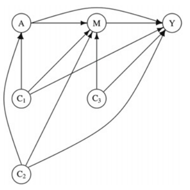
```
library(ggdag)
dag2 <- dagify(
   # relationship for each node
    A ~ C1 + C2 +C3 ,
    C2 ~ C1,
    M ~ A + C1 + C2 + C3,
    Y ~ M + A + C1 + C2 + C3,
    # specify exposure and outcome
    exposure = "A", outcome = "Y",
    # Location of each node
    coords = list(
      x = c(Y = 3, M = 2, A = 1,
            C1 = 1, C2 = 1, C3 = 2),
      y = c(Y = 2, M = 2, A = 2,
            C1 = 1, C2 = 0, C3 = 1))
  )
dag2 %>%
  ggplot(aes(x = x, y = y,
    xend = xend, yend = yend)) +
  geom_dag_point(shape=1) +
  #geom_dag_edges() +
  geom_dag_edges_arc(
    curvature = c(0, 0.35, 0, 0, 0,
        0, 0.35, 0, -0.35, 0, 0, 0)) +
  theme_void() +
  geom_dag_text(# subscripts
    color = "black",
    parse = TRUE,
    label = c("A","M","Y",
        expression(C[1]),
        expression(C[2]),
        expression(C[3])) )
```

dagR



```
library(dagR)
dag2 <- dag.init(
  outcome = NULL,
  exposure = NULL,
  covs = c(1, 1, 1, 1),
  arcs = # write in pairs
   c(1,0,  # C1 to A
     1,4,  # C1 to M
     1,-1, # C1 to Y
     1,2,  # C1 to C2
     2,0,  # C2 to A
     2,4,  # C2 to M
     2,-1, # C2 to Y
     3,4,  # C3 to M
     3,-1, # C3 to Y
     4,-1, # M to Y
     0, 4, # A to M
     0,-1  # A to Y
  ),
  # directed arcs, 0 per cov
  assocs = c(0, 0, 0, 0),
  xgap = 0.04,
  ygap = 0.05,
  len = 0.05,
  symbols = c("A", "C_1", "C_2",
             "C_3","M", "Y"),
  noxy = T
)
plot_dag2 <- dag.draw(dag2,
    noxy = T, legend = F)
```

igraph



```
library(igraph)
dag2 <- make_graph(
  edges = c("A","M",#A to M
            "A","Y", #A to Y
            "M","Y", #M to Y
            "C1","A", #C1 to A
            "C1","M", #C1 to M
            "C1","Y", #C1 to Y
            "C2","A", #C2 to A
            "C2","M", #C2 to M
            "C2","Y", #C2 to Y
            "C3","M", #C3 to M
            "C3","Y"),#C3 to Y
  directed = TRUE)
plot(dag2,
     layout = matrix(
         c(1, 2, 3, 1, 1, 2,
           2, 2, 2, 1, 0, 1),
         nrow = 6, ncol = 2
     ), #matrix of location
     edge.curved = c(
         0, 0.35, 0, 0, 0, 0,
         0.4, 0, -0.4, 0, 0, 0),
     vertex.size = 25,
     vertex.label= c(
         "A", "M", "Y",
         expression(C[1]),
         expression(C[2]),
         expression(C[3])),
     vertex.color = "white",
     edge.color = "black",
     vertex.label.color = "black",
     edge.arrow.size = 0.6 )
```

Figure 2: Complex mediation DAG example for each software. A denotes the exposure, Y the outcome, and M the mediator, and "C_1"/C_1, "C_2"/C_2, and "C_3"/C_3 denote the three covariates. The code used to generate each DAG is included below the relative plot.

nodes, black text, and arrows and nodes arranged chronologically from left to right. In Table 1, we highlight the design customization settings available for each software to further adapt the DAG to the user's preferences.

2.1 Ti*k*Z

Ti*k*Z is a LaTeX library for creating graphical figures, with extensive customization settings to generate a vast array of different images. It relies on portable graphics format (PGF), another LaTeX language as its base layer [30]. Tantau developed and maintains the two libraries. Tantau did not design the library specifically for DAGs,

Table 1: Summary of the characteristics and capabilities of each reviewed software. Where possible, the relative code/functions are mentioned. Note that "–" is used to denote when a feature is not available

	Ti*k*Z	DAGitty	ggdag	dagR	igraph
Visual design					
Curved arrows	`[bend`*direction*`=`*angle*`]`	`[pos="#,#"]` after edge specification	`+ geom_dag_edge_arc()`	—	`edge.curved=c()` in `plot()`
Subscripts	Underscore in inline math mode for node label ($_$)	—	`expression()` in + `geom_dag_text()`	—	`vertex.label=c()` in `plot()`
Math notation	Inline math mode for node label ($$)	—	`+ geom_dag_text()` using Unicode	—	`vertex.label=c()` in `plot()`
Auto-generated placements	—	This is default option in the plotting function	This is default option in the plotting function	This is default option in the plotting function	This is default option in the plotting function
Manual placements	`at` (x,y) after node command OR placement relative to other nodes [*direction*=`of`*other node*]	`[pos="x,y"]` after node specification OR give a list of x and y pairs in the `coordinates()`	`coords = list(x=c(), y=c())` in the `dagify()` function	`dag$x=c()` for x coordinates and `dag$y=c()` for y coordinates	`layout = matrix()` in `plot()` where each row is the corresponding (x,y) coordinate
Default visual settings	Black text and arrows, no circles	In R: black text and arrows, no circles; on website: gray circles and black text and gray arrows	Black shaded circles with white text, black arrows	Black text and arrows, no circles	Navy text, orange circles, gray arrows
Design customizations	Easy to add circles/shapes and change color, size, and weight of nodes and edges	In R: cannot add shapes around nodes, cannot customize color; on website: custom color, can remove circles	Easy to customize shape, color, size, and location of nodes and edges	No circles available, cannot customize color	Easy to customize/remove shapes, can change color, size, and location of nodes and edges
Analysis Capability					
Exposure, outcome framework	—	Specify the `[exposure]` or `[outcome]` after each node	Specify the `exposure` and `outcome` in the `dagify()` function	Outcome and exposure automatically included as node `#-1,0` respectively	—
Ancestor / descendant identification	—	`ancestors()`, `descendants()`		`dag.ancestors()`	`subcomponent()`; with `mode="in"` or `mode="out"`
Conditional independencies identification	—	`impliedConditionalIndependencies()`		—	—
Adjustment sets calculation	—	`adjustmentSets()`	`ggdag_adjustment_set()`	`dag.search()`	—
Data Simulation	—	`simulateSEM()`	`simulate_data()`	`data.sim()`	—
Utility					
Resources/online help	[30,56]	[20,37,38,40,41]	[42–45]	[23,39]	[22,47,48]
Learning curve	Straightforward code, relatively easy to customize	Straightforward to use website, easy to copy and paste code	Straightforward to use defaults, longer learning curve to customize	Longer learning curve required to specify nodes and edges, less intuitive	Longer learning curve required to specify edges, straightforward to plot
Base software	LaTeX	website / R	R	R	C/C++, R, Python

and thus, there is no built-in exposure, outcome, and covariates framework, analysis features, or automatic placement of nodes. Despite these limitations, Ti*k*Z serves as one of the leading software to create DAGs, thanks to its flexibility, which allows the user to easily specify the color, size, shape, and arrangement of nodes and edges. In addition, as Ti*k*Z is built into the LaTeX environment, one can use mathematical notation to label variables with inline math mode (denoted by $'s), including subscripts, Greek letters, and any other mathematical symbols. There are many tutorials and user-posted question and answer boards online that explain the different possible customization settings. One can also refer to the Ti*k*Z and PGF manual for full documentation of all of the package's capabilities [30]. For the purpose of creating causal diagrams, we recommend beginning with DAG-specific resources as Ti*k*Z's extensive language can be overwhelming, especially for a beginner in LaTeX. To code a DAG using Ti*k*Z, we first create the nodes and specify each node's location, shape, color, and label as desired. We then list the edges and can again customize any stylistic preferences. The code is both intuitive and readable.

As shown in Figure 1, we write the labels using plain text, add circles around the nodes, and use the stealth arrow type. We display the variables chronologically and arrange the exposure, outcome, and confounder such that all edges can be shown using straight lines. As shown in Figure 2, we use a similar design as in Figure 1, but here we write variable labels using inline math mode. This allows us to add subscripts to the confounder nodes, and label them as C_1, C_2, and C_3. Here, we incorporate curved arrows by specifying the angle of the desired curve. For those with less familiarity with LaTeX, there may be a steeper learning curve to create figures with Ti*k*Z. However, we believe that frequent users of LaTeX will find it easy to incorporate Ti*k*Z into their documents to produce DAGs. As shown Figures 1 and 2, it is obvious why Ti*k*Z is a popular resource to draw DAGs. We can easily create a clear and visually appealing DAG, with straightforward changes available to adapt the style to a user's preferences. In Table 1, we further see that Ti*k*Z's strength comes from its customizability and visual appeal, while its main limitation is that it is not designed for a DAG-specific framework and thus does not have analysis capabilities.

2.2 DAGitty

DAGitty is a browser-based interface, downloadable tool, and R library for creating, editing, and analyzing DAGs. The website interface and downloadable tool are accessible via http://www.dagitty.net/ and the R package is available on CRAN [20]. DAGitty's browser version provides a graphical user interface that allows users to draw and analyze causal diagrams. Its drag-and-click features make the tool very user-friendly and easy to learn. The website allows the user to select and label nodes, connect nodes via directional edges, and identify the exposure, outcome, and covariates. After one creates a causal diagram, one can explore the conditional independence, ancestor/descendant identification, and minimally sufficient adjustment sets [37,38]. The user can also copy and paste the model code into R after installing and loading the DAGitty library. Similarly, the R library has the functionality to obtain the conditional independencies, ancestor/descendant identification, and adjustment set lists directly in the statistical program. The DAGitty package in R additionally offers functions that can simulate data based on the specified DAG structure [20]. However, the simulation functionality is limited; the creators suggest employing it only for validation purposes and that one use other techniques or software for more complicated simulation studies [20,39]. DAGitty 0.9a, the oldest version of the software available, was released in 2010 with its first announcement via a letter in *Epidemiology* [40]. The most current version of the R package available is 3.1, which was updated in 2023 (as of August 2023). DAGitty developers also maintain the browser-based website regularly [41].

In Figure 1, we see the simple confounding DAG with DAGitty's output from R. Notably, there are no circles around the nodes, the lines are very light and thin, and the arrows run very close to the letters. Figure 2 shows the more complicated mediation DAG, with DAGitty's graph shown second to the left. We can see that DAGitty is not able to incorporate the subscripts on the nodes. It can plot the curved edges, but the placement and execution of the curves do not look as polished as in Ti*k*Z. The top curved arrow from A to Y appears condensed due to space limitations imposed by the RStudio plot output box and the R Markdown display region. If the curve had a larger radius from A to Y, then the display region in R would cut off the top part of the curve. This region limitation is not a problem on the DAGitty website.

To create both DAGs, we employ the website to set up the initial placement and then copy and paste code from the website into RStudio, where we use the DAGitty R library. The DAGitty website is user-friendly and intuitive. In summary, DAGitty has extensive analysis capabilities, but is less flexible in visual design. Overall, this is a suitable package for users who want to produce a quick DAG and can accept compromises on visual appearance.

2.3 ggdag

The R package ggdag allows users to plot and analyze causal graphs [21]. It is built on top of DAGitty to utilize DAGitty's powerful algorithms to analyze DAGs, while allowing users to employ ggplot and tidyverse to create professional, reproducible, and visually appealing DAGs [42]. It also enables the use of DAGitty objects in the context of tidyverse [21,43]. The R functions are flexible in the sense that they allow users to code their DAG structure using DAGitty syntax or ggdag-specific syntax. This feature allows ggdag to have the same analytical capability that DAGitty has, including identification of conditional independencies, ancestor/descendant relationships, and minimally sufficient adjustment sets lists [21]. ggdag additionally offers a visual display of adjustment sets via a colored graph [43,44]. Finally, this package has a wrapper function that allows one to apply DAGitty's simulating data algorithm to the structural equation model [43]. Thus, ggdag and DAGitty are both able to simulate data but under the same limitations. The initial release was in March 2018, and the current version 0.2.10 was updated in 2023 (as of August 2023) [43].

Figure 1 displays the simple confounding DAG with ggdag's output shown in the third panel. In ggdag, edges are specified with structural equation model notation in the `dagify()` function. The `ggplot()` plotting function then renders the defined dagify object. Figure 2 illustrates the more complicated mediation DAG, with ggdag's version displayed in the center column. We see that ggdag is able to incorporate curved edges and subscripts on the node labels. The curved edges have a nice bold arc, for which it was easy to control the radius and directionality using the `geom_dag_edge_arc()` add-on. The incorporation of both the subscript and the professional-looking curved edges makes this graph visually appealing. The creation of this graph took twice the amount of time as DAGitty, suggesting a longer learning curve; however, this might be eased with more frequent use. Incorporating the placement of each node, the location of each curved edge, and the subscripts each took the authors multiple Google searches for examples, vignettes, or user-posted question and answer boards [42–45]. The end product is very appealing, but it did require patience and time. For quicker use of ggdag, one might utilize DAGitty's web application to specify node and edge locations and then use the corresponding code and DAGitty object in ggdag's plotting function.

Table 1 highlights ggdag's performance in terms of analysis capability, and visual appeal. Notably, ggdag can incorporate subscripts and curved arrows, customize node placement, and write Greek symbols to label nodes through unique Unicode values in the ggdag label function [46]. In the utility category, ggdag has many vignettes, is well documented, and has many resources available online [42–45]; however, ggdag falls short compared to other software in ease-of-use due to the relative time needed by the authors to create Figure 2. Overall, ggdag is a valuable tool for users who want to create professional-looking DAGs in R.

2.4 dagR

dagR is an R package developed by Lutz P. Breitling, which was originally released in 2010 and most recently updated in 2022 (as of August 2023) [23,39]. The package allows users to plot DAGs, identify minimal sufficient adjustment sets, list ancestors of a given node, and simulate data from the specified causal diagram. dagR notably provides additional simulation capabilities compared to DAGitty (and ggdag) by allowing for a combination of binary and continuous variables within the same DAG [39]. Unlike DAGitty and ggdag, dagR does not provide functions to identify conditional independencies, and cannot directly identify descendants [23].

While dagR excels at simulating data and provides some analysis functions, it has limitations compared to other available software in terms of visualizations and user-friendliness. The package is lacking in customization settings for DAGs, as it does not allow for curved edges, circles around nodes, subscripts, or math notation. In Figure 1, we see that the visual design is fairly similar to DAGitty, with smaller node labels and thin edges. The dagR default settings print a legend below the DAG, allowing the user to provide longer labels or descriptions for the nodes. However, we found that the legend lines tend to overlay, reducing readability, as seen in Figure 1. In Figure 2, we employ the automatic placement of nodes feature, as without curved arrows, this achieves the clearest arrangement of all edges in the graph. Unfortunately, the DAG still has visual limitations, such as the overlaid arrow heads leading into variables M and Y. With more complex DAGs, it would be challenging to generate a readable graph without curved lines.

In addition, we note that dagR, despite the simple visual results, is one of the hardest to utilize. Compared to DAGitty and ggdag, there are few online resources available with example code and use descriptions. We also find the code itself to be the least intuitive. For example, all edges are specified together in a single vector using pairs of numbers, where each number refers to a different node. With a larger number of variables, it becomes difficult to track which number corresponds to each node, making the system cumbersome. In Table 1, we highlight dagR's strength in analysis, notably data simulation, as well as its limitations in visual design and utility. The package lags in terms of the readability of its graphs and code. Finally, we note that dagR developers provide a function to translate dagR objects to DAGitty [39]. Thus, if a user desires a more visually appealing graph, but has already written their causal structure using dagR (perhaps to simulate data), then they can convert the object to DAGitty, and use DAGitty or ggdag to plot.

2.5 igraph

Finally, we include igraph, an open-source network analysis tool that emphasizes efficiency and portability [22]. Csárdi and Nepusz began the development of igraph in 2006 [22], but many collaborators have since contributed to its growth. The most recent R update of igraph as of August 2023 is version 1.5.0.1, released in July 2023 [47]. The tool can be utilized in R, Python, Mathematica, and C/C++, but its core is written in C [22]. Here, we focus on igraph's implementation in R. This package excels at auto-placement of nodes and edges, utilizing its vast array of graph layout algorithms, making it a valuable tool for visualizing complex and large DAGs [22,48]. While igraph provides many functions to calculate various structural properties of graphs and conduct network analysis, it has limited causal analysis capabilities. It can identify ancestors and descendants (i.e., `subcomponent()`), but does not offer functions to directly compute conditional independencies and adjustment sets [48].

The igraph notation to specify arrows is similar to dagR, where edges are listed in a vector with every pair denoting the origin and destination nodes of the edge [22,48]. However, unlike in dagR, one can name nodes with characters, making the code more readable. While automatic arrangement of the nodes is the default, the user can specify the x and y coordinates for each node via a layout matrix. In Figure 1, we see that the igraph DAG allows for black text surrounded by a circle for each node with black lines connecting each node. Since the default visualization uses navy text, orange circles, and gray arrows, we use several add-on features to achieve the desired styling (i.e., `edge.color`, `vertex.size`, and `vertex.color` are used in the plotting function). In Figure 2, we again specify the color, size, label, and location of all nodes and edges. To draw curved lines, we use the `edge.curved` specification. To handle subscript notation, we use `vertex.label` and the `expression()` feature; one could easily specify Greek letters here using Unicode. A full list of plotting controls is conveniently located in the R Vignette [48]. The DAGs produced are clean, professional, and highly customizable, although fine-tuning all the plotting features can be time-consuming. In summary, igraph excels in its flexibility for the visual design of a DAG; however, it has limited analysis functions to answer causal inference questions and less intuitive code compared to Ti*k*Z, DAGitty, and ggdag.

3 Discussion

There are several ways to create DAGs using open-source software, each with different strengths and weaknesses. By focusing on two DAG structures (seen in Figures 1 and 2), we are able to compare the software and highlight key features. Our findings are summarized in Table 1 by displaying the software's visual design, analysis capability, and utility. In this review, we choose to focus on only five software, namely Ti*k*Z, DAGitty, ggdag, dagR, and igraph, and primarily restrict our scope to DAGs.

The graphs we create in Figures 1 and 2 attempt to employ circular nodes, all-black coloring, and chronological arrangement, but we note that there is some debate on best stylistic practices for DAGs. While some recommend that variables be arranged such that arrows flow in a single direction (e.g., left to right or top to bottom) [1,35], others arrange nodes by causal proximity [49]. Furthermore, some choose to limit circular nodes to latent or unobserved variables and use square nodes or no shape for observed variables [35,50]. In Table 1, we highlight which software would be adept at these design changes. To create an acyclic directed mixed graph [51], all the software highlighted except for dagR can display bidirected edges as needed. Meanwhile, Ti*k*Z-SWIG, a library using Ti*k*Z in LaTeX, is the leading software to draw Single World Intervention Graphs (SWIGs [52]) [53]. More work is needed to provide a synopsis of software available to generate the other types of causal graphs and evaluate their performance.

We recommend that one choose a package to draw DAGs appropriate to one's needs. Should the reader want to quickly produce an informal graph, we suggest using DAGitty, as the online platform allows one to generate a figure without writing any code. For DAG interpretations such as identifying minimally sufficient adjustment sets, ancestor/descendant relationships, and conditional independencies, DAGitty offers the widest range of functionality. When simulating data from a DAG, dagR provides the most flexibility. For visualizing large and complex DAGs, we recommend using igraph. Finally, for formal publication quality graphs, we recommend Ti*k*Z when the manuscript is being written in LaTeX and the rendered output is a PDF and recommend ggdag when the final result will be coded in R utilizing an R Markdown or Quarto compiler.

While the discussed software offer a wide range of capabilities, we are optimistic that drawing DAGs will become even easier as new tools arise and existing software improve. We write this article using the listed versions of each of the software; the discussed packages may have future updates that address some of the shortcomings we identify. We encourage developers and users to continue contributing to the open-source DAG software community and look forward to future developments. We anticipate that soon the option will exist to take a photo of a hand-drawn graph and convert it to code to render the DAG in digital form. Such software already exists for mathematical formulas, matrices, and chemical diagrams [54,55], and it is the logical next step for DAG drawing tools.

Acknowledgements: The authors would like to thank Caleb Miles, Linda Valeri, and Daniel Malinsky for their invaluable and constructive feedback. We would also like to extend our appreciation to the Columbia University Causal Inference Learning Group for creating a supportive environment and facilitating the informative discussions. We express our gratitude to the developers, blog and tutorial authors, and thread posters and commenters for providing such a rich basis of tools and knowledge for using software to work with DAGs. We would also like to thank our reviewers for their instructive and insightful commentary and feedback.

Funding information: This work was supported by grant T32DA031099 (Amy J. Pitts).

Author contributions: Both authors contributed equally to the manuscript.

Conflict of interest: The authors state that there is no conflict of interest.

References

[1] Greenland S, Pearl J, Robins JM. Causal diagrams for epidemiologic research. Epidemiology. 1999:37–48.
[2] Hernán MA, Robins JM. Causal Inference: What If. Boca Raton FL: Chapman & Hall/CRC; 2020.
[3] Pearl J. Causality. Cambridge, UK: Cambridge University Press; 2009.
[4] Imbens GW, Rubin DB. Causal inference in statistics, social, and biomedical sciences. Cambridge, UK: Cambridge University Press; 2015.
[5] Morgan SL, Winship C. Counterfactuals and causal inference. Cambridge, UK: Cambridge University Press; 2015.
[6] Spirtes P, Glymour CN, Scheines R. Causation, prediction, and search. Cambridge, MA: MIT Press; 2000.
[7] Suttorp MM, Siegerink B, Jager KJ, Zoccali C, Dekker FW. Graphical presentation of confounding in directed acyclic graphs. Nephrology Dialysis Transplant. 2015;30(9):1418–23.
[8] Shrier I, Platt RW. Reducing bias through directed acyclic graphs. BMC Med Res Methodol. 2008;8:1–15.
[9] VanderWeele TJ, Robins JM. Four types of effect modification: a classification based on directed acyclic graphs. Epidemiology. 2007;18(5):561–8.
[10] Knight CR, Winship C. The causal implications of mechanistic thinking: Identification using directed acyclic graphs (DAGs). Handbook of causal analysis for social research. Dordrecht: Springer Netherlands; 2013. p. 275–99.
[11] Holland PW. Causal inference, path analysis and recursive structural equations models. ETS Research Report Series. 1988;1988(1):i-50.
[12] Kohler U, Class F, Sawert T. Control variable selection in applied quantitative sociology: a critical review. Eur Sociol Rev. 2023;1–14:jcac078.
[13] Freedman DA. As others see us: A case study in path analysis. J Educ Stat. 1987;12(2):101–28.
[14] Costa JdJ, Bernardini F, Artigas D, Viterbo J. Mining direct acyclic graphs to find frequent substructures an experimental analysis on educational data. Inform Sci. 2019;482:266–78.
[15] Huynh QL, Thi TLL. Applying the directed acyclic graph to examine the factors related to the adoption of e-learning. J Knowledge Manag Econ Inform Technol. 2014;4(1):1–15.
[16] Imbens GW. Potential outcome and directed acyclic graph approaches to causality: Relevance for empirical practice in economics. J Econ Literature. 2020;58(4):1129–79.
[17] Yang Z, Zhao Y. Energy consumption, carbon emissions, and economic growth in India: Evidence from directed acyclic graphs. Econ Modell. 2014;38:533–40.
[18] Awokuse TO. Export-led growth and the Japanese economy: evidence from VAR and directed acyclic graphs. Appl Econ. 2006;38(5):593–602.
[19] Glymour M, Pearl J, Jewell NP. Causal inference in statistics: A primer. Chichester, UK: John Wiley & Sons; 2016.
[20] Textor J, van der Zander B, Gilthorpe MS, Liśkiewicz M, Ellison GT. Robust causal inference using directed acyclic graphs: the R package 'dagitty'. Int J Epidemiol. 2016;45(6):1887–94.
[21] Barrett M. ggdag: Analyze and Create Elegant Directed Acyclic Graphs; 2023. R package version 0.2.10. https://CRAN.R-project.org/package=ggdag.
[22] Csardi G, Nepusz T. The igraph software package for complex network research. Int J Complex Systems 2006;1695(5):1–9. https://igraph.org.
[23] Breitling LP. dagR: a suite of R functions for directed acyclic graphs. Epidemiology. 2010;21(4):586–7.
[24] Kalisch M, Mächler M, Colombo D, Maathuis MH, Bühlmann P. Causal inference using graphical models with the R Package pcalg. J Stat Softw. 2012;47(11):1–26.
[25] Scutari M. Bayesian Network Constraint-based structure learning algorithms: parallel and optimized implementations in the bnlearn R Package. J Stat Softw. 2017;77(2):1–20.
[26] Hauser A, Bühlmann P. Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs. J Machine Learn Res. 2012;13:2409–64. https://jmlr.org/papers/v13/hauser12a.html.
[27] Zhang K, Ramsey J, Gong M, Cai R, Shimizu S, Spirtes P, et al. Causal-learn's documentation; 2021. https://causal-learnreadthedocsio/en/latest/.
[28] Kalainathan D, Goudet O. Causal discovery toolbox: Uncover causal relationships in python; 2019. http://arXiv.org/abs/arXiv:190302278.
[29] Zhang K, Zhu S, Kalander M, Ng I, Ye J, Chen Z, et al. gCastle: A python toolbox for causal discovery; 2021. http://arXiv.org/abs/arXiv:211115155.
[30] Tantau T. The TikZ and PGF Packages-Manual for Version 3.1.10. Institut für Theoretische Informatik, Universität zu Lübeck: Lubeck, Germany, 2023.
[31] quiver. Quiver; Available from:https://q.uiver.app/.
[32] Elias Bareinboim CJ, Juan D. Correa. Causal Fusion; Available from: https://www.causalfusion.net/login.
[33] Ramsey JD, Zhang K, Glymour M, Romero RS, Huang B, Ebert-Uphoff I, et al. TETRAD-A toolbox for causal discovery. In:8th International Workshop on Climate Informatics; 2018. p. 1–4.
[34] Haughton D, Kamis A, Scholten P. A review of three directed acyclic graphs software packages: MIM, Tetrad, and WinMine. Amer Statist. 2006;60(3):272–86.

[35] Tennant PW, Murray EJ, Arnold KF, Berrie L, Fox MP, Gadd SC, et al. Use of directed acyclic graphs (DAGs) to identify confounders in applied health research: review and recommendations. Int J Epidemiol. 2021;50(2):620–32.

[36] R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria; 2022. Available from: https://www.R-project.org/.

[37] Van der Zander B, Liskiewicz M, Textor J. Constructing separators and adjustment sets in ancestral graphs. In Proceedings of the UAI 2014 Conference on Causal Inference: Learning and Prediction-Volume 1274, 2014. p. 11–24.

[38] Van der Zander B, Liśkiewicz M, Textor J, Yang Q, Wooldridge M, et al. Efficiently finding conditional instruments for causal inference. Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015:3243–9.

[39] Breitling LP, Duan C, Dragomir AD, Luta G. Using dagR to identify minimal sufficient adjustment sets and to simulate data based on directed acyclic graphs. Int J Epidemiol. 2022;50(6):1772–7.

[40] Textor J, Hardt J, Knüppel S. DAGitty: a graphical tool for analyzing causal diagrams. Epidemiology. 2011;22(5):745.

[41] Johannes Textor AA. Benito van der Zander. Draw and analyze causal diagrams. http://www.dagitty.net/.

[42] Barrett M. An introduction to directed acyclic graphs. URL: https://cranr-project.org/web/packages/ggdag/vignettes/intro-to-dags.html. 2020.

[43] Barrett M. ggdag: Analyze and Create Elegant Directed Acyclic Graphs. R package version 0.2.8; 2023. R package version 0.2.8. https://cran.r-project.org/web/packages/ggdag/ggdag.pdf.

[44] Barrett M. Common Structures of Bias. 2022. https://cranr-projectorg/web/packages/ggdag/vignettes/bias-structureshtml.

[45] Barrett M. An Introduction to ggdag. 2022. https://cranr-projectorg/web/packages/ggdag/vignettes/intro-to-ggdaghtml.

[46] Rahlf T. Data visualisation with R: 100 examples. Basel: Springer; 2017.

[47] Csárdi G, Nepusz T, Traag V, Horvát S, Zanini F, Noom D, et al. igraph: Network Analysis and Visualization in R; 2023. R package version 1.5.0.1. https://CRAN.R-project.org/package=igraph.

[48] Csárdi G, Nepusz T, Traag V, Horvát S, Zanini F, Noom D, et al. igraph (R interface). 2023. https://cranr-projectorg/web/packages/igraph/vignettes/igraphhtml.

[49] Miao W, Geng Z, Tchetgen Tchetgen EJ. Identifying causal effects with proxy variables of an unmeasured confounder. Biometrika. 2018;105(4):987–93.

[50] Silva R. Principled selection of impure measures for consistent learning of linear latent variable models. In: NIPS Workshop on Causality and Feature Selection; 2006.

[51] Richardson T. Markov properties for acyclic directed mixed graphs. Scandinav J Stat. 2003;30(1):145–57.

[52] Richardson TS, Robins JM. Single world intervention graphs (SWIGs): A unification of the counterfactual and graphical approaches to causality. Center for the Statistics and the Social Sciences, University of Washington Series Working Paper. 2013. Vol. 128. Issue 30. 2013.

[53] Richardson TS. TikZ/PGF shape library for constructing Single-World Intervention Graphs (SWIGs). University of Washington; 2021. SWIGs LaTeX. https://sites.stat.washington.edu/tsr/website/documents/2018/tikz-for-swigs/swig-examples.pdf.

[54] Costa DS, Mello CA, d'Amorim M. A comparative study on methods and tools for handwritten mathematical expression recognition. In: Proceedings of the 21st ACM Symposium on Document Engineering. Association for Computing Machinery; 2021. p. 1–4.

[55] Mathpix. Handwriting recognition; 2023. https://mathpix.com/handwriting-recognition.

[56] Grantham N. Draw DAGs with TikZ; 2022. https://www.nsgrantham.com/draw-dags-tikz.