Drawing and Analyzing Causal DAGs with DAGitty

Johannes Textor

January 11, 2020

DAGitty is a software for drawing and analyzing causal diagrams, also known as directed acyclic graphs (DAGs). Functions include identification of minimal sufficient adjustment sets for estimating causal effects, diagnosis of insufficient or invalid adjustment via the identification of biasing paths, identification of instrumental variables, and derivation of testable implications.

DAGitty is provided in the hope that it is useful for researchers and students in Epidemiology, Sociology, Psychology, and other empirical disciplines. The software should run in any modern web browser that supports JavaScript, HTML, and SVG.

This is the user manual for DAGitty version 3.0. The manual is updated with every release of a new stable version. DAGitty is available at dagitty.net. An R package 'dagitty' implementing the same functionality is also available on CRAN and at github.com/jtextor/dagitty.

Contents

1 Introduction

DAGitty is a web-based software for analyzing causal diagrams. It contains some of the fastest algorithms available for this purpose.

This manual describes how causal diagrams can be created (Section 3) and manipulated (Section 4) using DAGitty. In Section 5, DAGitty's capabilities to analyze

causal diagrams are described. A brief introduction to causal diagrams is given in Section 2. Advanced users might also be interested in the R package 'dagitty' [18], which implements all functionality of the web-based software and more.

1.1 Citing DAGitty

Developing and maintaining DAGitty takes time and effort. If you publish research results obtained with the help of DAGitty, please consider giving us credit by citing our work. The main reference for DAGitty is our paper describing the accompanying R package [18], which is based on the same software library, and therefore also serves as a reference for the web-based program. We have also published several research papers describing the specific algorithms used in DAGitty, such as for identification of biasing paths [17], adjustment sets [20], and instrumental variables [21].

1.2 Running DAGitty online

There are two ways to run DAGitty: either from the internet or from your own computer. To run DAGitty online, visit the URL `dagitty.net`. DAGitty should run in every modern browser. Specifically, I expect it to work well on recent versions of Firefox, Chrome, Opera, and Safari as well as on Internet Explorer (IE) version 9.0 or later, which all support scalable vector graphics (SVG). If you encounter any problems, please send me an e-mail so I can fix them (my contact information is at the end of this manual). Keep in mind that DAGitty is used by hundreds of people per day from all over the world – these people all benefit if the problem you found is fixed so please do consider investing the time to notify me if you encounter any bugs.

1.3 Installing DAGitty on your own computer

DAGitty can be "installed" on your computer for use without an internet connection. To do this, download the file `dagitty.net/dagitty.zip` which is a ZIP archive containing DAGitty's source. Unpack this ZIP file anywhere in your file system. To run DAGitty, just open the file `dags.html` in the unpacked folder.

Some features of DAGitty will not work in the offline version, because they are actually implemented on the web server. Currently, these features are:

- Exporting model drawings as PDF, JPEG or PNG files.
- Publishing models on-line.

1.4 Migrating from earlier versions of DAGitty

The following two issues are important for users of older DAGitty versions. New users can skip this section.

- The model code syntax has been completely changed in DAGitty version 3.0; the old syntax based on the DAG program by Sven Knüppel [5] was getting too limited to accommodate the new features that were being added. Therefore, I decided to switch to a very different, but much more extensible syntax closely based on the "dot" language used by graphviz. DAGitty will still be able to open model code from older versions (with one small caveat for very old code, see below) and will automatically convert this to the new syntax.
- Early versions of DAGitty supported only one exposure and one outcome variable. It has now been possible for quite some time to have more than one exposure and/or outcome variable. This means that the old model code convention where the variable in the first line is the exposure and the variable in the second line is the outcome no longer works. Hence, if you open a model created with an earlier version in DAGitty 2.0 or higher, exposure and outcome will appear like normal variables. To fix this, simply set exposure and outcome again and save the new model code.

2 A brief introduction to causal diagrams

In this section, we will briefly review what causal diagrams are and how they can be applied in empirical sciences. For a more detailed account, we recommend the book *Causal Inference in Statistics: A Primer* by Pearl, Glymour and Jewell [8], or the chapter *Causal Diagrams* in the Epidemiology textbook of Rothman, Greenland, and Lash [10]. Also take a look at the web page `dagitty.net/learn/`, where I am collecting several tutorials (some of them interactive) on specific DAG-related topics.

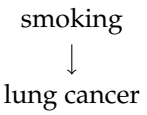
In Epidemiology, causal diagrams are also frequently called *DAGs*.[1] In a nutshell, a DAG is a graphic model that depicts a set of hypotheses about the causal process that generates a set of variables of interest. An arrow

[1] The term "DAG" is somewhat confusing to computer scientists and mathematicians, for whom a DAG is simply an abstract mathematical structure without specific semantics attached to it.

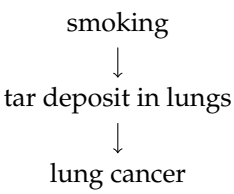
$X \to Y$ is drawn if there is a direct causal effect of X on Y. Intuitively, this means that the natural process determining Y is directly influenced by the status of X, and that altering X via external intervention would also alter Y. However, an arrow $X \to Y$ only represents that part of the causal effect which is *not* mediated by any of the other variables in the diagram. If one is certain that X does not have a direct causal influence on Y, then the arrow is omitted. This has two important implications: (1) arrows should follow time order, or else the diagram contradicts the basic principle that causes must precede their effects; (2) the omission of an arrow is a stronger claim than the inclusion of an arrow – the presence of an arrow depicts merely the "causal null hypothesis" that X *might* have an effect on Y.

Mathematically, the semantics of an arrow $X \to Y$ can be defined as follows. Given a DAG G and a variable Y in G, let $X_1, \ldots, X_n$ be all variables in G that have direct arrows $X_i \to Y$ (also called the *parents* of Y). Then G claims that the causal process determining the value of Y can be modelled as a mathematical function $Y := f(X_1, \ldots, X_n, \epsilon_Y)$, where ϵ_Y (the "causal residual") is a random variable that is jointly independent of all X_i.

For example, the sentence "smoking causes lung cancer" could be translated into the following simple causal diagram:

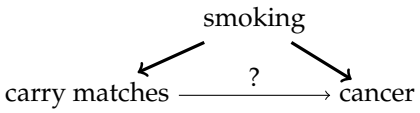


We would interpret this diagram as follows: (1) The variable "smoking" refers to a person's smoking habit prior to a later cancer disease status in that same person; (2) the natural process by which a person develops cancer might be influenced by the smoking habits of that person; (3) there exist no other variables that have a direct influence on both smoking habits and cancer. A slightly more complex version of this diagram might look as follows:



This diagram is about a person's smoking habits at a time t_1, the tar deposit in her lungs at a later time t_2, and finally the development of lung cancer at an even later time t_3. We claim that (1) the natural process which determines the amount of tar in the lungs is affected by smoking; (2) the natural process by which lung cancer develops is affected by the amount of tar in the lung; (3) the natural process by which lung cancer develops is not affected by the person's smoking other than indirectly via the tar deposit; and finally (4) no variables having relevant direct influence on more than one variable of the diagram were omitted.

In an epidemiological context, we are often interested in the putative effect of a set of variables, called *exposures*, on another set of variables called *outcomes*. A key question in Epidemiology (and many other empirical sciences) is: how can we infer the causal effect of an exposure on an outcome of interest from an observational study? Typically, a simple regression will not suffice due to the presence of *confounding factors*, which may lead to an over- or underestimation of the causal effect from the observed data. If the assumptions encoded in a given diagram hold, then it is sometimes possible to devise an *identification strategy* from that diagram, by which it would be possible to devise an unbiased estimate of a causal effect from observed data. One example identification strategy would be *covariate adjustment*. For example, consider the following diagram:



If we were to perform an association study on the relationship between carrying matches in one's pocket and developing lung cancer, we would probably find a correlation between these two variables. However, as the above diagram indicates, this correlation would not imply that carrying matches in your pocket causes lung cancer: Smokers are more likely to carry matches in their pockets, and also more likely to develop lung cancer. This is an example of a *confounded* association between two variables, which is mediated via the *biasing path* (bold). Now let us assume (unrealistically, and solely for didactic purposes) that the simplistic diagram above is an accurate representation of the process that generated our data. Under this assumption, would we adjust for smoking, e.g. by weighted averaging of separate effect estimates for smokers and non-smokers or by including smoking status as a covariate in a regression model, we would no longer find a correlation between carrying matches and lung cancer. In other words, adjustment for smoking would *close the biasing path*. In general, any set of covariates that closes all biasing paths (and does not open new ones or closes causal paths in the process) is called an *adjustment sets*. Adjustment sets will be explained in more detail in Section 5.5.1.

The purpose of DAGitty is to aid study design through devising identification strategies in (possibly complex) causal diagrams and, more generally, through the identification of causal and biasing paths as well as testable implications in a given diagram.

3 Loading, saving and sharing diagrams

This section covers the three basic steps of working with DAGitty: (1) loading a diagram; (2) manipulating the graphical layout of the diagram; and (3) saving the diagram. First of all, any causal diagram consists of vertices (variables) and arrows (direct causal effects). You can either create the diagram directly using DAGitty's graphical user interface (explained in the next section), or prepare a textual diagram description in a word processor and then import this description into DAGitty. In addition, DAGitty contains some pre-defined examples that you can use to become familiar with the program and with DAGs in general. To do so, just select one of the pre-defined examples from the *"Examples"* menu.

3.1 DAGitty's textual syntax for causal diagrams

The textual syntax in DAGitty is based on the 'dot' language by graphviz. In fact, many dot graphs should work directly in dagitty without modifications, although most of the style attributes of the dot language are not supported by dagitty. I believe it is best to introduce the syntax by a series of examples. Let us start by defining the example used in the introduction above.

```
dag{
  smoking
  "carry matches" [exposure]
  cancer [outcome]
  smoking -> "carry matches"
  smoking -> cancer
  "carry matches" -> cancer
}
```

This example shows the three basic components of the syntax in action:

- The enclosing statement `dag{ ... }`, which is always there. The DAG can also be given a name like so: `dag Smoking { ... }`
- The variable (vertex) statements. These consist of a variable name and a list of options enclosed in square brackets. For instance, the options "exposure" and "outcome" set a variable to be an exposure or outcome, respectively. Other relevant options are "latent" (for unobserved variables) and "adjusted" (for variables that have been adjusted for in a statistical analysis). It is necessary to double-quote the variable names if they contain spaces or other special characters, like for the variable "carry matches".
- The edge statements. These consist of a source variable, and edge type (which can be `->`, `<-`, or `<->`), and a target variable. Double-headed edges (`x<->y`) are simply an equivalent shortand for typing `x<-u->y; u[latent]`. Importantly, this means that double-headed edges are *not* meant to represent reciprocal causation (which is impossible to represent in a DAG). Instead, they are commonly used to depict unknown or unobserved confounders without specifying explicitly what those confounders are.

These three syntax components are in fact enough to define any DAG. We are now going to define the same DAG in various different ways to showcase various convenient features of the syntax that make DAG definitions more compact; it is not necessary to use any of these features, but they can save a lot of typing.

Variable statements can be omitted if the variable has no options, such as "smoking" in the above example. Every time a variable is used in an edge statement, that variable is automatically added as if there had been a corresponding node statement without an option.

```
dag{
  "carry matches" [exposure]
  cancer [outcome]
  smoking -> "carry matches"
  smoking -> cancer
  "carry matches" -> cancer
}
```

White-space is optional and several statements can be combined on one line. For clarity, it is recommended to insert semi-colons between different statements on the same line; however, this is not necessary. The following two versions are equivalent:

```
dag{
  "carry matches" [exposure]; cancer [outcome]
  smoking -> "carry matches"; smoking -> cancer;
  "carry matches" -> cancer
}
```

```
dag{
  "carry matches" [exposure] cancer [outcome]
  smoking -> "carry matches" smoking -> cancer
  "carry matches" -> cancer
}
```

Edge statements can be chained together such that entire paths can be defined at once:

```
dag{
  "carry matches" [exposure] ; cancer [outcome]
  smoking -> "carry matches" -> cancer
  smoking -> cancer
}
```

Arrows can also be written in reverse orientation, which is quite convenient when used together with edge chaining:

```
dag{
  "carry matches" [exposure] ; cancer [outcome]
  cancer <- smoking -> "carry matches" -> cancer
}
```

Another very useful feature for short DAG descriptions is variable grouping using curly braces. This allows you to define several arrows at once like so:

```
dag{
  "carry matches" [exposure] ; cancer [outcome]
  smoking -> {"carry matches" cancer}
   "carry matches" -> cancer
}
```

The curly braces open a new scope in which a "sub-graph" is defined. An arrow pointing to a sub-graph means that there will be arrows made to all variables in the sub-graph, and the sub-graph itself can also define its own internal arrows. This means that we can also write the above as:

```
dag{
  "carry matches" [exposure] ; cancer [outcome]
  smoking -> {"carry matches" -> cancer}
}
```

To save even more typing, several option names can be abbreviated using single letters like so:

```
dag{
  "carry matches" [e] ; cancer [o]
  smoking -> {"carry matches" -> cancer}
}
```

(a)

```
dag{
E -> D
A -> E
A -> Z
B -> Z
B -> D
Z -> E
Z -> D
}
```

(b)

```
dag{
A -> {Z E}
B -> {Z D}
Z -> {E D}
E -> D
}
```

(c)

```
dag{
{A B} ->
  {Z->{E->D}}
}
```

(d)

```
dag {
A [pos="0,-2"]
B [pos="2,-2"]
D [outcome,pos="2,0"]
E [exposure,pos="0,0"]
Z [pos="1,-1"]
A -> { E Z }
B -> { D Z }
E -> D
Z -> { D E }
}
```

(e)

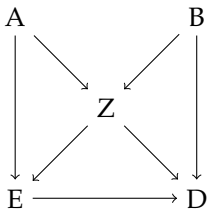


Figure 1: Example for a textual model definition with DAGitty**(a)** simple model code; **(b)** shorter model codel and **(c)** very short model code of the graph in (a) using clever nested grouping. **(d)** When the diagram is edited within DAGitty, vertex labels and adjustment status are augmented with additional layout coordinates for each variable, which are indicated as an option of the corresponding node. **(e)** Graphical layout corresponding to (d).

Like mentioned above, it is not necessary to use grouping or edge statement chaining; the only purpose of these tricks is to save some typing. In fact, once your textual syntax is entered in DAGitty, it will be converted back to a trivial form in which the variable and edge statements are all explicitly listed. (This is similar to what would happen in graphviz.)

The above examples covered only the structure of the DAG, but gave no layout information. A simple layout is automatically by DAGitty once you input a text description where DAGitty cannot detect any layout coordinates. The coordinates are also updated when you move nodes around or bend edges. See Figure 1 for how the layout information is added to the variable statements. You could of course enter your own layout information manually into the text syntax as well.

3.2 Loading a model text

To load a textually defined diagram into DAGitty, simply copy&paste the variable list, followed by a blank line, followed by the list of arrows into the *"Model code"* text box. Then click on *"Update DAG"*. DAGitty will

now generate a preliminary graphical layout for your diagram on the canvas, which may not yet look the way you intended, but can be freely modified.

3.3 Modifying the graphical layout of a diagram

To layout the vertices and arrows of your diagram more clearly than DAGitty did, simply drag the vertices with your mouse on the canvas. You may notice that DAGitty modifies the information in the *"Model code"* field on the fly, and augments it with additional position information for each vertex. In general, all changes you make to your diagram within DAGitty are immediately reflected in the model code.

3.4 Saving the diagram

To save your diagram locally, just copy&paste the contents of the *"Model code"* field to a text file, and save that file locally to your computer[2]. When you wish to continue working on the diagram, copy the model code back into DAGitty as explained above.

3.5 Exporting the diagram

DAGitty can export the diagram as a PDF or SVG vector graphic (publication quality) or a JPEG or PNG bitmap graphic (e.g. for inclusion in Powerpoint). Select the corresponding function from the *"Model"* menu. If you want to edit the graphical layout of the diagram or annotate it, it is recommended to export the diagram as an SVG file and open that in a vector graphics program such as Inkscape.

3.6 Publishing diagrams online

Part of the appeal of using DAGs is that the assumptions underlying one's research are made explicit, and the conclusions drawn from the data can be later re-checked if some of the assumptions are found to not hold. Of course, this requires to make the DAG available together with the data and interpretation. I have however seen many articles where people report having used DAGs but do not actually show them. If researchers, reviewers or editors deem it inappropriate to include the DAG (or its model code) in the manuscript itself, here's another option: Store the DAG on the DAGitty website and get a short URL under which this DAG will be accessible. Then include this URL in the manuscript, or its supporting information. For example, one of the DAGitty examples is stored at the URL `dagitty.net/mvcFQ`.

Here's how it works: Draw your DAG to full satisfaction, then choose *"Publish on dagitty.net"* from the *"Model"* menu. You have two options how to publish your DAG: anonymously, or linking it to an e-mail address. If you store the DAG anonymously, you will later on not be able to edit it or delete it from the server.

After choosing *"Publish on dagitty.net"* from the *"Model"* menu, a small form will appear where you can enter some metadata on the DAG, and provide your e-mail address if you so wish. Upon clicking *"Publish"*, the DAG will be sent to the dagitty.net server, and you will receive a URL under which the DAG is now available. If you provided your e-mail address, you will also receive a message requesting you to confirm your ownership of the DAG. This is simply done by clicking on a confirmation link. Only then will the DAG be linked to your e-mail address, and you will receive a password to use when deleting or modifying the published DAG.

If you did link your DAG to your e-mail address, you can delete it by choosing *"Delete on dagitty.net"* from the *"Model"* menu, which will prompt you to enter the DAG's URL and the password. If the URL and password match, the DAG will be deleted. Similarly, you can update a stored DAG using the *"Load from dagitty.net"* function from the *"Model"* menu, modifying it, and saving it again. You can view published DAGs (if you know their URL) by just putting the URL into your address bar of course, but you can also do so using the *"Load from dagitty.net"* function.

Please note that all DAGs stored on dagitty.net are meant to be public information. Do not store any data that you consider private or in any way secret. Once stored on dagitty.net, every person in the world who knows your DAG's URL can view it (but not your e-mail address if you provided one). Also note that there is no guarantee that dagitty.net will keep running forever. Storing your DAGs is done at your own risk. Still, you may find this feature useful, for instance to e-mail your DAGs to colleagues or to include links to DAGs in papers under review. For archival purposes, it may be more appropriate to include the DAG or the model code in the paper itself or its supporting information.

[2] This is most easily done by clicking in the text field, pressing *"CTRL + A"* to select the entire content of the text field, then pressing *"CTRL + C"* to copy the content. You can then paste the content in another program using *"CTRL + V"*.

4 Editing diagrams using the graphical user interface

You are free to make changes directly to the textual description of your diagram, which will be reflected on the canvas next time you click on *"Update DAG"*. However, you can also create, modify, and delete vertices and arrows graphically using the mouse.

4.1 Creating a new diagram

To create a new diagram, select *"New Model"* from the *"Model"* menu. You will be asked for the names of the exposure and the outcome variable, and an initial model containing just those variables and an arrow between them will be drawn. Then you can add variables and arrows to the model as explained below.

4.2 Adding new variables

To add a new variable to the model, double-click on a free space in the canvas (i.e., not on an existing variable) or press the *"n"* key. A dialog will pop up asking you for the name of the new variable. Enter the name into the dialog and press the enter key or click *"OK"*. If you click *"Cancel"*, no new variable will be created.

4.3 Renaming variables

To rename an existing variable, move the mouse pointer over that variable and hit the *"r"* key. A dialog will pop up allowing you to change the variable name.

4.4 Setting the status of a variable

Variables can have one of the following statuses:

- Exposure
- Outcome
- Unobserved (latent)
- Adjusted
- Other

To turn a variable into an exposure, move the mouse pointer over that variable and hit the *"e"* key; for an outcome, hit the *"o"* key instead. To toggle whether a variable is observed or unobserved, hit the *"u"* key; to toggle whether it is adjusted, hit the *"a"* key. Changing the status of variables may change the colors of the diagram vertices to reflect the new structure and information flow in the diagram (see below).

At present, these statuses are mutually exclusive – e.g., a variable cannot be both unobserved and adjusted or both exposure and unobserved. This could change in future versions of DAGitty.

4.5 Adding new arrows

To add a new arrow, double-click first on the source vertex (which will become highlighted) and then on the target vertex. The arrow will be inserted. If an arrow existed before in the opposite direction, that arrow will be deleted, because otherwise there would now be a cycle in the model.

Instead of double-clicking on a vertex, you can also move the mouse pointer over the vertex and press the key *"c"*. Arrows are by default drawn using a straight line, but you can change that moving the mouse pointer to the line, pressing and holding down the left mouse button, and "bending" the line by dragging as appropriate.

4.6 Deleting variables

To delete a variable, move the mouse pointer over that variable and hit the *"del"* key on your keyboard, or alternatively the *"d"* key (the latter comes in handy if you're on a Mac, which has no real delete key). All arrows to that variable will be deleted along with the variable. In contrast to DAGitty versions prior to 2.0, all variables can now be deleted including exposure and outcome.

4.7 Deleting arrows

An arrow is deleted just like it has been inserted, i.e., by double-clicking first on the start variable and then on the target variable. An arrow is also deleted automatically if a new one is inserted in the opposite direction (see above).

4.8 Choosing the style of display

At present, you can choose between two DAG diagram styles: "classic", where nodes and their labels are separate from each other, and SEM-like, where labels are inside nodes. Both have their advantages and disadvantages. By the way, "SEM" refers to structural equation modeling.

5 Analyzing diagrams

5.1 Paths

Causal diagrams contain two different kinds of paths between exposure and outcome variables.

- *Causal paths* start at the exposure, contain only arrows pointing away from the exposure, and end at the outcome. That is, they have the form $e \to x_1 \to \ldots \to x_k \to o$.
- *Biasing paths* are all other paths from exposure to outcome. For example, such paths can have the form $e \leftarrow x_1 \to \ldots \to x_k \to o$.

With respect to a set $\mathbf{Z}$ of conditioning variables (that can also be empty if we are not conditioning on anything), paths can be either *open* or *closed* (also called d-separated [7]). A path is *closed* by $\mathbf{Z}$ if one or both of the following holds:

- The path p contains a chain $x \to m \to y$ or a fork $x \leftarrow m \to y$ such that m is in $\mathbf{Z}$.
- The path p contains a collider $x \to c \leftarrow y$ such that c is not in $\mathbf{Z}$ and furthermore, $\mathbf{Z}$ does not contain any successor of c in the graph.

Otherwise, the path is *open*. The above criteria imply that paths consisting of only one arrow are always open, no matter the content of $\mathbf{Z}$. Also it is possible that a path is closed with respect to the empty set $\mathbf{Z} = \{\}$.

5.2 Coloring

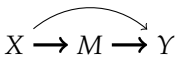
It is not easy to verify by hand which paths are open and which paths are closed, especially in larger diagrams. DAGitty highlights all arrows lying on open biasing paths in red and all arrows lying on open causal paths in green. This highlighting is optional and is controlled via the *"highlight causal paths"* and *"highlight biasing paths"* checkboxes.

5.3 Effect analysis

As mentioned above, arrows in DAGs represent *direct effects*. That is, in a DAG with three variables X, M, and Y, an arrow $X \to Y$ means that there is a causal effect of X on Y that is *not* mediated through the variable M. Often when building DAGs, people tend to forget this aspect and think only about whether any kind of causal effect exists, without paying attention to how it is mediated. This may result in DAGs with too many arrows.

To aid users with this, George Ellison (Leeds University) suggested to implement a function that identifies arrows for which also a corresponding indirect pathway exists. After drawing an initial DAG, one might reconsider these arrows and judge whether they are really necessary given the indirect pathways already present in the diagram.

For example, suppose after thinking about the pairwise causal relationships between our variables X, M, Y we came up with this DAG:



For the arrows drawn in bold, there is no corresponding indirect path – removing one of these arrows from the diagram means that there will no longer be any causal effect between the corresponding variables. These arrows are called *atomic direct effects* in DAGitty, and they can be highlighted – like in the above DAG – by ticking the checkbox with that name. On the other hand, for the thin arrow $X \to Y$, there is also the indirect pathway $X \to M \to Y$. One may therefore reconsider whether the arrow $X \to Y$ is truly necessary – perhaps the causal effect from X to Y is entirely mediated through M.

5.4 View mode

There are several ways to transform a given DAG such that it becomes better suited for a particular purpose. We call such a transformed DAG a *derived graph*. Currently DAGitty can display two kinds of derived graphs: correlation graphs, and moral graphs. These derived graphs can be shown by clicking on the respective radio button in the *"View mode"* field on the left-hand side of the screen.

5.4.1 The correlation graph

The correlation graph is not a DAG, but a simple graph with lines instead of arrows. It connects each pair of variables that, according to the diagram, could be statistically dependent. In other words, variables not connected by a line in the correlation graph must be statistically independent. These pairwise independencies are also listed in the *"Testable implications"* field on the right-hand side of the screen, and so the correlation graph could be seen as encoding a subset of those implications.

Although this is not implemented in DAGitty yet, it is also possible to take a given correlation graph (which can be obtained e.g. by thresholding a covariance matrix) and list all the DAGs that are "compatible" with it in

the sense that they entail exactly the given correlation graph [16].

5.4.2 The moral graph

To identify minimal sufficient adjustment sets, DAGitty uses the so-called "moral graph", which results from a transformation of the model to an undirected graph. This procedure is also highly recommended if you wish to verify the calculation by hand. See the nice explanation by Shrier and Platt [13] for details on this procedure.

In DAGitty, you can switch between display of the model and its moral graph choosing "moral graph" in the"view mode" section on the left-hand side of the page.

5.5 Causal effect identification

Some of the most important features of DAGitty are concerned with the question: how can causal effects be estimated from observational data? Currently, two types of causal effect identification are supported: adjustment sets, and instrumental variables.

5.5.1 Adjustment sets

Finding sufficient adjustment sets is one main purpose of DAGitty. In a nutshell, a sufficient adjustment set **Z** is a set of covariates such that adjustment, stratification, or selection (e.g. by restriction or matching) will minimize bias when estimating the causal effect of the exposure on the outcome (assuming that the causal assumptions encoded in the diagram hold). You can read more about controlling bias and confounding in Pearl's textbook, chapter 3.3 and epilogue [7]. Moreover, Shrier and Platt [13] give a nice step-by-step tutorial on how to test if a set of covariates is a sufficient adjustment set.

To identify adjustment sets, the diagram must contain at least one exposure and at least one outcome.

Total and direct effects. One can understand adjustment sets graphically by viewing an adjustment set as a set **Z** that closes all all biasing paths while keeping desired causal paths open (see previous section). DAGitty considers two kinds of adjustment sets:

- Adjustment sets for the *total effect* are sets that close all biasing paths and leave all causal paths open. In the literature, if the effect is not mentioned (e.g. [13, 5]), then usually this kind of adjustment set is meant.

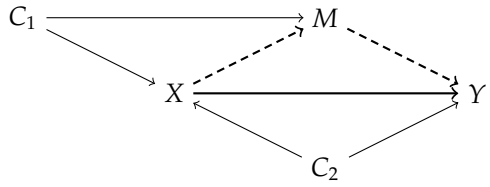


Figure 2: A causal diagram where the total and direct effects of exposure X on outcome Y are not equal. The total effect is the effect mediated only via the thick (both dashed and solid) arrows, while the direct effect is the effect mediated only via the thick arrow.

- Adjustment sets for the *direct effect* are sets that close all biasing paths and all causal paths, and leave only the direct arrow from exposure X to outcome Y (i.e., the path $X \to Y$, if it exists) open.

In a diagram where the only causal path between exposure and outcome is the path $X \to Y$, the total effect and the direct effect are equal. This is true e.g. for the diagram in Figure 1. An example diagram where the direct and total effects are not equal is shown in Figure 2.

As proved by Lauritzen et al. [6] (see also Tian et al. [19]), it suffices to restrict our attention to the part of the model that consists of exposure, outcome, and their ancestors for identifying sufficient adjustment sets. This is indicated by DAGitty by coloring irrelevant nodes in gray. The relevant variables are colored according to which node they are ancestors of (exposure, outcome, or both) – see the legend on the left-hand side of the screen. The highlighting may be turned on and off by toggling the *"highlight ancestors"* checkbox.

Minimal sufficient adjustment sets. A *minimal* sufficient adjustment set is a sufficient adjustment set of which no proper subset is itself sufficient. For example, consider again the causal diagram in Figure **??**. The following three sets are sufficient adjustment sets for the total and direct effects, which are equal in this case:

$$\{A, B, Z\}$$

$$\{A, Z\}$$

$$\{B, Z\}$$

Each of these sets is sufficient because it closes all biasing paths and leaves the causal path open. The sets $\{A, Z\}$ and $\{B, Z\}$ are minimal sufficient adjustment sets while the set $\{A, B, Z\}$ is sufficient, but not minimal. In contrast, the set $\{Z\}$ is *not* sufficient, since this would open

the path $E \leftarrow A \rightarrow Z \leftarrow B \leftarrow D$: Because both E and D depend on Z, adjusting for Z will induce additional correlation between E and D.

Finding minimal sufficient adjustment sets. To find minimal sufficient adjustment sets, select the option *"Adjustment (total effect)"* or *"Adjustment (direct effect)"* in the *"Causal effect identification"* field. DAGitty will then calculate all minimal sufficient adjustment sets and display them in that field. Any changes made to the diagram will be instantly reflected in the list of adjustment sets.

Forcing adjustment for specific covariates. You can also tell DAGitty that you wish a specific covariate to be included into every adjustment set. To do this, move the mouse over the vertex of that covariate and press the *a* key. DAGitty will then update the list of minimal sufficient adjustment sets accordingly – every set displayed is now minimal in the sense that removing any variable *except those you specified* will render that set insufficient. However, when you adjust for an intermediate or another descendant of the exposure, DAGitty will tell you that it is no longer possible to find a valid adjustment set.

Avoiding adjustment for unobserved covariates. You can tell DAGitty that a certain variable is unobserved (e.g. not measured at present, or not measurable because it is a latent variable) by moving the mouse over that covariate and pressing the *u* key. DAGitty will only calculate adjustment sets that do not contain unobserved variables. However, if too many or some important variables are unobserved, then it may be impossible to close all biasing paths.

5.5.2 Instrumental variables

Sometimes it is not possible to estimate a causal effect by simple covariate adjustment. For example, this is the case whenever there is an unobserved confounder that directly effects the exposure and outcome variables. However, this does not necessarily mean that it is impossible to estimate the causal effect at all. *Instrumental variable regression* is a technique that is often used in situations wit unobserved confounders. Note that this technique depends on linearity assumptions. For further information on instrumental variables, please refer to the literature [2, 4]. DAGitty can find instrumental variables in DAGs, as explained below.

The validity of an instrumental variable I depends on two causal conditions – exogeneity and exclusion restriction. These two conditions can be expressed in the language of DAGs and paths as follows: (1) there must be an open path between I and the exposure X; and (2) all paths between I and the outcome Y must be closed in a modified graph where all edges out of X are removed. A variable that fulfills these two conditions is called an *instrumental variable* or simply an *instrument*.

Instrumental variables can also be generalized such that the two conditions are required to hold *conditional on a set of covariates* $\mathbf{Z}$ [3]. The two conditions then read as follows: (1) there must be a path between I and X that is opened by $\mathbf{Z}$; and (2) all paths between I and Y must be closed by $\mathbf{Z}$ in a modified graph where all edges out of X are removed. A variable that fulfills these two conditions is called a *conditional instrument*.

DAGitty will find both "classic" and conditional instruments when the option *"Instrumental Variable"* is selected under the *"Causal effect identification"* field. Note that DAGitty will not always list *all* possible instruments; instead, it will restrict itself to a certain well-defined subset that we call "ancestral instruments". However, whenever any instrument or conditional instrument exists at all, then DAGitty is guaranteed to find one. Note also that if there are several instruments available, then it is best to choose the one that is most strongly correlated with X (conditional on $\mathbf{Z}$ in the case of a conditional instrument).

For details regarding ancestral instruments and how DAGitty computes them, please refer to the research paper where we describe these methods [21].

5.6 Testable implications

Any implications that are obtained from a causal diagram, such as possible adjustment sets or instrumental variables, are of course dependent on the assumptions encoded in the diagram. To some extent, these assumptions can be tested via the (conditional) independences implied by the diagram: If two variables X and Y are d-separated by a set $\mathbf{Z}$, then X and Y should be conditionally independent given $\mathbf{Z}$. The converse is not true: Two variables X and Y can be independent given a set $\mathbf{Z}$ even though they are not d-separated in the diagram. Furthermore, two variables can also be d-separated by the empty set $\mathbf{Z} = \emptyset$. In that case, the diagram implies that X and Y are *unconditionally* independent.

DAGitty displays all minimal testable implications in the *"Testable implications"* text field. Only such implications will be displayed that are in fact testable, i.e.,

that do not involve any unobserved variables. Note that the set of testable implications displayed by DAGitty does not constitute a "basis set" [7]. Future versions will allow choosing between different basis sets.

In general, the less arrows a diagram contains, the more testable predictions it implies. For this reason, "simpler" models with fewer arrows are in general easier to falsify (Occam's razor).

6 Acknowledgements

I would like to thank my collaborators Maciej Liśkiewicz and Benito van der Zander (both at the Institute for Theoretical Computer Science, University of Lübeck, Germany) for our collaborations on developing efficient algorithms to analyze causal diagrams.

I also thank Michael Elberfeld, Juliane Hardt, Sven Knüppel, Keith Marcus, Judea Pearl, Sabine Schipf, and Felix Thoemmes (in alphabetical order) for enlightening discussions (either in person, per e-mail, or on the SEMnet discussion list) about DAGs that made this program possible. Furthermore, I thank Robert Balshaw, George Ellison, Marlene Egger, Angelo Franchini, Ulrike Förster, Mark Gilthorpe, Dirk van Kampen, Jeff Martin, Jillian Martin, Karl Michaëlsson, David Tritchler, Eric Vittinghof, and other users for sending feedback and bug reports that greatly helped to improve DAGitty.

The development of DAGitty was sponsored by funding from the Institute of Genetics, Health and Therapeutics at Leeds University, UK. I thank George Ellison for arranging this generous support.

7 Legal notice

Use of DAGitty is (and will always be) freely permitted and free of charge. You may download a copy of DAGitty's source code from its website at `www.dagitty.net`. The source code is available under the GNU General Public License (GPL), either version 2.0, or any later version, at the licensee's choice; see the file `LICENSE.txt` in the download archive for details. In particular, the GPL permits you to modify and redistribute the source as you please as long as the result remains itself under the GPL.

8 Bundled libraries

DAGitty ships along with the JavaScript library *Prototype.js*, a framework that makes life with JavaScript much easier. Only some parts of Prototype (mainly those focusing on data structures) are included to keep the code small. Developed by the Prototype Core Team and licensed under the MIT license [15].

Furthermore, DAGitty contains some modified code from the *Dracula Graph Library* by Philipp Strathausen, which is also licensed under the MIT license [14].

I am grateful to the authors of these libraries for their valuable work.

9 Bundled examples

DAGitty contains some builtin examples for didactic and illustrative purposes. Some of these examples are taken from published papers or talks given at scientific meetings. These are, in inverse chronological order:

- van Kampen 2014 [22]
- Polzer et al., 2012 [9]
- Schipf et al., 2010 [11]
- Shrier & Pratt, 2008 [13]
- Sebastiani et al.[3], 2005 [12]
- Acid & de Campos, 1996 [1]

Another example was provided by Felix Thoemmes via personal communication (2013).

10 Author contact

I would be glad to receive feedback from those who use DAGitty for research or educational purposes. Also, you are welcome to send me your suggestions or requests for features that you miss in DAGitty.

Johannes Textor
Tumor Immunology Department
Radboud University Medical Center
Nijmegen, The Netherlands

`johannes.textor@gmx.de`
`johannes-textor.name`
Twitter: `@JohannesTextor`

[3] The example actually shows only a small part of their DAG.

References

[1] Silvia Acid and Luis M. De Campos. An algorithm for finding minimum d-separating sets in belief networks. In *Proceedings of the 12th Conference of Uncertainty in Artificial Intelligence*, pages 3–10, 1996.

[2] Joshua D. Angrist, Guido W. Imbens, and Donald B. Rubin. Identification of causal effects using instrumental variables. *Journal of the American Statistical Association*, 91(434):444–55, 1996.

[3] Carlos Brito and Judea Pearl. Generalized instrumental variables. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pages 85–93, 2002.

[4] Guido Imbens. Instrumental variables: An econometrician's perspective. *Statistical Science*, 29(3):323–58, 2014.

[5] Sven Knüppel and Andreas Stang. DAG program: identifying minimal sufficient adjustment sets. *Epidemiology*, 21(1):159, 2010.

[6] Steffen L. Laurizen, A. Philip Dawid, Birgitte N. Larsen, and Hanns-Georg Leimer. Independence properties of directed markov fields. *Networks*, 20(5):491–505, 1990.

[7] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[8] Judea Pearl, Madelyn Glymour, and Nicholas P Jewell. *Causality Inference in Statistics: A Primer*. Wiley, New York, NY, USA, 1st edition, 2016.

[9] Ines Polzer, Christian Schwahn, Henry Völzke, Torsten Mundt, and Reiner Biffar. The association of tooth loss with all-cause and circulatory mortality. Is there a benefit of replaced teeth? A systematic review and meta-analysis. *Clinical Oral Investigations*, 16(2):333–351, 2012.

[10] Kenneth J. Rothman, Sander Greenland, and Timothy L. Lash. *Modern Epidemiology*. Wolters Kluwer, 2008.

[11] Sabine Schipf, Robin Haring, Nele Friedrich, Matthias Nauck, Katharina Lau, Dietrich Alte, Andreas Stang, Henry Völzke, and Henri Wallaschofski. Low total testosterone is associated with increased risk of incident type 2 diabetes mellitus in men: Results from the study of health in pomerania (SHIP). *The Aging Male*, 14(3):168–75, 2011.

[12] Paola Sebastiani, Marco F. Ramoni, Vikki Nolan, Clinton T. Baldwin, and Martin H. Steinberg. Genetic dissection and prognostic modeling of overt stroke in sickle cell anemia. *Nature Genetics*, 37:435–40, 2005.

[13] Ian Shrier and Robert W. Platt. Reducing bias through directed acyclic graphs. *BMC Medical Research Methodology*, 8(70), 2008.

[14] Philipp Strathausen. Dracula graph layout and drawing framework. http://www.graphdracula.net, 2010.

[15] Prototype Core Team. Prototype–javascript library. http://www.prototypejs.org, 2010.

[16] Johannes Textor, Alexander Idelberger, and Maciej Liśkiewicz. Learning from pairwise marginal independencies. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, pages 882–91. AUAI Press, 2015.

[17] Johannes Textor and Maciej Liśkiewicz. Adjustment criteria in casual diagrams: an algorithmic perspective. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 681–88. AUAI Press, 2011.

[18] Johannes Textor, Benito van der Zander, Mark S. Gilthorpe, Maciej Liśkiewicz, and George TH Ellison. Robust causal inference using directed acyclic graphs: the R package 'dagitty'. *International Journal of Epidemiology*, 45(6):1887–1894, December 2016.

[19] Jin Tian, Azaria Paz, and Judea Pearl. Finding minimal d-separators. Technical Report R-254, UCLA, 1998.

[20] Benito van der Zander, Maciej Liśkiewicz, and Johannes Textor. Constructing separators and adjustment sets in ancestral graphs. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pages 907–16. AUAI Press, 2014.

[21] Benito van der Zander, Johannes Textor, and Maciej Liśkiewicz. Efficiently finding conditional instruments for causal inference. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 3243–49. AAAI Press, 2015.

[22] Dirk van Kampen. The ssq model of schizophrenic prodromal unfolding revised: An analysis of its causal chains based on the language of directed graphs. *European Psychiatry*, 29(7):437–48, 2014.