Accessible Computation of Tight Symbolic Bounds on Causal Effects using an Intuitive Graphical Interface

by Gustav Jonzon, Michael C Sachs, and Erin E Gabriel

Abstract Strong untestable assumptions are almost universal in causal point estimation. In particular settings, bounds can be derived to narrow the possible range of a causal effect. Symbolic bounds apply to all settings that can be depicted using the same directed acyclic graph and for the same effect of interest. Although the core of the methodology for deriving symbolic bounds has been previously developed, the means of implementation and computation have been lacking. Our R-package causaloptim aims to solve this usability problem by providing the user with a graphical interface through Shiny. This interface takes input in a form that most researchers with an interest in causal inference will be familiar: a graph drawn in the user's web browser and a causal query written in text using common counterfactual notation.

1 Introduction

A common goal in many different areas of scientific research is to determine causal relationships between one or more exposure variables and an outcome. Prior to any computation or inference, we must clearly state all assumptions made, i.e., all subject matter knowledge available, regarding the causal relationships between the involved variables as well as any additional variables, called confounders, that may not be measured but influence at least two other variables of interest. A popular tool in applied research for encoding these causal assumptions is a directed acyclic graph (DAG), in which directed edges represent direct causal influences (Greenland et al., 1999). Such a DAG not only clearly states the assumptions made by the researcher, but also comes with a sound methodology for causal inference, in the form of identification results (theorems on when an estimand is estimable) as well as derivation of expressions of causal estimands in terms of observable quantities (Pearl, 2009).

Unfortunately, point identification of a desired causal effect typically requires an assumption of no unmeasured confounders, in some form. When there are unmeasured confounders, it is sometimes still possible to derive bounds on the effect, i.e., a range of possible values for the causal effect in terms of the observed data distribution. Symbolic bounds are algebraic expressions for the bounds on the causal effect written in terms of probabilities that can be estimated using observed data. Alexander Balke and Judea Pearl first used linear programming to derive tight symbolic bounds in a simple binary instrumental variable (IV) setting (Balke and Pearl, 1997). Balke wrote a program in `C++` to take a linear programming problem as text file input, perform variable reduction, conversion of equality constraints into inequality constraints, and perform the vertex enumeration algorithm of Mattheiss (1973). This program has since been used by researchers in the field of causal inference (Balke and Pearl, 1997; Cai et al., 2008; Sjölander, 2009; Sjölander et al., 2014) but it is not particularly accessible because of the technical challenge of translating the DAG plus causal query into the constrained optimization problem and determining whether it is linear. Moreover, the program is not optimized and hence does not scale well to more complex problems. Since they only cover a simple instrumental variable setting, it has also not been clear to what extent their techniques extend to more general settings, nor how to apply them to more complex queries. Thus, applications of this approach have been limited to a small number of settings and few attempts to generalize the method to more widely applicable settings have been made.

Recent developments have expanded the applicability by generalizing the techniques and the causal DAGs and effects to which they apply (Sachs et al., 2022). These new methods have been applied in novel observational and experimental settings (Gabriel et al., 2022a, 2023, 2022b). Moreover, through the `R` package **causaloptim** (Sachs et al., 2023), these computations are now accessible. With **causaloptim**, the user needs only to give input in a way they would usually express their causal assumptions and state their target causal estimand; through a DAG and counterfactual expression. Providing DAGs through textual input is an awkward experience for most users, as DAGs are generally communicated pictorially. Our package **causaloptim** provides a user-friendly graphical interface (GUI) through a web browser, where the user can draw their DAG in a way that is familiar to them. The methodology that underpins **causaloptim** is not universal however; some restrictions on the DAG and query are imposed. These are validated and communicated to the user through the graphical interface, which guides the user through providing the DAG and query, adding any extra conditions beyond those encoded in the DAG, computing, interpreting and exporting the bounds for various

further analyses.

There exist few other R-packages related to causal bounds and none to our knowledge for computation of symbolic bounds. **bpbounds** (Palmer et al., 2018) provides a text-based interface to compute numeric bounds for the original single instrumental variable example of Balke and Pearl and extends this by being able to compute bounds given different types of data input including a ternary rather than binary instrument. There is also a standalone program written in Java by the TETRAD Project (https://github.com/cmu-phil/tetrad) that includes a GUI and has a wrapper for R. Its focus, however, is on causal discovery in a given sample data set, and although it can also compute bounds, it can do so only numerically for the given data set.

In this paper we describe our R package **causaloptim**, first focusing on the graphical and programmatic user interfaces in the next 2 sections. Then we highlight some of our interesting functions and data structures that may be useful in other contexts. We provide a summary of the theoretical background and methods, while referring to the companion paper Sachs et al. (2022) for the details. We illustrate the use of the package with some numeric examples and close with a discussion and summary.

2 Graphical user interface

In the following, we will work through the binary instrumental variable example, where we have 3 observed binary variables X, Y, Z, and we want to determine the average causal effect of X on Y given by the total causal risk difference, in the presence of unmeasured confounding by U_R and an instrumental variable Z. Our causal DAG is given by $Z \rightarrow X \rightarrow Y$ and $X \leftarrow U_R \rightarrow Y$ and our causal query is $P(Y(X = 1) = 1) - P(Y(X = 0) = 1)$, where we use $Y(X = x)$ to denote the counterfactual outcome Y if X were intervened upon to have value x. The package can handle more variables and complex settings including multiple exposures, outcomes and nested counterfactuals (although only a single exposure and outcome may be set by clicking vertices in the GUI, the query may easily be modified to a more complex one), but this classic example will serve to showcase its functionality.

causaloptim includes a GUI implemented in **shiny** (Chang et al., 2022). The interface is launched in the user's default web browser by calling `specify_graph()`. Once the **shiny** app is launched, the user is presented with an interactive display as shown in Figure 1, in which they can draw their causal DAG. This display is divided into a left side $\mathcal{L}$ and right side $\mathcal{R}$ to classify the vertices according to the class of DAGs that the method covers. This division signifies the following: Existence of unmeasured confounding is assumed *within each* of these sides, but *not between* them, and any causal influence between the two sides *must originate* in $\mathcal{L}$. Generally speaking, the outcome variables of interest will typically be on the right side, things like instruments or randomized treatments will be on the left side, and confounded exposures of interest on the right side. Thus, for the example, we would want to put the instrumental variable on the left side, but the exposure and outcome on the right side. In the web version of this article an interactive version of this interface is shown at the end of this section.

The DAG that we aim to construct is depicted in Figure 2.

2.1 Specifying the setting by drawing a causal diagram and adding attributes

The DAG is drawn using a point-and-click device (e.g., a mouse) to add vertices representing variables (by Shift-click) and name them (using any valid variable name in R without underscores), and to draw edges representing direct causal influences (Shift+drag) between them. The vertices may also be moved around, renamed and deleted (as can the edges) as also described in an instruction text preceding the DAG interface. As shown in Figure 3, for the example we add a vertex Z on the left side, and vertices X and Y on the right side. Then the $Z \rightarrow X$ and $X \rightarrow Y$ edges are added by Shift+clicking on a parent vertex and dragging to the child vertex. There is no need to add the unmeasured confounder variable U_R as it is assumed and added automatically. The same would have applied to U_L, but a confounder is added to a side only if it contains at least two variables.

Importantly, the nodes may be selected and assigned additional information. In $\mathcal{R}$, a variable may be assigned as unobserved (click+'u'). All observed variables are assumed categorical and their cardinality (i.e., number of levels) may be set (click+'c' brings up a prompt for this this number; alternatively a short-cut click+'any digit' is provided), with the default being binary. Although the causal query (i.e., the causal effect of interest) is entered subsequently, the DAG interface provides a convenient short-cut; a node X may be assigned as an exposure (click+'e') and another Y as outcome (click+'y'), whereupon the default query is the total causal risk difference $P(Y(X = 1) = 1) - P(Y(X = 0) = 1)$. Finally, an edge may be assigned as representing an assumed monotonic influence (click+'m'), i.e., that increasing the value of the exposure can not decrease the value of the outcome, e.g, $X(Z = 0) \leq X(Z = 1)$. The nodes and edges change appearance according to their assigned

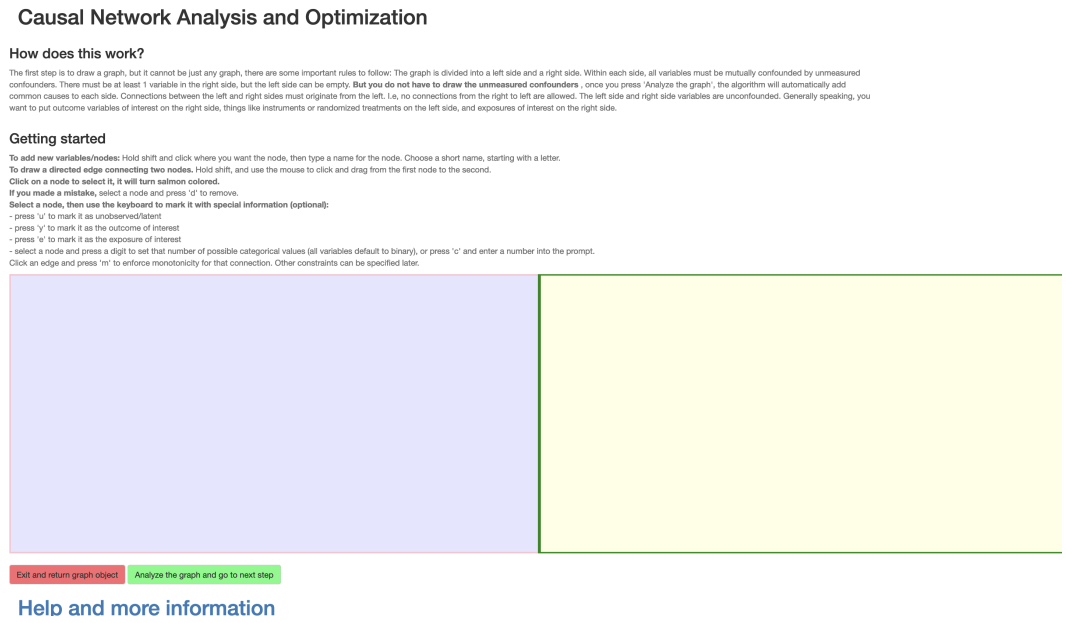


Figure 1: The Shiny web interface at launch

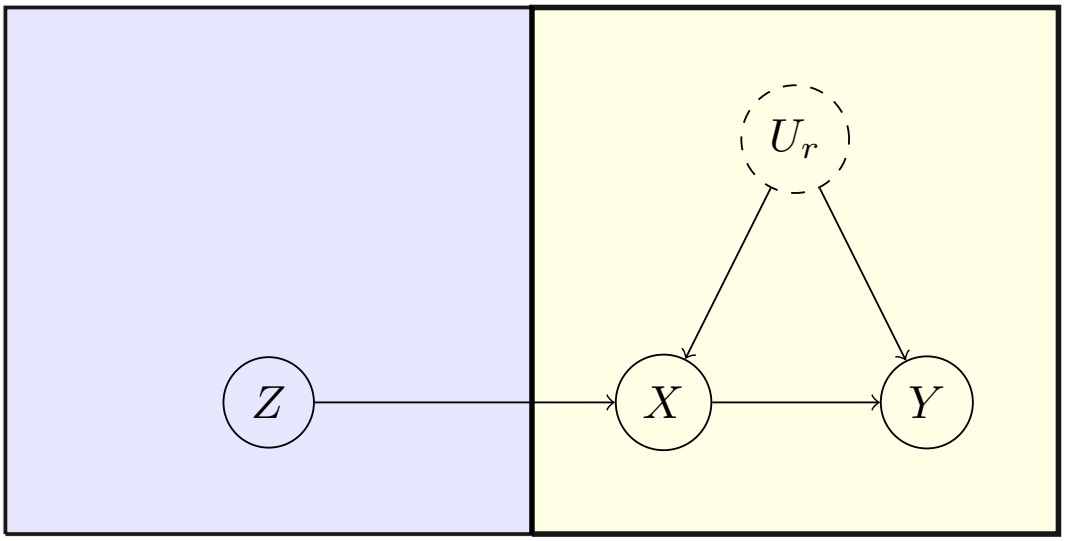
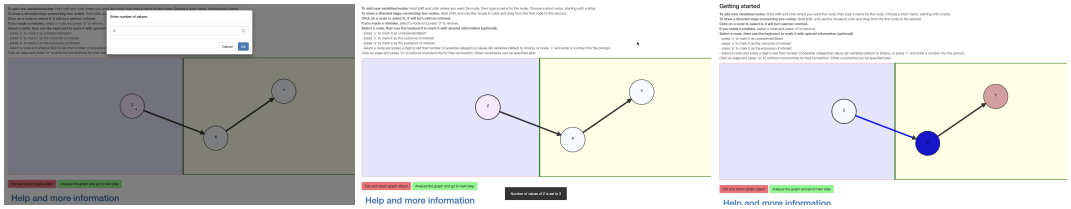


Figure 2: Causal DAG for the IV example.

(a) Adding and naming variables

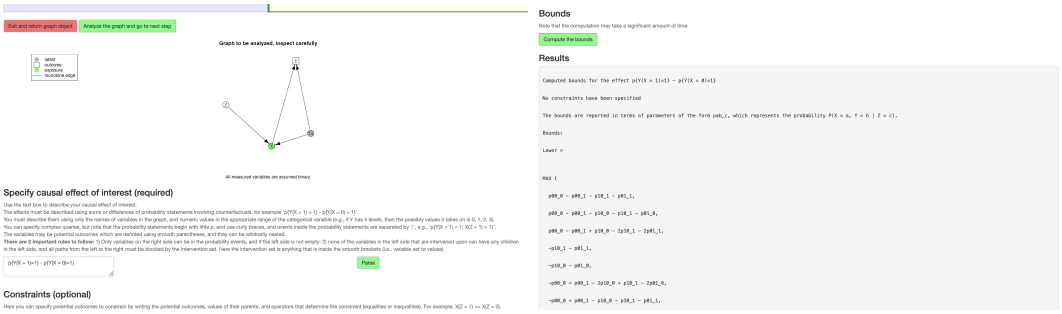
(b) Adding directed edges

Figure 3: Constructing the DAG



(a) Setting the number of categories **(b)** Confirmation message **(c)** Setting exposure and outcome

Figure 4: Setting attributes



(a) Graphical summary of the DAG with added confounding **(b)** Computing the bounds

Figure 5: The causal DAG and bounds

characteristics (Figure 4) and violations to the restrictions characterizing the class of DAGs are detected and communicated to the user.

Once the DAG has been drawn, the user may click the button "Analyze the graph", upon which the DAG is interpreted and converted into an annotated **igraph**-object (Csardi and Nepusz, 2006) as described in the implementation details below, and the results are displayed in graphical form to the user (Figure 5). The addition of U_R, the common unmeasured cause of X and Y, is added and displayed in this static plot.

2.2 Specifying the causal query

Next, the user is asked to specify the causal query, i.e., causal effect of interest. If no outcome variable has been assigned in the DAG then the input field for the causal query is left blank and a query needs to be specified. In our example, since we have assigned an exposure and outcome using the DAG interface, the total causal risk difference $P(Y(X = 1) = 1) - P(Y(X = 0) = 1)$ is suggested.

2.3 Specifying optional additional constraints

The user is also given the option to provide additional constraints besides those imposed by the DAG. This may be considered an optional advanced feature where, e.g., monotonicity of a certain direct influence of Z on X may be assumed by entering $X(Z = 1) \geq X(Z = 0)$, with any such extra constraints separated by line breaks. If this feature is used, the input is followed by clicking the button "Parse", which identifies and fixes them.

2.4 Computing the symbolic tight bounds on the query under the given constraints

As the final step, the button "Compute the bounds" is clicked, whereupon the constraints and objective are compiled into an optimization problem which is then solved for tight causal bounds on the query symbolically in terms of observational quantities (conditional probabilities of the observed variables in the DAG) and the expressions are displayed alongside information on how the parameters are to be interpreted in terms of the given variable names (Figure 5). During computation, a progress indicator is shown, and the user should be aware that complex and/or high-dimensional problems may take significant time. The interface also provides a feature to subsequently convert the bounds to LaTeX-code using standard probabilistic notation for publication purposes.

Once done, clicking "Exit and return objects to R" stops the **shiny** app and returns all information about the DAG, query and computed bounds to the R-session. This information is bundled in a list containing the graph, query, parameters and their interpretation, the symbolic tight bounds as expressions as well as implementations as R-functions and further log information about the formulation and optimization procedures.

3 Programmatic user interface

Interaction may also be done entirely programmatically as we illustrate with the same binary instrumental variable example. First we create the `igraph` object using the `graph_from_literal` function. Once the basic graph is created, the necessary vertex and edge attributes are added. The risk difference is defined as a character object. The `analyze_graph` function is the workhorse of **causaloptim**; it translates the causal graph, constraints, and causal effect of interest into a linear programming problem. This linear programming object, stored in `obj` in the code below, gets passed to `optimize_effect_2` which performs vertex enumeration to obtain the bounds as symbolic expressions in terms of observable probabilities. Note that when we textually enter the DAG, we also manually add the unmeasured confounding. Although the unmeasured confounder is designated as binary in the code, this is *not* assumed in the the computations where it is treated as possibly continuous, e.g.

```
graph <- igraph::graph_from_literal(Z -+ X, X -+ Y,
                                    Ur -+ X, Ur -+ Y)
V(graph)$leftside <- c(1, 0, 0, 0)
V(graph)$latent   <- c(0, 0, 0, 1)
V(graph)$nvals    <- c(2, 2, 2, 2)
E(graph)$rlconnect     <- c(0, 0, 0, 0)
E(graph)$edge.monotone <- c(0, 0, 0, 0)

riskdiff <- "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}"
obj <- analyze_graph(graph, constraints = NULL, effectt = riskdiff)
bounds <- optimize_effect_2(obj)
bounds
#> lower bound =
#> MAX {
#>   p00_0 - p00_1 - p10_1 - p01_1,
#>   p00_0 - p00_1 - p10_0 - p10_1 - p01_0,
#>   p00_0 - p00_1 + p10_0 - 2p10_1 - 2p01_1,
#>   -p10_1 - p01_1,
#>   -p10_0 - p01_0,
#>   -p00_0 + p00_1 - 2p10_0 + p10_1 - 2p01_0,
#>   -p00_0 + p00_1 - p10_0 - p10_1 - p01_1,
#>   -p00_0 + p00_1 - p10_0 - p01_0
#> }
#> ----------------------------------------
#> upper bound =
#> MIN {
#>   1 - p10_1 - p01_0,
#>   1 + p00_0 + p10_0 - 2p10_1 - p01_1,
#>   2 - p00_1 - p10_0 - p10_1 - 2p01_0,
#>   1 - p10_1 - p01_1,
#>   1 - p10_0 - p01_0,
#>   1 + p00_1 - 2p10_0 + p10_1 - p01_0,
#>   2 - p00_0 - p10_0 - p10_1 - 2p01_1,
#>   1 - p10_0 - p01_1
#> }
```

The resulting bounds object contains character strings representing the bounds and logs containing detailed information from the vertex enumeration algorithm. The bounds are printed to the console but more features are available to facilitate their use. The `interpret_bounds` function takes the bounds and parameter names as input and returns an R function implementing vectorized forms of the symbolic expressions for the bounds.

```
bounds_function <- interpret_bounds(bounds$bounds, obj$parameters)
str(bounds_function)
```

```
#> function (p00_0 = NULL, p00_1 = NULL, p10_0 = NULL, p10_1 = NULL, p01_0 = NULL,
#>     p01_1 = NULL, p11_0 = NULL, p11_1 = NULL)
```

The results can also be used for numerical simulation using `simulate_bounds`. This function randomly generates counterfactuals and probability distributions that satisfy the constraints implied by the DAG and optional constraints. It then computes and returns the bounds as well as the true causal effect.

If one wants to bound a different effect using the same causal graph, the `update_effect` function can be used to save some computation time. It takes the object returned by `analyze_graph` and the new effect string and returns an object of class `linearcausalproblem` that can be optimized: `obj2 <- update_effect(obj, "p{Y(X = 1) = 1}")`.

Finally, LaTeX-code may also be generated using the function `latex_bounds` as in `latex_bounds(bounds$bounds, obj$parameters)` yielding

$$\text{Lower bound} = \max \left\{ \begin{array}{l} P(X=0,Y=0|Z=0) - P(X=0,Y=0|Z=1) - P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=1), \\ P(X=0,Y=0|Z=0) - P(X=0,Y=0|Z=1) - P(X=1,Y=0|Z=0) - P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=0), \\ P(X=0,Y=0|Z=0) - P(X=0,Y=0|Z=1) + P(X=1,Y=0|Z=0) - 2P(X=1,Y=0|Z=1) - 2P(X=0,Y=1|Z=1), \\ -P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=1), \\ -P(X=1,Y=0|Z=0) - P(X=0,Y=1|Z=0), \\ -P(X=0,Y=0|Z=0) + P(X=0,Y=0|Z=1) - 2P(X=1,Y=0|Z=0) + P(X=1,Y=0|Z=1) - 2P(X=0,Y=1|Z=0), \\ -P(X=0,Y=0|Z=0) + P(X=0,Y=0|Z=1) - P(X=1,Y=0|Z=0) - P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=1), \\ -P(X=0,Y=0|Z=0) + P(X=0,Y=0|Z=1) - P(X=1,Y=0|Z=0) - P(X=0,Y=1|Z=0) \end{array} \right\}$$

$$\text{Upper bound} = \min \left\{ \begin{array}{l} 1 - P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=0), \\ 1 + P(X=0,Y=0|Z=0) + P(X=1,Y=0|Z=0) - 2P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=1), \\ 2 - P(X=0,Y=0|Z=1) - P(X=1,Y=0|Z=0) - P(X=1,Y=0|Z=1) - 2P(X=0,Y=1|Z=0), \\ 1 - P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=1), \\ 1 - P(X=1,Y=0|Z=0) - P(X=0,Y=1|Z=0), \\ 1 + P(X=0,Y=0|Z=1) - 2P(X=1,Y=0|Z=0) + P(X=1,Y=0|Z=1) - P(X=0,Y=1|Z=0), \\ 2 - P(X=0,Y=0|Z=0) - P(X=1,Y=0|Z=0) - P(X=1,Y=0|Z=1) - 2P(X=0,Y=1|Z=1), \\ 1 - P(X=1,Y=0|Z=0) - P(X=0,Y=1|Z=1) \end{array} \right\}.$$

4 Implementation and program overview

An overview of the main functions and their relations is depicted as a flow chart in Figure 6. All functions may be called individually by the user in the R-console and all input, output and interaction available through the **shiny** app is available through the R-console as well.

Sachs et al. (2022) define the following class of problems for which the query in general is not identifiable, but for which a methodology to derive symbolic tight bounds on the query is provided. The causal DAG consists of a finite set $\mathcal{W} = \{W_1, \ldots, W_n\} = \mathcal{W}_\mathcal{L} \cup \mathcal{W}_\mathcal{R}$ of categorical variables with $\mathcal{W}_\mathcal{L} \cap \mathcal{W}_\mathcal{R} = \varnothing$, no edges going from $\mathcal{W}_\mathcal{R}$ to $\mathcal{W}_\mathcal{L}$ and no external common parent between $\mathcal{W}_\mathcal{L}$ and $\mathcal{W}_\mathcal{R}$, but *importantly* external common parents $\mathcal{U}_\mathcal{L}$ and $\mathcal{U}_\mathcal{R}$ of variables within $\mathcal{W}_\mathcal{R}$ and $\mathcal{W}_\mathcal{R}$ may not be ruled out. Nothing is assumed about any characteristics of these confounding variables $\mathcal{U}_\mathcal{L}$ and $\mathcal{U}_\mathcal{R}$.

The causal query may in principle be any linear combination (although **causaloptim** version 0.9.8 implements contrasts with unit coefficients only) of joint probabilities of factual and counterfactual outcomes expressed in terms of the variables in $\mathcal{W}$ and may always be expressed as a sum of probabilities of response function variables of the DAG. It is subject to the restriction that each outcome variable is in $\mathcal{W}_\mathcal{R}$ and if $\mathcal{W}_\mathcal{L} \neq \varnothing$ it is also subject to a few regularity conditions as detailed in Sachs et al. (2022). Tight bounds on the query may then be derived symbolically in terms of conditional probabilities of the observable variables $\mathcal{W}$.

Algorithms 1 and 2 in Sachs et al. (2022) construct the constraint space and causal query in terms of the joint probabilities of the response function variables and in **causaloptim** are implemented in the functions `create_R_matrix` and `create_effect_vector` respectively. Both are called as sub-procedures of the function `analyze_graph` to translate the causal problem to that of optimizing an objective function over a constraint space. The implementation of Algorithm 1 involves constructing the response functions themselves as actual R-functions. Evaluating these correspond to evaluating the structural equations of the causal DAG.

The conditions on the DAG suffice to ensure that the causal query will depend only on the response functions corresponding to the variables in $\mathcal{W}_\mathcal{R}$ and that the exhaustive set of constraints on their probabilities are linear in a subset of conditional probabilities of observable variables (Proposition 2 in Sachs et al. (2022)), and the conditions on the query in turn ensure that it may be expressed as a linear combination of joint probabilities of the response functions of the variables in $\mathcal{W}_\mathcal{R}$ (Proposition 3 in Sachs et al. (2022)). Once this formulation

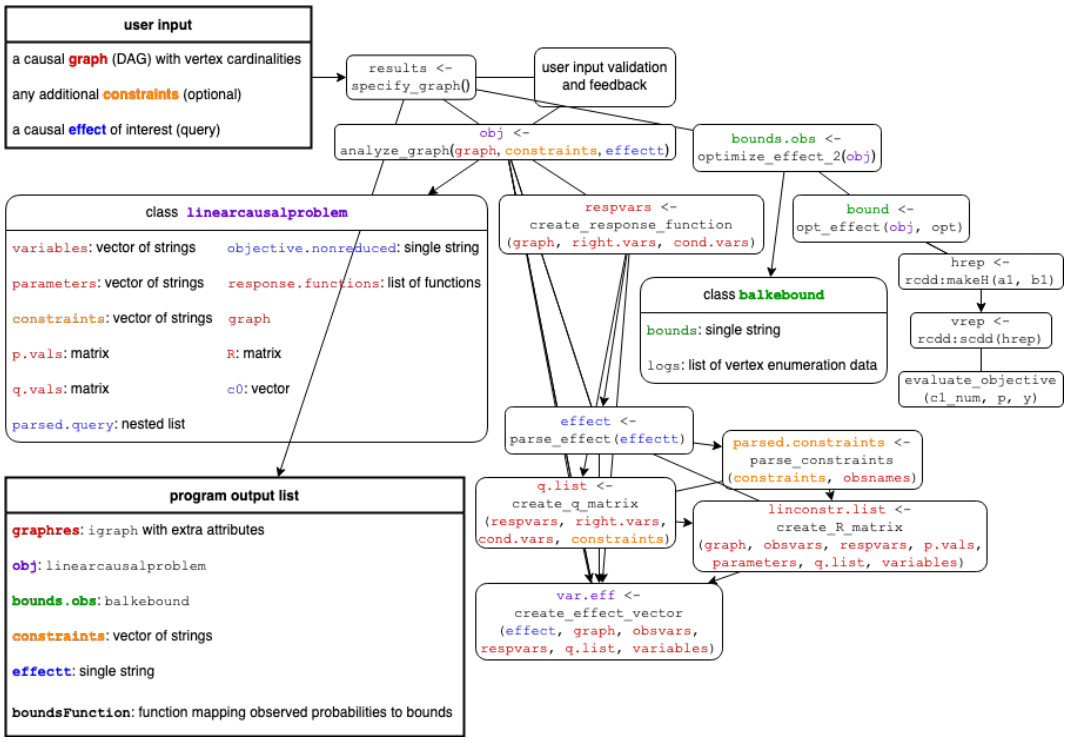


Figure 6: A function overview flow chart, listing user input, program output and overall structure. The function `specify_graph` launches the GUI. Undirected edges correspond to function calls and arrow heads to direct input. Objects are color coded according to their sources of information, with red corresponding to the DAG, blue to the query and purple to a mixture of them, as in the optimization problem of class `linearcausalproblem`. Finally, green corresponds to the subsequent optimization process producing the bounds of class `balkebound`.

of the causal problem as a linear program has been set up, a vertex enumeration method is employed to compute the extrema symbolically in terms of conditional probabilities of the observable variables.

The main and interesting functions will be described in some detail below. We begin however with an overview of how they are tied together by the **shiny** app.

`specify_graph`

The graphical interface is launched by `specify_graph()`, or preferably `results <- specify_graph()`. Once the **shiny** app is stopped, the input, output and other useful information is returned by the function, so we recommend assigning it to a variable so they are saved in the R-session and may easily be further analyzed and processed. All further function calls will take place automatically as the user interacts with the web interface. Thus, from a basic user perspective, `specify_graph` is the main function. The core functionality however is implemented in the functions `analyze_graph`, `optimize_effect_2` and their subroutines.

The `JavaScript` that handles the communication between the **shiny** server and the input as the user draws a DAG through the web interface uses on the project `directed-graph-creator`, an interactive tool for creating directed graphs, created using `d3.js` and hosted at https://github.com/cjrd/directed-graph-creator, which has been modified for the purpose of causal diagrams. The modification binds the user inputs as they interact with the graph to **shiny** so that the directed graph and its attributes set by the user are reactively converted into an `igraph`-object for further processing. Since directed graphs are common in many computational and statistical problems, this **shiny** interface may also be valuable to many other R-package authors and maintainers who may wish to provide their users with an accessible and intuitive way to interact with their software.

The server listens to a reactive function that, as the user draws the DAG, collects infor-

mation about the current edges, collects and annotates vertices, adds left- and right-side confounding, and returns an annotated `igraph`-object, comprising information about the connectivity along with some additional attributes; for each variable, its name, cardinality, latency-indicator and side-indicator, and for each edge, a monotonicity-indicator and (to detect and communicate violations on direction) a right-to-left-indicator. The server meanwhile also monitors the DAG for any violation of the restriction that each edge between $\mathcal{L}$ and $\mathcal{R}$ must go *from* $\mathcal{L}$ *to* $\mathcal{R}$, and if detected directly communicates this to the user through a text message in the **shiny** app. The app makes use **shiny** modules which makes testing the interface easier, and **causaloptim** includes many automated tests to make sure that the GUI functionality works as intended.

`analyze_graph`

The function `analyze_graph` takes a DAG (in the form of an `igraph` object), optional constraints, and a string representing the causal effect of interest and proceeds to construct and return a linear optimization problem (a `linearcausalproblem`-object) from these inputs.

First, some basic data structures are created to keep track of the observed variables, their possible values, the latent variables, and whether they are in $\mathcal{L}$ or $\mathcal{R}$. Once these basic data-structures have been created, the first task of the algorithm is to create the response function variables (for each variable, observed or not, except $U_{\mathcal{L}}$ and $U_{\mathcal{R}}$). Probabilities of these will be the entities $\mathbf{q}$ in which the objective function (representing the target causal effect) is expressed and will constitute the points in the space it is optimized over, where this space itself is constrained by the the relationships between them and observed conditional probabilities $\mathbf{p}$.

`create_response_function`

The function `create_response_function` returns a list `respvars` that has a named entry for each observed variable, containing its response function variable and response function. If X is an observed variable with n response functions, then they are enumerated by $\{0, \ldots, n-1\}$. Its entry `respvars$X` contains the response function variable R_X of X, and is a list with two entries. The first, `respvars$X$index`, is a vector containing all the possible values of R_X, i.e., the integers $(0, \ldots, n-1)$. The second, `respvars$X$values` is itself a list with n entries; each containing the particular response function of X corresponding to its index. Each such response function is an actual `R`-function and may be evaluated by passing it any possible values of the parents of X as arguments.

Next, the response function variables are used in the creation of a matrix of unobserved probabilities. Specifically the joint probabilities $P(\mathbf{R}_{\mathcal{R}} = \mathbf{r}_{\mathcal{R}})$ for each possible value-combination $\mathbf{r}_{\mathcal{R}}$ of the response function variables $\mathbf{R}_{\mathcal{R}}$ of the right-side-variables $\mathbf{W}_{\mathcal{R}}$. In Sachs et al. (2022), the possible value-combinations $\mathbf{r}_{\mathcal{R}}$ are enumerated by $\gamma \in \{1, \ldots, \aleph_{\mathcal{R}}\}$ with corresponding probabilities $q_\gamma := P(\mathbf{R}_{\mathcal{R}} = \mathbf{r}_\gamma)$ being components of the vector $\mathbf{q} \in [0,1]^{\aleph_{\mathcal{R}}}$.

`create_R_matrix`

The constraints that the DAG and observed conditional probabilities $\mathbf{p}$ (in `p.vals`) impose on the unobserved probabilities $\mathbf{q}$ (represented by `variables`) are linear. Specifically, there exists a matrix whose entries are the coefficients relating `p.vals` to `variables`. This matrix is called P in Sachs et al. (2022), where its existence is guaranteed by Proposition 2 and its construction is detailed in Algorithm 1, which is implemented in the function `create_R_matrix`. This function returns back a list with two entries; a vector of strings representing the linear constraints on the unobserved $\mathbf{q} \in [0,1]^{\aleph_{\mathcal{R}}}$ imposed by and in terms of the observed $\mathbf{p} \in [0,1]^B$ and the numeric matrix $R \in \{0,1\}^{(B+1)\times\aleph_{\mathcal{R}}}$ of coefficients corresponding to these constraints as well as the probabilistic ones and given by $R = \begin{pmatrix} \mathbf{1} \\ P \end{pmatrix}$ where $P \in \{0,1\}^{B\times\aleph_{\mathcal{R}}} : \mathbf{p} = P\mathbf{q}$, so $R\mathbf{q} = \begin{pmatrix} 1 \\ \mathbf{p} \end{pmatrix}$.

This determines the constraint space as a compact convex polytope in **q**-space, i.e., in $\mathbb{R}^{\aleph_{\mathcal{R}}}$. To create the matrix, we define a recursive function `gee_r` that takes two arguments; a positive integer `i` being the index $i \in \{1, \ldots, n\}$ of a variable $W_i \in \mathcal{W}$ (i.e. the i^{th} component of **W** or, equivalently, the `i`th entry of `obsvars`) and a vector `r` being a value $\mathbf{r} \in \nu(\mathbf{R})$ in the set $\nu(\mathbf{R})$ of all possible value-vectors of the joint response function variable **R**. This recursive function is called for each variable in `obsvars` and for each possible value of the response function variable vector. The base case is reached if the variable has no parents, in which case the list corresponding to the response function variable R_{W_i} of W_i is extracted from `respvars`. From this list, the entry whose index matches the `i`th index of `r` (i.e. the one corresponding to the response function variable value r_i =`r[i]`) is extracted and finally its value, i.e., the corresponding response function itself, is extracted and is evaluated on an empty list of arguments, since it is a constant function and determined only by the value r_i.

The recursive case is encountered when `parents` is non-empty. If so, then for each parent in `parents`, its index in `obsvars` is determined and `gee_r` is recursively called with the same vector `r` as first argument but now with this particular index (i.e. that of the current parent) as second argument. The numeric values returned by these recursive calls are then sequentially stored in a vector `lookin`, whose entries are named by those in `parents`. Just as in the base case, the response function corresponding to the particular value r_i of the response function variable R_{W_i} (i.e. the response function of the variable `obsvars[i]` that has the index `r[i]`) is extracted from `respvars` and is now evaluated with arguments given by the list `lookin`. Note that `gee_r(r, i)` corresponds to the value $w_i = g^*_{W_i}(\mathbf{r})$ in Sachs et al. (2022).

Then the values that match the observed probabilities are recorded, the corresponding entries in the current row of the matrix `R` are set to 1 and a string representing the corresponding equation is constructed and added to the vector of constraints.

`parse_effect`

Now that the constraint space has been determined, the objective function representing the causal query needs to be specified as a linear function of the components of **q**, i.e., `variables`. First, the causal query that has been provided by the user as a text-string in `effectt` is passed to the function `parse_effect`, which identifies its components including nested counterfactuals and creates a data structure representing the causal query. This structure includes nested lists which represent all interventional paths to each outcome variable in the query.

Once the nested list `effect` is returned back to `analyze_graph`, it checks that the requirements (see Proposition 3 in Sachs et al. (2022)) on the query are fulfilled before creating the linear objective function. Despite these regularity conditions, a large set of possible queries may be entered using standard counterfactual notation, using syntax described in the accompanying instruction text along with examples such as $P(Y(M(X = 0), X = 1) = 1) - P(Y(M(X = 0), X = 0) = 1)$; the natural direct effect (Pearl, 2001) of a binary exposure X at level $M = 0$ on a binary outcome Y *not* going through the mediator M, in the presence of unmeasured confounding between M and Y (Sjölander, 2009).

`create_effect_vector`

Now that the required characteristics of the query have been established, the corresponding objective function will be constructed by the function `create_effect_vector` which returns a list `var.eff` of string-vectors; one for each term in the query. Each such vector contains the names (strings in `variables`) of the response function variables of the right-side (i.e. the components of **q**) whose sum corresponds the that particular term. The function `create_effect_vector` implements Algorithm 2 of Sachs et al. (2022) with the additional feature that if the user has entered a query that is incomplete in the sense that there are omitted mediating variables on paths from base/intervention variables to the outcome variable, then this is interpreted as the user intending the effects of the base/intervention variables to be propagated through the mediators, so that they are set to their "natural" values under this intervention. These mediators are detected and their values are set accordingly.

We define a recursive function `gee_rA` that takes three arguments; a positive integer `i` (the index i of a variable $W_i \in \mathcal{W}$ =`obsvars`), a vector `r` (a value $\mathbf{r} \in \nu(\mathbf{R})$ in the set $\nu(\mathbf{R})$ of all possible value-vectors of the joint response function variable $\mathbf{R}$) and a string `path` that represents an interventional path and is of the form "X -> ... -> Y" if not `NULL`. The base case is reached either if `path` is non-`NULL` and corresponds to a path to the intervention set or if `parents` is empty. In the former case, the corresponding numeric intervention-value is returned, and in the latter case, the value of the corresponding response function called on the empty list of arguments is returned just as in the base case of `gee_r`. The recursive case is encountered when `path` is `NULL` and `parents` is non-empty. This recursion proceeds just as in `gee_r`, but now rather with a recursive call to `gee_rA`, whose third argument is now `path = paste(gu, "->", path)` where the string in `gu` is the name of the parent variable in `parents` whose index `i` in `obsvars` is the second argument of this recursive call. This construction traces the full path taken from the outcome of interest to the variable being intervened upon. Note that `gee_rA(r, i, path)` corresponds to the value $w_i = h_{W_i}^{A_i}(\mathbf{r}, W_i)$ in Sachs et al. (2022). A matrix is now created just as in the observational case, but this time using `gee_rA` instead of `gee_r` .

`optimize_effect_2`

Once the constraints on $\mathbf{q}$ as well as the effect of interest in terms of $\mathbf{q}$ have been established, it remains only to optimize this expression over the constraint space. Here, $\mathbf{c}$ denotes the constant gradient vector of the linear objective function and P denotes the coefficient matrix of the linear restrictions on $\mathbf{q}$ in terms of $\mathbf{p}$ imposed by the causal DAG. By adding the probabilistic constraints on $\mathbf{q}$ we have arrived at e.g. the following linear program giving a tight lower bound on the average causal effect $\theta_{\mathbf{q}} = P\{Y(X=1)=1\} - P\{Y(X=0)=1\}$ in the simple instrumental variable problem of the introductory section:

$$\begin{aligned}\min_{\mathbf{q}} \theta_{\mathbf{q}} &= \min\{\mathbf{c}^\top \mathbf{q} \mid \mathbf{q} \in \mathbb{R}^{16}, \mathbf{q} \geq \mathbf{0}_{16\times 1}, \mathbf{1}_{1\times 16}\mathbf{q} = 1, P\mathbf{q} = \mathbf{q}\} \\ &= \max\{\begin{pmatrix}1 & \mathbf{q}^\top\end{pmatrix}\mathbf{y} \mid \mathbf{y} \in \mathbb{R}^9, \mathbf{y} \geq \mathbf{0}_{9\times 1}, \begin{pmatrix}\mathbf{1}_{16\times 1} & P^\top\end{pmatrix} \leq \mathbf{c}\} \\ &= \max\{\begin{pmatrix}1 & \mathbf{q}^\top\end{pmatrix}\bar{\mathbf{y}} \mid \bar{\mathbf{y}} \text{ is a vertex of } \{\mathbf{y} \in \mathbb{R}^9 \mid \mathbf{y} \geq \mathbf{0}_{9\times 1}, R^\top \leq \mathbf{c}\}\}\end{aligned}$$

Since we allow the user to provide additional linear inequality constraints (e.g. it may be quite reasonable to assume the proportion of "defiers" in the study population of our example to be quite low), the actual primal and dual linear programs may look slightly more complicated, but this small example still captures the essentials.
In general, given the matrix of linear constraints on the observable probabilities implied by the DAG and an optional user-provided matrix inequality, we construct the coefficient matrix and right hand side vector of the dual polytope.

The optimization via vertex enumeration step in **causaloptim** is implemented in the function `optimize_effect_2` which uses the double description method for vertex enumeration, as implemented in the **rcdd** package (Geyer et al., 2021). This step of vertex enumeration has previously been the major computational bottleneck. The approach is now based on `cddlib` (https://people.inf.ethz.ch/fukudak/cdd_home/), which has an implementation of the Double Description Method (dd). Any convex polytope can be dually described as either an intersection of half-planes (which is the form we get our dual constraint space in) or as a minimal set of vertices of which it is the convex hull (which is the form we want it in) and the dd algorithm efficiently converts between these two descriptions. `cddlib` also uses exact rational arithmetic, so there is no need to worry about any numerical instability issues. The vertices of the dual polytope are obtained and stored as rows of a matrix with `hrep <- rcdd::makeH(a1, b1); vrep <- rcdd::scdd(hrep); vertices <- vrep$output[vrep$output[, 1] == 0 & vrep$output[, 2] == 1, -c(1, 2), drop=FALSE]`.

The rest is simply a matter of plugging them into the dual objective function, evaluating the expression and presenting the results. The first part of this is done by `apply(vertices, 1, function(y) evaluate_objective(c1_num, p, y))` (here `(c1_num,p)`$= \left(\begin{pmatrix}b_\ell^\top & 1\end{pmatrix}, p\right)$ separates the dual objective gradient into its numeric and symbolic parts).

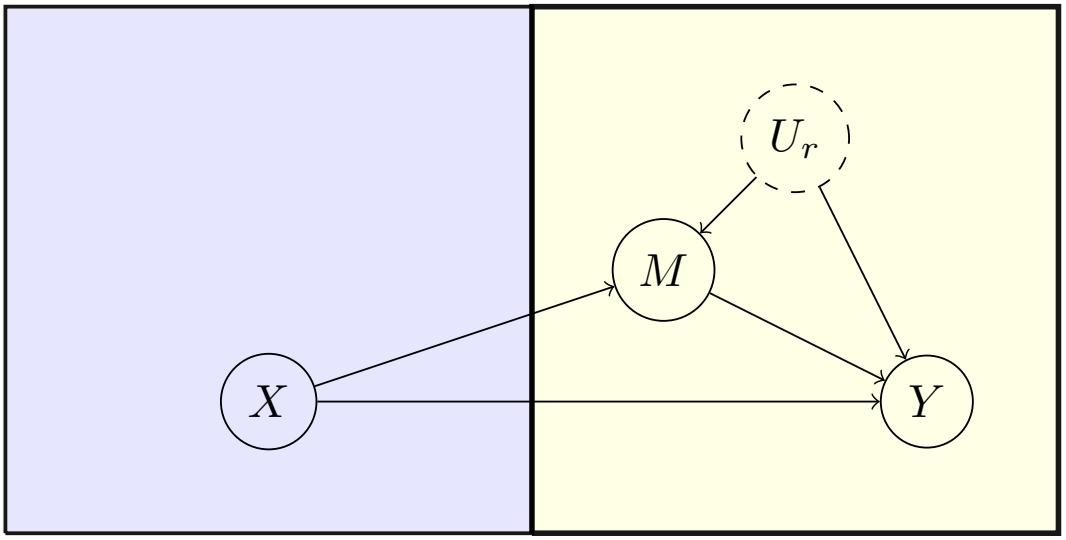


Figure 7: Causal DAG for mediation example

causaloptim also contains a precursor to to `optimize_effect_2`, called `optimize_effect`. This legacy function uses the original optimization procedure written in `C++` by Alexander Balke and involves linear program formulation followed by the vertex enumeration algorithm of Mattheiss (1973). This has worked well for very simple settings but has been found to struggle on more complex problems and thus been insufficient for the ambitions of **causaloptim**. The efficiency gains of `optimize_effect_2` over the legacy code have reduced the computation time for several setting from hours to milliseconds.

5 Numeric examples

We illustrate with a couple of applications. Although the problem formulations are entered via the textual interface below, they may of course just as well be entered via the graphical interface prior to numeric evaluation. This is done by launching the GUI and assigning its results to a variable as in `results <- specify_graph()`, following steps similar to the initial example in the section "Graphical User Interface" and clicking "Exit and return objects to R", whereupon all information about the DAG, query and bounds are accessible via `results`. Regardless if drawn via the GUI or entered via code, the DAG may be subsequently visualized using the `plot` function.

5.1 A Mediation Analysis

In Sjölander (2009), the author derives bounds on natural direct effects in the presence of confounded intermediate variables and applies them to data from the Lipid Research Clinics Coronary Primary Prevention Trial (Freedman et al., 1992), where subjects were randomized to cholestyramine treatment and presence of coronary heart disease events as well as levels of cholesterol were recorded after a 1-year follow-up period. We let X be a binary treatment indicator, with $X = 0$ indicating actual cholestyramine treatment and $X = 1$ indicating placebo. We further let Y be an indicator of the occurrence of coronary heart disease events within follow-up, with $Y = 0$ indicating event-free follow-up and $Y = 1$ indicating an event. We finally let M be a dichotomized (cut-off at 280 mg/dl) cholesterol level indicator, with $M = 0$ indicating levels < 280 mg/dl and $M = 1$ indicating levels ≥ 280 mg/dl. The causal assumptions are summarized in the DAG shown in Figure 7, where U_r is unmeasured and confounds the effect of M on Y.

```
b <- igraph::graph_from_literal(X -+ Y, X -+ M, M -+ Y,
                                Ur -+ Y, Ur -+ M)
V(b)$leftside <- c(1, 0, 0, 0)
```

```
V(b)$latent   <- c(0, 0, 0, 1)
V(b)$nvals    <- c(2, 2, 2, 2)
E(b)$rlconnect     <- c(0, 0, 0, 0, 0)
E(b)$edge.monotone <- c(0, 0, 0, 0, 0)
```

Note that although the unmeasured confounder `Ur` appears to be designated as binary in this code, this is *not* assumed in the the computations, where nothing is assumed about its distribution.

Using the data from Table IV of Sjölander (2009), we compute the observed conditional probabilities.

```
# parameters of the form pab_c, which represents
# the probability P(Y = a, M = b | X = c)
p00_0 <- 1426/1888 # P(Y=0,M=0|X=0)
p10_0 <- 97/1888 # P(Y=1,M=0|X=0)
p01_0 <- 332/1888 # P(Y=0,M=1|X=0)
p11_0 <- 33/1888 # P(Y=1,M=1|X=0)
p00_1 <- 1081/1918 # P(Y=0,M=0|X=1)
p10_1 <- 86/1918 # P(Y=1,M=0|X=1)
p01_1 <- 669/1918 # P(Y=0,M=1|X=1)
p11_1 <- 82/1918 # P(Y=1,M=1|X=1)
```

We proceed to compute bounds on the controlled direct effect $CDE(0) = P(Y(M = 0, X = 1) = 1) - P(Y(M = 0, X = 0) = 1)$ of X on Y not passing through M at level $M = 0$, the controlled direct effect $CDE(1) = P(Y(M = 1, X = 1) = 1) - P(Y(M = 1, X = 0) = 1)$ at level $M = 1$, the natural direct effect $NDE(0) = P(Y(M(X = 0), X = 1) = 1) - P(Y(M(X = 0), X = 0) = 1)$ of X on Y at level $X = 0$ and the natural direct effect $NDE(1) = P(Y(M(X = 1), X = 1) = 1) - P(Y(M(X = 1), X = 0) = 1)$ at level $X = 1$.

```
CDE0_query <- "p{Y(M = 0, X = 1) = 1} - p{Y(M = 0, X = 0) = 1}"
CDE0_obj <- analyze_graph(b, constraints = NULL, effectt = CDE0_query)
CDE0_bounds <- optimize_effect_2(CDE0_obj)
CDE0_boundsfunction <- interpret_bounds(bounds = CDE0_bounds$bounds,
                                        parameters = CDE0_obj$parameters)
CDE0_numericbounds <- CDE0_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                          p10_0 = p10_0, p10_1 = p10_1,
                                          p01_0 = p01_0, p01_1 = p01_1,
                                          p11_0 = p11_0, p11_1 = p11_1)

CDE1_query <- "p{Y(M = 1, X = 1) = 1} - p{Y(M = 1, X = 0) = 1}"
CDE1_obj <- update_effect(CDE0_obj, effectt = CDE1_query)
CDE1_bounds <- optimize_effect_2(CDE1_obj)
CDE1_boundsfunction <- interpret_bounds(bounds = CDE1_bounds$bounds,
                                        parameters = CDE1_obj$parameters)
CDE1_numericbounds <- CDE1_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                          p10_0 = p10_0, p10_1 = p10_1,
                                          p01_0 = p01_0, p01_1 = p01_1,
                                          p11_0 = p11_0, p11_1 = p11_1)
NDE0_query <- "p{Y(M(X = 0), X = 1) = 1} - p{Y(M(X = 0), X = 0) = 1}"
NDE0_obj <- update_effect(CDE0_obj, effectt = NDE0_query)
NDE0_bounds <- optimize_effect_2(NDE0_obj)
NDE0_boundsfunction <- interpret_bounds(bounds = NDE0_bounds$bounds,
                                        parameters = NDE0_obj$parameters)
NDE0_numericbounds <- NDE0_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                          p10_0 = p10_0, p10_1 = p10_1,
                                          p01_0 = p01_0, p01_1 = p01_1,
                                          p11_0 = p11_0, p11_1 = p11_1)

NDE1_query <- "p{Y(M(X = 1), X = 1) = 1} - p{Y(M(X = 1), X = 0) = 1}"
```

Table 1: Bounds on the controlled and natural direct effects.

	lower	upper
CDE(0)	-0.20	0.39
CDE(1)	-0.78	0.63
NDE(0)	-0.07	0.56
NDE(1)	-0.55	0.09

```
NDE1_obj <- update_effect(CDE0_obj, effectt = NDE1_query)
NDE1_bounds <- optimize_effect_2(NDE1_obj)
NDE1_boundsfunction <- interpret_bounds(bounds = NDE1_bounds$bounds,
                                        parameters = NDE1_obj$parameters)
NDE1_numericbounds <- NDE1_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                          p10_0 = p10_0, p10_1 = p10_1,
                                          p01_0 = p01_0, p01_1 = p01_1,
                                          p11_0 = p11_0, p11_1 = p11_1)
```

We obtain the same symbolic bounds as Sjölander (2009) and the resulting numeric bounds are given in Table 1 which of course agree with those of Table V in Sjölander (2009).

5.2 A Mendelian Randomization Study of the Effect of Homocysteine on Cardiovascular Disease

Mendelian randomization (Davey Smith and Ebrahim, 2003) assumes certain genotypes may serve as suitable instrumental variables for investigating the causal effect of an associated phenotype on some disease outcome.

In Palmer (2011), the authors investigate the effect of homocysteine on cardiovascular disease using the 677CT polymorphism (rs1801133) in the Methylenetetrahydrofolate Reductase gene as an instrument. They use observational data from Meleady et al. (2003) in which the outcome is binary, the treatment has been made binary by a suitably chosen cut-off at $15\mu mol/L$, and the instrument is ternary (this polymorphism can take three possible genotype values).

With X denoting the treatment, Y the outcome and Z the instrument, the DAG is the one in Figure 2 (although now with a ternary instrument) and the conditional probabilities are given as follows.

```
params <- list(p00_0 = 0.83, p00_1 = 0.88, p00_2 = 0.72,
               p10_0 = 0.11, p10_1 = 0.05, p10_2 = 0.20,
               p01_0 = 0.05, p01_1 = 0.06, p01_2 = 0.05,
               p11_0 = 0.01, p11_1 = 0.01, p11_2 = 0.03)
```

The computation using **causaloptim** is done using the following code.

```
# Input causal DAG
b <- graph_from_literal(Z -+ X, X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(1, 0, 0, 0)
V(b)$latent <- c(0, 0, 0, 1)
V(b)$nvals <- c(3, 2, 2, 2)
E(b)$rlconnect <- c(0, 0,  0, 0)
E(b)$edge.monotone <- c(0, 0, 0, 0)
# Construct causal problem
obj <- analyze_graph(b, constraints = NULL,
                     effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
# Compute bounds on query
bounds <- optimize_effect_2(obj)
# Construct bounds as function of parameters
boundsfunction <- interpret_bounds(bounds = bounds$bounds,
```

```
                                    parameters = obj$parameters)
# Insert observed conditional probabilities
numericbounds <- do.call(boundsfunction, as.list(params))
round(numericbounds, 2)

#>   lower upper
#> 1 -0.09  0.74
```

Our computed bounds agree with those computed using **bpbounds** as well as those estimated using Theorem 2 of Richardson and Robins (2014), who independently derived expressions for tight bounds that are applicable to this setting.

6 Summary and discussion

The methods and algorithms described in Sachs et al. (2022) to compute symbolic expressions for bounds on non-identifiable causal effects are implemented in the package **causaloptim**. Our aim was to provide a user-friendly interface to these methods with a graphical interface to draw DAGs, specify causal effects with commonly used notation for counterfactuals, and take advantage of an efficient implementation of vertex enumeration to reduce computation times. These methods are applicable to a wide variety of causal inference problems which appear in biomedical research, economics, social sciences and more. Aside from the graphical interface, programming with the package is encouraged to promote reproducibility and advanced use. Our package includes automated unit tests and also tests for correctness by comparing the symbolic bounds derived using our program to independently derived bounds in particular settings.

Our implementation uses a novel approach to draw DAGs using `JavaScript` in a web browser that can then be passed to `R` using **shiny**. This graphical approach can be adapted and used in other settings where graphs need to be specified and computed on, such as other causal inference settings, networks, and multi-state models. Other algorithms and data structures that could be more broadly useful include the representation of structural equations as `R` functions, recursive evaluation of response functions, and parsing of string equations for causal effects and constraints.

Note that **causaloptim** computes bounds only in settings with tightness guarantees. Although the set of available causal queries is quite large, the set of admissible DAGs can appear restrictive. The usefulness of the methodology and implementation extends to a much larger class of problems however. Tight bounds for restricted settings can be used to derive potentially not tight but narrow valid bounds in less restrictive settings. In Cai et al. (2007) and Cui and Tchetgen (2021) the authors make use of the tight Balke-Pearl bounds to derive narrow (but not necessarily tight) valid bounds on causal effects subject to causal DAGs that fall outside the scope for tightness guarantees using this methodology. We believe the tight bounds computed by **causaloptim** hold great potential for such applications.

References

A. Balke and J. Pearl. Bounds on treatment effects from studies with imperfect compliance. *Journal of the American Statistical Association*, 92(439):1171–1176, 1997. [p53]

Z. Cai, M. Kuroki, and T. Sato. Non-parametric bounds on treatment effects with non-compliance by covariate adjustment. *Statistics in Medicine*, 26(16):3188–3204, 2007. doi: https://doi.org/10.1002/sim.2766. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.2766. [p66]

Z. Cai, M. Kuroki, J. Pearl, and J. Tian. Bounds on direct effects in the presence of confounded intermediate variables. *Biometrics*, 64(3):695–701, 2008. [p53]

W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. *shiny: Web Application Framework for R*, 2022. URL https://CRAN.R-project.org/package=shiny. R package version 1.7.4. [p54]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL https://igraph.org. [p56]

Y. Cui and E. T. Tchetgen. A semiparametric instrumental variable approach to optimal treatment regimes under endogeneity. *Journal of the American Statistical Association*, 116 (533):162–173, 2021. doi: 10.1080/01621459.2020.1783272. URL https://doi.org/10.1080/01621459.2020.1783272. PMID: 33994604. [p66]

G. Davey Smith and S. Ebrahim. 'Mendelian randomization': can genetic epidemiology contribute to understanding environmental determinants of disease? *International Journal of Epidemiology*, 32(1):1–22, 02 2003. ISSN 0300-5771. doi: 10.1093/ije/dyg070. URL https://doi.org/10.1093/ije/dyg070. [p65]

L. S. Freedman, B. I. Graubard, and A. Schatzkin. Statistical validation of intermediate endpoints for chronic diseases. *Statistics in Medicine*, 11(2):167–178, 1992. doi: https://doi.org/10.1002/sim.4780110204. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.4780110204. [p63]

E. E. Gabriel, M. C. Sachs, and A. Sjölander. Causal bounds for outcome-dependent sampling in observational studies. *Journal of the American Statistical Association*, 117(538):939–950, 2022a. [p53]

E. E. Gabriel, M. C. Sachs, and A. Sjölander. Sharp nonparametric bounds for decomposition effects with two binary mediators. *Journal of the American Statistical Association*, page In press, 2022b. [p53]

E. E. Gabriel, A. Sjölander, and M. C. Sachs. Nonparametric bounds for causal effects in imperfect randomized experiments. *Journal of the American Statistical Association*, 118(541): 684–692, 2023. [p53]

C. J. Geyer, G. D. Meeden, and incorporates code from cddlib written by Komei Fukuda. *rcdd: Computational Geometry*, 2021. URL https://CRAN.R-project.org/package=rcdd. R package version 1.5. [p62]

S. Greenland, J. Pearl, and J. M. Robins. Causal diagrams for epidemiologic research. *Epidemiology*, pages 37–48, 1999. [p53]

T. H. Mattheiss. An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities. *Operations Research*, 21(1):247–260, 1973. [p53, 63]

R. Meleady, P. M. Ueland, H. Blom, A. S. Whitehead, H. Refsum, L. E. Daly, S. E. Vollset, C. Donohue, B. Giesendorf, I. M. Graham, A. Ulvik, Y. Zhang, and A.-L. Bjorke Monsen. Thermolabile methylenetetrahydrofolate reductase, homocysteine, and cardiovascular disease risk: the European Concerted Action Project. *The American journal of clinical nutrition*, 77(1):63–70, Jan. 2003. ISSN 0002-9165. doi: 10.1093/ajcn/77.1.63. Place: United States. [p65]

T. Palmer, R. Ramsahai, V. Didelez, and N. Sheehan. *bpbounds: R package implementing Balke-Pearl bounds for the average causal effect*, 2018. URL https://github.com/remlapmot/bpbounds. [p54]

T. M. Palmer. Nonparametric bounds for the causal effect in a binary instrumental-variable model. *Stata Journal*, 11(3):345–367(23), 2011. URL https://journals.sagepub.com/doi/pdf/10.1177/1536867X1101100302. [p65]

J. Pearl. Direct and indirect effects. In *Proceedings of the Seventeenth Conference on Uncertainty and Artificial Intelligence, 2001*, pages 411–420. Morgan Kaufman, 2001. [p61]

J. Pearl. *Causality*. Cambridge university press, 2009. [p53]

T. S. Richardson and J. M. Robins. Ace bounds; sems with equilibrium conditions. *Statistical Science*, 29(3):363–366, 2014. [p66]

M. C. Sachs, G. Jonzon, A. Sjölander, and E. E. Gabriel. A general method for deriving tight symbolic bounds on causal effects. *Journal of Computational and Graphical Statistics*, 0(0):1–10, 2022. doi: 10.1080/10618600.2022.2071905. URL https://doi.org/10.1080/10618600.2022.2071905. [p53, 54, 58, 60, 61, 62, 66]

M. C. Sachs, G. Jonzon, A. Sjölander, and E. E. Gabriel. *causaloptim: An Interface to Specify Causal Graphs and Compute Bounds on Causal Effects*, 2023. URL https://github.com/sachsmc/causaloptim. R package version 0.9.8. [p53]

A. Sjölander. Bounds on natural direct effects in the presence of confounded intermediate variables. *Statistics in Medicine*, 28(4):558–571, 2009. [p53, 61]

A. Sjölander, W. Lee, H. Källberg, and Y. Pawitan. Bounds on causal interactions for binary outcomes. *Biometrics*, 70(3):500–505, 2014. [p53]

A. Sjölander. Bounds on natural direct effects in the presence of confounded intermediate variables. *Statistics in Medicine*, 28(4):558–571, 2009. doi: https://doi.org/10.1002/sim.3493. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.3493. [p63, 64, 65]

Gustav Jonzon
Department of Medical Epidemiology and Biostatistics, Karolinska Institutet, Sweden

https://ki.se/meb
gustav.jonzon@ki.se

Michael C Sachs
Department of Public Health, University of Copenhagen, Denmark

Supported in part by Novo Nordisk Fonden Grant NNF22OC0076595

https://biostat.ku.dk/
ORCiD: 0000-0002-1279-8676
michael.sachs@sund.ku.dk

Erin E Gabriel
Department of Public Health, University of Copenhagen, Denmark

Supported in part by Novo Nordisk Fonden Grant NNF22OC0076595

https://biostat.ku.dk/
ORCiD: 0000-0002-0504-8404
erin.gabriel@sund.ku.dk