Py-Tetrad and RPy-Tetrad: A New Python Interface with R Support for Tetrad Causal Search

Joseph D. Ramsey JDRAMSEY@ANDREW.CMU.EDU
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA

Bryan Andrews ANDR1017@UMN.EDU
Department of Psychiatry & Behavioral Sciences, University of Minnesota, Minneapolis, MN

Abstract

We give novel Python and R interfaces for the (Java) Tetrad project for causal modeling, search, and estimation. The Tetrad project is a mainstay in the literature, having been under consistent development for over 30 years. Some of its algorithms are now classics, like PC and FCI; others are recent developments. It is increasingly the case, however, that researchers need to access the underlying Java code from Python or R. Existing methods for doing this are inadequate. We provide new, up-to-date methods using the JPype Python-Java interface and the Reticulate Python-R interface, directly solving these issues. With the addition of some simple tools and the provision of working examples for both Python and R, using JPype and Reticulate to interface Python and R with Tetrad is straightforward and intuitive.

Keywords: Tetrad, Python, R, JPype, Reticulate, causal search

1. Introduction

Tetrad (Ramsey et al., 2018)[1] is a well-established package for causal modeling, search, and estimation that has been continuously developed since the early 1990s. Not only is it the source for several now-classic algorithms, such as the celebrated Peter/Clark (PC) (Spirtes et al., 2000) and Fast Causal Inference (FCI) (Spirtes et al., 2000) algorithms, but it contains implementations of more recent state-of-the-art algorithms as well. Algorithms in Tetrad include, but are not limited to:

- Fast Greedy Equivalence Search (FGES) (Ramsey et al., 2017), an implementation of the Greedy Equivalent Search (GES) (Chickering, 2002) algorithm;
- Build Pure Clusters (BPC) (Silva et al., 2006), an algorithm for searching over latent variable structures using tests of variable tetrads;
- Greedy Relaxations of Sparsest Permutation (GRaSP) (Lam et al., 2022), a permutation-based algorithm that provides high-accuracy output for dense graphs.

However, it is increasingly the case that users need to access the underlying code of Tetrad. This can present an obstacle to those unfamiliar with Java since most data scientists now primarily use Python (VanRossum and Drake, 2010) or R (R Core Team, 2023). Python is beneficial for machine learning and is also increasingly used in the sciences. At the same

1. https://github.com/cmu-phil/tetrad

time, R has been a mainstay of scientific and statistical research for many years. Python and R are scripting languages that can mock up ideas quickly, wrangle data, generate plots, do statistical analyses, etc. In addition, Python has become a go-to language for scientific algorithmic development, so a great deal of software has become readily available for general use in Python that is not readily accessible from Java.

The Python project, JPype (Nelson and Scherer, 2020), provides a ready solution for accessing the underlying code of Tetrad from Python; it allows easy access, using Python-native syntax, to any Java class, method, or field, so we have created a package, py-tetrad,[2] to show how to use JPype to interact with Tetrad Java code. Py-tetrad provides three things. First, it provides simple tools for translating datasets and graphs between Python and Tetrad. Second, it provides a class, TetradSearch, that handles everyday operations without needing to resort to JPype programming explicitly. And third, it provides numerous examples of using TetradSearch and JPype to interface Python with Tetrad.

To access Tetrad from R, we take advantage of the Reticulate R package,[3] which provides indirect access to Tetrad through py-tetrad. We call the resulting project rpy-tetrad; it is located in a subdirectory of the py-tetrad project.[4] R through Reticulate can share data and graphs directly with py-tetrad, and py-tetrad can then translate them to Java. Routing the connection from R to Java through Python has the advantage of fewer "moving parts"; updates to py-tetrad are immediately accessible to R, so the projects do not need to be updated separately. As with py-tetrad, rpy-tetrad provides numerous examples.

Installation instructions for py-tetrad and rpy-tetrad may be found in the ReadMe files on their respective GitHub pages, and example files for use may also be found in these directories on GitHub. These instructions will be kept up to date if changes to Java, Tetrad, or Python require it. The example files in py-tetrad and rpy-tetrad will also be kept up to date as changes to Java, Tetrad, or Python happen.

It is worth noting that the idea of interfacing Python and R with Tetrad is not new; previous work had produced the packages py-causal (for Python) and r-causal (for R) using the Causal Command tool[5] (see the comparison in Table 1). For those using py-causal or r-causal, we recommend transitioning to py-tetrad and rpy-tetrad for two reasons. First, the Java versions used in py-causal and r-causal are now outdated and no longer receive updates. Using py-tetrad and rpy-tetrad, users can benefit from the significant improvements made to Tetrad in recent years. Second, JPype enables access to the entire Tetrad codebase from Python, not just a select portion. While this may not be essential for users interested in specific methods already supported by py-causal or r-causal, it dramatically facilitates exploring aspects of Tetrad that are not directly supported.[6]

2. https://github.com/cmu-phil/py-tetrad
3. https://rstudio.github.io/reticulate/
4. https://github.com/cmu-phil/py-tetrad/tree/main/pytetrad/R
5. https://github.com/bd2kccd/causal-cmd
6. We have had many complaints about the use of javabridge for py-causal, especially on Macs, where it can be almost impossible to install. We have also had complaints about the use of rjava for r-causal in its handling of data transfers. JPype, so far, appears to be bug-free and is easy to install, which is yet another reason why we switched to it.

	py-causal	r-causal	py-tetrad	rpy-tetrad
Tetrad Version	6.8.0	6.8.0	7.5.0+	7.5.0+
Last Developed	2017	2017	2023+	2023+
Tetrad Access	javabridge	rjava	JPype	JPype
Python Version	2.7+	-	3.5+	3.5+
R Version	-	3.3.0+	-	4+

Table 1: Comparison of py-causal, r-causal, py-tetrad, rpy-tetrad

2. Prepackaged Tools

Prepackaged methods are provided in the 'tools' directory of the project[7] to translate datasets between Tetrad and Python and to translate graphs from Tetrad to Python. A class, TetradSearch,[8] is provided to give access to Tetrad without using JPype calls; this class encapsulates a wide swath of Tetrad functionality and is accessible from R. These are not necessarily the only tools that will eventually be provided, but they are helpful for a wide variety of tasks.

The dataset translators serve two purposes. First, they provide translations of data between Tetrad and Python in a way that makes them useful for most purposes. They are simple, fast, and effective; even large datasets are translated quickly. They can translate data that is continuous, discrete, or a mixture of continuous and discrete columns. From Python to Tetrad, discrete columns are detected by the column type; these column types can be set directly in the data frame if the Python data loader does not already set them to the appropriate type. Second, Python has an abundance of tools for data preprocessing and wrangling that researchers can now directly incorporate into their pipelines.

Several graph translation methods from Tetrad to Python are provided to handle a variety of purposes. Users can, of course, write their own graph translators for special purposes if the provided formats are inadequate. One can do this by wrapping one of these methods and extending its functionality.

A graph can be retrieved from Tetrad in the following formats:

1. As a GeneralGraph object in causal-learn. The causal-learn package[9] supports a graph object in Python compatible with Tetrad's EdgeListGraph; the translation is direct, and the returned graph can be manipulated directly in Python.

2. As an edge matrix in PCALG (Kalisch et al., 2012). This is an edge matrix in which tail, arrow, star, circle, and null (no endpoint) endpoints are represented as distinct integers, which is compatible with Tetrad's EdgeListGraph.

3. As a simple DOT string. This is the form of a graph used by Graphviz (Ellson et al., 2002) to plot graphs. Graphviz is a sophisticated graph plotting tool with many options; our DOT format gives a basic DOT output. For more nuanced DOT

7. https://github.com/cmu-phil/py-tetrad/tree/main/pytetrad/tools
8. https://github.com/cmu-phil/py-tetrad/blob/main/pytetrad/tools/TetradSearch.py
9. https://causal-learn.readthedocs.io/en/latest/index.html

outputs, the user can implement a similar method in Python (as we do for our Python example below).

4. As a model specification for the lavaan (Rosseel et al., 2017) R package. Any directed and acyclic graph can be saved in this format.

If one uses JPype directly, the entire codebase of Tetrad is at one's fingertips; code examples are given in the py-tetrad project to show how to do this, but we will give an example below to show how powerful this can be. Of course, when accessing the Tetrad codebase, it helps to know the various classes, methods, and fields that one can use in one's JPype code. For this, it helps to have the documentation for the Tetrad package available for reference. This is given in the usual Java format as a set of Javadocs,[10] which can be accessed online, though one should check for updates;[11] the current Tetrad version for these is 7.4.0, though these are updated as revisions to Tetrad are made.

It should be noted as well that JPype allows for Java interfaces to be implemented in Python, so it is possible to define tests or scores in Python for use in Java search methods. An example of this is provided in the py-tetrad package.[12]

3. A Code Example in Python

A typical py-tetrad workflow starts by loading a dataset, wrangling the data, and then optionally plotting histograms and scatter plots. Python provides robust tools for these tasks, so there is no need to rely on Tetrad for these steps. One then converts the data into a Tetrad dataset object before passing it to a Tetrad search procedure. The procedure returns a graph in Tetrad format, which can optionally be converted back to Python for further processing, such as calculating statistics in a simulation study or aggregating multiple runs in a bootstrapped study; examples of both are available in py-tetrad.

We give an example of a bootstrapped study below. Tetrad has built-in bootstrapping facilities, which we will showcase in our subsequent R example for comparison, but we will do the bootstrapping in Python to show how it can be done. The bootstrapped study uses the Apple Watch Fitbit data (Fuller et al., 2020), which is available in easily loaded format at the indicated location.[13] This code example is included in the py-tetrad repository.[14]

First, we load the data and make a knowledge object. This example has a mixture of continuous and discrete columns, so we need to correctly set the column types for our variables so they will be translated to Tetrad properly. We also use Tetrad's knowledge facility to put variables into "knowledge tiers" to ensure that variables in later tiers cannot cause variables in earlier tiers. We use the Degenerate Gaussian mixed-type score (Andrews et al., 2019) with the SP-FCI algorithm (see Raskutti and Uhler (2018) and Ogarrio et al. (2016)).[15]

10. The version of Javadocs at the time of writing of this report is at https://www.phil.cmu.edu/tetrad-javadocs/7.4.0/
11. https://github.com/cmu-phil/py-tetrad
12. https://github.com/cmu-phil/py-tetrad/blob/main/pytetrad/general_scoring_example.py.
13. https://github.com/cmu-phil/example-causal-datasets/tree/main/real/apple-watch-fitbit
14. https://github.com/cmu-phil/py-tetrad/blob/main/pytetrad/jpype_example.py
15. The SP algorithm, since it looks at all permutations of the variables, is exponential and so scales in Tetrad comfortably only to about 11 variables (we have 10 here). Still, the implementation in Tetrad

```
import pandas as pd
import graphviz as gviz

import jpype
import jpype.imports
jpype.startJVM(classpath=[f"resources/tetrad-gui-current-launch.jar"])

import pytetrad.tools.translate as ptt
import pytetrad.tools.visualize as ptv
import edu.cmu.tetrad.search as ts
import edu.cmu.tetrad.data as td


tiers = [['age', 'gender', 'height', 'weight', 'resting_heart', 'device', 'activity'],
         ['steps', 'heart_rate', 'calories', 'distance']]

df = pd.read_csv("resources/aw-fb-pruned18.data.mixed.numeric.txt", sep="\t")
df = df[tiers[0] + tiers[1]]
df = df.astype({col: int for col in ["gender", "device", "activity"]})

knowledge = td.Knowledge()
knowledge.setTierForbiddenWithin(0, True)
for col in tiers[0]: knowledge.addToTier(0, col)
for col in tiers[1]: knowledge.addToTier(1, col)
```

Now we bootstrap and record the results. Note that since some columns are discrete and some continuous, we need to use a score suitable for mixed data types. Finally, we visualize and print the results. The bootstrapped results are shown in Table 2; the graph plot is shown in Figure 1.

```
reps = 10
graphs = []
for rep in range(reps):
    data = ptt.pandas_data_to_tetrad(df.sample(frac=1, replace=True))

    score = ts.score.DegenerateGaussianScore(data, True)
    score.setPenaltyDiscount(2)
    test = ts.test.ScoreIndTest(score, data)

    alg = ts.SpFci(test, score)
    alg.setKnowledge(knowledge)
    graphs.append(alg.search())
```

allows up to 11 variables per knowledge tier, so we are able to extend its functionality to more variables. SP is being substituted in SP-FCI for FGES in the GFCI algorithm.

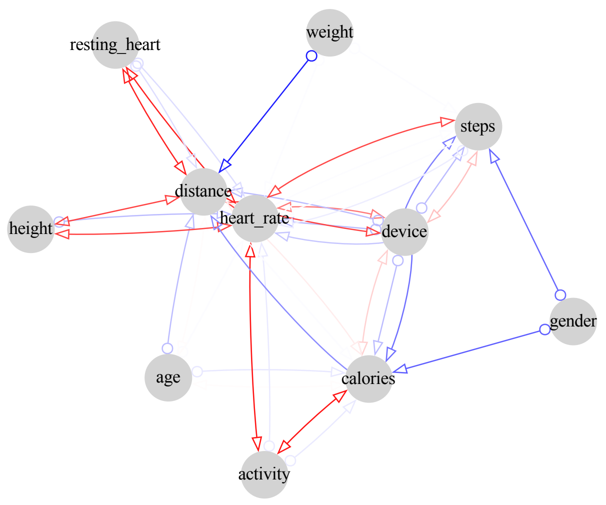

Figure 1: Graph for the JPype example in the text for "Apple Watch Fitbit."

```
probs = ptv.graphs_to_probs(graphs)
graph_attr = {"viewport": "600", "outputorder": "edgesfirst"}
gdot = gviz.Graph(format="pdf", engine="neato", graph_attr=graph_attr)

gdot = ptv.write_gdot(gdot, probs, length=2)
gdot.render(filename="apple_fitbit", cleanup=True, quiet=True)
gdot.clear()
```

4. A Code Example in R

We provide support for R via py-tetrad and the R project, Reticulate, in a project we call "rpy-tetrad," located in the 'R' subdirectory of the py-tetrad project. Instructions for setting up the project to run in R or RStudio; these instructions, and rpy-tetrad itself, have been tested and shown to work on Mac, Linux, and Windows. Note that the interface in R to Tetrad is limited to what is available in the TetradSearch class in py-tetrad, so if more methods are needed in R, more methods must be added to the py-tetrad class so R can access them; this is straightforward. One cannot use JPype commands directly from R; these must be routed through Python.

We give an example in this section; more examples are available in the 'R' subdirectory of py-tetrad. We use data from a NASA Airfoil Self-Noise experiment (Brooks et al., 2014) with six variables. In this example, the data loader in R loads the data as a mixture of continuous and discrete (integer) variables, which the converter in py-tetrad will translate as a mixture of continuous data in Tetrad. To enforce the constraint that all variables are interpreted as continuous, we need first to tell R that the columns in this data frame are

Adjacency	$\leftrightarrow$	$\circ\!\!\rightarrow$	$\rightarrow$	$\leftarrow\!\!\circ$	$\leftarrow$
(activity, calories)	0.93	0.07	-	-	-
(activity, heart rate)	0.93	0.07	-	-	-
(calories, device)	0.22	-	-	0.30	0.48
(device, distance)	0.70	0.30	-	-	-
(device, heart rate)	-	0.45	-	0.30	0.25
(device, steps)	-	0.19	0.51	0.30	-
(distance, resting heart)	0.80	-	-	0.20	-
(heart rate, height)	0.71	-	-	0.29	-
(heart rate, resting heart)	0.80	-	-	0.20	-
(distance, weight)	-	-	-	0.86	-
(distance, height)	0.71	-	-	0.03	-
(heart rate, steps)	-	0.78	-	0.02	0.01
(gender, steps)	-	0.72	-	-	-
(calories, gender)	-	-	-	0.71	-
(calories, distance)	0.06	-	0.47	-	-
(age, distance)	-	0.29	-	-	-
(age, calories)	-	0.05	-	-	-
(age, heart rate)	-	0.01	-	-	-
(heart rate, weight)	-	-	-	0.01	-

Table 2: Results of a 100-fold bootstrapping of the Apple Watch Fitbit data, as described in the text. Frequencies for each edge type encountered over 100 folds are given.

all to be interpreted as 'numeric'; a line in the code does this. Then a test defined over continuous variables ('Fisher Z') and a score defined over continuous variables ("SEM BIC") can be used to run the search algorithm. TetradSearch will report if the search, test, or score is for one data type but another data type is given.

As in the Python example, we here appeal to background knowledge. We use the feature in the TetradSearch class that allows us to specify background knowledge as 'temporal tiers,' where variables in later tiers are known not to cause variables in earlier tiers. For this example, we know that 'pressure' (sound pressure) in this NASA experiment cannot cause any other experimental or airfoil design variables, so we put it in a later tier. (One may also forbid or require particular edges as part of background knowledge.) If we use such background knowledge in a search, we expect the algorithm to honor it. Not all algorithms in Tetrad are designed to honor background knowledge; if one chooses an algorithm that does not and provides background knowledge, an error will be reported. Examples of algorithms that are not currently designed to honor background knowledge include ICA LiNGAM, Direct LiNGAM, and CCD, though this may change.

This example is written to work in RStudio, so that a histogram/scatterplot plot matrix (Figure 2) is displayed in the Plots window, and the plot of the graph returned by the

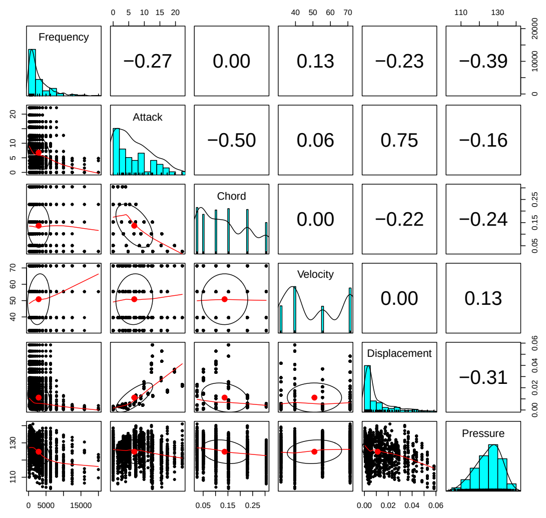

Figure 2: Plot matrix for the R code "NASA Airfoil Self-Noise" example; this is displayed in the Plots window in RStudio. The psych package in R produced this plot matrix.

algorithm is displayed in the Viewer window (e.g., Figure 3). We have included this code example in the py-tetrad repository.[16]

Tetrad also has a bootstrapping facility that TetradSearch makes available if one prefers to use it. (As in the Python example, the user may prefer to write their own bootstrapping code.) If automatic bootstrapping is used, a graph is plotted that shows frequencies of occurrence of each edge observed in any bootstrapping models. We show bootstrapping results with 30 folds (Figure 4).[17] We use here the GRaSP algorithm, a permutation algorithm that relaxes the faithfulness assumption, with the linear Gaussian BIC score.

```
setwd("~/py-tetrad/pytetrad")
library(reticulate)

data <- read.table("./resources/airfoil-self-noise.continuous.txt",
                   header=TRUE)
i <- c(1, 6)
data[ , i] <- apply(data[ , i], 2, function(x) as.numeric(x))

library(psych)
pairs.panels(data, method = "pearson")
```

16. https://github.com/cmu-phil/py-tetrad/blob/main/pytetrad/R/sample_r_code7.R
17. It should be noted that the TetradSearch class can also be used in Python as well for those who don't wish to make JPype calls themselves. Examples of this are provided in the py-tetrad package.

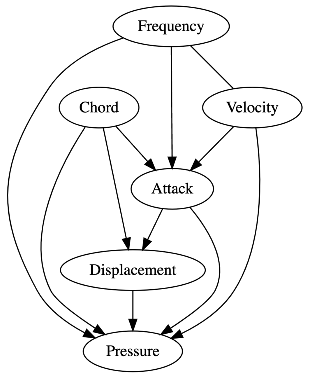

Figure 3: Plot of the graph for the R code ”NASA Airfoil Self-Noise” example; this is displayed in the Viewer window in RStudio. This plot is produced by Graphviz using the DiagrammaR package in R.

```
source_python("tools/TetradSearch.py")
ts <- TetradSearch(data)

ts$add_to_tier(1, "Attack")
ts$add_to_tier(1, "Chord")
ts$add_to_tier(1, "Velocity")
ts$add_to_tier(1, "Displacement")
ts$add_to_tier(1, "Frequency")
ts$add_to_tier(2, "Pressure")

ts$use_sem_bic(penalty_discount=2)
ts$use_fisher_z()
ts$run_grasp()

print(ts$get_string())

library(DiagrammeR)
dot <- ts$get_dot()
grViz(dot)

ts$set_bootstrapping(numberResampling = 30)
ts$run_grasp()
dot <- ts$get_dot()
grViz(dot)
```

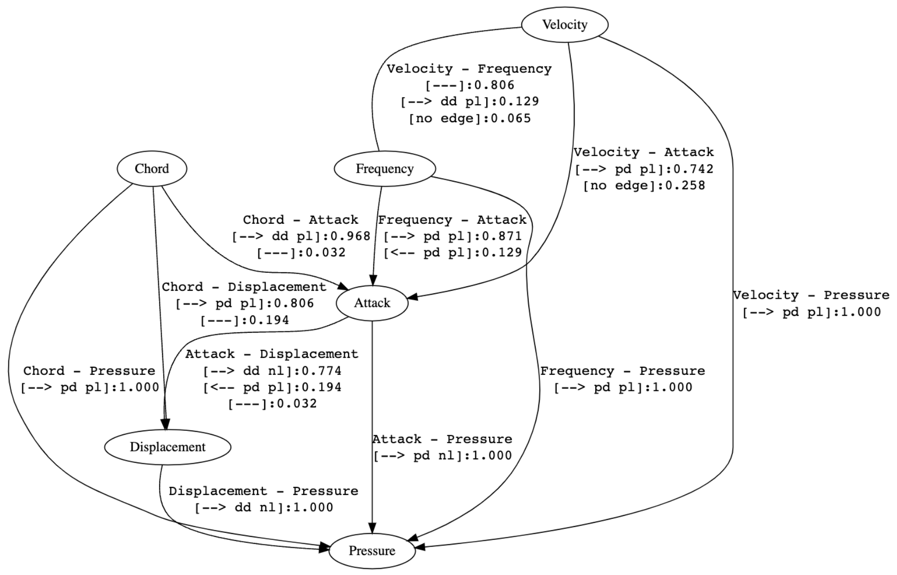

Figure 4: Plot of the 30-fold bootstrapping graph for the R code "NASA Airfoil Self-Noise" example displayed in the Viewer window in RStudio. This plot is produced by Graphviz using the DiagrammaR package in R.

5. Conclusion

In Python, the JPype package allows access to arbitrary code in Tetrad. In both Tetrad and R, the TetradSearch class allows users to access Tetrad's most commonly used functionality without directly using JPype. If one wishes to do JPype programming, the entire codebase of Tetrad becomes available for Python scripts. These Python or R scripts can be published, shared, and reused. Despite its performance advantages over Python in many areas, Java is not a good scripting language, so publishing Java classes as scripts is a bit forced. Also, accessing other languages from Java is tricky, whereas accessing Java from Python with JPype is not, as we have shown. So py-tetrad scripts can easily take cognizance of the entire functionality available in Python, including functionality from Java.

One known issue is that the installation process can be a bit cumbersome. We hope to simplify this process so users can install py-tetrad using pip and rpy-tetrad using CRAN. Also, initial responses to these tools have been overwhelmingly positive, but feedback is welcome for issues encountered in py-tetrad or rpy-tetrad. Installing Grasphviz in Python is also challenging but must be left as an exercise for the reader. Suggestions for new features to include in py-tetrad, rpy-tetrad, or Tetrad are welcome.[18]

6. Citations and Bibliography

Acknowledgments

We thank our anonymous reviewers for their detailed comments. We thank Yasuhiro Shimodaira and Kelvin Lim for their feedback, especially with our R implementation, and for ensuring our projects run well on Windows. We also thank Peter Spirtes for encouraging this project and setting the goal of breaking down barriers between projects. We especially thank the authors of JPype and Reticulate for their excellent tools. Ramsey's work on this project was partly funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Andrews' work on this project was funded and supported by the Comorbidity: Substance Use Disorders and Other Psychiatric Conditions Training Program T32DA037183.

References

Bryan Andrews, Joseph Ramsey, and Gregory F Cooper. Learning high-dimensional directed acyclic graphs with mixed data-types. In *The 2019 ACM SIGKDD Workshop on Causal Discovery*, pages 4–21. PMLR, 2019.

Thomas F Brooks, D Stuart Pope, and Michael A Marcolini. Airfoil Self-Noise. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5VW2C.

David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*, pages 483–484. Springer, 2002.

Daniel Fuller, Javad Rahimipour Anaraki, Bo Simango, Faramarz Dorani, Arastoo Bozorgi, Hui Luan, and Fabien Basset. Using machine learning methods to predict physical activity types with apple watch and fitbit data using indirect calorimetry as the criterion. 2020.

Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H Maathuis, and Peter Bühlmann. Causal inference using graphical models with the r package pcalg. *Journal of statistical software*, 47:1–26, 2012.

Wai-Yin Lam, Bryan Andrews, and Joseph Ramsey. Greedy relaxations of the sparsest permutation algorithm. In *Uncertainty in Artificial Intelligence*, pages 1052–1062. PMLR, 2022.

18. Bug reports and feature suggestions may be sent to us by email or, preferably, submitted to our GitHub Issue Tracker at https://github.com/cmu-phil/pytetrad/issues.

Karl E Nelson and Martin K Scherer. Jpype. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2020.

Juan Miguel Ogarrio, Peter Spirtes, and Joe Ramsey. A hybrid causal search algorithm for latent variable models. In *Conference on probabilistic graphical models*, pages 368–379. PMLR, 2016.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2023. URL https://www.R-project.org/.

Joseph Ramsey, Madelyn Glymour, Ruben Sanchez-Romero, and Clark Glymour. A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International journal of data science and analytics*, 3:121–129, 2017.

Joseph D Ramsey, Kun Zhang, Madelyn Glymour, Ruben Sanchez Romero, Biwei Huang, Imme Ebert-Uphoff, Savini Samarasinghe, Elizabeth A Barnes, and Clark Glymour. Tetrad—a toolbox for causal discovery. In *8th international workshop on climate informatics*, 2018.

Garvesh Raskutti and Caroline Uhler. Learning directed acyclic graph models based on sparsest permutations. *Stat*, 7(1):e183, 2018.

Yves Rosseel, Daniel Oberski, Jarrett Byrnes, Leonard Vanbrabant, Victoria Savalei, Ed Merkle, Michael Hallquist, Mijke Rhemtulla, Myrsini Katsikatsou, Mariska Barendse, et al. Package 'lavaan'. *Retrieved June*, 17(1):2017, 2017.

Ricardo Silva, Richard Scheines, Clark Glymour, Peter Spirtes, and David Maxwell Chickering. Learning the structure of linear latent variable models. *Journal of Machine Learning Research*, 7(2), 2006.

Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000.

Guido VanRossum and Fred L Drake. *The python language reference*. Python Software Foundation Amsterdam, Netherlands, 2010.