grapl

GRAPL

A computational library for nonparametric structural causal modelling, analysis and inference

Prof Max A. Little, UoB/MIT
maxl@mit.edu

Why we need a structural causal modelling library

- ❑ Intuitive language for arbitrarily complex structural causal models (SCMs)
- ❑ Model import/export
- ❑ Analysis of topological and causal relationships
- ❑ Automated derivation of any complex, factorized, non-parametric distributions: joint, marginal, conditional, interventional
- ❑ Outputs for display publication/CAS
- ❑ Implementation in a widely-used language (Python)

grapl

GRAPL library: selected features

- A simple, text-based domain-specific language (DSL) for DAGs and ADMGs
- Derivation of factorized, marginalized nonparametric distributional models for arbitrary DAGs
- Computation of interventional distributions in arbitrarily complex DAGs/ADMGs
- Various algorithms for analysis of causal influence in DAGs/ADMGs (e.g. c-components/districts, node interventions, local Markov conditional independence relations, topological sorting)
- Latex format output distributions

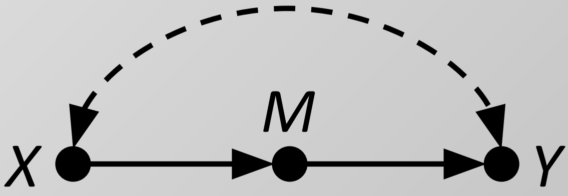
grapl

Example: Simple front-door ADMG

```
>>> import grapl.algorithms as algs
>>> import grapl.dsl as dsl

>>> grapl_obj = dsl.GraplDSL()
>>> dag_grapl = ' "Front door adjustment"; \
>>>   X; Y; M; \
>>>   X -> M; \
>>>   M -> Y; \
>>>   X <-> Y; '
>>> G = grapl_obj.readgrapl(dag_grapl)

>>> id_str, expr, isident = algs.idfixing(G, {'X'}, {'Y'})
>>> if isident:
>>>     print(id_str) # p_{X}(Y)=\sum_{M,X'}[p(Y|M,X')p(M|X)p(X')]
>>> else:
>>>     print('Interventional distribution not identifiable')
```



$$p_X(Y) = \sum_{M,X'} [p(Y|M,X')p(M|X)p(X')]$$

grapl

How GRAPL works: under the hood

- `admg.py: class ADMG()` ADMG graph object and methods for construction, topological analysis, manipulation
- `dsl.py: class GraplDSL()` A DSL language lexer and parser object, implemented using PLY, for describing DAGs/ADMGs
- `algorithms.py`: Core causal inference algorithms: Richardson fixing, DAG factorization, truncated factorization, local Markov independences, Tian factorization
- `expr.py: class Expr()` Non-parametric distribution expression object and methods for adding and substituting variables, fixing and marginal fixing, cancelling common sub-expressions, marginalizing, simplifying and converting to Latex strings

Selected GRAPL functions

`algorithms.truncfactor`

Truncated factorization ("g-formula") for DAGs.

Parameters:

G (*ADMG*) – DAG object representing the causal graph (must not have bidirects)
X (*Set* of *Strings*) – Interventional variables where each string is a random variable name (must not be empty)
Y (*Set* of *Strings*) – Effect variables, each string is a random variable name (if empty, all variables in G other than the set X)
prefactor (*Boolean*) – If True, joint distribution is Markov factored before fixing

Returns: (*String*, *Expr*, *Boolean*)

If *G* is a DAG, factored interventional distribution string, corresponding *Expr* object, and True. Otherwise, returns '', None, False.

Selected GRAPL functions

`algorithms.localmarkov`

Compute all local Markov independences for DAGs.

Parameters:

G (*ADMG*) – DAG object representing the causal graph (must not have bidirects)

Returns: (*Set* of *Strings*, *Boolean*)

If *G* is a DAG, set of strings representing Markov independences, True. Otherwise returns empty set, False.

Selected GRAPL methods

`Expr.cancel`

An algorithm for cancelling variables in a distribution expression (Expr object). This seeks to greedily match and remove terms appearing in both numerator and denominator of an expression. Returns True if any changes to the expression occurred as a result, and False otherwise.

`Expr.marginal`

An algorithm for marginalizing out variables in a distribution expression (Expr object). Greedily removes variables appearing in both the numerator and the set of marginal variables. Returns True if any changes to the expression occurred as a result, and False otherwise.

`Expr.simplify`

An algorithm for simplifying a distribution (Expr object), by successive cancellation and marginalization until a fixed point is reached. Returns True if simplifications were possible, and False otherwise.

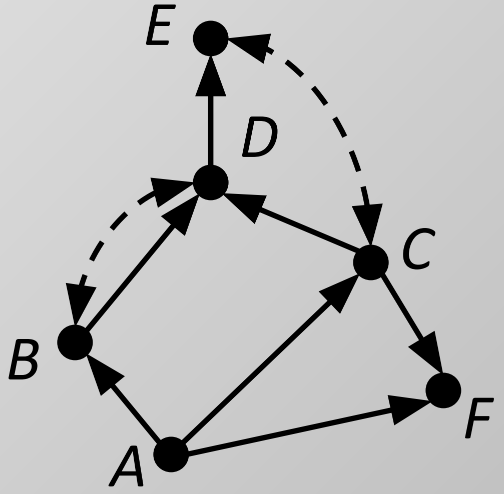
grapl

Example: Complex ADMG

```
>>> import grapl.algorithms as algs
>>> import grapl.dsl as dsl

>>> grapl_obj = dsl.GraplDSL()
>>> G = grapl_obj.readgrapl(open(filename,'r').read())

>>> D = G.districts() # [{'A'},{'F'},{'B','D'},{'E','C'}]
>>> V = G.de({'B','F'}) # {'B','D','E','F'}
>>> dist_str, fac_expr = algs.admgfactor(G)
>>> print(dist_str)
```



$$p(A,B,C,D,E,F) = p(D|A,B,C)p(E|D,A,C)p(F|A)p(B|A)p(A,C)$$

grapl

Coming soon (v1.5)

- ❑ Conditional interventional distributions (c.f. IDC algorithm)
- ❑ Expressions for expectations e.g. ATEs, CATEs based on interventional distributions
- ❑ Auto-generate numerical functions to compute empirical average/median treatment effects, from data
- ❑ Automated derivation of bootstrap weights (c.f. causal bootstrapping)
- ❑ Input DAGs/ADMGs directly from vector graphical drawing software
- ❑ Requests …?

Thank you!

Github: https://github.com/max-little/GRAPL, contributions welcome.

Citation: Little, M. A., (2022). GRAPL: A computational library for nonparametric structural causal modelling, analysis and inference. *Journal of Open Source Software*, **7**(76), 4534, https://doi.org/10.21105/joss.04534

Prof Max A. Little, UoB/MIT
maxl@mit.edu