Adjunct

`Adjunct` (Averaging Decomposable graphs via Junction Trees) is a program for Bayesian learning of decomposable graphs from data. For background on the problem and the algorithmic techniques used by `Adjunct`, see the related papers:

- K. Kangas, M. Koivisto, and T. Niinimäki. Learning chordal Markov networks by dynamic programming. *In Advances in Neural Information Processing Systems 27 (NIPS)*, pages 2357–2365. Curran Associates, Inc., 2014.
- K. Kangas, T. Niinimäki, and M. Koivisto. Averaging of Decomposable Graphs by Dynamic Programming. *In Proceedings of the 31st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015 (to appear).

1 Bayesian learning

A decomposable model is an undirected graphical model where the graph structure is *decomposable* (or *triangulated*, or *chordal*).

Given data on a finite set of variables V, we are interested in learning graph structures that fit the data well. In the Bayesian approach we evaluate a graph by the probability of the data under the graph, the *likelihood*, obtained by integrating out the model parameters.

For a decomposable graph $\mathcal{G}$ on V the likelihood is a *decomposable function*

$$\varphi(\mathcal{G}) = \frac{\prod_{C \in \mathcal{C}(\mathcal{G})} \varphi_c(C)}{\prod_{S \in \mathcal{S}(\mathcal{G})} \varphi_c(S)}\,, \quad \varphi_c : 2^V \to \mathbb{R}\,,$$

where $\mathcal{C}(\mathcal{G})$ denotes the set of *cliques* of $\mathcal{G}$ and $\mathcal{S}(\mathcal{G})$ denotes the multiset of *separators* arising from specific clique intersections. The function is induced by the *local component* function φ_c defined for all subsets of V.

By multiplying the likelihood with a *structure prior* and dividing by a normalizing constant, we obtain a posterior distribution over decomposable graphs. Learning problems include finding graphs that maximize the posterior probability and computing various marginals of the posterior, such as the probability that an individual edge is present in the graph.

The input to `Adjunct` is a decomposable function φ given explicitly as its local component φ_c, in a form a logarithmic "scores". `Adjunct` comes with a program `dmscore` (see section 4) for computing BDeu scores for a given data set.

Given φ, `Adjunct` can perform the following learning tasks:

1. Compute the maximum of φ over all decomposable graphs,

$$\max_{\mathcal{G}} \varphi(\mathcal{G}) \,.$$

   When φ is the likelihood function, this is the task of finding a maximum-a-posteriori graph under the uniform prior over decomposable graphs.

2. Compute the sum of $\varphi\tau$ over all decomposable graphs,

$$\sum_{\mathcal{G}} \varphi(\mathcal{G})\tau(\mathcal{G}) \,,$$

   where $\tau(\mathcal{G})$ is number of rooted junctions trees of $\mathcal{G}$. This gives any marginal of the posterior that can be expressed as a decomposable function, under the uniform prior over rooted junction trees.

3. Draw independent samples from the distribution proportional to $\varphi\tau$.

4. Estimate edge posterior probabilities under the distribution proportional to φ via importance sampling.

2 Building

`Adjunct` compiles as `C++11` and has been tested on the `GCC` and `clang` compilers. The simplest way to compile is to run the `make` program in the subdirectory `adjunct`. You can change the default compiler (`GCC`) by editing the first line in `Makefile`.

Alternatively, you can compile `Adjunct` manually, e.g.,

```
g++ -std=c++11 -O3 -o adjunct
    adjunct.cpp common.cpp maximization.cpp sampling.cpp
    sampling_adaptive.cpp sampling_naive.cpp tools.cpp
```

3 Usage

Running `Adjunct` with no arguments will produce usage instructions:

```
adjunct [-flags] <input file> [<maximum width>]
        [<action [arg ...]>]
```

The only mandatory argument is `input file`, which specifies the input function (see section 4).

The optional `action` argument determines what to do with the scores. If the argument is "`max`" or not given, `Adjunct` will find a graph that maximizes the function. For example, running

```
adjunct -tsh bridges.score
```

will produce

```
==================================== Tree
{0,1}
+--{1,2,4}              {1}
+--{0,9}                {0}
   +--{8,9,10}          {9}
      +--{8,11}         {8}
      |  +--{3,11}      {11}
      |  |  +--{3,5}    {3}
      |  +--{7,11}      {11}
      +--{6,8}          {8}
==================================== Score
-2630.457413
```

The optional `-flags` argument controls what is printed for the resulting graph (see section 5). In this example, a junction tree of the graph and its score are printed.

The `maximum width` argument restricts the maximum size of cliques. For example, running

```
adjunct -t bridges.score 2
```

will produce the highest scoring network with at most two vertices per clique.

Sampling

If the `action` argument is "`sample`", `Adjunct` will first compute the sum tables and then draw independent samples of decomposable graphs. The full arguments are:

```
adjunct [-flags] <input file> [<maximum width>]
        sample [<n> [<seed>]]
```

Here, `n` is the number of samples to draw (1 by default) and "`seed`" is a seed for the random number generator. If no seed is specified, Unix time in seconds is used. For example,

```
adjunct -c asia.score sample 5 0
```

will produce five samples in a compact form:

```
50{49{37{164}}}{24}{96}
164{38{7}{50{24}}}{96}
38{7}{50{24}{96}}{164}
38{7}{50{24}}{164{96}}
50{38{7}{164}}{56}{96}
```

Producing output in this form (e.g. into a file) is convenient since `Adjunct` can parse the form and later display it in other formats.

Reading trees from a file/command line

If the `action` argument is "`file`", `Adjunct` expects an additional file argument, which must contain trees in the compact form. `Adjunct` will read those trees and display them according to the output flags. For example, if the trees of the previous section are in `trees.txt`, running

```
adjunct -s asia.score file trees.txt
```

will produce the scores of the sampled trees:

```
-22483.301209
-22478.557123
-22478.557123
-22478.557123
-22481.641032
```

A single tree argument can also be given directly on the command line with the `action` argument "`tree`". For example,

```
adjunct -s asia.score tree 50{49{37{164}}}{24}{96}
```

will produce

```
-22483.301209
```

Enumerating decomposable graphs

Finally, the `action` argument "`enum`" will perform a bruteforce enumeration of all decomposable graphs and compute the true edge probabilities under the uniform prior over decomposable models. This enumeration is feasible only up to around 8 vertices (a few minutes on a standard desktop computer).

4 Input format and DMscore

`Adjunct` comes with three example input files, `asia.score`, `bridges.score` and `flare.score`. Each input file contains the target function in a form of logarithmic scores corresponding to (unnormalized) probabilities, one score for each subset of the variables.

The first four lines of an input file are

```
DMST
n
subset_scores
colex_order m
```

where n is the number of variables and m is the maximum clique size for which the scores have been computed.

The rest of the input file contains the scores themselves, in the order specified by the fourth line. Currently, the only supported order is the colexicographic order (also known as the binary lexicographic order). An example of the order for $n = 4$, $m = 2$:

```
0000  =  {}
0001  =  {1}
0010  =  {2}
0011  =  {1,2}
0100  =  {3}
0101  =  {1,3}
0110  =  {2,3}
1000  =  {4}
1001  =  {1,4}
1010  =  {2,4}
1100  =  {3,4}
```

Note that the first score line belongs to the empty set and the score should always be equal to 0 (probability 1). In fact, certain optimizations of `Adjunct` rely on this assumption.

Cliques larger than m are treated as having 0 probability and they will not appear in any produced networks (assuming at least ome network has a positive probability).

DMscore

The program `DMscore` produces BDeu (Bayesian Dirichlet equivalent uniform) score files from data sets. The program can be compiled similarly by running

`make` in the directory `dmscore`. To produce score files, run

```
dmscore <datafile> <equivalent sample size>
        [<max clique size>]
```

See `example.dat` for an example of a data file on 8 variables and 200 records. Each record is on a separate line and contains numeric values starting from 0 for each variable, separated by spaces.

Run e.g.

```
dmscore example.dat 1
```

to produce BDeu scores with equivalent sample size 1.

5 Output flags

Output flags control what is printed for each produced graph, i.e., the graphs produced by maximization or sampling, or graphs read from a file or the command line. The output is printed in the same order as the flags are specified.

Supported flags are:

- s: the decomposable score of the graph
- k: the cliques and separators of the graph and their local scores
- t: a textual junction tree representation of the graph
- c: a compact representation of the graph, readable by `Adjunct`
- j: the number of distinct junction trees of the graph
- r: the number of distinct *rooted* junction trees of the graph
- m: the adjacency matrix of the graph
- d: a .dot file of the graph

Additional control flags may appear anywhere among the output flags:

- h: print a header line before each type of output
- v: verbose, print information on the progress of the computation
- e: print estimates of edge probabities after sampling is complete
- n: use naive sampling instead of the default adaptive sampling (this is slower but uses no extra memory)

Default flags are `-ksthv`.