A Guide to the CP-net Generator

Thomas E. Allen

Version 0.70 (December 2015)

1 Introduction

The CP-net Generator (GenCPnet) generates acyclic conditional preference networks (CP-nets) [2] uniformly at random with respect to a specified set. It is possible to specify parameters such as the number of nodes, bound on indegree, and the size of domains.

GenCPnet implements the method described in our paper, "Generating CP-nets Uniformly at Random" [1]. If you use or adapt this software, we kindly ask that you would cite our paper. GenCPnet is free software, released under the GNU Public License version 3. GenCPNet is written in C++ and designed to run on a GNU Linux system. Throughout this manual, we assume the reader has access to such a system and is familiar with using the command line to perform simple instructions. Please email questions or bug reports to thomas.allen@uky.edu.

2 Obtaining GenCPnet

Presently the software and documentation is housed at

`http://cs.uky.edu/~goldsmit/papers/GeneratingCPnetCode.html`.

The code can be downloaded as a zipped archive via a link on that page. Alternatively, you can download the archive directly with:

```
wget http://cs.uky.edu/~goldsmit/papers/gencpnet_0.70.zip
```

or

```
wget http://cs.uky.edu/~teal223/gencpnet_0.70.zip
```

3 Installing GenCPnet

The GNU MultiPrecision (GMP) library must be installed to compile or run GenCPnet. If GMP is not present, you can install it with your distribution's package manager. For more information on GMP, see `https://gmplib.org/`.

Once GMP is present, it should be straightforward to build GenCPnet on a GNU Linux system:

```
unzip gencpnet_0.70.zip
cd gencpnet_0.70
make
```

If the build is successful, the message `Success!` will appear on the last line of the output. It should then be possible to run GenCPNet directly within the build directory. For example, try:

```
./gencpnet --help
```

You may also wish to copy the executable file `gencpnet` to a suitable directory in your path. For example,

```
make install
```

copies the executable file to your `~/bin` directory by default. If you wish to make the program available to all users on the system, you can do so with a command such as:

```
sudo cp gencpnet /usr/local/bin
```

You may need assistance from your system administrator with these latter steps. Finally, you may wish to edit `Makefile` itself. The comments in the file provide additional installation and customization details.

4 Using the CP-net Generator

Typing `./gencpnet --version` from within the `gencpnet_0.70` build directory shows the current version:

```
$ ./gencpnet --version
CP-net Generator 0.70
Copyright (C) 2015 Thomas E. Allen
This is free software; see the source for copying conditions.
CP-net Generator comes with ABSOLUTELY NO WARRANTY.
```

If you are working in a different directory and the GenCPnet executable file is installed to a directory in your path, you should omit the `./` and simply type `gencpnet` in all of the examples in this guide.

```
$ gencpnet --version
```

Note that the `$` in these examples is the command line prompt and should not be typed.

4.1 The `-n` parameter: a simple binary-valued example

We begin with a couple of very simple examples. First, we show how to generate a single CP-net with 3 nodes and binary domains:

```
$ mkdir examples
$ ./gencpnet -n 3 examples
Building distribution tables for CP-nets with the following specs:
Number of nodes: n = 3
Bound on in-degree c = 2
Homogeneous domains of size d = 2
Probability of incompleteness i = 0
Generating 1 random CP-nets with these specs.
Generation complete.
```

Executing GenCPnet with the `-n` option specifies that the CP-net thus generated should be sampled from CP-nets with $n = 3$ nodes. By default, the CP-nets in the set have unbounded indegree (a node may have 0, 1, or 2 parents), and the domains are binary. In this case the generated example is written in XML format to the `examples` directory:

```
$ ls -l examples
total 4
-rw-rw-r-- 1 user user 1932 Dec 15 14:49 cpnet_n3c2d2_0000.xml
```

The `cpnet_n3c2d2_0000.xml` filename describes the parameters of the generated CP-net instance:

- `n3` indicates that the CP-net has $n = 3$ nodes;
- `c2` indicates that a node can at most $c = 2$ parents;
- `d2` indicates that the domains are all of size $d = 2$.
- `_0000` is an index number. If we had generated multiple CP-nets using the `-g` option described below, then these would be numbered sequentially from 0000.

Note that if we attempt to perform the identical command a second time, an error message would result:

```
Error: filename cpnet_n3c2d2_0000.xml already exists.
Delete file(s) first or output to another directory.
```

By throwing the error message, we avoid the unfortunate possibility of overwriting the sets that we have previously generated, since these may be needed for later experiments or in case later researchers wish to confirm our results using the same input.

The format of the XML file is described in detail at the site `http://www.ece.iastate.edu/~gsanthan/crisner.html`. Here we only describe it briefly. Note that since our algorithm is randomized, the output will likely differ on each execution.

```
<PREFERENCE-SPECIFICATION>

<PREFERENCE-VARIABLE>
 <VARIABLE-NAME>x1</VARIABLE-NAME>
 <DOMAIN-VALUE>1</DOMAIN-VALUE>
 <DOMAIN-VALUE>2</DOMAIN-VALUE>
</PREFERENCE-VARIABLE>

<PREFERENCE-VARIABLE>
  <VARIABLE-NAME>x2</VARIABLE-NAME>
  <DOMAIN-VALUE>1</DOMAIN-VALUE>
  <DOMAIN-VALUE>2</DOMAIN-VALUE>
 </PREFERENCE-VARIABLE>

 <PREFERENCE-VARIABLE>
  <VARIABLE-NAME>x3</VARIABLE-NAME>
  <DOMAIN-VALUE>1</DOMAIN-VALUE>
  <DOMAIN-VALUE>2</DOMAIN-VALUE>
 </PREFERENCE-VARIABLE>

 <PREFERENCE-STATEMENT>
   <STATEMENT-ID>p1_1</STATEMENT-ID>
   <PREFERENCE-VARIABLE>x1</PREFERENCE-VARIABLE>
   <PREFERENCE>2:1</PREFERENCE>
 </PREFERENCE-STATEMENT>

 <PREFERENCE-STATEMENT>
   <STATEMENT-ID>p2_1</STATEMENT-ID>
   <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
   <CONDITION>x1=1</CONDITION>
   <CONDITION>x3=1</CONDITION>
   <PREFERENCE>1:2</PREFERENCE>
 </PREFERENCE-STATEMENT>

 <PREFERENCE-STATEMENT>
   <STATEMENT-ID>p2_2</STATEMENT-ID>
```

```
37    <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
38    <CONDITION>x1=1</CONDITION>
39    <CONDITION>x3=2</CONDITION>
40    <PREFERENCE>1:2</PREFERENCE>
41  </PREFERENCE-STATEMENT>
42
43  <PREFERENCE-STATEMENT>
44    <STATEMENT-ID>p2_3</STATEMENT-ID>
45    <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
46    <CONDITION>x1=2</CONDITION>
47    <CONDITION>x3=1</CONDITION>
48    <PREFERENCE>2:1</PREFERENCE>
49  </PREFERENCE-STATEMENT>
50
51  <PREFERENCE-STATEMENT>
52    <STATEMENT-ID>p2_4</STATEMENT-ID>
53    <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
54    <CONDITION>x1=2</CONDITION>
55    <CONDITION>x3=2</CONDITION>
56    <PREFERENCE>1:2</PREFERENCE>
57  </PREFERENCE-STATEMENT>
58
59  <PREFERENCE-STATEMENT>
60    <STATEMENT-ID>p3_1</STATEMENT-ID>
61    <PREFERENCE-VARIABLE>x3</PREFERENCE-VARIABLE>
62    <CONDITION>x1=1</CONDITION>
63    <PREFERENCE>1:2</PREFERENCE>
64  </PREFERENCE-STATEMENT>
65
66  <PREFERENCE-STATEMENT>
67    <STATEMENT-ID>p3_2</STATEMENT-ID>
68    <PREFERENCE-VARIABLE>x3</PREFERENCE-VARIABLE>
69    <CONDITION>x1=2</CONDITION>
70    <PREFERENCE>2:1</PREFERENCE>
71  </PREFERENCE-STATEMENT>
72
73  </PREFERENCE-SPECIFICATION>
```

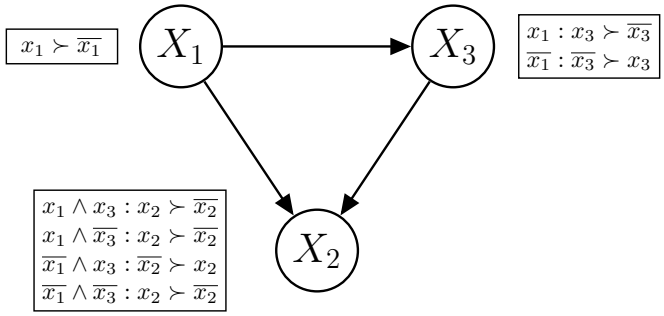
The PREFERENCE-VARIABLE enclosures tell us that we have three variables, X_1, X_2, and X_3, each with the values 1 and 2. These can map to $\mathrm{Dom}(X_1) = \{x_1, \overline{x_1}\}$, etc. The preference statement in lines 21–25

```
<PREFERENCE-STATEMENT>
  <STATEMENT-ID>p1_1</STATEMENT-ID>
  <PREFERENCE-VARIABLE>x1</PREFERENCE-VARIABLE>
  <PREFERENCE>2:1</PREFERENCE>
</PREFERENCE-STATEMENT>
```

tells us that X_1 has no dependencies (parents), and that outcomes with $X_1 = 2$ are always preferred to those with $X_1 = 1$. The preference statement in lines 66–71

```
<PREFERENCE-STATEMENT>
  <STATEMENT-ID>p3_2</STATEMENT-ID>
  <PREFERENCE-VARIABLE>x3</PREFERENCE-VARIABLE>
  <CONDITION>x1=2</CONDITION>
  <PREFERENCE>2:1</PREFERENCE>
</PREFERENCE-STATEMENT>
```

tells us that X_1 is a parent of X_3 and in particular that, when $X_1 = 2$, outcomes with $X_3 = 2$ are preferred to those with $X_3 = 1$. Altogether, the XML specification corresponds to the following CP-net:



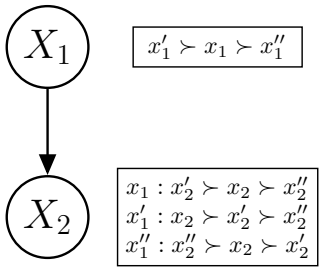
4.2 The `-d` parameter: a two-node multi-valued example

It is possible to generate CP-nets with multivalued domains with the `-d` option. Note that our method assumes domain sizes are *homogeneous*, that is, the same for all variables. For example, we can generate an example with 2 nodes ($n = 2$) and three-valued domains ($d = 3$) with:

```
$ ./gencpnet -n 2 -d 3 examples
Building distribution tables for CP-nets with the following specs:
Number of nodes: n = 2
Bound on in-degree c = 1
Homogeneous domains of size d = 3
Probability of incompleteness i = 0
Generating 1 random CP-nets with these specs.
Generation complete.
$ cat examples/cpnet_n2c1d3_0000.xml
 <PREFERENCE-SPECIFICATION>

 <PREFERENCE-VARIABLE>
  <VARIABLE-NAME>x1</VARIABLE-NAME>
  <DOMAIN-VALUE>1</DOMAIN-VALUE>
  <DOMAIN-VALUE>2</DOMAIN-VALUE>
  <DOMAIN-VALUE>3</DOMAIN-VALUE>
 </PREFERENCE-VARIABLE>

 <PREFERENCE-VARIABLE>
  <VARIABLE-NAME>x2</VARIABLE-NAME>
  <DOMAIN-VALUE>1</DOMAIN-VALUE>
  <DOMAIN-VALUE>2</DOMAIN-VALUE>
  <DOMAIN-VALUE>3</DOMAIN-VALUE>
 </PREFERENCE-VARIABLE>

 <PREFERENCE-STATEMENT>
   <STATEMENT-ID>p1_1</STATEMENT-ID>
   <PREFERENCE-VARIABLE>x1</PREFERENCE-VARIABLE>
   <PREFERENCE>2:1</PREFERENCE>
   <PREFERENCE>1:3</PREFERENCE>
 </PREFERENCE-STATEMENT>

 <PREFERENCE-STATEMENT>
   <STATEMENT-ID>p2_1</STATEMENT-ID>
   <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
   <CONDITION>x1=1</CONDITION>
```

```
37    <PREFERENCE>2:1</PREFERENCE>
38    <PREFERENCE>1:3</PREFERENCE>
39  </PREFERENCE-STATEMENT>
40
41  <PREFERENCE-STATEMENT>
42    <STATEMENT-ID>p2_2</STATEMENT-ID>
43    <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
44    <CONDITION>x1=2</CONDITION>
45    <PREFERENCE>1:2</PREFERENCE>
46    <PREFERENCE>2:3</PREFERENCE>
47  </PREFERENCE-STATEMENT>
48
49  <PREFERENCE-STATEMENT>
50    <STATEMENT-ID>p2_3</STATEMENT-ID>
51    <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
52    <CONDITION>x1=3</CONDITION>
53    <PREFERENCE>3:1</PREFERENCE>
54    <PREFERENCE>1:2</PREFERENCE>
55  </PREFERENCE-STATEMENT>
56
57  </PREFERENCE-SPECIFICATION>
```

Again, each generated instance is random and thus probably differs from what is shown above. This time the `PREFERENCE-VARIABLE` enclosures show that there are two variables, X_1 and X_2, each with *three* values, 1, 2, and 3. We can write these as $\mathrm{Dom}(X_1) = \{x_1, x_1', x_1''\}$, etc. The first `PREFERENCE-STATEMENT` enclosure in lines 26–31 tells us that the preference over X_1 is unconditional, with $x_1' \succ x_1 \succ x_1''$. The following three preference statements (rules) in lines 33–55 give the CPT for X_2. Note that there is one rule for each assignment to X_1, the parent of X_2. Thus the corresponding CP-net is:



4.3 The `-c` and `-g` parameters: multiple CP-nets with bounded indegree

Usually we want to generate a set of CP-nets for an experiment. We can specify the size of the set with the `-g` parameter. Also, a bound on indegree is customarily assumed; we can specify this bound with the `-c` parameter.

Suppose we want to generate 10 CP-nets uniformly at random. The CP-nets should have 10 nodes. No node should have more than 4 parents. The domains are binary. We can do this by typing:

```
$ ./gencpnet -n 10 -c 4 -d 2 -g 10 temp
Building distribution tables for CP-nets with the following specs:
Number of nodes: n = 10
Bound on in-degree c = 4
Homogeneous domains of size d = 2
Probability of incompleteness i = 0
Generating 10 random CP-nets with these specs.
Generation complete.
```

The options `-n`, `-c`, and `-d`, specify the number of nodes n, bound c on indegree, and the size d of all domains, as described in our paper. The `-g` option specifies the number of CP-nets to generate. In this case the resulting CP-nets are written to the `temp` subdirectory, which we can create using `mkdir` as shown above. Note that if we had failed to create the new subdirectory, we would receive an error message:

```
Error: cannot open output file cpnet_n10c4d2_0000.xml
Make sure specified directory temp is accessible.
```

The files are named `cpnet_n10c4d2_0000.xml`, `cpnet_n10c4d2_0001.xml`, etc. The numbers after the `n`, `c`, and `d` correspond respectively to the number of nodes, bound on indegree, and homogeneous domain size as described above. The numerals `0000`, `0001`, ..., `0009` index the generated instances.

If the `-c` parameter is omitted, GenCPnet assumes a default value of 5. It is possible to generate CP-nets with unbounded indegree by specifying a "bound" of $c = n - 1$, or by specifying an "infinite" bound, i.e., `-c 99`, in which case the program sets c to $n - 1$ internally. For example,

```
./gencpnet -n 20 -c 19 temp
```

generates just one CP-net with binary valued domains, $n = 20$ nodes, and indegree that is effectively unbounded. However, with high probability, the resulting CP-net will have one node with indegree 19, resulting in a conditional preference table with $2^{19} = 524288$ entries. The resulting file will be nearly 1 GB in size. If the tables are considerably larger even than this, it may be impossible to output the resulting CP-net representation. For example:

```
$ ./gencpnet -n 60 -c 59 temp
Building distribution tables for CP-nets with the following specs:
Number of nodes: n = 60
Bound on in-degree c = 59
Homogeneous domains of size d = 2
Probability of incompleteness i = 0
Sorry, there is not enough memory for CPTs with 576460752303423488 (5.76461e+17) entries
Aborting generation.
```

A suitable bound on indegree is thus conventionally assumed. For example, the command

```
./gencpnet -n 60 -c 5 -g 100 temp
```

should complete in approximately 10–15 seconds, producing 100 files, each around 270 KB in size.

Note that presently the number of nodes is limited to $n = 63$. An error message will result for $n \geq 64$. This is a due to our implementation, which assumes a processor with 64-bit words.

4.4 Generating DT problems with `-t` and `-h`

Dominance Testing (DT), determining whether one outcome is preferred to another, is an important reasoning problem when working with CP-nets. With GenCPnet the `-t` option can be used to generate *DT problem instances*, as well as CP-net instances, that are quiprobable with respect to a set of DT problems parameterized by the number of nodes n, bound c on indegree, and homogeneous domain size d. It is also possible to constrain the Hamming distance, h, of the resulting problem set—that is, to fix the number of variables in which each outcome pair differs.

For example, the following generates 10 random CP-nets with 5 nodes, bound 2 on indegree, and binary domains.

```
./gencpnet -n 5 -c 2 -g 10 -t 1 problems
```

For each CP-net, one pair of outcomes (unconstrained by Hamming distance) is also generated. A listing of the directory gives:

```
-rw-rw-r-- 1 user user 4006 Dec 10 17:01 cpnet_n5c2d2_0000.xml
-rw-rw-r-- 1 user user 3783 Dec 10 17:01 cpnet_n5c2d2_0001.xml
-rw-rw-r-- 1 user user 4006 Dec 10 17:01 cpnet_n5c2d2_0002.xml
-rw-rw-r-- 1 user user 4006 Dec 10 17:01 cpnet_n5c2d2_0003.xml
-rw-rw-r-- 1 user user 4006 Dec 10 17:01 cpnet_n5c2d2_0004.xml
-rw-rw-r-- 1 user user 3783 Dec 10 17:01 cpnet_n5c2d2_0005.xml
-rw-rw-r-- 1 user user 3277 Dec 10 17:01 cpnet_n5c2d2_0006.xml
-rw-rw-r-- 1 user user 3500 Dec 10 17:01 cpnet_n5c2d2_0007.xml
-rw-rw-r-- 1 user user 3500 Dec 10 17:01 cpnet_n5c2d2_0008.xml
-rw-rw-r-- 1 user user 4006 Dec 10 17:01 cpnet_n5c2d2_0009.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0000_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0001_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0002_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0003_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0004_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0005_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0006_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0007_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0008_0000.xml
-rw-rw-r-- 1 user user 1452 Dec 10 17:01 dt_n5c2d2_0009_0000.xml
```

Here the XML files prefaced with `cpnet_` describe CP-nets as explained above. Those prefaced with `dt_` describe DT problem instances. Note that the filename for each DT instance contains *two* indices. The first is the same as the index of the corresponding CP-net. The second is numbered sequentially from 0000. If we had generated multiple DT problems for CP-net 0000, then the corresponding DT XML files would be labeled `0000_0000`, `0000_0001`, etc.

Recall that a DT problem $\mathcal{N} \models o \succ o'$ consists of a CP-net $\mathcal{N}$ and a pair of outcomes o and o'. It is assumed, of course, that $\mathcal{N}$, o, and o' share the same set of variables $\mathcal{V}$ with associated domains. Consider the randomly generated contents of `dt_n5c2d2_0000_0000.xml`:

```
<PREFERENCE-QUERY>
  <PREFERENCE-SPECIFICATION-FILENAME>cpnet_n5c2d2_0000.xml</PREFERENCE-SPECIFICATION-FILENAME>
  <QUERY-TYPE>DOMINANCE</QUERY-TYPE>
  <OUTCOME>
    <LABEL>BETTER</LABEL>
    <ASSIGNMENT>
      <PREFERENCE-VARIABLE>x1</PREFERENCE-VARIABLE>
      <VALUATION>2</VALUATION>
    </ASSIGNMENT>
    <ASSIGNMENT>
      <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
      <VALUATION>1</VALUATION>
    </ASSIGNMENT>
    <ASSIGNMENT>
      <PREFERENCE-VARIABLE>x3</PREFERENCE-VARIABLE>
      <VALUATION>2</VALUATION>
    </ASSIGNMENT>
    <ASSIGNMENT>
      <PREFERENCE-VARIABLE>x4</PREFERENCE-VARIABLE>
      <VALUATION>2</VALUATION>
    </ASSIGNMENT>
    <ASSIGNMENT>
      <PREFERENCE-VARIABLE>x5</PREFERENCE-VARIABLE>
      <VALUATION>2</VALUATION>
    </ASSIGNMENT>
  </OUTCOME>
```

```
    <OUTCOME>
      <LABEL>WORSE</LABEL>
      <ASSIGNMENT>
        <PREFERENCE-VARIABLE>x1</PREFERENCE-VARIABLE>
        <VALUATION>2</VALUATION>
      </ASSIGNMENT>
      <ASSIGNMENT>
        <PREFERENCE-VARIABLE>x2</PREFERENCE-VARIABLE>
        <VALUATION>2</VALUATION>
      </ASSIGNMENT>
      <ASSIGNMENT>
        <PREFERENCE-VARIABLE>x3</PREFERENCE-VARIABLE>
        <VALUATION>2</VALUATION>
      </ASSIGNMENT>
      <ASSIGNMENT>
        <PREFERENCE-VARIABLE>x4</PREFERENCE-VARIABLE>
        <VALUATION>1</VALUATION>
      </ASSIGNMENT>
      <ASSIGNMENT>
        <PREFERENCE-VARIABLE>x5</PREFERENCE-VARIABLE>
        <VALUATION>1</VALUATION>
      </ASSIGNMENT>
    </OUTCOME>
  </PREFERENCE-QUERY>
```

The `PREFERENCE-SPECIFICATION-FILENAME` enclosure gives the XML file specifying CP-net $\mathcal{N}$. The two `outcome` enclosures specify the outcomes o and o'. In this case, the problem is whether $\overline{x_1}\,x_2\,\overline{x_3}\,\overline{x_4}\,\overline{x_5} \succ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,x_4\,x_5$ with respect to CP-net $\mathcal{N}$ as defined in the file `cpnet_n5c2d2_0000.xml`. Again, we refer the reader to the page `http://www.ece.iastate.edu/~gsanthan/crisner.html` for additional details on the XML specification.

Note that in the problem given above, the Hamming distance is 3, since the two values differ in the values of X_2, X_4, and X_5. We could constrain the Hamming distance to any given value between 1 and n inclusive with the `-h` option. For example:

```
./gencpnet -n 5 -c 2 -g 10 -t 1 -h 2 problems
```

We can use `-t 10` to generate 10 DT problems for each of 10 CP-nets—a total of 100 DT problems:

```
./gencpnet -n 20 -c 4 -g 10 -t 10 problems
```

4.5 Other command line parameters

The `--quiet` parameter can be used to silence all most of the output to the console. Conversely, `--verbose` provides additional output for debugging purposes.

An experimental `-i` parameter allows specifying a *degree of incompleteness*. For example, `-i 0.60` specifies that each conditional preference rule will be specified with probability 0.6. Note that this does not mean that *exactly* 60% of the rules of each table will be filled; it is only a probability invoked for each rule. Thus the actual generated table could be complete or empty. **This is an experimental feature. As such, it could be removed or its behavior may be modified in a future release.**

Finally, the `--count` and `--countdags` parameters only output the *number* of CP-nets and directed acyclic graphs (DAGs) respectively, as described in our paper. For example, the following BASH shell script

```
echo "The number of acyclic, unbounded, binary CP-nets:"
echo -e "n\ta(n)"
for i in $(seq 1 8); do
    echo -n -e $i '\t'
```

```
    # Here we use c=9999 as "infinity"; that is, unbounded indegree
    ./gencpnet -n $i -c 9999 -d 2 --count --quiet .
    echo
done
```

produces the output:

```
The number of acyclic, unbounded, binary CP-nets:
n    a(n)
1    2
2    12
3    488
4    481776
5    157549032992
6    4059976627283664056256
7    5242534484601779604747295174905035666965 76
8    1427153634467948627654814418603596566233315529747076624457951655059580698906328832
```

Note also the use of the `--quiet` option. The complete script is included in the distribution as `count.sh`.

5 License

GenCPnet is free software, released under the GNU Public License version 3. You should have received a copy of `gpl-3.0.txt` in the archive. If not, see `http://www.gnu.org/licenses/`.

If you use GenCPnet in your research, we hope you will choose to cite our paper. BibTeX format is:

```
@inproceedings{generating_cpnets,
  title  = {Generating {CP}-nets Uniformly at Random},
  author = {Thomas E. Allen and
    Judy Goldsmith and Hayden Elizabeth Justice and Nicholas Mattei and Kayla Raines},
  booktitle={Procedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)},
  note = {To appear},
  year = {2016}
}
```

References

[1] Thomas E. Allen, Judy Goldsmith, Hayden Elizabeth Justice, Nicholas Mattei, and Kayla Raines. Generating CP-nets uniformly at random. In *Procedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016. To appear.

[2] C. Boutilier, R.I. Brafman, C. Domshlak, H.H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.