Python Multiwinner Package Manual

January 29, 2017

Contents

1 Introduction **1**

2 Main Components of the Package **1**

2.1 Running a Single Experiment 2

2.1.1 Converting 2D Data to Preference Orders 2

2.1.2 Computing Elections Winners 4

2.1.3 Visualizing Elections 5

2.2 Running a Sequence of Experiments 5

2.2.1 The `gendiag.py` Program 6

2.2.2 Generating and Visualizing Histograms 8

3 Extending the Package **9**

3.1 Adding New Voting Rules 9

4 Final Remarks **9**

1 Introduction

This is a rudimentary manual for using the Python Multiwinner Package software. The package is implemented in Python (2.7.3) and provides a number of features:

1. generation of preference profiles based on the two-dimensional Euclidean domain;
2. computing winners of multiwinner elections;
3. visualizing election results;
4. running series of experiments.

2 Main Components of the Package

Below we describe the main components of the package.

2.1 Running a Single Experiment

The main program here is `experiment.py` which takes care of running a single experiment (generating a 2D Euclidean election, computing results according to various rules, and preparing visual representation of the results). This program uses `2d2pref.py`, `winner.py`, and `visualize.py` to achieve its goals.

Program `experiment.py` is invoked as follows:

```
  python experiment.py <description.input
```

where `description.input` is a file specifying how the experiment should be conducted (what 2D Euclidean election to generate, which rules to run, and for what committee sizes). Below we provide an example of an `.input` file that uses most of the features of `experiment.py`:

```
candidates               # switch to generating candidates
gauss 1 0 0.5 125        # generate 125 points with sigma=0.5,
                         # centered at (1,0)
uniform -2 -2 1 1 50     # generate 50 points distributed uniformly
                         # on the square (-2,-2)(1,1)
voters                   # switch to generating voters
circle 0 0 2 400         # generate 400 points distributed uniformly
                         # on a disc centered at (0,0) with radius 2
generate input-data      # save the generated points to file input-data.in
                         # generate preference-based election and save
                         # it to file input-data.out
stv  20 input-a-stv      # compute the election result for 20 winners using
                         # STV and save the result to input-a-stv.{win,png}
stv  40 input-b-stv      # compute the election result for 40 winners using
                         # STV and save the result to input-b-stv.{win,png}
```

There two main outcomes of running `experiment.py`. The `.win` files which contain the sets of candidates and voters (both including their positions in the 2D model of the generated election) and the winner set (also including their 2D positions). The `.png` files contain visualizations of the election results. In most typical ways of using `experiment.py`, a single run generates only a single election, but possibly runs several rules on it (possibly for several different committee sizes).

`experiment.py` uses `2d2pref.py`, `winner.py`, and `visualize.py` to perform its actions. Usually there is no need to invoke these programs manually. Nonetheless, below we describe how to run them and their file formats (the format of `.win` files may be useful).

2.1.1 Converting 2D Data to Preference Orders

Program `2d2pref.py` converts elections from the 2D Euclidean domain to the ordinal model. The program is ran as follows:

```
python 2d2pref.py <2d_election.in >ordinal_election.out
```

The `2d_election.in'` file has the following format:

1. A single line with two numbers, m and n (the number of candidates and the number of voters).
2. m lines with descriptions of candidates. Each line contains three items, numbers x and y (the position of the candidate in the 2D space) and a single word associated with the candidate (currently not used, but mandatory)
3. n lines with descriptions of voters (in the same format as candidates).

For example, the following file defines four candidates and two voters:

```
4 2
1 1 #
2 1 #
2 2 #
10 0 #
-2 -2 #
3 3 #
```

The generated `ordinal_election.out` file has the following format:

1. A single line with two numbers m and n (the number of candidates and the number of voters)
2. m lines with the description of candidates (consecutive numbers that are the candidate IDs and then the position of the candidate in the 2D space and the word associated with the candidate)
3. n lines with descriptions of voters. Each line lists the IDs of the candidates from the one most preferred by the voter (closest to him or her in the Euclidean distance) to the least preferred one (the farthest). The voter's preference order is then followed by his or her position in the 2D space and the word associated with him or her.

For example, for the above 2D election, `2d2pref.py` outputs:

```
4 2
0     1.0 1.0 #
1     2.0 1.0 #
2     2.0 2.0 #
3     10.0 0.0 #
0 1 2 3    -2.0 -2.0
2 1 0 3    3.0 3.0
```

This file describes an election with candidate set $C = \{0, 1, 2, 3\}$ and with two voters, one with preference order $0 \succ 1 \succ 2 \succ 3$ and one with preference order $2 \succ 1 \succ 0 \succ 3$.

2.1.2 Computing Elections Winners

The heart of the package is the `winner.py` program, which computes the results of multi-winner elections. The invocation is:

```
python winner.py rule k <ordinal_election.out >result.win
```

Then the program reads in election `ordinal_election.out`, computes a winning committee of size `k` using multiwinner voting rule `rule` and writes out the results to the file `result.win`. Available rules include `bloc`, `kborda`, `stv`, and many others. To see a list of available rules run:

```
python winner.py --help
```

The input ordinal election is exactly in the format produced by `2d2pref.py`. The `result.win` file is in the same format as the election, except for the following two changes:

1. The first line contains three numbers (after the numbers m and n of candidates and winners, there is also the committee size k)
2. After the $m + n$ lines describing the election, there are another k lines describing the election winners (each line starts with the ID of the candidate in the committee, followed by his or her 2D position and the associated word)

For example, running:

```
python winner.py kborda 2 <election.out >election.win
```

where `election.out` is the ordinal election from the example in the preceding section, `winner.py` produces file `election.win` with the following contents:

```
4 2 2
0    1.0 1.0 #
1    2.0 1.0 #
2    2.0 2.0 #
3    10.0 0.0 #
0 1 2 3    -2.0 -2.0
2 1 0 3    3.0 3.0
1    2.0 1.0 #
2    2.0 2.0 #
```

Note that the `.win` files contain all the information about the election and, thus, there is no need to store the `.in` or `.out` files.

While running, `winner.py` provides additional output (e.g., indicating where in each algorithm it is). This information is sent to the standard error output (which means that normally it is displayed on the screen, but is not sent to the `result.win` file).

Tie breaking. `winner.py` uses a simplified variant of the parallel-universes tie-breaking. Whenever during computation of some rule it reaches a tie, it makes a random choice regarding what decision to make.

2.1.3 Visualizing Elections

Program `visualize.py` reads in a `.win` file and creates its graphical presentation as a `.png` image. The invocation is:

```
python visualize.py result
```

The program reads in file `result.win` (note that the program adds the extension `.win` automatically) and creates its graphical representation in the file `result.png`.

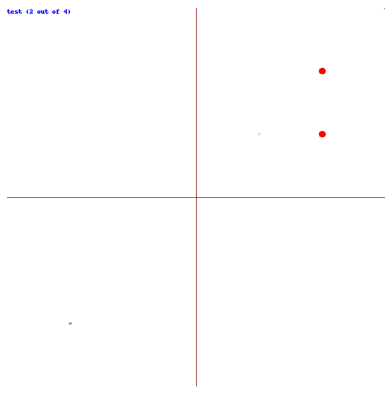
Currently, the program is hard-coded with the following parameters:

1. The created image has resolution 600×600.
2. The image represents square $[-3, 3] \times [-3, 3]$.
3. The colors and shapes of voters, candidates, and winners are fixed. Voters are dark gray, candidates are light gray, winners are red (and bigger).

For example, running:

```
python visualize.py result
```

where `result.win` is the file generated in the previous section, generates file `result.png` with the following content (note that point $(10, 0)$ is not shown; the picture only shows the $[-3, 3] \times [-3, 3]$ area):



2.2 Running a Sequence of Experiments

Using `experiment.py`, it is possible to compute the results of a number of rules (possibly for different committee sizes) on a single election. For example, consider the following `kborda-cm.input` file:

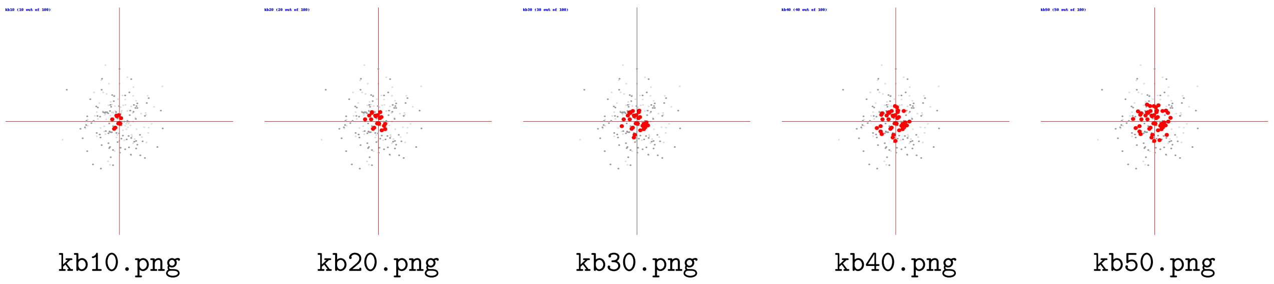


Figure 1: Images produced using KBORDA-CM.INPUT.

```
candidates
gauss 0 0 0.5 100
voters
gauss 0 0 0.5 100
generate kborda-cm
kborda 10 kb10
kborda 20 kb20
kborda 30 kb30
kborda 40 kb40
kborda 50 kb50
```

After invoking:

```
python experiment <kborda-cm.input
```

we get five images that show how the k-Borda committee changes as we change its size (see Figure 1). In such cases, it is necessary to prepare the appropriate .INPUT file manually. On the othe hand, the PYTHON MULTIWINNER PACKAGE provides support for running a single rule on a series of elections.

2.2.1 The `gendiag.py` Program

The GENDIAG.PY is invoked as follows:

```
  python gendiag.py input_template rule from to k
```

where:

1. `input_template` is a name of the template of the `.input` file to use (a template is just like a regular `.input` file, but it ends befofe—and not including—the `generate` line).

2. `rule` is the voting rule to use.

3. `from` is the first sequence number to use (see below)

4. `to` is the last sequence number to use (see below)
5. `k` is the committee size

With such parameters, the program works as follows:

1. It creates directory `data_input_template` and copies there all the python scripts from the current directory (currently we assume that all the scripts from Python Multiwinner Package are present in the current directory).
2. For all positive integers i between `from` and `to` it executes the following:
   (a) It creates an input file that starts with the contents of the `input_template` and that is supplemented with generating the election and running voting rule `rule` with committee size `k`.
   (b) It runs the `experiment.py` program on the created input file (the names of the respective `.win` and `.png` files—without the extension—are `rule_k-`i).
   (c) It deletes the `.in` and `.out` files.

In effect, the directory `data_input_template` is populated with the results of `to` $-$ `from` $+\,1$ `.win` files (and their `.png` files).

For example, we can use the following file (which we named `uniform50`):

```
candidates
uniform -3 -3 3 3 50
voters
uniform -3 -3 3 3 50
```

Invoking:

```
python gendiag.py uniform50 kborda 1 1000 25
```

will run the k-Borda rule on 1000 different elections, each with 50 candidates and 50 voters generated uniformly at random on the $[-3, 3] \times [-3, 3]$ square (warning! this computation may take a moment; the reader may wish to first reduce the value `1000` to something much smaller).

The naming scheme of the files generated using `gendiag.py` is such that it is possible to run several instances of the program in parallel, to speed up computation. For example, we could invoke:

```
python gendiag.py uniform50 kborda 1 1000 25 &
python gendiag.py uniform50 kborda 1001 2000 25 &
python gendiag.py uniform50 kborda 2001 3000 25 &
python gendiag.py uniform50 kborda 3001 4000 25 &
```

to generated 4000 election results on a computer with at least four CPU cores.

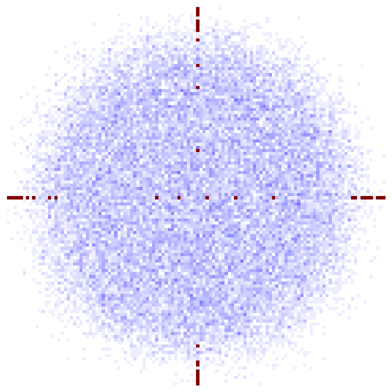


Figure 2: `kborda_25_hist.png`.

2.2.2 Generating and Visualizing Histograms

Python Multiwinner Package provides tools for constructing histograms that show how frequently a winners appear in a given area (depending on the voting rule, the committee size, and the distributions of voters and candidates). The exact explanation of how such histograms are generated and how they can be interpreted are provided by Elkind et al. [**?**]. Below we describe how the Python Multiwinner Package supports generating such histograms.

To generate histogram, we should do the following:

1. Run gendiag.py to generate appropriate sequence of elections (we should start sequence numbers from 1); we assume we have done so as in the preceding section to obtain 1000 election results.
2. Enter directory `data_input_template`.
3. Invoke `histogram.py rule_k to`. This generates file `rule_k.hist` (the actual numerical data for the histogram).
4. Invoke `hisogram_draw.py rule_k.hist 0.0004` (the second parameter is the ε value of Elkind et al. [**?**]). We obtain file `rule_k_hist.png` with the histogram.

For example, after running the example from the previous section, to obtain the histogram we should execute the following commands:

```
cd data_uniform50
python histogram.py kborda_25 1000
python histogram_draw.py kborda_25.hist 0.0004
```

In effect, we obtain file `kborda_25_hist.png` presented in Figure 2.

Currently the main parameters (e.g., the resolution of the generated pictures) of both `histogram.py` and `histogram_draw.py` are hard-coded and require modifications in the source code to change.

3 Extending the Package

There are many ways in which the `Python Multiwinner Package` can be extended. We welcome all contributions. Below we describe one particular type of extension that many users may find important.

3.1 Adding New Voting Rules

Currently the easiest way to add a new voting rule is to directly edit the source of WINNER.PY and to add a function computing the new rule just before the line:

```
if __name__ == "__main__":
```

at the end of the file.

The new code should follow the following structure:

```
RULES += [("new_rule", "a more detailed description")]

def new_rule( V, k ):
  n = len(V)   # number of voters
  m = len(V[0] # number of candidates

  # execute the rule here

  return LIST_OF_k_WINNERS
```

The candidate set contains integers $0, \ldots, \mathtt{m} - \mathtt{1}$. The profile,`V`, is a list of preference orders. Each preference order is a list of integers (i.e., of the names of the candidates) in the order from the most to the least preferred one.

4 Final Remarks

We welcome all sorts of feedback (but please bare in mind that this package is in its very preliminary form; a lot of things, indeed, could be done better). The package is freely available for research and personal use (however, we make no promises regarding its usefulness; use at your own risk!). For commercial use, please contact the authors.