Preferential Voting Tools
pref_voting

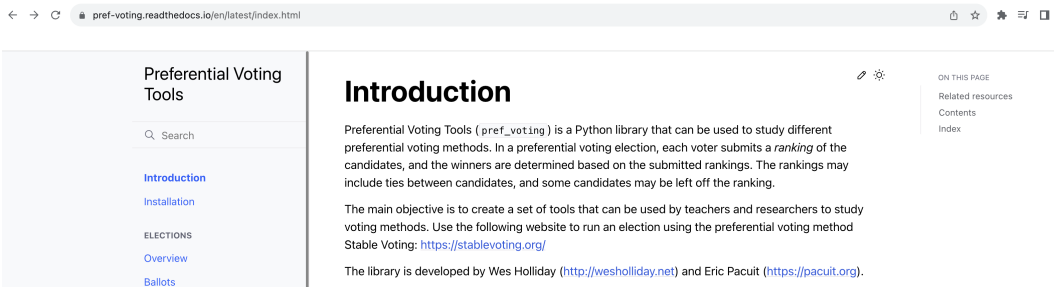
Wes Holliday (UC Berkeley) Eric Pacuit (University of Maryland)

October 18, 2024

Preferential Voting Tools (`pref_voting`)

A Python library that can be used to study and run elections with different preferential voting methods (graded voting methods and cardinal voting methods are also included for comparison).

https://pref-voting.readthedocs.io/



Notebook Available on Colab

https://colab.research.google.com/drive/1QcmITfZEuRv7N2cyeqRbsZk3XZRjrOdd?usp=sharing

Election Data

- `Profile`: each voter linearly orders the candidates;
- `ProfileWithTies`: each voter ranks the candidates, allowing ties and omissions of candidates;
- `GradeProfile`: each voter assigns grades from some finite list of grades to selected candidates (with approval ballots as a special case);
- `UtilityProfile`: each voter assigns a real number to each candidate;
- `SpatialProfile`: each voter and each candidate is placed in a multi-dimensional space;
- `MajorityGraph`: an edge from candidate A to candidate B represents that more voters rank A above B than vice versa;
- `MarginGraph`: a weighted version of a `MajorityGraph`, where the weight on an edge represents the margin of victory.

Election Data

- `Profile`: each voter linearly orders the candidates;

```
from pref_voting.profiles import Profile

prof = Profile(
    [[2, 1, 0, 3], [3, 2, 0, 1], [3, 1, 0, 2]], # linear rankings
    rcounts=[2, 1, 3], # number of voters for each ranking
    cmap={0: 'a', 1: 'b', 2: 'c', 3: 'd'} # candidate mapping
    )

prof.display()
```

[15] ✓ 0.0s

```
+---+---+---+
| 2 | 1 | 3 |
+---+---+---+
| c | d | d |
| b | c | b |
| a | a | a |
| d | b | c |
+---+---+---+
```

Election Data

- `ProfileWithTies`: each voter ranks the candidates, allowing ties and omissions of candidates;

```
from pref_voting.profiles_with_ties import ProfileWithTies

prof = ProfileWithTies(
    [{0:1, 1:1}, {0:1, 1:1, 2:2},  {0:3, 1:1, 2:2}, {0:1, 3:0, 2:1}], # rankings with ties
    rcounts=[2, 2, 1, 3], # number of voters for each ranking
    candidates=[0, 1, 2, 3], # candidates in the election
    cmap={0: 'a', 1: 'b', 2: 'c', 3: 'd'} # candidate map
    )

prof.display()
```

[16] ✓ 0.0s

2	2	1	3
a b	a b	b	d
	c	c	a c
		a	

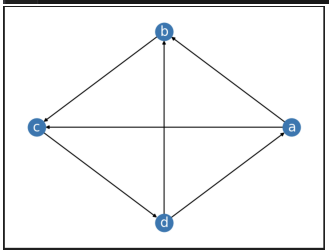
Election Data

- `MajorityGraph`: an edge from candidate A to candidate B represents that more voters rank A above B than vice versa;

```python
from pref_voting.weighted_majority_graphs import MajorityGraph

mg = MajorityGraph(
    [0, 1, 2, 3], # candidates in the election
    [(0, 1), (1, 2), (0, 2), (3, 0), (3, 1), (2, 3)], # edges in the majority graph
    cmap={0: 'a', 1: 'b', 2: 'c', 3: 'd'} # candidate mapping
    )

mg.display()
```



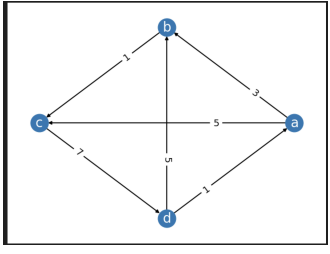
Election Data

- `MarginGraph`: a weighted version of a `MajorityGraph`, where the weight on an edge represents the margin of victory.

```
from pref_voting.weighted_majority_graphs import MarginGraph
mg = MarginGraph(
    [0, 1, 2, 3], # candidates in the election
    [(0, 1, 3), (1, 2, 1), (0, 2, 5), (3, 0, 1), (3, 1, 5), (2, 3, 7)], # weighted edges in the
    margin graph
        cmap={0: 'a', 1: 'b', 2: 'c', 3: 'd'} # candidate mapping
)

mg.display()
```

[27] ✓ 0.0s Py



Election Data

- `UtilityProfile`: each voter assigns a real number to each candidate;

```
from pref_voting.utility_profiles import UtilityProfile

uprof = UtilityProfile([
    {0:1.0, 1: 0.5, 2: 1.25}, # utilities for each candidate
    {0:-0.5, 1: 9.5, 2: 1.05},
    {0:3.0, 1: -2.0, 2: 1.0},
    {0:0.5, 1: 0.0, 2: 3.0}
    ])

uprof.display()
```

[22] ✓ 0.0s

Voter	0	1	2
1	1	0.5	1.25
2	−0.5	9.5	1.05
3	3	−2	1
4	0.5	0	3

Election Data

- `GradeProfile`: each voter assigns grades from some finite list of grades to selected candidates (with approval ballots as a special case);

```
from pref_voting.grade_profiles import GradeProfile

gprof = GradeProfile([
    {0:1, 1:3, 2:3}, # grades for each candidate
    {0:3, 1:2, 2:1},
    {0:1, 1:1, 2:1},
],
gcounts=[1, 2, 1], # number of voters for each grade assignment
grades=[0, 1, 2, 3], # the grades in the profile
cmap = {0: 'a', 1: 'b', 2: 'c'} # candidate mapping
)

gprof.display(show_totals=True)
```

[24] ✓ 0.0s

```
...
        1    2    1    Sum    Median
--  ---  ---  ---  -----  --------
a     1    3    1      8         1
b     3    2    1      8         2
c     3    1    1      6         1
```

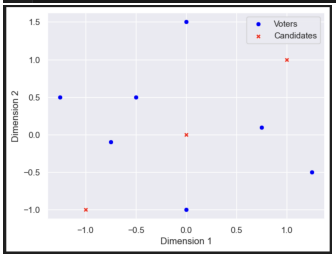
Election Data

- `SpatialProfile`: each voter and each candidate is placed in a multi-dimensional space;

```python
from pref_voting.spatial_profiles import SpatialProfile

sprof = SpatialProfile(
    cand_pos={0:(0, 0), 1: (1, 1), 2:(-1, -1)}, # candidate positions
    voter_pos={0:(0, 1.5), 1:(-1.25, 0.5), 2:(0.75, 0.1), 3:(0, -1), 4:(1.25, -0.5), 5:(-0.75, -0.1), 6:(-0.5, 0.5)}, # voter positions
    candidate_types={0:'L', 1:'L', 2:'C' }) # assignment of candidates to types (or parties)

# sprof.display() # display the spatial profile
sprof.view()
```



to_latex()

```
prof = Profile(
    [[0, 1, 2], [2, 1, 0], [1, 2, 0]],
    rcounts=[2, 1, 4],
    cmap={0: 'a', 1: 'b', 2: 'c'})

print(prof.to_latex())
```

[36] ✓ 0.0s

```
\begin{tabular}{ccc}
$2$ & $1$ & $4$\\\hline
$a$ & $c$ & $b$\\
$b$ & $b$ & $c$\\
$c$ & $a$ & $a$
\end{tabular}
```

2	1	4
a	*c*	*b*
b	*b*	*c*
c	*a*	*a*

Properties of Profiles

```
prof = Profile([
    [2, 1, 0, 3],
    [3, 2, 0, 1],
    [3, 1, 0, 2]],
    rcounts=[2, 2, 3])

prof.display()

print(f"The margin of 1 over 3 is {prof.margin(1, 3)}")
print(f"The Plurality scores are {prof.plurality_scores()}")
print(f"The Copeland scores are {prof.copeland_scores()}")
print(f"The Borda scores are {prof.borda_scores()}")
print(f"The Condorcet winner is {prof.condorcet_winner()}")
print(f"The weak Condorcet winner is {prof.weak_condorcet_winner()}")
print(f"The Condorcet loser is {prof.condorcet_loser()}")
print(f"The profile is uniquely weighted: {prof.is_uniquely_weighted()}")
```

https://pref-voting.readthedocs.io/en/latest/profiles.html

Properties of Profiles

```
+---+---+---+
| 2 | 2 | 3 |
+---+---+---+
| 2 | 3 | 3 |
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| 3 | 1 | 2 |
+---+---+---+
The margin of 1 over 3 is -3
The Plurality scores are {0: 0, 1: 0, 2: 2, 3: 5}
The Copeland scores are {0: -3.0, 1: -1.0, 2: 1.0, 3: 3.0}
The Borda scores are {0: 7, 1: 10, 2: 10, 3: 15}
The Condorcet winner is 3
The weak Condorcet winner is [3]
The Condorcet loser is 0
The profile is uniquely weighted: False
```

https://pref-voting.readthedocs.io/en/latest/profiles.html

Transforming Between Election Data

Object	Method	Output
`Profile`/`ProfileWithTies`	`majority_graph()`	`MajorityGraph`
`Profile`/`ProfileWithTies`	`margin_graph()`	`MarginGraph`

Transforming Between Election Data

Object	Method	Output
`Profile`/`ProfileWithTies`	`majority_graph()`	`MajorityGraph`
`Profile`/`ProfileWithTies`	`margin_graph()`	`MarginGraph`
`Profile`	`to_profile_with_ties()`	`ProfileWithTies`
`Profile`	`randomly_truncate()`	`ProfileWithTies`
`ProfileWithTies`	`to_linear_profile()`	`Profile`

Transforming Between Election Data

Object	Method	Output
`Profile`/`ProfileWithTies`	`majority_graph()`	`MajorityGraph`
`Profile`/`ProfileWithTies`	`margin_graph()`	`MarginGraph`
`Profile`	`to_profile_with_ties()`	`ProfileWithTies`
`Profile`	`randomly_truncate()`	`ProfileWithTies`
`ProfileWithTies`	`to_linear_profile()`	`Profile`
`MarginGraph`	`debord_profile()`	`Profile`
`MarginGraph`	`minimal_profile()`	`Profile`

Transforming Between Election Data

Object	Method	Output
`Profile`/`ProfileWithTies`	`majority_graph()`	`MajorityGraph`
`Profile`/`ProfileWithTies`	`margin_graph()`	`MarginGraph`
`Profile`	`to_profile_with_ties()`	`ProfileWithTies`
`Profile`	`randomly_truncate()`	`ProfileWithTies`
`ProfileWithTies`	`to_linear_profile()`	`Profile`
`MarginGraph`	`debord_profile()`	`Profile`
`MarginGraph`	`minimal_profile()`	`Profile`
`UtilityProfile`	`to_ranking_profile()`	`ProfileWithTies`
`GradeProfile`	`to_ranking_profile()`	`ProfileWithTies`

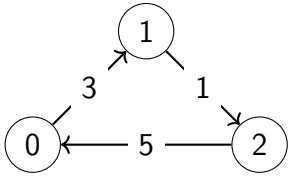
Transforming Between Election Data

Object	Method	Output
`Profile`/`ProfileWithTies`	`majority_graph()`	`MajorityGraph`
`Profile`/`ProfileWithTies`	`margin_graph()`	`MarginGraph`
`Profile`	`to_profile_with_ties()`	`ProfileWithTies`
`Profile`	`randomly_truncate()`	`ProfileWithTies`
`ProfileWithTies`	`to_linear_profile()`	`Profile`
`MarginGraph`	`debord_profile()`	`Profile`
`MarginGraph`	`minimal_profile()`	`Profile`
`UtilityProfile`	`to_ranking_profile()`	`ProfileWithTies`
`GradeProfile`	`to_ranking_profile()`	`ProfileWithTies`
`UtilityProfile`	`to_approval_profile()`	`GradeProfile`
`UtilityProfile`	`to_k_approval_profile()`	`GradeProfile`

Transforming Between Election Data

Object	Method	Output
`Profile`/`ProfileWithTies`	`majority_graph()`	`MajorityGraph`
`Profile`/`ProfileWithTies`	`margin_graph()`	`MarginGraph`
`Profile`	`to_profile_with_ties()`	`ProfileWithTies`
`Profile`	`randomly_truncate()`	`ProfileWithTies`
`ProfileWithTies`	`to_linear_profile()`	`Profile`
`MarginGraph`	`debord_profile()`	`Profile`
`MarginGraph`	`minimal_profile()`	`Profile`
`UtilityProfile`	`to_ranking_profile()`	`ProfileWithTies`
`GradeProfile`	`to_ranking_profile()`	`ProfileWithTies`
`UtilityProfile`	`to_approval_profile()`	`GradeProfile`
`UtilityProfile`	`to_k_approval_profile()`	`GradeProfile`
`SpatialProfile`	`to_utility_profile()`	`UtilityProfile`

MarginGraph to Profile




```
mg = MarginGraph([0, 1, 2], [(0, 1, 3), (1, 2, 1), (2, 0, 5)])

debord_prof = mg.debord_profile()
debord_prof.display()

min_prof = mg.minimal_profile()
min_prof.display()
```

[45] ✓ 0.0s

```
+---+---+---+---+---+
| 1 | 1 | 3 | 3 | 1 |
+---+---+---+---+---+
| 0 | 2 | 2 | 1 | 0 |
| 1 | 0 | 0 | 2 | 1 |
| 2 | 1 | 1 | 0 | 2 |
+---+---+---+---+---+
+---+---+---+
| 2 | 3 | 4 |
+---+---+---+
| 0 | 1 | 2 |
| 1 | 2 | 0 |
| 2 | 0 | 1 |
+---+---+---+
```

SpatialProfile to UtilityProfile

https://pref-voting.readthedocs.io/en/latest/spatial_profiles.html#utility-functions

```
from pref_voting.utility_functions import *

uprof = sprof.to_utility_profile(utility_function=linear_utility)

uprof = sprof.to_utility_profile(utility_function=quadratic_utility)

uprof = sprof.to_utility_profile(utility_function=rm_utility)

uprof = sprof.to_utility_profile(utility_function=mixed_rm_utility)

uprof = sprof.to_utility_profile(utility_function=matthews_utility)

uprof = sprof.to_utility_profile(utility_function=shepsle_utility)

uprof = sprof.to_utility_profile(utility_function=city_block_utility)
```

Generating Elections - `Profile`

- `generate_profile`
  - Wrapper function that uses the `prefsampling` package to generate profiles using standard probability models: https://comsoc-community.github.io/prefsampling/index.html.

- Enumerating `Profile` objects
  - `enumerate_anon_profile`

Generating a `Profile`

```python
from pref_voting.generate_profiles import generate_profile

prof = generate_profile(3, 11) # default is the impartial culture

prof = generate_profile(3, 11, prob_model="IAC") # impartial anonymous culture

prof = generate_profile(3, 11, prob_model="mallows", phi=0.8, normalize_phi=True)

prof = generate_profile(3, 11, prob_model="urn", alpha=3)

prof = generate_profile(3, 11, prob_model="euclidean", num_dims=5)

prof = generate_profile(3, 11, prob_model="single_peaked_conitzer", num_dims=5)

prof = generate_profile(3, 11, prob_model="plackett_luce", alphas=[3, 1, 1])
```

Generating Elections - SpatialProfile

- generate_spatial_profile
- generate_spatial_profile_polarized
- generate_spatial_profile_polarized_cands_randomly_polarized_voters

Generating a SpatialProfile

```python
from pref_voting.generate_spatial_profiles import generate_spatial_profile, generate_spatial_profile_polarized,
generate_spatial_profile_polarized_cands_randomly_polarized_voters, generate_covariance

sprof = generate_spatial_profile(3, 100, num_dims=3) # 3 candidates, 100 voters, 3 dimensions

sprof = generate_spatial_profile_polarized(
    [((-1, 0), generate_covariance(2, 0.1, 0), 2), # 2 candidates in this cluster
     ((1, 0), generate_covariance(2, 0.1, 0), 2), # 2 candidates in this cluster
    ],
    [((-1, 0), generate_covariance(2, 0.25, 0), 100), # 100 voters in this cluster
     ((1, 0), generate_covariance(2, 0.25, 0), 100), # 100 voters in this cluster
    ],
)

sprof = generate_spatial_profile_polarized_cands_randomly_polarized_voters(
    [((-1, 0), generate_covariance(2, 0.1, 0), 2), # 2 candidates in this cluster
     ((1, 0), generate_covariance(2, 0.1, 0), 2),  # 2 candidates in this cluster
    ],
    1000, # number of voters
    [((-1, 0), generate_covariance(2, 0.25, 0), 0.6), # 60% of voters in this cluster
     ((1, 0), generate_covariance(2, 0.25, 0), 0.4)], # 40% of voters in this cluster],
)
```

Other Tools

- Generating a `MarginGraph`
  - `generate_margin_graph`
  - `generate_margin_graph_bradley_terry`

Other Tools

- Generating a `MarginGraph`
  - `generate_margin_graph`
  - `generate_margin_graph_bradley_terry`
- Generating *edge ordered* tournaments:
  - `generate_edge_ordered_tournament`
  - `generate_edge_ordered_tournament_infinite_limit`

Other Tools

- Generating a `MarginGraph`
  - `generate_margin_graph`
  - `generate_margin_graph_bradley_terry`
- Generating *edge ordered* tournaments:
  - `generate_edge_ordered_tournament`
  - `generate_edge_ordered_tournament_infinite_limit`
- Enumerating `MarginGraph` objects
  - `enumerate_canonical_edge_ordered_tournaments`
  - `enumerate_uniquely_weighted_margin_graphs`

Other Tools

- Generating a `MarginGraph`
  - `generate_margin_graph`
  - `generate_margin_graph_bradley_terry`
- Generating *edge ordered* tournaments:
  - `generate_edge_ordered_tournament`
  - `generate_edge_ordered_tournament_infinite_limit`
- Enumerating `MarginGraph` objects
  - `enumerate_canonical_edge_ordered_tournaments`
  - `enumerate_uniquely_weighted_margin_graphs`
- Generating a `UtilityProfile`
  - `generate_utility_profile_uniform`
  - `generate_utility_profile_normal`

Collective Decision Methods

- `VotingMethod`: given *edata*, outputs a list of candidates, representing tied winners;
- `ProbVotingMethod`: given *edata*, outputs a dictionary whose keys are candidates and whose values are probabilities;
- `SocialWelfareFunction`: given *edata*, outputs a ranking of the candidates.

Voting Methods

Currently, there are 79 voting methods that are implemented.

- Scoring Rules:
  pref-voting.readthedocs.io/en/latest/scoring_methods.html
- Iterative Methods:
  pref-voting.readthedocs.io/en/latest/iterative_methods.html
- C1 Methods
  pref-voting.readthedocs.io/en/latest/c1_methods.html
- Margin Based Methods
  pref-voting.readthedocs.io/en/latest/margin_based_methods.html
- Combined Methods
  pref-voting.readthedocs.io/en/latest/combined_methods.html
- Other Methods
  pref-voting.readthedocs.io/en/latest/other_methods.html

Voting Methods

```
from pref_voting.voting_methods import *
from pref_voting.generate_profiles import generate_profile

prof = generate_profile(4, 1000)
vms = [plurality, borda, instant_runoff, instant_runoff_put, minimax, split_cycle, ranked_pairs, river]
for vm in vms:
    print(f"The {vm.name} winners: {vm(prof)}")
    print(f"\trestricted to the candidates [1, 2, 3]: {vm(prof, curr_cands=[1, 2, 3])}")
```

[103] ✓ 0.0s

```
The Plurality winners: [3]
        restricted to the candidates [1, 2, 3]: [3]
The Borda winners: [3]
        restricted to the candidates [1, 2, 3]: [3]
The Instant Runoff winners: [0]
        restricted to the candidates [1, 2, 3]: [3]
The Instant Runoff PUT winners: [0]
        restricted to the candidates [1, 2, 3]: [3]
The Minimax winners: [0]
        restricted to the candidates [1, 2, 3]: [3]
The Split Cycle winners: [0]
        restricted to the candidates [1, 2, 3]: [3]
The Ranked Pairs winners: [0]
        restricted to the candidates [1, 2, 3]: [3]
The River winners: [0]
        restricted to the candidates [1, 2, 3]: [3]
```

Axioms

The `Axiom` objects include functions that check whether a collective decision procedure satisfies a given axiom with respect to some *edata*:

- `has_violation`: check whether there is at least one violation of the axiom by the procedure for the given *edata*
- `find_all_violations`: enumerate all violations together with relevant data.

Axioms

- dominance axioms
  pref-voting.readthedocs.io/en/latest/dominance_axioms.html
- invariance axioms
  pref-voting.readthedocs.io/en/latest/invariance_axioms.html
- monotonicity axioms
  pref-voting.readthedocs.io/en/latest/monotonicity_axioms.html
- strategic axioms
  pref-voting.readthedocs.io/en/latest/strategic_axioms.html
- variable voter axioms
  pref-voting.readthedocs.io/en/latest/variable_voter_axioms.html
- variable candidate axioms
  pref-voting.readthedocs.io/en/latest/variable_candidate_axioms.html
- SWF axioms
  pref-voting.readthedocs.io/en/latest/swf_axioms.html

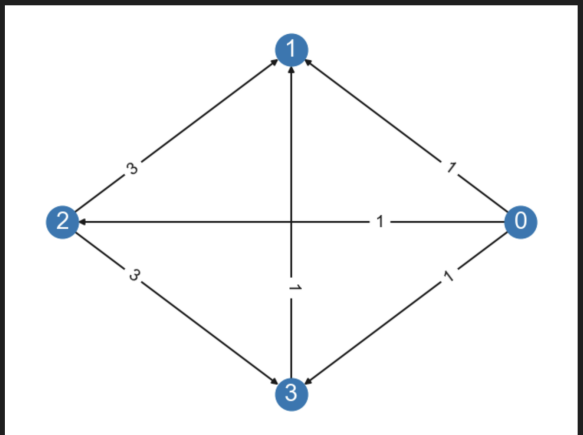
Condorcet Winner

```
from pref_voting.dominance_axioms import *

vm = borda
num_cands = 4
num_voters = 3
num_trials = 1000

for t in range(num_trials):
    prof = generate_profile(num_cands, num_voters)
    if condorcet_winner.has_violation(prof, vm, only_resolute=True, verbose=True):
        break
```

Condorcet Winner




```
Profile([[0, 2, 3, 1], [2, 3, 1, 0], [0, 2, 1, 3]], rcounts=[1, 1, 1], cmap={0: '0', 1: '1', 2: '2', 3: '3'})
The Condorcet winner 0 is not the unique winner:
Borda winner is {2}
```

Immunity to Spoilers

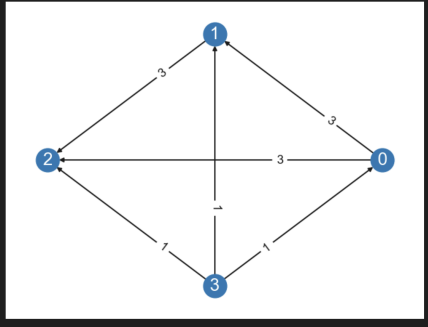
```
from pref_voting.variable_candidate_axioms import *

vm = borda
num_cands = 4
num_voters = 3
num_trials = 1000

for t in range(num_trials):
    prof = generate_profile(num_cands, num_voters)
    if immunity_to_spoilers.has_violation(prof, vm, verbose=True):
        break
```

Immunity to Spoilers

```
Immunity to Spoilers violation for Borda.
3 wins without 1 in the election and is majority preferred to 1 but both lose when 1 is included:
+---+---+---+
| 1 | 1 | 1 |
+---+---+---+
| 3 | 3 | 0 |
| 0 | 0 | 1 |
| 1 | 1 | 2 |
| 2 | 2 | 3 |
+---+---+---+
Profile([[3, 0, 1, 2], [3, 0, 1, 2], [0, 1, 2, 3]], rcounts=[1, 1, 1], cmap={0: '0', 1: '1', 2: '2', 3: '3'})
```




```
Winners in full election: [0]
Winners in election without 1: [0, 3]
```

Calculating Condorcet Efficiency - 1

```
from pref_voting.analysis import binomial_confidence_interval

min_error = 0.05
min_num_samples = 1000
max_num_samples = 100_000

voting_methods = [plurality,  borda,  instant_runoff]

num_candidates = 4
num_voters = 11

has_condorcet_winner = []
elect_condorcet_winner = {vm.name: [] for vm in voting_methods}

num_samples = 0
error_ranges = []
```

Calculating Condorcet Efficiency - 2

```
while num_samples < min_num_samples or (any([(err[1] - err[0]) > min_error for err in
error_ranges]) and num_samples < max_num_samples):

    num_samples += 1
    prof = generate_profile(num_candidates, num_voters)
    cw = prof.condorcet_winner()

    has_condorcet_winner.append(cw is not None)
    for vm in voting_methods:
        if cw is not None:
            elect_condorcet_winner[vm.name].append(vm(prof) == [cw])

    error_ranges = [binomial_confidence_interval(elect_condorcet_winner[vm.name])
                    if len(elect_condorcet_winner[vm.name]) > 0 else (0, np.inf)
                    for vm in voting_methods]
```

Calculating Condorcet Efficiency - 3

```
print(f"For {num_samples} samples:\n")
print(f"The percentage of profiles with a Condorcet winner is {np.mean
(has_condorcet_winner)}")
err = binomial_confidence_interval(has_condorcet_winner)
print(f"\tError: {err[1]-err[0]}")

print()
for vm in voting_methods:
    print(f"The Condorcet efficiency of {vm.name} is {np.mean(elect_condorcet_winner[vm.
    name])}")
    err = binomial_confidence_interval(elect_condorcet_winner[vm.name])
    print(f"\tError: {err[1]-err[0]}")
```

Calculating Condorcet Efficiency - Output

```
For 1764 samples:

The percentage of profiles with a Condorcet winner is 0.8378684807256236
        Error: 0.03495235683503972

The Condorcet efficiency of Plurality is 0.6258457374830853
        Error: 0.04997642712974326
The Condorcet efficiency of Borda is 0.8640054127198917
        Error: 0.03561495608588128
The Condorcet efficiency of Instant Runoff is 0.9242219215155616
        Error: 0.027667207860292087
```

pref_voting.analysis.condorect_efficiency_data

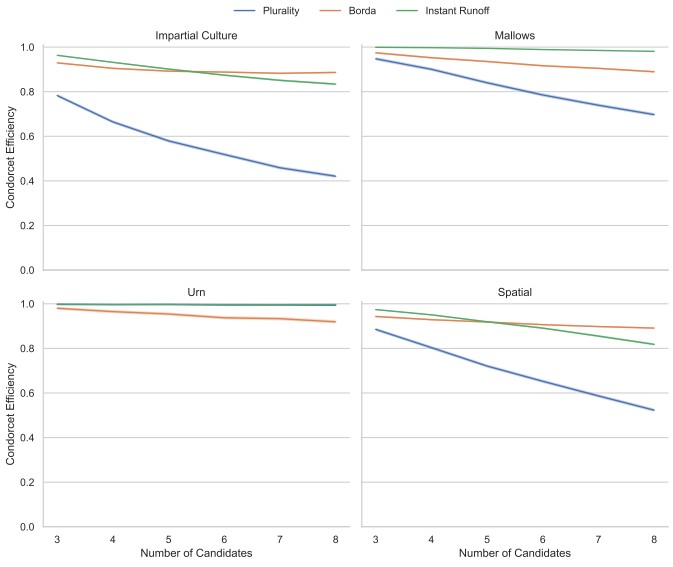
```
from pref_voting.analysis import condorcet_efficiency_data

prob_models = {
    "Impartial Culture": lambda nc, nv: generate_profile(nc, nv),
    "Mallows": lambda nc, nv: generate_profile(nc, nv, probmodel="mallows", phi=0.8,
    normalise_phi=True),
    "Urn": lambda nc, nv: generate_profile(nc, nv, probmodel="urn", alpha=5),
    "Spatial": lambda nc, nv: generate_spatial_profile(nc, nv, 3).to_utility_profile().
    to_ranking_profile().to_linear_profile()
}

df = condorcet_efficiency_data(
    [plurality, borda, instant_runoff], # voting methods to evaluate
    numbers_of_candidates=[3, 4, 5, 6, 7, 8], # numbers of candidates to check
    numbers_of_voters=[100], # numbers of voters to check
    prob_models=prob_models, # dictionary of the probability models to use
    min_num_samples=1000, # minimum number of samples
    max_num_samples=100_000, # maximum number of samples
    min_error=0.01, # the minimum error
    use_parallel=True, # use parallel processing
    num_cpus=12) # number of cpus to use when doing parallel processing
```

[114] ✓ 5m 25.6s

pref_voting.analysis.condorect_efficiency_data



Percentage of Violations: Immunity to Spoilers - 1

```
from pref_voting.analysis import binomial_confidence_interval
from pref_voting.variable_candidate_axioms import immunity_to_spoilers

min_error = 0.01
min_num_samples = 1000
max_num_samples = 100_000

axiom = immunity_to_spoilers
voting_methods = [plurality, borda, instant_runoff]

num_candidates = 4
num_voters = 11

has_violation = {vm.name: [] for vm in voting_methods}
num_samples = 0
error_ranges = []
```

Percentage of Violations: Immunity to Spoilers - 2

```
while num_samples < min_num_samples or (any([(err[1] - err[0]) > min_error for err in error_ranges])
and num_samples < max_num_samples):

    num_samples += 1
    prof = generate_profile(num_candidates, num_voters)

    for vm in voting_methods:
        has_violation[vm.name].append(axiom.has_violation(prof, vm))

    error_ranges = [binomial_confidence_interval(has_violation[vm.name])
                    if len(has_violation[vm.name]) > 0 else (0, np.inf)
                    for vm in voting_methods]

print(f"For {num_samples} samples:\n")

for vm in voting_methods:
    print(f"The percentage of violations of {axiom.name} for {vm.name} is {np.mean
    (elect_condorcet_winner[vm.name])}")
    err = binomial_confidence_interval(elect_condorcet_winner[vm.name])
    print(f"\tError: {err[1]-err[0]}")
```

Percentage of Violations: Immunity to Spoilers - Output

```
[131]  ✓  2m 30.6s
...  For 28445 samples:


   The percentage of violations of Immunity to Spoilers for Plurality is 0.6258457374830853
           Error: 0.04997642712974326
   The percentage of violations of Immunity to Spoilers for Borda is 0.8640054127198917
           Error: 0.03561495608588128
   The percentage of violations of Immunity to Spoilers for Instant Runoff is 0.9242219215155616
           Error: 0.027667207860292087
```

Thank you!

https://pref-voting.readthedocs.io/

https://pypi.org/project/pref-voting/

https://github.com/voting-tools/pref_voting/tree/main/howto

https://pref.tools/pref_voting/