Charlie Murphy
6 August 2020

Bias Reduction

Description

This routine seeks is an aggressive parallel version of the Republican or Democrat seat share increasing program that already exists. No changes have been made to those programs with the exception of putting them into the aggressive parallel format and adding the improved population constraint.

Files

`chain_xtended_polish_lt_fracs_dem.py`	Markov Chain that has been modified to only accept plans that increase the number of fractional Democratic seats won
`bias_input.py`	Input file for both versions
`republican_bias_parallel_agressive.py`	Version that increases Republican bias
`democratic_bias_parallel_agressive.py`	Version that increases Democratic bias

Dependencies

- `calc_fracwins_comp.py`
- `chain_xtended_polish_lt_fracs.py`
- `conditional_dump.py`
- `get_electionsinfo.py`
- `pop_constraint.py`
- `splice_assignment_fn.py`

Set Up

`chain_xtended_polish_lt_fracs.py` and `chain_xtended_polish_lt_fracs_dem.py` must be placed in the gerrychain directory. The following line of code must be added to `__init__.py`:

```
from .chain_xtended_polish_lt_fracs import MarkovChain_xtended_polish_lt_fracs
from .chain_xtended_polish_lt_fracs_dem import MarkovChain_xtended_polish_lt_fracs_dem
```

`bias_input.py` must be placed in the folder `input_templates`. All other files and dependencies should be placed in the `redistricting` folder.

Inputs

`my_electiondatafile`	The path to the shapefile with the election data
`ex_dist_name`	The name of the txt or csv file that contains the initial seed; this must be placed in the `example_districts` folder
`state`	The two-letter abbreviation for the state; this is used to find the correct election composite in `get_electionsinfo.py`

`popkey`	The name of the field in `my_electionfile` with the population data
`geotag`	The name of the field in `ex_dist_name` that corresponds to the GEOID of the precincts
`my_apportionent`	The name of the field in `ex_dist_name` that corresponds to the district each precinct is assigned to
`markovchainlength`	The number of iterations the Markov Chain should go through; in the parallel version, this is the number of iterations for each processor
`poptol`	The percent from ideal that each district may be allowed to differ when merged; for best results make this approximately half of `max_pop_deviation`
`maxsplits`	The maximum allowed number of county splits
`electionvol`	The win volatility
`boundary_margin`	The percentage change in the number of cut edges that will be allowed at accepted each step of the Markov Chain
`max_pop_deviation`	The maximum allowed population deviation as calculated in Dave’s
`seat_min`	The minimum number of Democratic seats allowed
`my_electionproxy`	A single election that may be used as a proxy for partisan intent (while the code still needs this parameter to run, all of the partisan calculations I believe are done using the election composite)
`poolsize`	The number of processors
`time_interval`	How often each processor checks the status of the other processors (in seconds)

Combined Workflow

Description

This routine balances population, eliminates fracks, and smooths plans while making minimal changes to the number of county splits and partisan intent of the plan. First the population is balanced in stage 0. This is the only stage where county splits are allowed and then only 1 between any pair of merged districts. Then in stage 1, fracks are eliminated. In both the first two stages plans are accepted if the decrease the number of cut edges. If gerrychain takes more than the number of tries specified by the cutoff parameter, then the number of cut edges is allowed to increase by the percentage specified by the margin parameter. In stage 2, plans are only accepted if they reduce the number of cut edges. In all stages, the partisan intent of the plan is preserved as no new plan will be accepted if it deviates by more than the win_margin parameter from the previous plan in terms of number of fractional Democratic seats.

Files

`chain_xtended_combined_workflow.py`	Markov Chain that has been modified to only accept plans the meet the required criteria in each stage
`combined_input.py`	Input file for both the single-processor and parallel versions
`combined_workflow.py`	Single processor version of the routine
`combined_workflow_parallel_agressive.py`	Aggressive parallel version of the routine

Dependencies

- `calc_fracwins_comp.py`
- `district_list.py`
- `fracking.py`
- `get_electionsinfo.py`
- `pop_constraint.py`
- `splice_assignment_fn.py`
- `tree_proposals.py`
- `total_splits.py`

Set Up

`chain_xtended_combined_workflow.py` must be placed in the gerrychain directory. The following line of code must be added to `__init__.py`:

```
from .chain_xtended_combined_workflow import MarkovChain_xtended_combined_workflow
```

`tree_proposals.py` must be placed in the `proposals` folder of the gerrychain directoy and following line of code should replace line 2 in that `__init__.py`:

```
from .tree_proposals import recom, reversible_recom, ReCom, recom_frack, recom_merge
```

`combined_input.py` must be placed in the folder `input_templates`. All other files and dependencies should be placed in the `redistricting` folder.

Inputs

`my_electiondatafile` The path to the shapefile with the election data

`ex_dist_name` The name of the txt or csv file that contains the initial seed; this must be placed in the `example_districts` folder

`state` The two-letter abbreviation for the state; this is used to find the correct election composite in `get_electionsinfo.py`

`popkey` The name of the field in `my_electionfile` with the population data

`geotag` The name of the field in `ex_dist_name` that corresponds to the GEOID of the precincts

`my_apportionent` The name of the field in `ex_dist_name` that corresponds to the district each precinct is assigned to

`markovchainlength` The number of iterations the Markov Chain should go through; in the parallel version, this is the number of iterations for each processor

`poptol` The percent from ideal that each district may be allowed to differ when merged; for best results make this approximately half of `max_pop_deviation`

`electionvol` The win volatility

`max_pop_deviation` The maximum allowed population deviation as calculated in Dave's

`win_margin` The percentage change in the fractional seat share that will be allowed at accepted each step of the Markov Chain

`cutoff` The number of unsuccessful tries before the smoothing bound is loosened

`margin` The percentage increase in the number of cut edges that is allowed every time the cutoff is met

`poolsize*` The number of processors

`time_interval*` How often each processor checks the status of the other processors (in seconds)

**parallel version only*

County Split Reduction

Description

This routine seeks to reduce the number of county splits within a redistricting plan without exceeding the maximum population deviation. This routine does not limit changes to the number of cut edges or the fractional seats won. Of note, like previous versions, reducing the number of county splits is a timing consuming process.

Files

`county_splits_input.py`	Input file
`county_splits_reduction_parallel_agressive.py`	Aggressive parallel version of the routine

Dependencies

- `chain_xtended.py`
- `district_list.py`
- `get_electionsinfo.py`
- `pop_constraint.py`
- `splice_assignment_fn.py`
- `total_splits.py`

Set Up

`chain_xtended.py` must be placed in the gerrychain directory. The following line of code must be added to `__init__.py`:

```
from .chain_xtended import MarkovChain_xtended
```

`county_splits_input.py` must be placed in the folder `input_templates`. All other files and dependencies should be placed in the `redistricting` folder.

Inputs

`my_electiondatafile`	The path to the shapefile with the election data
`ex_dist_name`	The name of the txt or csv file that contains the initial seed; this must be placed in the `example_districts` folder
`state`	The two-letter abbreviation for the state; this is used to find the correct election composite in `get_electionsinfo.py`
`popkey`	The name of the field in `my_electionfile` with the population data
`geotag`	The name of the field in `ex_dist_name` that corresponds to the GEOID of the precincts
`my_apportionent`	The name of the field in `ex_dist_name` that corresponds to the district each precinct is assigned to

`markovchainlength`	The number of iterations the Markov Chain should go through; this is the number of iterations for each processor
`poptol`	The percent from ideal that each district may be allowed to differ when merged; for best results make this approximately half of `max_pop_deviation`
`max_pop_deviation`	The maximum allowed population deviation as calculated in Dave’s
`poolsize`	The number of processors
`time_interval`	How often each processor checks the status of the other processors (in seconds)

Fracking

Description

This file provides a number of useful functions that provide information about fracking as well as other information about county splits.

Set Up

Place `fracking.py` in the `redistricting` folder.

Functions

`get_county_field` This function takes in a partition and returns the county field.

`fracking` This function takes in a partition and returns the number of fracks. A district is considered fracked if it has at least two discontinuous portions within a county.

`fracking_total_splits` This function takes in a partition and returns the number of fracks and the number of county splits. If you are trying to get both pieces of information, this is more efficient than calling `fracking` and `total_splits`.

`fracking_merge` This function takes in a partition and two districts. It returns the number fracks, number of counties split between those two districts, and the number of total county splits. This is used to tell whether two districts being merged split more than one county.

`get_intersections` This helper function takes in a partition and the county field and returns a partition split by both districts and counties.

`num_pieces` This helper function takes in a partition and the county field and returns the total number of pieces when the map is split by both districts and counties.

`get_fracks` This function takes in a partition, the county field, and the result of `get_intersections`. It returns the district number of 2 districts that share a county where at least one of them is fracked.

`get_fracked_subgraph` This function takes in a partition and returns the subgraph of two districts in a county where at least one of them is fracked.

Fracking Reduction

Description

This routine seeks to eliminate fracks within a redistricting plan without adding any additional county splits and maintaining the fractional democratic seats. This is achieved by combining the portions two districts, where one of them has a frack, that lie within the fracked county. Also note that in situations where the fracked portions of a district exclusively border different districts within the same county that this program will get stuck.

Files

`chain_xtended_fracking.py`	Markov Chain that has been modified to only accept plans that reduce the number of fracks
`fracking_input.py`	Input file for both the single-processor and parallel versions
`fracking_reduction.py`	Single processor version of the routine
`fracking_reduction_parallel_agressive.py`	Aggressive parallel version of the routine

Dependencies

- `calc_fracwins_comp.py`
- `district_list.py`
- `fracking.py`
- `get_electionsinfo.py`
- `pop_constraint.py`
- `splice_assignment_fn.py`
- `tree.py`
- `tree_proposals.py`

Set Up

`chain_xtended_fracking.py` and `tree.py` must be placed in the gerrychain directory. The following line of code must be added to `__init__.py`:

```
from .chain_xtended_fracking import MarkovChain_xtended_fracking
```

`tree_proposals.py` must be placed in the `proposals` folder of the gerrychain directoy and following line of code should replace line 2 in that `__init__.py`:

```
from .tree_proposals import recom, reversible_recom, ReCom, recom_frack, recom_merge
```

`fracking_input.py` must be placed in the folder `input_templates`. All other files and dependencies should be placed in the `redistricting` folder.

Inputs

`my_electiondatafile`	The path to the shapefile with the election data
`ex_dist_name`	The name of the txt or csv file that contains the initial seed; this must be placed in the `example_districts` folder
`state`	The two-letter abbreviation for the state; this is used to find the correct election composite in `get_electionsinfo.py`
`popkey`	The name of the field in `my_electionfile` with the population data
`geotag`	The name of the field in `ex_dist_name` that corresponds to the GEOID of the precincts
`my_apportionent`	The name of the field in `ex_dist_name` that corresponds to the district each precinct is assigned to
`markovchainlength`	The number of iterations the Markov Chain should go through; in the parallel version, this is the number of iterations for each processor
`poptol`	The percent from ideal that each district portion may be allowed to differ when merged; since this program does not combine whole districts, a higher poptol may be help make the program run faster
`win_margin`	The percentage change in the fractional seat share that will be allowed at accepted each step of the Markov Chain
`electionvol`	The win volatility
`boundary_margin`	The percentage change in the number of cut edges that will be allowed at accepted each step of the Markov Chain
`max_pop_deviation`	The maximum allowed population deviation as calculated in Dave’s
`poolsize*`	The number of processors
`time_interval*`	How often each processor checks the status of the other processors (in seconds)

**parallel version only*

Pop Constraint

Description

This file includes functions that calculate the population deviation as it is done in Dave's and includes a population constraint that can be used to only accept plans that do not exceed a certain population deviation.

Set Up

Place `pop_constraint.py` in the `redistricting` folder.

Functions

`pop_deviation`	This function takes in a partition and returns the population deviation as calculated by Dave's.
`pop_constraint`	This function takes in a maximum population deviation and returns a lambda function that can be used as a constraint in gerrychain.
`get_edge`	This function takes in a partition and returns an edge that is between the two bordering districts with the greatest difference in population.
`get_districts`	This function takes in a partition and returns a tuple of the two bordering districts with the greatest difference in population.
`get_pop_subgraph`	This function takes in a partition and returns a subgraph of a county that is split between the bordering districts with the greatest difference in population. I did not end up finding this useful in any of my routines, but to my knowledge it does work.

Population Balance

Description

This routine seeks to balance the population balance so as to reduce the population deviation as calculated in Dave's. This is achieved by at each step combining the bordering districts with the greatest difference in population/ County splits are limited to one between any pair of districts. Changes to the number of cut edges and the fractional Democratic seats won are restricted. Of note, this program is very good at eliminating large population deviations, however, it does run considerably slower when trying to reduce the population deviation below 2%.

Files

`chain_xtended_pop_balance.py`	Markov Chain that has been modified to only accept plans that reduce the population deviation
`pop_balance_input.py`	Input file for both all versions
`population_balance.py`	Single processor version of the routine
`population_balance_parallel.py`	Parallel version of the routine
`population_balance_parallel_agressive.py`	Aggressive parallel version of the routine

Dependencies

- `calc_fracwins_comp.py`
- `district_list.py`
- `fracking.py`
- `get_electionsinfo.py`
- `pop_constraint.py`
- `splice_assignment_fn.py`
- `tree_proposals.py`

Set Up

`chain_xtended_pop_balance.py` must be placed in the gerrychain directory. The following line of code must be added to `__init__.py`:

```
from .chain_xtended_pop_balance import MarkovChain_xtended_pop_balance
```

`tree_proposals.py` must be placed in the `proposals` folder of the gerrychain directoy and following line of code should replace line 2 in that `__init__.py`:

```
from .tree_proposals import recom, reversible_recom, ReCom, recom_frack, recom_merge
```

`pop_balance_input.py` must be placed in the folder `input_templates`. All other files and dependencies should be placed in the `redistricting` folder.

Inputs

`my_electiondatafile` The path to the shapefile with the election data

`ex_dist_name` The name of the txt or csv file that contains the initial seed; this must be placed in the `example_districts` folder

`state` The two-letter abbreviation for the state; this is used to find the correct election composite in `get_electionsinfo.py`

`popkey` The name of the field in `my_electionfile` with the population data

`geotag` The name of the field in `ex_dist_name` that corresponds to the GEOID of the precincts

`my_apportionent` The name of the field in `ex_dist_name` that corresponds to the district each precinct is assigned to

`markovchainlength` The number of iterations the Markov Chain should go through; in the parallel versions, this is the number of iterations for each processor

`poptol` The percent from ideal that each district portion may be allowed to differ when merged; in my experience setting this 0.01 yields the best results

`win_margin` The percentage change in the fractional seat share that will be allowed at accepted each step of the Markov Chain

`electionvol` The win volatility

`boundary_margin` The percentage change in the number of cut edges that will be allowed at accepted each step of the Markov Chain

`poolsize*` The number of processors

`time_interval**` How often each processor checks the status of the other processors (in seconds)

**parallel versions only*

***aggressive parallel version only*

Proportional Seats Deviation

Description

This file provides functions to find the proportional seats deviation.

Set Up

Place `proportional_seats_deviation.py` in the `redistricting` folder.

Functions

`prop_dev` This function takes in a partition, election composite, win volatility, and the number of proportional seats and return the deviation of the fractional seats won in the partition to the number of proportional seats.

`prop_frac_dev` This function takes in a partition, election composite, win volatility, and the Democratic vote share return the deviation of the fractional seats won in the partition to the number of proportional seats as calculated as the Democratic vote share multiplied by the total number of seats.

Proportional Seats

Description

This routine seeks to get the fractional number of Democratic seats won as close to proportional as possible. There are two versions. The first, `proportional_seats_parallel_agressive.py`, defines proportional seats as the whole number of seats closest to the number of seats times the Democratic vote share. The second, `proportional_fractional_seats_parallel_agressive.py`, defines proportional seats as the fractional number of seats equivalent to the number of seats times the Democratic vote share. County splits are limited to not more than one between every pair of bordering districts. There are no restrictions on the number of cut edges.

Files

`chain_xtended_prop_dev.py`	Markov Chain that has been modified to only accept plans that reduce difference between the fractional seats won and the whole number seats closest to the proportional fractional seats.
`chain_xtended_prop__frac_dev.py`	Markov Chain that has been modified to only accept plans that reduce difference between the fractional seats won and the closest to the proportional fractional seats.
`prop_dev_input.py`	Input file for `proportional_seats_parallel_agressive.py`
`prop_frac_dev_input.py`	Input file for `proportional_fractional_seats_parallel_agressive.py`
`proportional_seats_parallel_agressive.py`	See description
`proportional_fractional_seats_parallel_agressive.py`	See description

Dependencies

- `calc_fracwins_comp.py`
- `district_list.py`
- `fracking.py`
- `get_electionsinfo.py`
- `pop_constraint.py`
- `proportional_seats_deviation.py`
- `splice_assignment_fn.py`
- `total_splits.py`
- `tree_proposals.py`

Set Up

chain_xtended_prop_dev.py and chain_xtended_prop_frac_dev.py must be placed in the gerrychain directory. The following lines of code must be added to __init__.py:

```
                from .chain_xtended_prop_dev import MarkovChain_xtended_prop_dev
           from .chain_xtended_prop_frac_dev import MarkovChain_xtended_prop_frac_dev
```

tree_proposals.py must be placed in the proposals folder of the gerrychain directoy and following line of code should replace line 2 in that __init__.py:

```
      from .tree_proposals import recom, reversible_recom, ReCom, recom_frack, recom_merge
```

prop_dev_input.py and prop_frac_dev_input must be placed in the folder input_templates. All other files and dependencies should be placed in the redistricting folder.

Inputs

my_electiondatafile	The path to the shapefile with the election data
ex_dist_name	The name of the txt or csv file that contains the initial seed; this must be placed in the example_districts folder
state	The two-letter abbreviation for the state; this is used to find the correct election composite in get_electionsinfo.py
popkey	The name of the field in my_electionfile with the population data
geotag	The name of the field in ex_dist_name that corresponds to the GEOID of the precincts
my_apportionent	The name of the field in ex_dist_name that corresponds to the district each precinct is assigned to
markovchainlength	The number of iterations the Markov Chain should go through; in the parallel version, this is the number of iterations for each processor
poptol	The percent from ideal that each district may be allowed to differ when merged; for best results make this approximately half of max_pop_deviation
electionvol	The win volatility
max_pop_deviation	The maximum allowed population deviation as calculated in Dave’s
proportional_seats*	The whole number seats closest to the proportional fractional seats
vote_share**	The Democratic vote share
poolsize	The number of processors
time_interval	How often each processor checks the status of the other processors (in seconds)

*prop_dev_input.py only

**prop_frac_dev_input.py only

Smoothing

Description

This routine seeks to reduce the number of cut edges within a redistricting plan without exceeding the maximum population deviation. County splits are limited to one between any pair of districts. This routine does not limit changes to the number fractional seats won.

Files

`chain_xtended_smoothing.py`	Markov Chain that has been modified to only accept plans that reduce the number of cut edges
`smoothing_input.py`	Input file
`smoothing_parallel_agressive.py`	Aggressive parallel version of the routine

Dependencies

- `district_list.py`
- `fracking.py`
- `get_electionsinfo.py`
- `pop_constraint.py`
- `splice_assignment_fn.py`
- `total_splits.py`
- `tree_proposals.py`

Set Up

`chain_xtended_smoothing.py` must be placed in the gerrychain directory. The following line of code must be added to `__init__.py`:

```
from .chain_xtended_smoothing import MarkovChain_xtended_smoothing
```

`tree_proposals.py` must be placed in the `proposals` folder of the gerrychain directoy and following line of code should replace line 2 in that `__init__.py`:

```
from .tree_proposals import recom, reversible_recom, ReCom, recom_frack, recom_merge
```

`smoothing_input.py` must be placed in the folder `input_templates`. All other files and dependencies should be placed in the `redistricting` folder.

Inputs

`my_electiondatafile`	The path to the shapefile with the election data
`ex_dist_name`	The name of the txt or csv file that contains the initial seed; this must be placed in the `example_districts` folder

`state`	The two-letter abbreviation for the state; this is used to find the correct election composite in `get_electionsinfo.py`
`popkey`	The name of the field in `my_electionfile` with the population data
`geotag`	The name of the field in `ex_dist_name` that corresponds to the GEOID of the precincts
`my_apportionent`	The name of the field in `ex_dist_name` that corresponds to the district each precinct is assigned to
`markovchainlength`	The number of iterations the Markov Chain should go through; this is the number of iterations for each processor
`poptol`	The percent from ideal that each district may be allowed to differ when merged; for best results make this approximately half of `max_pop_deviation`
`max_pop_deviation`	The maximum allowed population deviation as calculated in Dave’s
`poolsize`	The number of processors
`time_interval`	How often each processor checks the status of the other processors (in seconds)

Tree

Description

A single additional function, `recursive_tree_part_recom`, has been added to this file. This function works exactly the same as the function `recursive_tree_part` with exception that it takes the parameter `pop_target` as a tuple instead of a float. This is so that districts or district portions can be merged where the target population for each is not the same. This is used by the fracking reduction program since it only combines the portion of each district that lie within the fracked county.

Set Up

Place `tree.py` in the gerrychain directory

Tree Proposals

Description

This file adds two additional functions to the addition tree proposals file. This allows for versions of ReCom where the districts merged are chosen in ways other than randomly selecting a cut edge. Only the functions add are documented here.

Set Up

Place `tree_proposals.py` in the `proposals` folder of the gerrychain directoy and following line of code should replace line 2 in that `__init__.py`:

```
      from .tree_proposals import recom, reversible_recom, ReCom, recom_frack, recom_merge
```

Functions

`recom_merge` This version of ReCom works just like the normal version, except that the Markov Chain inputs both the partition and a tuple of the two districts to be merged.

`recom_frack` This version of ReCom takes in the partition and a subgraph from the Markov Chain. Currently this program is only used by the fracking reduction routine, but it can be applied to any situation that meets the above parameters.