iowa

May 6, 2021

https://github.com/Toblerity/Fiona/issues/944

[133]:
```
import fiona
```

[134]:
```
import matplotlib.pyplot as plt
from gerrychain import (GeographicPartition, Partition, Graph, MarkovChain,
                        proposals, updaters, constraints, accept, Election)
from gerrychain.proposals import recom
from functools import partial
import pandas
```

[135]:
```
import maup
import numpy as np
import geopandas
import matplotlib.pyplot as plt
from gerrychain import (GeographicPartition, Partition, Graph, MarkovChain,
                        proposals, updaters, constraints, accept, Election)
from gerrychain.updaters import Tally, cut_edges, exterior_boundaries,␣
↪exterior_boundaries_as_a_set
from networkx import is_connected, connected_components
```

[136]:
```
graph = Graph.from_file("./IA_counties.zip", ignore_errors=True)
precincts = geopandas.read_file("./IA_counties.zip")
```

```
C:\Users\darre\anaconda3\envs\gerrychain\lib\site-
packages\gerrychain\graph\graph.py:162: UserWarning: Geometry is in a geographic
CRS. Results from 'area' are likely incorrect. Use 'GeoSeries.to_crs()' to re-
project geometries to a projected CRS before this operation.

  areas = df.geometry.area.to_dict()
```

[137]:
```
components = list(connected_components(graph))
biggest_component_size = max(len(c) for c in components)
problem_components = [c for c in components if len(c) != biggest_component_size]
print(is_connected(graph))
for component in problem_components:
    for node in component:
        graph.remove_node(node)
```

```
print(is_connected(graph))
```

```
True
True
```

[138]:
```
from gerrychain.constraints.contiguity import contiguous_components, contiguous
from gerrychain import Partition
```

[139]:
```
election = Election("PRES16", {"Dem": "PRES16D", "Rep": "PRES16R"})

initial_partition = GeographicPartition(
    graph,
    assignment="CD",
    updaters={
        "cut_edges": cut_edges,
        "population": Tally("TOTPOP", alias="population"),
        "PRES16": election
    }
)
```

[140]:
```
contiguous_components(initial_partition)
```

[140]:
```
{'1': [<Graph [20 nodes, 31 edges]>],
 '4': [<Graph [39 nodes, 73 edges]>],
 '3': [<Graph [16 nodes, 26 edges]>],
 '2': [<Graph [24 nodes, 45 edges]>]}
```

[141]:
```
for district, pop in initial_partition["population"].items():
    print("District {}: {}".format(district, pop))
```

```
District 1: 761548
District 4: 761571
District 3: 761612
District 2: 761624
```

[142]:
```
sum_population = sum(initial_partition["population"].values())
ideal_population = sum_population / len(initial_partition)

# We use functools.partial to bind the extra parameters (pop_col, pop_target,␣
 ↪epsilon, node_repeats)
# of the recom proposal.
proposal = partial(recom,
                   pop_col="TOTPOP",
                   pop_target=ideal_population,
                   epsilon=.02,
                   node_repeats=2
                  )
```

[143]:
```
compactness_bound = constraints.UpperBound(
    lambda p: len(p["cut_edges"]),
    2*len(initial_partition["cut_edges"])
)

pop_constraint = constraints.
 ↪within_percent_of_ideal_population(initial_partition, .02)
```

[144]:
```
from gerrychain import MarkovChain
from gerrychain.constraints import single_flip_contiguous, contiguous
from gerrychain.proposals import propose_random_flip
from gerrychain.accept import always_accept
steps = 1000
chain = MarkovChain(
    proposal=proposal,
    constraints=[single_flip_contiguous, compactness_bound, pop_constraint],
    accept=always_accept,
    initial_state=initial_partition,
    total_steps=steps
)
```

[145]:
```
def district_diff(partition1, partition2):
    percentage_change = []
    for (district1, graph1), (district2, graph2) in␣
 ↪zip(contiguous_components(partition1).items(),␣
 ↪contiguous_components(partition2).items()):
        if district1 == district2:
            set1 = set(graph1[0].nodes)
            set2 = set(graph2[0].nodes)
            if set1 != set2:
                set_diff1 = set1 - set2
                set_diff2 = set2 - set1
                set_intersection = set1 & set2
                diff = len(set_intersection)/len(set1)
                if diff > 1:
                    percentage_change.append(0)
                else:
                    percentage_change.append(diff)
            else:
                percentage_change.append(1)
    return percentage_change
```

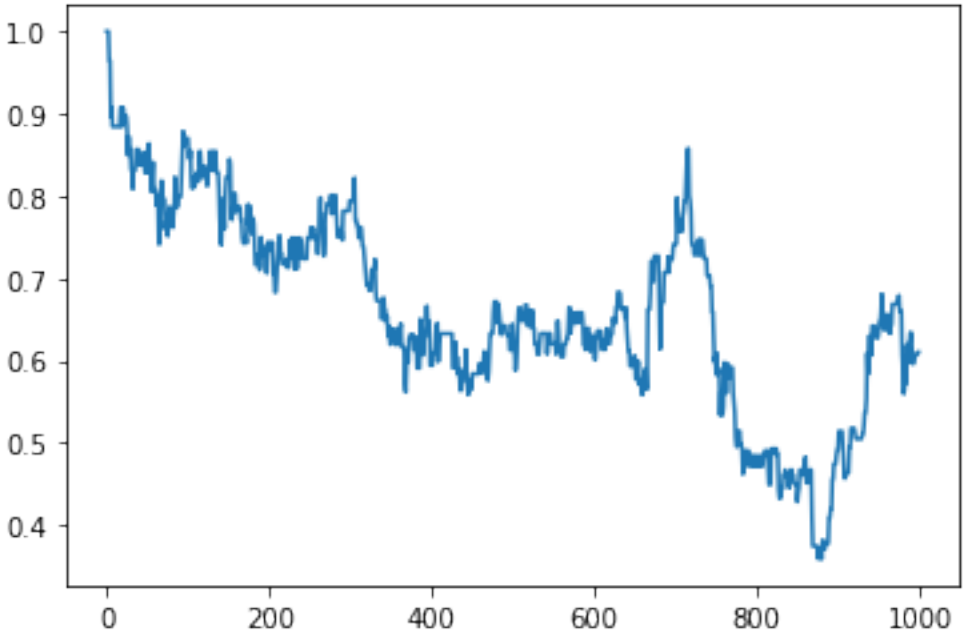
[146]:
```
last1 = None
best_partition = None
best_partition_similarity = 1
district_percent_change_per_partition = []
for partition in chain.with_progress_bar():
```

```
    district_differences = district_diff(initial_partition, partition)
    district_percent_change_per_partition.append(district_differences)
    last1 = partition
    partition_similarity = np.mean(district_differences)
    if best_partition_similarity > partition_similarity:
        best_partition_similarity = partition_similarity
        best_partition = partition
```

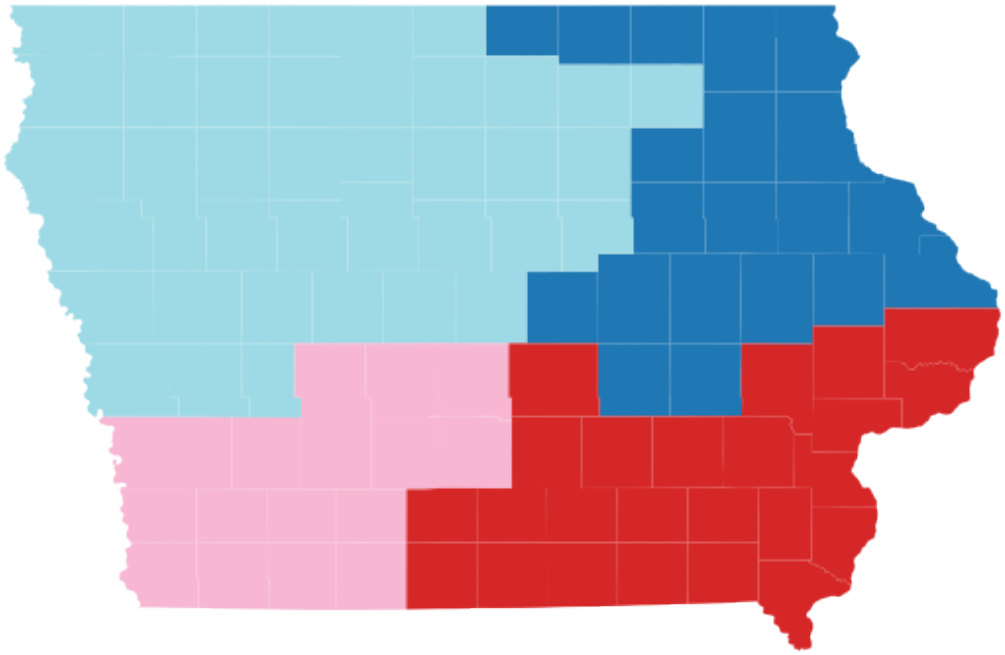
```
100%|      | 1000/1000 [04:50<00:00,  3.44it/s]
```

[148]:
```
import matplotlib.pyplot as plt
y = np.mean(district_percent_change_per_partition, axis=1)
plt.plot(y)
```

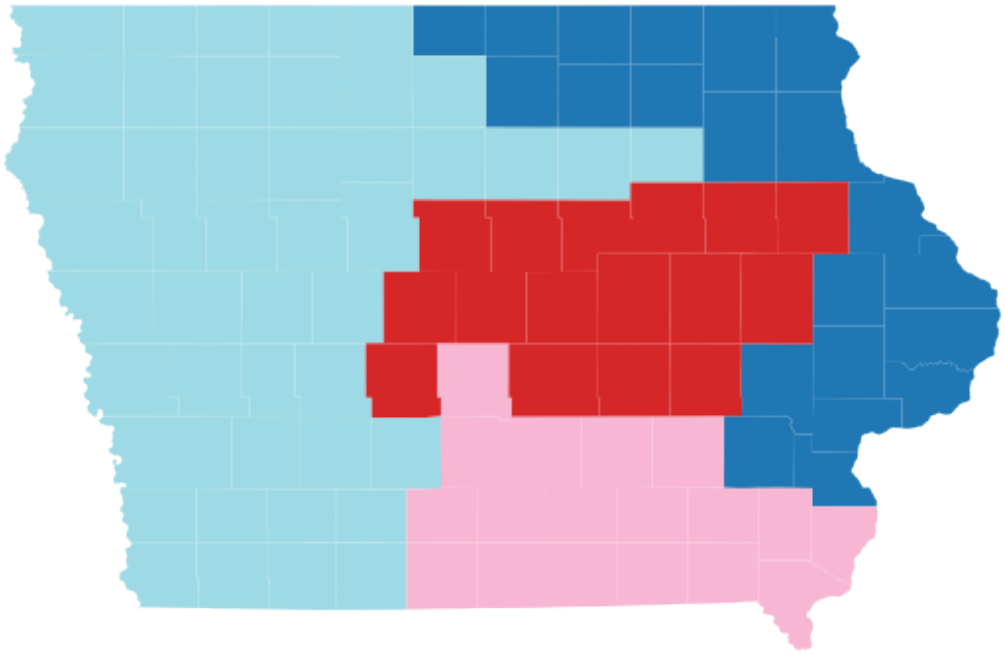
[148]:
```
[<matplotlib.lines.Line2D at 0x21046a172e0>]
```



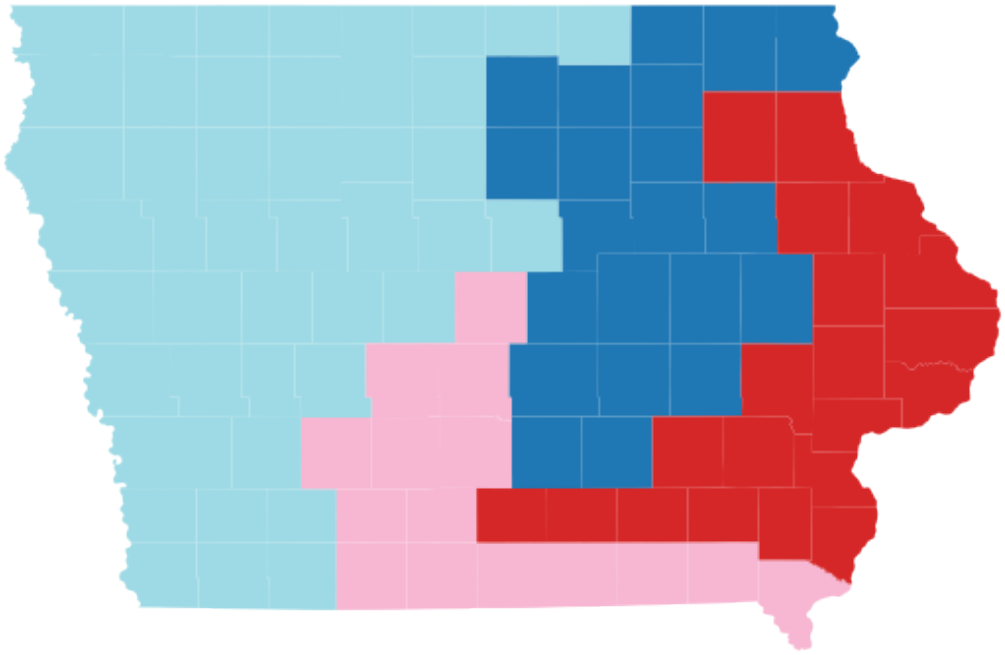
[149]:
```
initial_partition.plot(figsize=(10, 10), cmap="tab20")
plt.axis('off')
plt.show()
```



[150]:
```
best_partition.plot(figsize=(10, 10), cmap="tab20")
plt.axis('off')
plt.show()
```



[151]:
```
last1.plot(figsize=(10, 10), cmap="tab20")
plt.axis('off')
plt.show()
```



[152]:
```
    for (district1, graph1), (district2, graph2) in␣
↪zip(contiguous_components(initial_partition).items(),␣
↪contiguous_components(last1).items()):
        if district1 == district2:
            set1 = set(graph1[0].nodes)
            set2 = set(graph2[0].nodes)
            set_diff1 = set1 - set2
            set_diff2 = set2 - set1
            print(set_diff1)
            print(set_diff2)
```

```
{4, 5, 40, 15, 81, 89, 27, 62}
{96, 33, 75, 45, 80, 51, 22, 54, 29}
{96, 33, 75, 45, 80, 29, 95}
{97, 67, 7, 40, 43, 77, 46, 15, 55, 58, 30}
{97, 67, 7, 43, 77, 46, 55, 58, 30}
{34, 35, 36, 68, 48, 28, 94, 95}
{34, 35, 36, 68, 48, 51, 22, 54, 28, 94}
{4, 5, 81, 89, 27, 62}
```

[153]:
```
last1
```

[153]: <GeographicPartition [4 parts]>