Racial Preferences and School Segregation

Aya Konishi

December 11, 2020

1 Introduction

Since the landmark *Brown v. Board* (1954) decision, which mandated racial desegregation of public schools in the United States, levels of school segregation have fluctuated. The percentage of Black students in the South who attend schools that are at least 50% White peaked in the late 1980's at 44%, and decreased in the 2000's to 23% (Orfield et al., 2014). Also, the percentage of Black students in predominantly minority schools has changed very little (from around 75% in the 1960's to around 70% in the 1990's) (Orfield, 1983).

Evidently, school segregation exists currently, but why is it a problem? The academic achievement gap for Black children increases as they spend time in segregated schools (Hanushek and Rivkin, 2009). Also, since public schools are partially funded through property taxes, a concentration of high-income families in a school district leads to increased funding for the school (Augenblick et al., 1997). There is a racial wealth gap; the median White family in 2016 had $146,984, while the median Black family had $3,557 and the median Latino family had $6,591 (Collins et al., 2018). This indicates that racially segregated schools lead to disproportionate school funding between children of color (especially Latino and Black children) and White children.

Segregation doesn't only exist in schools; it also appears in neighborhoods (Frey and Myers, 2002). Neighborhood racial segregation has been found to increase infant mortality, the rates of heart disease, and the incidence of low birth weight in Black people especially (Kramer and Hogue, 2009). In addition, it makes over-policing of Black communities easier, and also increases disparities in the quality of public services available to residents (Bass, 2001).

It is clear that segregation among schools and neighborhoods both have negative effects. However, since school districts tend to be larger than neighborhoods, and *Brown v. Board* mandated school desegregation in public schools, it is easier to focus on school desegregation than neighborhood desegregation. Since they both are crucial to eliminating racial disparities, we will investigate whether mandating school desegregation will also result in neighborhood desegregation.

This leads to the question: can policy regarding school district composition affect local neighborhood segregation? More specifically, if local governments

encourage diverse composition of school districts (by giving incentives for over-represented groups to leave), will this decrease neighborhood segregation?

2 Model

Previously, Schelling modeled racial segregation with agent-based models (Schelling, 2006). He showed that individuals' preferences about the racial composition of their neighbors could lead to intense segregation if their preferences are over a certain threshold. In Schelling's model, red and blue agents are distributed across a grid, and at each step, each agent assesses their local Moore neighborhood (the eight cells bordering the agent's current cell, as shown in Figure 3). If the color composition of their neighbors is over a certain threshold, the agent moves randomly to an empty location. Figure 1 shows how the Schelling model results in extreme segregation after 20 steps.

(a) Steps = 0 (b) Steps = 20

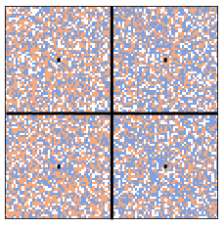
Figure 1: Schelling model with red agents' tolerance 0.5, blue agents' tolerance 0.8. Agents distributed evenly across 100×100 grid, with 20% of cells empty.

To incorporate schools into the model, we adapt the Schelling model by adding school district composition into the decision policy of each agent. The grid is separated into n school districts, with the cell in the center designated as the school. Figure 2 shows the grid divided into 4 and 9 school districts. The black lines are overlayed on the figure; the model allows for agents to be present on the borders between school districts.

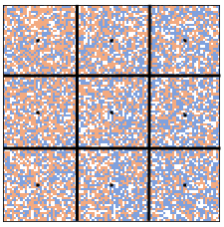
In the new model, agents make decisions with respect to two factors: neighborhood composition and school district composition. For each agent, we assign a value of 1 or 0 to the variables $move_{nbhd}$ and $move_{district}$. If the agent is inclined to move neighborhoods (within the same district), we set $move_{nbhd} = 1$, and 0 otherwise. If the agent is inclined to move districts, we set $move_{district} = 1$, and 0 otherwise.

2.1 Neighborhood Composition

All agents of the same color have the same preference of neighborhood composition. We designate p_{blue} as the minimum fraction of like-colored neighborhood agents that blue agents tolerate. In other words, if fewer than p_{blue} of



(a) 4 school districts



(b) 9 school districts

Figure 2: 100×100 grids divided into school districts, with the schools marked in black.

neighbors are blue, the blue agent will be inclined to move neighborhoods (so for these agents $move_{nbhd} = 1$). Similarly, if fewer than p_{red} of neighbors are red, the red agent will be inclined to move neighborhoods (so for these agents $move_{nbhd} = 1$). Otherwise, we set $move_{nbhd} = 0$.

2.2 School District Composition

In an effort to curb inter-school district segregation, we assume that the local government or school board incentivizes moving districts if there are too many agents of a certain kind in a school district. We define $frac_{blue}$ as the fraction of agents in a school district that are blue, and define $frac_{red}$ as the fraction of agents in a school district that are red. We let p_{school} be the tolerance of the local government of

$$|frac_{blue} - frac_{red}|$$

Thus, if $|frac_{blue} - frac_{red}| > p_{school}$, then the government will give incentives for certain agents to leave the district.

If

$$frac_{blue} - frac_{red} > p_{school}$$

, then the blue agents are inclined to leave the district, so for these agents we set $move_{district} = 1$. On the other hand, if

$$frac_{red} - frac_{blue} > p_{school}$$

, then the red agents are inclined to leave the district, and for these agents we set $move_{district} = 1$.

2.3 Weighing Districts and Neighborhoods

The following decision policy is used for each agent at each step:

- If $move_{nbhd} = 1$ and and $move_{district} = 0$, then the agent moves randomly to another neighborhood within their district.

- If $move_{nbhd} = 1$ and $move_{district} = 1$, they move to another district randomly.
- If $move_{nbhd} = 0$ and $move_{district} = 1$, they move to another district with probability p_{change}.
- If $move_{nbhd} = 0$ and $move_{district} = 0$, they don't move.

p_{change} is a measure of the desirability of the incentive that the government provides for residents to move to another district.

3 Data

We set $p_{blue} = 0.8$ and $p_{red} = 0.5$ using data from the Detroit Area Study, conducted in 1992 (Farley et al., 1994). It asked around 1500 Detroit residents to indicate how comfortable they would be in neighborhoods of different racial compositions, how likely they would move from the neighborhood, and how likely they would be to move into the neighborhood. Given that this information is from over 30 years ago, it might not be an accurate representation of current opinions. In addition, the data was collected in the Detroit Metropolitan Area, which probably is not a good representation of opinions across the United States. However, we use the data collected from this study because they collected random samples, and the questions asked simulated real situations as closely as possible.

4 Results

As mentioned previously, to investigate whether school district policies affect neighborhood segregation, we consider grids with 4 school districts, and grids with 9 school districts.

For grids with 4 school districts, we distribute the likelihood of blue, red, and empty cells as follows:

- 0.2 probability of an empty cell everywhere
- 0.45 probability of red cells, 0.35 probability of blue cells on the two left districts
- 0.35 probability of red cells, 0.45 probability of blue cells on the two right districts

For grids with 9 school districts, we distribute the likelihood of blue, red, and empty cells as follows:

- 0.2 probability of an empty cell everywhere
- 0.45 probability of red cells, 0.35 probability of blue cells on the three left districts

- 0.4 probability of red cells and 0.4 probability of blue cells on the three middle districts
- 0.35 probability of red cells, 0.45 probability of blue cells on the three right districts

We distribute the agents in this way because if all agents were distributed evenly across the grid,

$$|frac_{blue} - frac_{red}|$$

would be close to zero, so the school district make-up would not invoke policy for desegregation of schools. If there is a slight bias in distribution of agents in a school district, the school district make-up is more likely to violate a school districts p_{school} preference.

In addition, we set $p_{blue} = 0.8$, $p_{red} = 0.5$, and $p_{change} = 0.1$ in all simulations. We use grids of 100×100.

To measure whether neighborhood segregation changes with respect to minimizing school district segregation, we use various measures of segregation and isolation.

4.1 Neighborhood Segregation

One measure of segregation used is the fraction of like-colored neighbors in an agent's Moore neighborhood (as shown in Figure 3). We let i_{color} be the number of cells in the Moore neighborhood that are the same color as the ith cell. Let n be the number of cells in the grid. Thus, we compute the overall level of segregation in the grid as (White, 1986):

$$\frac{\sum_{i=1}^{n} \frac{i_{color}}{8}}{n}$$

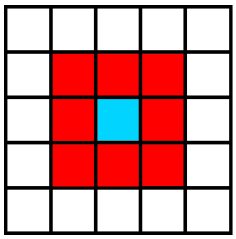
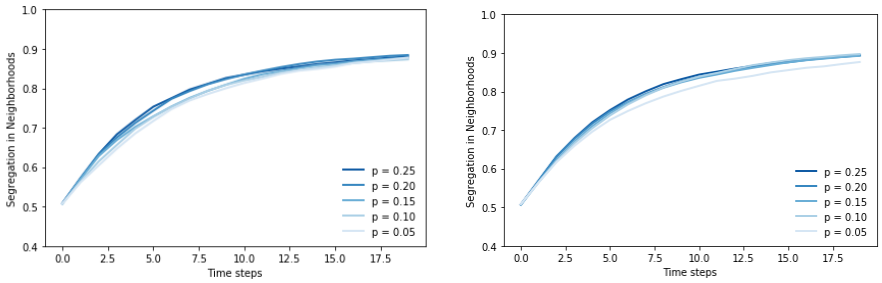


Figure 3: Moore Neighborhood

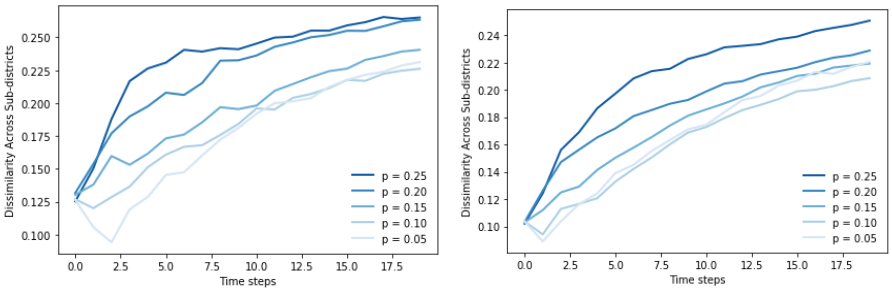
Figure 4a shows the level of segregation, averaged over 20 trials, over 20 steps in the model of a grid with 4 school districts. p_{school} is varied over the set $\{0.25, 0.20, 0.15, 0.1, 0.05\}$, each variation is plotted. Similarly, Figure 4b shows the same simulation, but with 9 school districts.



(a) 4 school districts (b) 9 school districts

Figure 4: Levels of neighborhood segregation over 20 time steps, averaged over 20 iterations. p_{school} varies and takes values among $\{0.25, 0.20, 0.15, 0.1, 0.05\}$.

4.2 Dissimilarity



(a) 4 school districts (b) 9 school districts

Figure 5: Levels of dissimilarity over 20 time steps, averaged over 20 iterations. p_{school} varies and takes values among $\{0.25, 0.20, 0.15, 0.1, 0.05\}$.

We want to consider segregation in local neighborhoods, but the previous segregation metric mentioned only considers the Moore neighborhood of an agent. Given that this neighborhood is small, especially compared to the size of the board, we consider larger local neighborhoods.

We consider these neighborhoods by using the metric of dissimilarity. This measure indicates the proportion of a group that would need to move districts to create a uniform distribution of red and blue agents. In our analysis, we let the districts be smaller than the school districts, but larger than Moore neighborhoods. Figure 6 illustrates the 36 sub-districts that were considered in finding the level of dissimilarity.

We let n be the number of sub-districts, r_i be the number of red agents in sub-district i, b_i be the number of blue agents in sub-district i, r_{tot} be the total number of red agents, and b_{tot} the total number of blue agents. We compute the overall level of dissimilarity as (White, 1986):

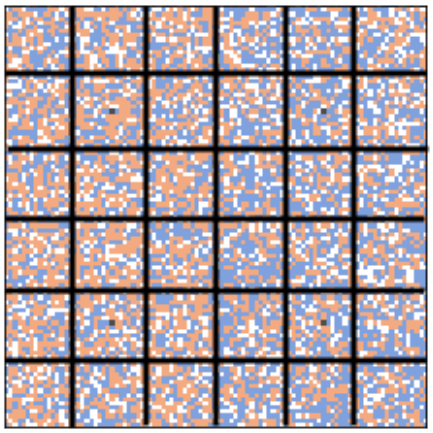
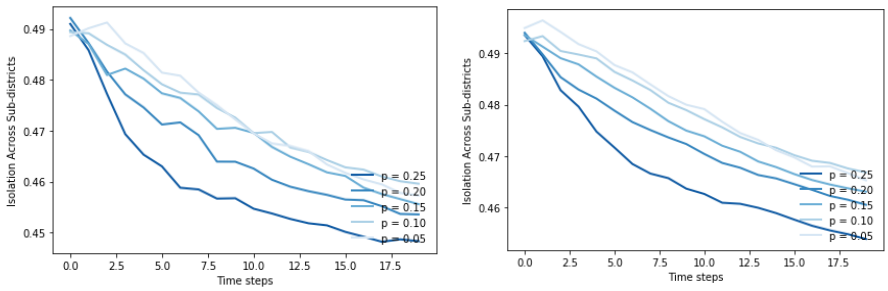


Figure 6: 100×100 grid divided into 36 sub-districts, which are considered in measures of dissimilarity and isolation.

$$\frac{1}{2}\sum_{i=1}^{n}\left|\frac{r_i}{r_{tot}} - \frac{b_i}{b_{tot}}\right|$$

Figure 5a shows the level of dissimilarity, averaged over 20 trials, over 20 steps in the model of a grid with 4 school districts. p_{school} is varied over the set $\{0.25, 0.20, 0.15, 0.1, 0.05\}$, each variation is plotted. Similarly, Figure 5b shows the same simulation, but with 9 school districts.

4.3 Isolation



(a) 4 school districts

(b) 9 school districts

Figure 7: Levels of isolation over 20 time steps, averaged over 20 iterations. p_{school} varies and takes values among $\{0.25, 0.20, 0.15, 0.1, 0.05\}$.

We also used a measure of isolation to measure the probability of a red agent

interacting with a blue agent in their sub-district. Similarly to calculating the level of dissimilarity, we categorized the grid into 36 sub-districts, as shown in Figure 6. Let n be the number of sub-districts in a grid. Let r_i be the number of red agents in sub-district i, b_i be the number of blue agents in sub-district i, r_{tot} be the number of red agents total, and pop_{tot} be the number of total agents in the grid. We calculate isolation in the following way (White, 1986):

$$\sum_{i=1}^{n} \frac{r_i}{r_{tot}} \frac{b_i}{pop_{tot}}$$

Figure 7a shows the level of isolation, averaged over 20 trials, over 20 steps in the model of a grid with 4 school districts. p_{school} is varied over the set $\{0.25, 0.20, 0.15, 0.1, 0.05\}$, each variation is plotted. Similarly, Figure 7b shows the same simulation, but with 9 school districts.

5 Conclusion

5.1 Discussion of findings

Evidently, with different metrics for segregation and isolation, the answer to our question varies. Firstly, with stricter school district or local government policies for integrating schools (smaller p_{school}) neighborhood segregation measured with Moore neighborhoods decreases slightly. With the case of 9 school districts, when $p_{school} = 0.05$, the level of segregation seems to be slightly smaller compared to other values of p_{school}.

Secondly, when considering dissimilarity, we see that higher p_{school} is, the higher the proportion of agents that would need to move districts to have a uniform distribution of agents. With $p_{school} = 0.05$, after 20 steps, the dissimilarity is around 0.225 with 4 school districts, and around 0.21 with 9 school districts. It seems that with more school districts, dissimilarity increases.

Thirdly, when looking at isolation, we see that if p_{school} is smaller, then the probability of a red agent interacting with a blue agent in their school district increases (meaning that isolation decreases). With 9 school districts, isolation is less than with 4 school districts.

Therefore, we can conclude that with more school districts, and lower p_{school}, there is less neighborhood segregation with respect to Moore neighborhoods, less isolation, and more dissimilarity. Thus, it seems that school districts mandating school integration does reduce neighborhood-level segregation.

5.2 Improvements

This model is extremely simple; many improvements and extensions could be made. Firstly, it has been shown that the 'tipping' that occurs in Schelling segregation model (where segregation emerges quickly depending on the preferences of the agents) can be attributed to the fact that their decision policy to move is a threshold function, rather than a continuous one (Bruch and Mare,

2006). One improvement could be to change the agents' decision policies into continuous ones that Bruch and Mare proposed, and see whether the results change.

Secondly, changing the initial conditions to be *already* segregated with respect to neighborhoods, and investigating whether a school district policy introduced later in the simulation could be more realistic. After all, the initial conditions of would never be relatively integrated with respect to neighborhoods (since the U.S. has rates of residential segregation) (Williams and Collins, 2016). In each simulation, the locations of the agents (and the level of segregation, however it is measured) stabilize after around 20 steps. This may not be the case if the initial conditions are intensely segregated with respect to neighborhoods.

Thirdly, it would be useful to have agents weigh their options of where to move, and not just move randomly to another neighborhood or district. For example, it would be more accurate if agents considered the composition of the destinations, the proximity to their current location, or the proximity to the school when deciding to move.

References

Augenblick, J. G., Myers, J. L., & Anderson, A. B. (1997). Equity and adequacy in school funding. *The Future of Children*, 63–78.

Bass, S. (2001). Policing space, policing race: Social control imperatives and police discretionary decisions. *Social Justice*, *28*(1 (83), 156–176.

Bruch, E. E., & Mare, R. D. (2006). Neighborhood choice and neighborhood change. *American Journal of sociology*, *112*(3), 667–709.

Collins, C., Asante-Muhammed, D., Hoxie, J., & Terry, S. (2018). Dreams deferred: How enriching the 1% widens the racial wealth divide. *Washington, DC: Institute for Policy Studies. https://ips-dc. org/wp-content/uploads/2019/01/IPS_RWD-Report_FINAL-1.15*, *19*.

Farley, R., Steeh, C., Krysan, M., Jackson, T., & Reeves, K. (1994). Stereotypes and segregation: Neighborhoods in the detroit area. *American Journal of Sociology*, *100*(3), 750–780. http://www.jstor.org/stable/2782404

Frey, W. H., & Myers, D. (2002). Neighborhood segregation in single-race and multirace america: A census 2000 study of cities and metropolitan areas. *Fannie Mae Foundation*.

Hanushek, E. A., & Rivkin, S. G. (2009). Harming the best: How schools affect the black-white achievement gap. *Journal of policy analysis and management*, *28*(3), 366–393.

Kramer, M. R., & Hogue, C. R. (2009). Is segregation bad for your health? *Epidemiologic reviews*, *31*(1), 178–194.

Orfield, G. (1983). Public school desegregation in the united states, 1968-1980.

Orfield, G., Frankenberg, E., Ee, J., & Kuscera, J. (2014). *Brown at 60: Great progress, a long retreat and an uncertain future*. Civil rights project/Proyecto derechos civiles.

Schelling, T. C. (2006). *Micromotives and macrobehavior*. WW Norton & Company.

White, M. J. (1986). Segregation and diversity measures in population distribution. *Population index*, 198–221.

Williams, D. R., & Collins, C. (2016). Racial residential segregation: A fundamental cause of racial disparities in health. *Public health reports*.

School Choice

December 10, 2020

This code is adapted from Think Complexity by Alan Downey

[60]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from utils import decorate, savefig

from scipy.signal import correlate2d
from Cell2D import Cell2D, draw_array

from matplotlib.colors import LinearSegmentedColormap
import math
```

[61]:
```
def locs_where(condition):
    """Find cells where a logical array is True.

    condition: logical array

    returns: list of location tuples
    """
    return list(zip(*np.nonzero(condition)))
```

[126]:
```
# make a custom color map
palette = sns.color_palette('muted')
colors = 'white', palette[1], palette[0], 'black'
cmap = LinearSegmentedColormap.from_list('cmap', colors)


class Schelling(Cell2D):
    """Represents a grid of Schelling agents."""

    options = dict(mode='same', boundary='wrap')

    kernel = np.array([[1, 1, 1],
                       [1, 0, 1],
                       [1, 1, 1]], dtype=np.int8)
```

```
    def __init__(self, n, p_r, p_b, p_school, p_change, num_districts):
        """Initializes the attributes.

        n: number of rows
        p_w: threshold on the fraction of similar neighbors for red agents
        p_r: threshold on the fraction of similar neighbors for blue agents
        p_school: if a is the fraction of blue agents in the district, b is the␣
↪fraction of red agents, p_school is the threshold for |a - b|
        p_change: if the agent likes their neighborhood, but is inclined to␣
↪move districts, they will move with p_change probability
        """

        self.n = n

        self.p_r = p_r
        self.p_b = p_b
        self.p_school = p_school
        self.p_change = p_change

        self.num_districts = num_districts

        # 0 is empty, 1 is red, 2 is blue, 3 is school
        choices = np.array([0, 1, 2], dtype=np.int8)
        probs = [0.2, 0.4, 0.4]

        if num_districts == 4:
            left = np.random.choice(choices, (n, math.floor(n/2)), p = [0.2, 0.
↪45, 0.35])
            right = np.random.choice(choices, (n, n - math.floor(n/2)), p = [0.
↪2, 0.35, 0.45])
            self.array = np.zeros((n,n))

            self.array[:, 0:math.floor(n/2)] = left
            self.array[:, math.floor(n/2):] = right

        if num_districts == 9:
            left = np.random.choice(choices, (n, math.floor(n/3)), p = [0.2, 0.
↪45, 0.35])
            middle = np.random.choice(choices, (n, math.floor(n/3)), p = [0.2,␣
↪0.4, 0.4])
            right = np.random.choice(choices, (n, n - 2*math.floor(n/3)), p =␣
↪[0.2, 0.35, 0.45])
            self.array = np.zeros((n,n))

            self.array[:, 0:math.floor(n/3)] = left
```

```
            self.array[:, math.floor(n/3):math.floor(2*n/3)] = middle
            self.array[:, math.floor(2*n/3):] = right

        #add the schools, equally spaced throughout the grid

        if self.num_districts == 4:
            self.array[math.floor(n/4), math.floor(n/4)] = 3
            self.array[math.floor(n/4), math.floor(3*n/4)] = 3
            self.array[math.floor(3*n/4), math.floor(n/4)] = 3
            self.array[math.floor(3*n/4), math.floor(3*n/4)] = 3

        if self.num_districts == 9:
            self.array[math.floor(n/6), math.floor(n/6)] = 3
            self.array[math.floor(n/6), math.floor(n/2)] = 3
            self.array[math.floor(n/6), math.floor(5*n/6)] = 3

            self.array[math.floor(n/2), math.floor(n/6)] = 3
            self.array[math.floor(n/2), math.floor(n/2)] = 3
            self.array[math.floor(n/2), math.floor(5*n/6)] = 3

            self.array[math.floor(5*n/6), math.floor(n/6)] = 3
            self.array[math.floor(5*n/6), math.floor(n/2)] = 3
            self.array[math.floor(5*n/6), math.floor(5*n/6)] = 3

         # create an array where the value of each cell designates the district␣
↪number of the cell
        self.district = np.zeros((self.n, self.n))

        divide_by = math.sqrt(self.num_districts)

        def interval(i):
            return math.floor(i*n/divide_by)

        #assign districts
        index = 1
        for i in range(math.ceil(divide_by)):
            for j in range(math.ceil(divide_by)):
                self.district[interval(i): interval(i+1), interval(j):␣
↪interval(j+1)] = index

                index = index + 1


    def count_neighbors(self):
        """Surveys neighboring cells.

        returns: tuple of
```

```
            empty: True where cells are empty
            school: True where there is a school
            frac_red: fraction of red neighbors around each cell
            frac_blue: fraction of blue neighbors around each cell
            frac_same: fraction of neighbors with the same color
        """
        a = self.array

        empty = a==0
        red = a==1
        blue = a==2
        school = a==3

        # count red neighbors, blue neighbors, and total
        num_red = correlate2d(red, self.kernel, **self.options)
        num_blue = correlate2d(blue, self.kernel, **self.options)
        num_neighbors = num_red + num_blue

        with np.errstate(invalid='ignore'):
            # compute fraction of similar neighbors
            frac_red = num_red / num_neighbors
            frac_blue = num_blue / num_neighbors

        # no neighbors is considered the same as no similar neighbors
        # (this is an arbitrary choice for a rare event)
        frac_red[num_neighbors == 0] = 0
        frac_blue[num_neighbors == 0] = 0

        # for each cell, compute the fraction of neighbors with the same color
        frac_same = np.where(red, frac_red, frac_blue)

        # for empty cells, frac_same is NaN
        frac_same[empty] = np.nan
        frac_same[school] = np.nan

        return empty, school, frac_red, frac_blue, frac_same


    def count_district(self):
        """Surveys cells within the same school district.
        returns: tuple of
            empty: True where cells are empty
            school: True where there is a school
            frac_red: fraction of red agents in the school district as the cell
            frac_blue: fraction of blue agents in the school district as the␣
↪cell
```

```
            frac_same: fraction of agents with the same color in the school␣
↪district as the cell
        """

        a = self.array

        empty = a==0
        red = a==1
        blue = a==2
        school = a==3

        frac_red = np.zeros((self.n, self.n))
        frac_blue = np.zeros((self.n, self.n))

        # iterate over all 4 districts
        for dis_number in range(1,self.num_districts + 1):
            # get array of cells that are red in this district
            red_district = np.logical_and(red, self.district == dis_number)

            #count number of red agents in this district
            num_red = np.count_nonzero(red_district)

            # put number of red neighbors in this district in each cell
            num_red_array = np.where(self.district == dis_number, num_red, 0)

            # get array of cells that are blue in this district
            blue_district = np.logical_and(blue, self.district == dis_number)

            #count number of blue agents in this district
            num_blue = np.count_nonzero(blue_district)

            # put number of blue neighbors in this district in each cell
            num_blue_array = np.where(self.district == dis_number, num_blue, 0)

            num_total = num_red + num_blue

            frac_red += num_red_array/num_total
            frac_blue += num_blue_array/num_total

            #if there are no agents in the district, set frac to 0
            frac_red[num_total == 0] = 0
            frac_blue[num_total == 0] = 0

        frac_same = np.where(red, frac_red, frac_blue)

        # for empty cells, frac_same is NaN
        frac_same[empty] = np.nan
```

```
        frac_same[school] = np.nan

        return empty, school, frac_red, frac_blue, frac_same

    def segregation(self):
        """Computes the average fraction of similar neighbors.

        returns: fraction of similar neighbors, averaged over cells
        """
        _, _, _, _, frac_same_nbhd = self.count_neighbors()
        _, _, _, _, frac_same_district = self.count_district()
        return np.nanmean(frac_same_nbhd), np.nanmean(frac_same_district)

    def isolation(self, districts):
        """Computes the isolation wrt school districts"""

        isolate = 0

        isol_dist = np.zeros((self.n,self.n))

        divide_by = math.ceil(math.sqrt(districts))

        def interval(i):
            return math.floor(i*self.n/divide_by)

        #assign districts
        index = 1
        for i in range(divide_by):
            for j in range(divide_by):
                isol_dist[interval(i): interval(i+1), interval(j):␣
↪interval(j+1)] = index

                index = index + 1

        n_red = np.count_nonzero(np.array(self.array == 1))

        for i in range(1, districts + 1):
            red_dist = np.logical_and(isol_dist == i, self.array == 1)
            blue_dist = np.logical_and(isol_dist == i, self.array == 2)

            n_red_dist = np.count_nonzero(red_dist)
            n_blue_dist = np.count_nonzero(blue_dist)

            n_dist = n_red_dist + n_blue_dist

            isolate += (n_red_dist/n_red)*(n_blue_dist/n_dist)
```

```
        return isolate

    def dissimilarity(self, districts):
        """Computes the dissimilarity wrt school districts"""
        dissimilarity = 0

        diss_dist = np.zeros((self.n,self.n))

        divide_by = math.ceil(math.sqrt(districts))

        def interval(i):
            return math.floor(i*self.n/divide_by)

        #assign districts
        index = 1
        for i in range(divide_by):
            for j in range(divide_by):
                diss_dist[interval(i): interval(i+1), interval(j):␣
↪interval(j+1)] = index

                index = index + 1

        n_red = np.count_nonzero(np.array(self.array == 1))
        n_blue = np.count_nonzero(np.array(self.array == 2))

        for i in range(1, districts + 1):

            red_dist = np.logical_and(diss_dist == i, self.array == 1)
            blue_dist = np.logical_and(diss_dist == i, self.array == 2)

            n_red_dist = np.count_nonzero(red_dist)
            n_blue_dist = np.count_nonzero(blue_dist)

            dissimilarity += abs((n_red_dist/n_red) - (n_blue_dist/n_blue))

        return dissimilarity/2


    def step(self):
        """Executes one time step.

        returns: fraction of similar neighbors, averaged over cells
        """
        def move_nbhd(empty, unhappy_nbhd, unhappy_district):
            # identify locations of cells that should move neighborhoods (stay␣
↪in district)
```

```
            move_nbhd = np.logical_and(unhappy_nbhd, np.
↪logical_not(unhappy_district))
            move_nbhd_locs = locs_where(move_nbhd)

            orig_district = [self.district[j] for j in move_nbhd_locs]

            # shuffle the unhappy cells
            if len(move_nbhd_locs):
                np.random.shuffle(move_nbhd_locs)

            for index in range(len(move_nbhd_locs)):
                source = move_nbhd_locs[index]

                # get the district number of the source location
                source_dis = orig_district[index]

                # empty spots in the original district
                dest_empty = np.logical_and(empty, np.isin(self.district,␣
↪[source_dis]))

                # find the locations of the empty cells
                empty_locs = locs_where(dest_empty)

                # for each unhappy district cell, choose a random destination␣
↪(but need to make sure the destination is not in the same district)
                num_empty = np.sum(dest_empty)

                if num_empty == 0:
                    continue

                i = np.random.randint(num_empty)
                dest = empty_locs[i]

                # move
                a[dest] = a[source]
                a[source] = 0
                empty[source] = True
                empty[dest] = False


        def move_district(empty, unhappy_nbhd, unhappy_district):
            #identify locations of cells that should move districts

            poss_move = np.logical_and(np.logical_not(unhappy_nbhd),␣
↪unhappy_district)

            poss_move_locs = locs_where(poss_move)
```

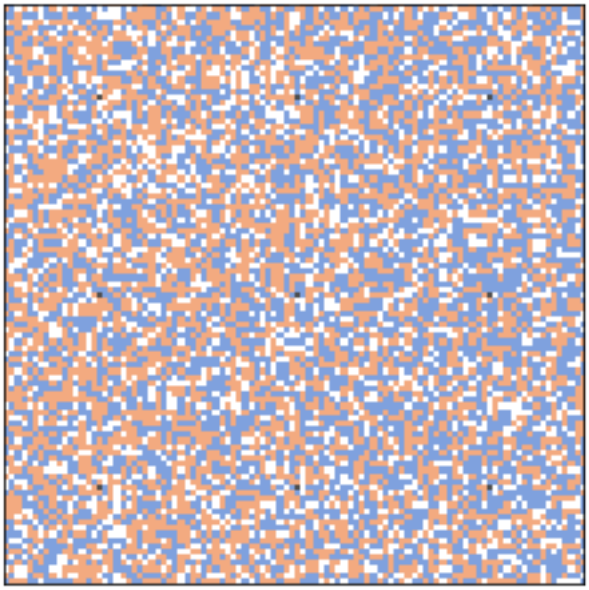
```
            n_poss_move = np.shape(poss_move_locs)[0]

            change = np.random.choice([False, True], n_poss_move, p = [1 - self.
↪p_change, self.p_change])

            for i in range(n_poss_move):
                poss_move[poss_move_locs[i][0]][poss_move_locs[i][1]] =␣
↪change[i]

            move_district = np.logical_or(np.logical_and(unhappy_nbhd,␣
↪unhappy_district), poss_move)

            #move_district = np.logical_and(np.logical_and(unhappy_nbhd,␣
↪unhappy_district), poss_move)
            move_district_locs = locs_where(move_district)

            #get the district of origin of each agent that wants to move␣
↪districts
            orig_district = [self.district[j] for j in move_district_locs]

            # shuffle the unhappy cells
            if len(move_district_locs):
                np.random.shuffle(move_district_locs)

            for index in range(len(move_district_locs)):
                source = move_district_locs[index]

                # get the district number of the source location
                source_dis = orig_district[index]

                # get array of possible districts to move to
                dest_dis = np.setdiff1d(range(1,self.num_districts + 1),␣
↪[source_dis])

                # empty spots not in the original district
                dest_empty = np.logical_and(empty, np.isin(self.district,␣
↪dest_dis))

                # find the locations of the empty cells
                empty_locs = locs_where(dest_empty)

                # for each unhappy district cell, choose a random destination␣
↪(but need to make sure the destination is not in the same district)
                num_empty = np.sum(dest_empty)
```

```
                if num_empty == 0:
                    continue

                i = np.random.randint(num_empty)
                dest = empty_locs[i]

                # move
                a[dest] = a[source]
                a[source] = 0

                empty[source] = True
                empty[dest] = False


        a = self.array

        red = a==1
        blue = a==2

        empty, _, _, _, frac_same_nbhd = self.count_neighbors()
        _, _, frac_red_district, frac_blue_district, frac_same_district = self.
↪count_district()

        # find the unhappy cells (ignore NaN in frac_same)
        with np.errstate(invalid='ignore'):
            # find cells unhappy with their neighborhood
            unhappy_nbhd = np.logical_or(np.logical_and(frac_same_nbhd < self.
↪p_r, red), np.logical_and(frac_same_nbhd < self.p_b, blue))

            # find cells unhappy with their school district
            unhappy_district = np.logical_or(np.logical_and(frac_red_district -␣
↪frac_blue_district > self.p_school, red), np.logical_and(frac_blue_district␣
↪- frac_red_district > self.p_school, blue))

            num_empty = np.sum(empty)

            move_nbhd(empty, unhappy_nbhd, unhappy_district)

            move_district(empty, unhappy_nbhd, unhappy_district)

        # check that the number of empty cells is unchanged
        num_empty2 = np.sum(a==0)
        assert num_empty == num_empty2

        # return the average fraction of similar neighbors
        return [np.nanmean(frac_same_nbhd), np.nanmean(frac_same_district)]
```

```
    def draw(self):
        """Draws the cells."""
        return draw_array(self.array, cmap=cmap, vmax=3)
```

[127]:
```
grid = Schelling(n=100, p_r = 0.5, p_b = 0.8, p_school = 0.10, p_change = 0.1,␣
↪num_districts = 9)

grid.draw()
```

[127]: <matplotlib.image.AxesImage at 0x7fa0839ed650>



[130]:
```
grid.isolation(36)
```

[130]: 0.5040785348246798

[131]:
```
grid.dissimilarity(36)
```

[131]: 0.08721323042966245

[132]:
```
for i in range(20):
    grid.step()

_, _, frac_red_district, frac_blue_district, _ = grid.count_district()
frac_red_district
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:119:
RuntimeWarning: divide by zero encountered in true_divide
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:120:
RuntimeWarning: divide by zero encountered in true_divide
```

[132]:
```
array([[0.55316973, 0.55316973, 0.55316973, …, 0.47066667, 0.47066667,
        0.47066667],
       [0.55316973, 0.55316973, 0.55316973, …, 0.47066667, 0.47066667,
        0.47066667],
       [0.55316973, 0.55316973, 0.55316973, …, 0.47066667, 0.47066667,
        0.47066667],
       …,
       [0.48571429, 0.48571429, 0.48571429, …, 0.44410876, 0.44410876,
        0.44410876],
       [0.48571429, 0.48571429, 0.48571429, …, 0.44410876, 0.44410876,
        0.44410876],
       [0.48571429, 0.48571429, 0.48571429, …, 0.44410876, 0.44410876,
        0.44410876]])
```

[133]:
```
grid.isolation(36)
```

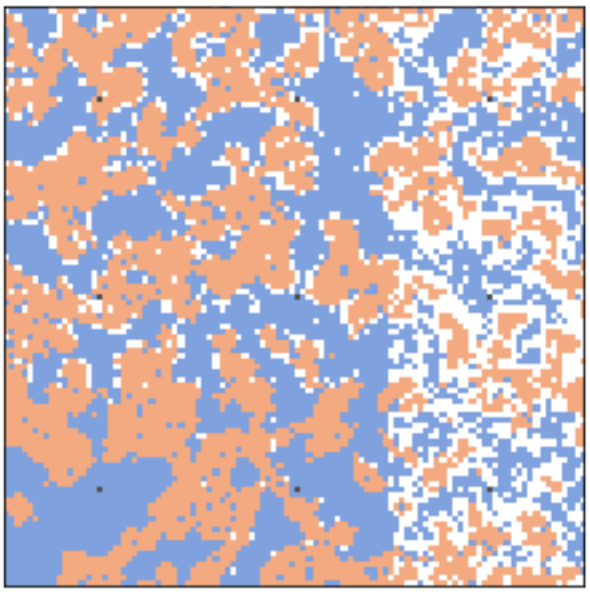
[133]:
```
0.4712634377640559
```

[134]:
```
grid.dissimilarity(36)
```

[134]:
```
0.21548692349146376
```

[135]:
```
grid.draw()
```

[135]:
```
<matplotlib.image.AxesImage at 0x7fa083da2e90>
```



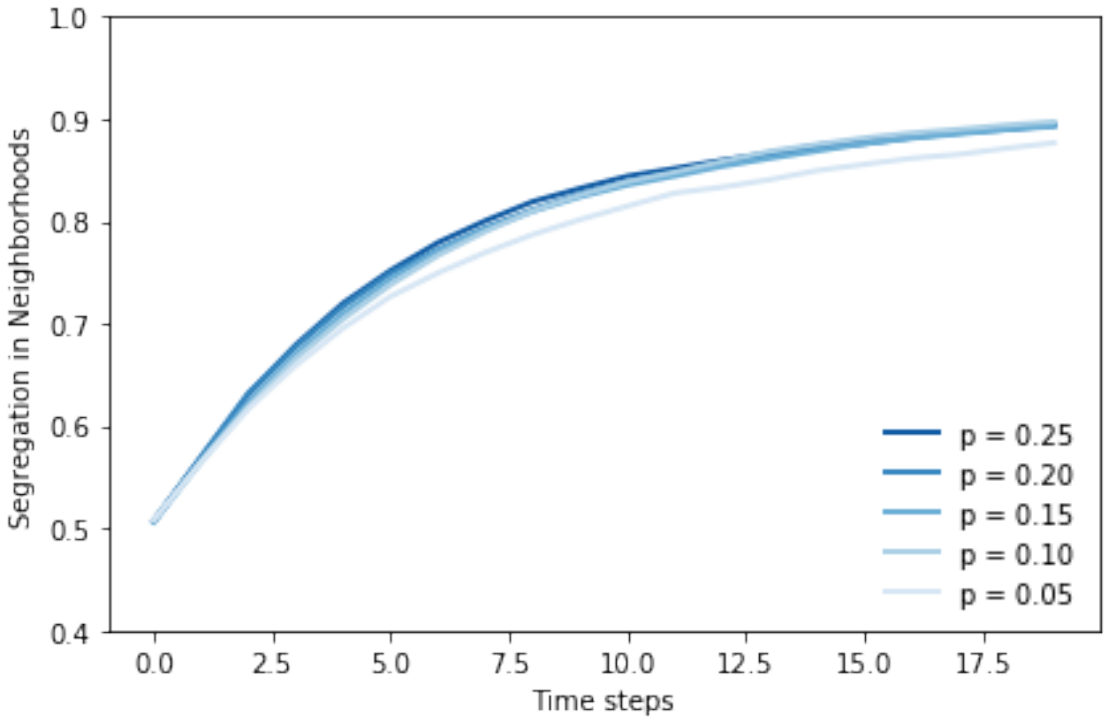
[120]:
```
frac_same_nbhd, _ = grid.segregation()
frac_same_nbhd
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:119:
RuntimeWarning: divide by zero encountered in true_divide
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:120:
RuntimeWarning: divide by zero encountered in true_divide
```

[120]:
```
0.8913422204040978
```

[20]:
```
from utils import set_palette
set_palette('Blues', 5, reverse=True)

num_steps = 20
np.random.seed(17)
for p in [0.25, 0.2, 0.15, 0.1, 0.05]:
    seg_nbhd = np.zeros((num_steps))
    for n in range(20):
        grid = Schelling(n=100, p_r = 0.5, p_b = 0.75, p_school = p, p_change =␣
 ↪0.1, num_districts = 4)
        seg_nbhd += np.array([grid.step()[0] for i in range(num_steps)])

    seg_nbhd /= 20
    plt.plot(range(num_steps), seg_nbhd, label='p = %.2f' % p)
    print(p, seg_nbhd[-1], seg_nbhd[-1] - p)

decorate(xlabel='Time steps', ylabel='Segregation in Neighborhoods',
                loc='lower right', ylim=[0.4, 1])
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:134:
RuntimeWarning: divide by zero encountered in true_divide
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:135:
RuntimeWarning: divide by zero encountered in true_divide

0.25 0.8943053419368558 0.6443053419368558
0.2 0.8938840519831632 0.6938840519831633
0.15 0.8927087099466057 0.7427087099466056
0.1 0.8972025028241306 0.7972025028241306
0.05 0.8766515675840824 0.8266515675840823
```

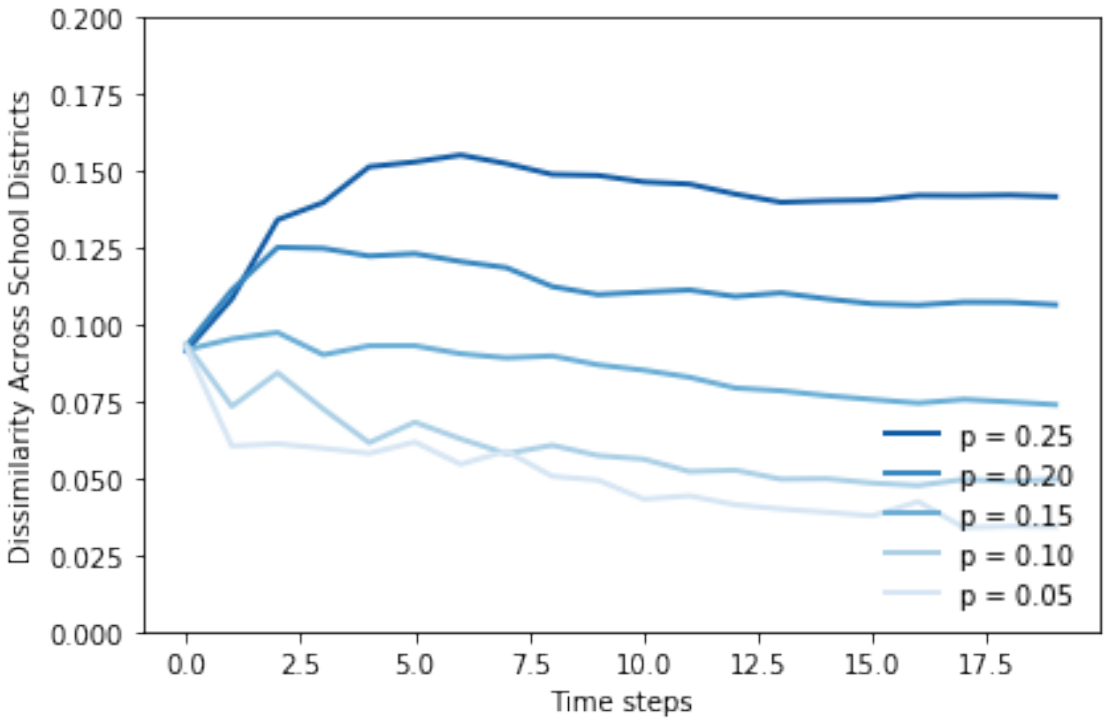


[ ]:
```
from utils import set_palette
set_palette('Blues', 5, reverse=True)

num_steps = 20
np.random.seed(17)
for p in [0.25, 0.2, 0.15, 0.1, 0.05]:
    seg_nbhd = np.zeros((num_steps))
    for n in range(20):
        grid = Schelling(n=100, p_r = 0.5, p_b = 0.75, p_school = p, p_change =␣
↪0.1, num_districts = 9)
        seg_nbhd += np.array([grid.step()[0] for i in range(num_steps)])

    seg_nbhd /= 20
    plt.plot(range(num_steps), seg_nbhd, label='p = %.2f' % p)
    print(p, seg_nbhd[-1], seg_nbhd[-1] - p)

decorate(xlabel='Time steps', ylabel='Segregation in Neighborhoods',
                loc='lower right', ylim=[0.4, 1])
```

[100]:
```
from utils import set_palette
set_palette('Blues', 5, reverse=True)

num_steps = 20
```

```
np.random.seed(17)
for p in [0.25, 0.2, 0.15, 0.1, 0.05]:
    dissimilarity = np.zeros((num_steps))
    for n in range(20):
        grid = Schelling(n=100, p_r = 0.5, p_b = 0.8, p_school = p, p_change =␣
↪0.1, num_districts = 9)
        for i in range(num_steps):
            dissimilarity[i] += grid.dissimilarity()
            grid.step()

    dissimilarity /= 20
    plt.plot(range(num_steps), dissimilarity, label='p = %.2f' % p)

decorate(xlabel='Time steps', ylabel='Dissimilarity Across School Districts',
                loc='lower right', ylim=[0, 0.2])
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:144:
RuntimeWarning: divide by zero encountered in true_divide
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:145:
RuntimeWarning: divide by zero encountered in true_divide
```

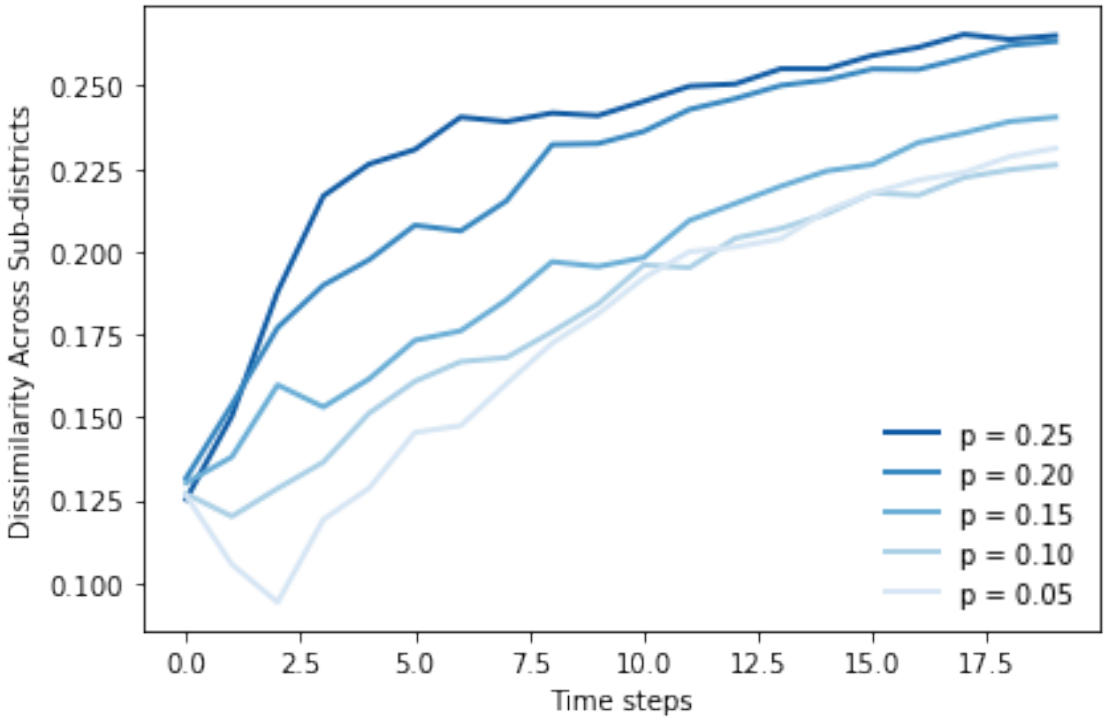


[145]:
```
num_steps = 20
np.random.seed(17)
```

```
dissimilarity_4 = np.zeros((len(p_school), num_steps))
p_index = 0

for p in [0.25, 0.2, 0.15, 0.1, 0.05]:
    for n in range(20):
        grid = Schelling(n=100, p_r = 0.5, p_b = 0.8, p_school = p, p_change =␣
↪0.1, num_districts = 4)
        for i in range(num_steps):
            dissimilarity_4[p_index][i] += grid.dissimilarity(36)
            grid.step()

    dissimilarity_4[p_index] /= 20
    p_index = p_index + 1
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:119:
RuntimeWarning: divide by zero encountered in true_divide
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:120:
RuntimeWarning: divide by zero encountered in true_divide
```

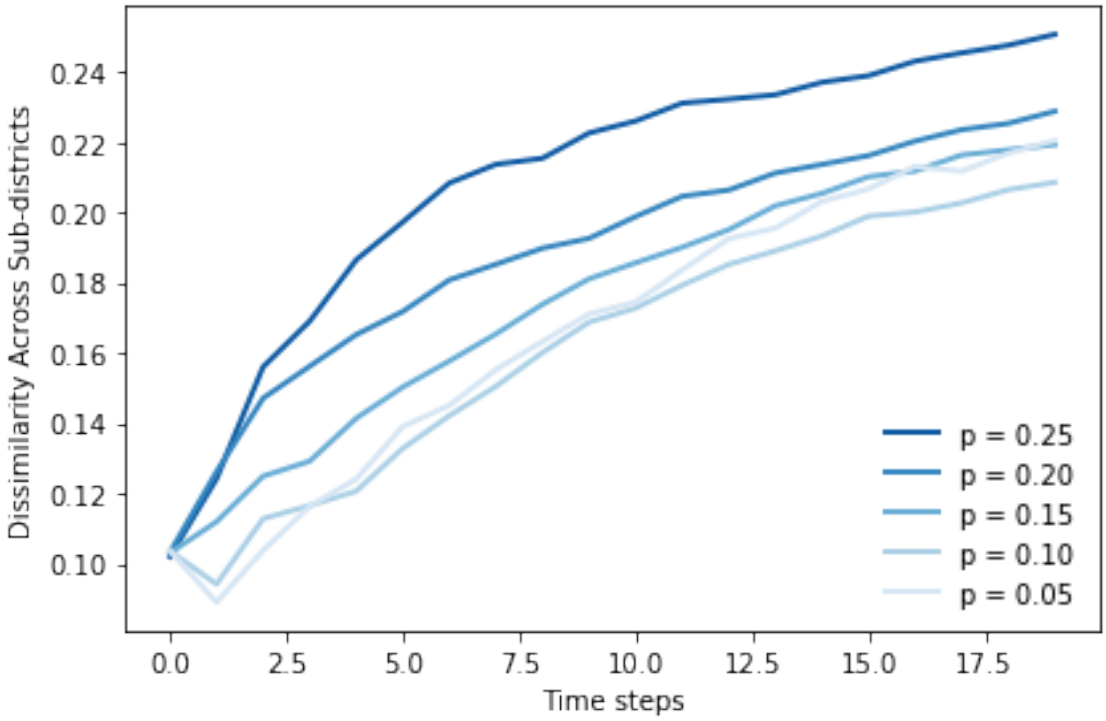
[146]:
```
from utils import set_palette
set_palette('Blues', 5, reverse=True)

p_index = 0

for p in p_school:
    plt.plot(range(num_steps), dissimilarity_4[p_index], label='p = %.2f' % p)
    p_index = p_index + 1

decorate(xlabel='Time steps', ylabel='Dissimilarity Across Sub-districts',
                loc='lower right')
```



[147]:
```
num_steps = 20
np.random.seed(17)

dissimilarity_9 = np.zeros((len(p_school), num_steps))
p_index = 0

for p in [0.25, 0.2, 0.15, 0.1, 0.05]:
    for n in range(20):
        grid = Schelling(n=100, p_r = 0.5, p_b = 0.8, p_school = p, p_change =␣
↪0.1, num_districts = 9)
        for i in range(num_steps):
            dissimilarity_9[p_index][i] += grid.dissimilarity(36)
            grid.step()

    dissimilarity_9[p_index] /= 20
    p_index = p_index + 1
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:119:
RuntimeWarning: divide by zero encountered in true_divide
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:120:
RuntimeWarning: divide by zero encountered in true_divide
```

[148]:
```
from utils import set_palette
set_palette('Blues', 5, reverse=True)

p_index = 0

for p in p_school:
    plt.plot(range(num_steps), dissimilarity_9[p_index], label='p = %.2f' % p)
    p_index = p_index + 1

decorate(xlabel='Time steps', ylabel='Dissimilarity Across Sub-districts',
                loc='lower right')
```

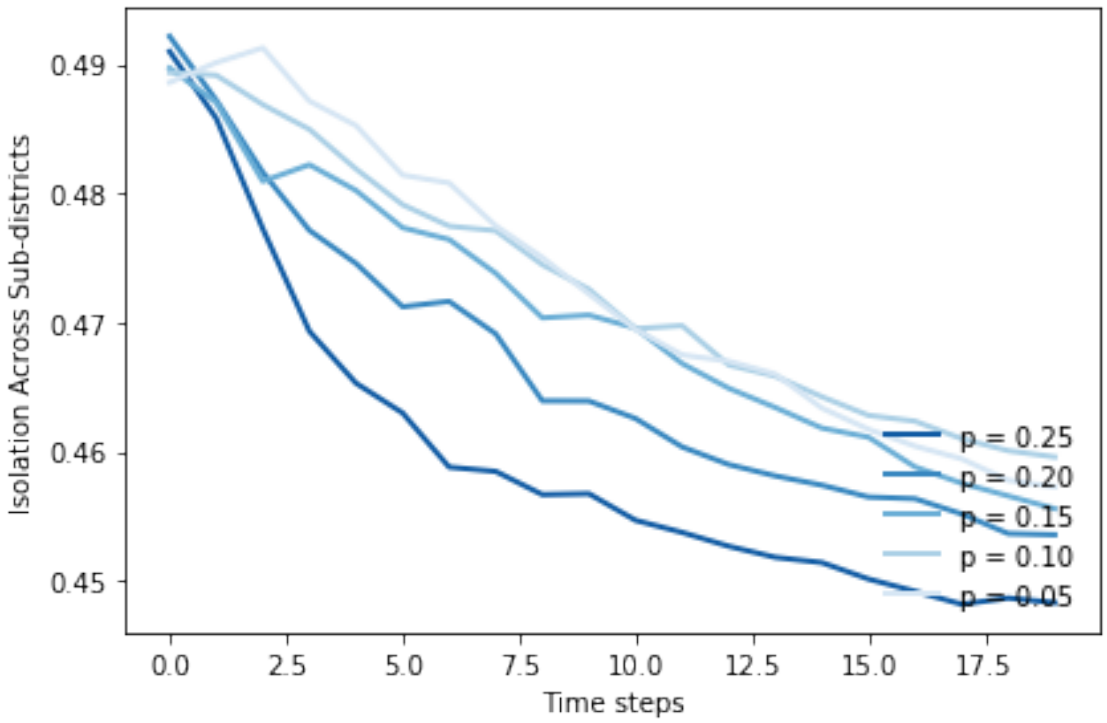


[149]:
```
num_steps = 20
np.random.seed(17)
p_school = [0.25, 0.2, 0.15, 0.1, 0.05]
isolation_4 = np.zeros((len(p_school), num_steps))
p_index = 0

for p in p_school:
    for n in range(20):
        grid = Schelling(n=100, p_r = 0.5, p_b = 0.8, p_school = p, p_change =␣
↪0.1, num_districts = 4)
        for i in range(num_steps):
```

```
            isolation_4[p_index][i] += grid.isolation(36)
            grid.step()

    isolation_4[p_index] /= 20
    p_index = p_index + 1
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:119:
RuntimeWarning: divide by zero encountered in true_divide
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:120:
RuntimeWarning: divide by zero encountered in true_divide
```

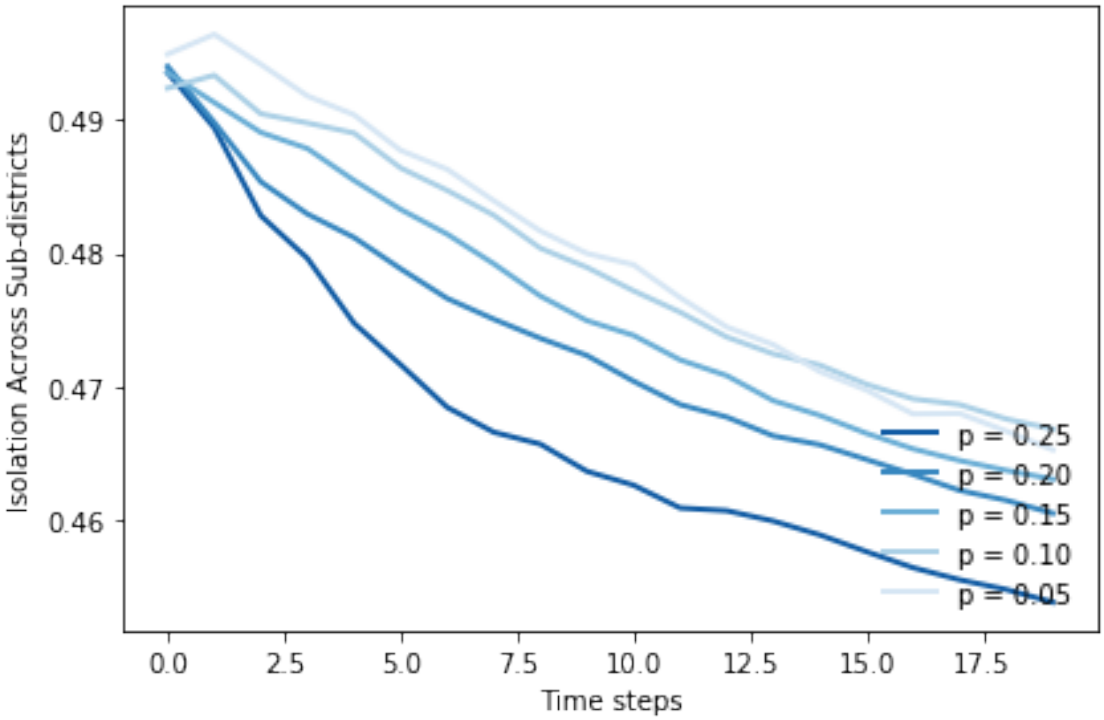
[150]:
```
from utils import set_palette
set_palette('Blues', 5, reverse=True)

p_index = 0

for p in p_school:
    plt.plot(range(num_steps), isolation_4[p_index], label='p = %.2f' % p)
    p_index = p_index + 1

decorate(xlabel='Time steps', ylabel='Isolation Across Sub-districts',
                loc='lower right')
```



[151]:
```
num_steps = 20
np.random.seed(17)
p_school = [0.25, 0.2, 0.15, 0.1, 0.05]
isolation_9 = np.zeros((len(p_school), num_steps))
p_index = 0

for p in p_school:
    for n in range(20):
        grid = Schelling(n=100, p_r = 0.5, p_b = 0.8, p_school = p, p_change =␣
↪0.1, num_districts = 9)
        for i in range(num_steps):
            isolation_9[p_index][i] += grid.isolation(36)
            grid.step()

    isolation_9[p_index] /= 20
    p_index = p_index + 1
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:119:
RuntimeWarning: divide by zero encountered in true_divide
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:120:
RuntimeWarning: divide by zero encountered in true_divide
```

[152]:
```
from utils import set_palette
set_palette('Blues', 5, reverse=True)

p_index = 0

for p in p_school:
    plt.plot(range(num_steps), isolation_9[p_index], label='p = %.2f' % p)
    p_index = p_index + 1

decorate(xlabel='Time steps', ylabel='Isolation Across Sub-districts',
                loc='lower right')
```


Isolation Across Sub-districts
0.49
0.48
0.47
0.46
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Time steps
p = 0.25
p = 0.20
p = 0.15
p = 0.10
p = 0.05