BrunoFBessa_5881890_P6_code

July 4, 2021

0.1 SFI5904 - Complex Networks

Project 6: Dynamics of difusion and probabilistic automata First Semester of 2021

Professor: Luciano da Fontoura Costa (luciano@ifsc.usp.br)

Student: Bruno F. Bessa (num. 5881890, bruno.fernandes.oliveira@usp.br) Universidade de São Paulo, São Carlos, Brazil.

Implement all the automata analyzed in CDT-22, reproducnf the figures of this work.

0.2 Code

The sections below have the code used to perform experiments and generate results.

[167]:
```
import random
import numpy as np
import pandas as pd
from scipy import signal
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
```

[126]:
```
# Deterministic Automata

class DeterministicAutomata():

    def __init__(self, states: tuple, deterministic_transitions: tuple) -> None:
        self.states = states
        self.deterministic_transitions = deterministic_transitions

    def execute(self) -> list:
        list_states = []

        for _state in list(deterministic_transitions):
            list_states.append(_state)

        return list_states
```

```python
class ProbabilisticAutomata:

    def __init__(self, states:dict, number_transitions: int) -> None:
        self.states = states
        self.number_transitions = number_transitions

    def execute(self) -> list:
        list_states = []

        for n in range(0, number_transitions-1):
            for key, value in states.items():
                if random.random() < value:
                    list_states.append(key)

        print(len(list_states), number_transitions)

        return list_states

class ProbabilisticSequentialAutomata:

    def __init__(self, states:dict, number_transitions: int, current_state: int = None) -> None:
        self.states = states
        self.number_transitions = number_transitions
        self.current_state = list(self.states)[0]

    def execute(self) -> list:
        list_states = []

        for n in range(0, self.number_transitions):

            inner_automata = self.states[self.current_state]
            elements, probabilities = list(inner_automata.keys()), list(inner_automata.values())
            transition_state = (np.random.choice(elements, 1, p=probabilities))[0]

            list_states.append(transition_state)
            self.current_state = transition_state

        return list_states

class ProbabilisticSequentialAutomata2:

    def __init__(self, states:dict, number_transitions: int, current_state: int = None) -> None:
        self.states = states
```

```
        self.number_transitions = number_transitions
        self.current_state = list(self.states)[0]

    def execute(self) -> list:
        list_states = []

        for n in range(0, self.number_transitions):

            inner_automata = self.states[self.current_state]
            elements, probabilities = list(inner_automata.keys()),␣
↪list(inner_automata.values())
            transition_state = (np.random.choice(elements, 1,␣
↪p=probabilities))[0]

            if transition_state > 1:
                list_states.append(transition_state/transition_state)
            else:
                list_states.append(transition_state)
            self.current_state = transition_state

        return list_states
```

[ ]:
```
class ProbabilisticSequentialAutomata2:

    def __init__(self, states:dict, number_transitions: int, current_state: int␣
↪= None) -> None:
        self.states = states
        self.number_transitions = number_transitions
        self.current_state = list(self.states)[0]

    def execute(self) -> list:
        list_states = []


        for n in range(0, number_transitions):
            inner_automata = self.states[self.current_state]
            elements, probabilities = list(inner_automata.keys()),␣
↪list(inner_automata.values())
            transition_state = (np.random.choice(elements, 1, probabilities))[0]
            if transition_state > 1:
                list_states.append(transition_state/transition_state)
            else:
                list_states.append(transition_state)
            self.current_state = transition_state

        return list_states
```

```
[162]: def stemplot(pattern: list) -> None:

    pattern = [float(x) for x in pattern]
    x = range(1, len(pattern)+1)

    fig = plt.figure(figsize=(6, 2))
    ax = fig.add_axes([0.3, 0.3, 0.8, 0.8])
    plt.stem(x, pattern)
    ax.set_title("Stemplot for generated pattern")
    plt.show()

def waveplot(pattern: list) -> None:

    T_MAX = 1000
    pattern_sw = []
    for x in pattern:
        for slope in range(0, int(T_MAX/len(pattern))):
            pattern_sw.append(x)

    t = np.linspace(1, len(pattern)+1, T_MAX, endpoint=False)
    fig = plt.figure(figsize=(6, 2))
    ax = fig.add_axes([0.3, 0.3, 0.8, 0.8])
    plt.plot(t, pattern_sw)
    ax.set_title("Square wave for generated pattern")
    plt.show()


def barplot(pattern: list) -> None:

    pattern = np.array([float(x)==1.0 for x in pattern])
    barprops = dict(aspect="auto", cmap="plasma", interpolation='nearest')
    fig = plt.figure(figsize=(6, 2))
    ax = fig.add_axes([0.3, 0.3, 0.8, 0.8])
    ax.set_axis_off()
    ax.imshow(pattern.reshape((1, -1)), **barprops)
    ax.set_title("Barplot for generated pattern")
    plt.show()

def lines_2dplot(x_pattern: list, y_pattern:list, title_comp: str = None) ->␣
 ↪None:

    x_pattern_lines = []
    for x in range(0, len(x_pattern)):
        if x_pattern[x] == 1:
            x_pattern_lines.append(x)

    y_pattern_lines = []
```

```
    for y in range(0, len(y_pattern)):
        if y_pattern[y] == 1:
            y_pattern_lines.append(y)


    plt.figure(figsize=(5, 5))
    for x in x_pattern_lines:
        plt.axvline(x, color = 'y')
    for y in y_pattern_lines:
        plt.axhline(y, color = 'y')

    ax = plt.axes()
    ax.set_facecolor("blue")

    ax.set_title("Two-dimensional patterns obtained by barcodes" + " " ␣
↪+title_comp)
    plt.show()

def circles_2dplot(x_pattern: list, y_pattern:list, title_comp: str = None) ->␣
↪None:

    x_pattern_lines = []
    for x in range(0, len(x_pattern)):
        if x_pattern[x] == 1:
            x_pattern_lines.append(x)

    y_pattern_lines = []
    for y in range(0, len(y_pattern)):
        if y_pattern[y] == 1:
            y_pattern_lines.append(y)

    plt.figure(figsize=(10, 10))
    circles = []
    for x in x_pattern_lines:
        for y in y_pattern_lines:
            circle = plt.Circle((x, y), 15, color="y", fill=False)
            circles.append(circle)

    ax = plt.gca()
    ax.cla()
    ax.grid(False)
    ax.set_xlim((0, len(x_pattern)))
    ax.set_ylim((0, len(y_pattern)))
    ax.set_facecolor("red")

    for circle in circles:
        ax.add_patch(circle)
```

```
    ax.set_title("Circles placed at crossing points (barcode patterns)" + " "␣
 ↪+title_comp)
    plt.show()

def densityplot(data: pd.DataFrame) -> None:

    plot = sns.displot(df, x="freq", hue="automaton", kind="kde")
    plot.fig.suptitle("Dentity of relative number of 1s")
    plt.show()
```

0.3 Results

[ ]:
```
# Deterministic Automata

states_da = ("0", "1")
deterministic_transitions = ("1", "0", "1", "1")
DA_pattern = DeterministicAutomata(states_da, deterministic_transitions).
 ↪execute()

stemplot(DA_pattern)
waveplot(DA_pattern)
barplot(DA_pattern)
```

[ ]:
```
# Probabilistic Automata
states_pa_a = {0: {0: 0.9, 1: 0.1}, 1:{0: 0.9, 1: 0.1}}
states_pa_b = {0: {0: 0.2, 1: 0.8}, 1:{0: 0.2, 1: 0.8}}
states_pa_c = {0: {0: 0.5, 1: 0.5}, 1:{0: 0.5, 1: 0.5}}


number_transitions = 200

for _ in range(0, 3):
    PA_a_pattern = ProbabilisticSequentialAutomata(states=states_pa_a,␣
 ↪number_transitions=number_transitions).execute()
    barplot(PA_a_pattern)

for _ in range(0, 3):
    PA_b_pattern = ProbabilisticSequentialAutomata(states_pa_b,␣
 ↪number_transitions).execute()
    barplot(PA_b_pattern)

for _ in range(0, 3):
    PA_c_pattern = ProbabilisticSequentialAutomata(states_pa_c,␣
 ↪number_transitions).execute()
```

```
    barplot(PA_c_pattern)
```

[ ]:
```
# Probabilistic Sequential Automata
states_sp_d = {0: {0: 0.9, 1: 0.1}, 1:{0: 0.882, 1: 0.098, 2: 0.02},
             2: {2: 0.2, 3: 0.8}, 3:{2: 0.194, 3: 0.776, 4: 0.03},
             4: {4: 0.5, 5: 0.5}, 5:{4: 0.495, 5: 0.495, 0: 0.01}}


number_transitions = 500
SPA_pattern = ProbabilisticSequentialAutomata(states_sp_d, number_transitions).
 ↪execute()
waveplot(SPA_pattern)
```

[ ]:
```
# Probabilistic Sequential Automata
states_sp_e = {0: {0: 0.9, 1: 0.1}, 1:{0: 0.882, 1: 0.098, 2: 0.02},
             2: {2: 0.2, 3: 0.8}, 3:{2: 0.194, 3: 0.776, 4: 0.03},
             4: {4: 0.5, 5: 0.5}, 5:{4: 0.495, 5: 0.495, 0: 0.01}}


number_transitions = 500
SPA_pattern = ProbabilisticSequentialAutomata2(states_sp_e, number_transitions).
 ↪execute()
waveplot(SPA_pattern)
```

[ ]:
```
# Vertical and horizontal lines generated with binary automata
lines_2dplot(PA_a_pattern, PA_a_pattern, "same pattern in x and y")

lines_2dplot(PA_a_pattern,
             ProbabilisticSequentialAutomata(states=states_pa_a,␣
 ↪number_transitions=number_transitions).execute(),
            "different patterns in x and y")

lines_2dplot(PA_a_pattern,
             PA_c_pattern,
            "different patterns in x and y")
```

[ ]:
```
# Circles centered at intersections of vertical and horizontal lines
# generated with binary automata

circles_2dplot(PA_a_pattern, PA_a_pattern, "same pattern in x and y")
```

[ ]:
```
# Density of the relative number os 1s

N = 200
list_columns = ["freq", "automata"]
df = pd.DataFrame(columns=list_columns)
```

```
states_pa_a = {0: {0: 0.9, 1: 0.1}, 1:{0: 0.9, 1: 0.1}}
states_pa_b = {0: {0: 0.2, 1: 0.8}, 1:{0: 0.2, 1: 0.8}}
states_pa_c = {0: {0: 0.5, 1: 0.5}, 1:{0: 0.5, 1: 0.5}}


for i in range (0, N):
    _a = float(np.sum(ProbabilisticSequentialAutomata(states=states_pa_a,␣
 ↪number_transitions=number_transitions).execute()))
    _b = float(np.sum(ProbabilisticSequentialAutomata(states=states_pa_b,␣
 ↪number_transitions=number_transitions).execute()))
    _c = float(np.sum(ProbabilisticSequentialAutomata(states=states_pa_c,␣
 ↪number_transitions=number_transitions).execute()))

    df.loc[len(df)] = [_a, "Automaton a"]
    df.loc[len(df)] = [_b, "Automaton b"]
    df.loc[len(df)] = [_c, "Automaton c"]

densityplot(df)
```

[ ]:
```
# Density of the relative number os 1s

N = 200
list_columns = ["freq", "automata"]
df = pd.DataFrame(columns=list_columns)

states_pa_55_45 = {0: {0: 0.55, 1: 0.45}, 1:{0: 0.45, 1: 0.55}}
states_pa_60_40 = {0: {0: 0.40, 1: 0.60}, 1:{0: 0.60, 1: 0.40}}
states_pa_50_50 = {0: {0: 0.50, 1: 0.50}, 1:{0: 0.50, 1: 0.50}}


number_transitions = 100
aaa = []

for i in range (0, N):
    _a = float(np.sum(ProbabilisticSequentialAutomata(states=states_pa_55_45,␣
 ↪number_transitions=number_transitions).execute()))
    _b = float(np.sum(ProbabilisticSequentialAutomata(states=states_pa_60_40,␣
 ↪number_transitions=number_transitions).execute()))
    _c = float(np.sum(ProbabilisticSequentialAutomata(states=states_pa_50_50,␣
 ↪number_transitions=number_transitions).execute()))

    df.loc[len(df)] = [_a, "Automaton 55%/45%"]
    df.loc[len(df)] = [_b, "Automaton 60%/40%"]
    df.loc[len(df)] = [_c, "Automaton 50%/40%"]

densityplot(df)
```