SOUNDSTALLATION 5

Ramin Akhavijou

This document outlines the process of creating a custom instrument utilizing sensors and microcontrollers, including coding, information retrieval, data collection, data visualization, demonstration, and machine learning. It provides detailed instructions for each of these stages, catering to individuals interested in understanding or constructing similar musical instruments.

Overview

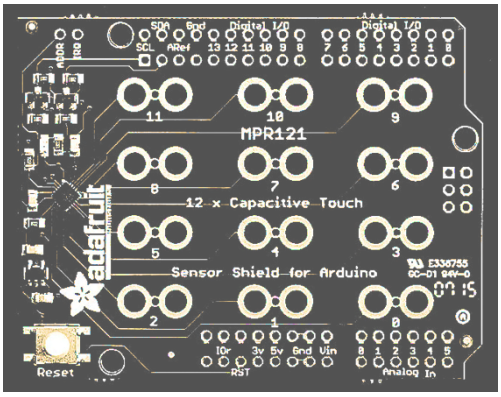
In this Soundstallation, I have incorporated two touch sensors. One of them is connected to an acoustic instrument that I built myself, while the other is connected to a photo frame. To enable touch sensing, I utilized a touch shield connected to an Arduino, which provided me with 12 capacitive sensors. I used 11 of these sensors for the sound library, and the remaining one was used to switch between libraries. Consequently, each instrument offered a total of 22 different sounds. I connected the touch shield to the instrument's strings using alligator clips, attaching one side of the clip to the shield and the other side to the string. As the sensor detects anything electrically conductive or containing water, I was able to touch different parts of the strings, tuning pegs, and screws for triggering the sounds, however, I specifically chose to play

the screws on the instruments. Once the signal is captured by the Arduino, it is transmitted to Max/MSP to trigger the sounds.

The second touch sensor was connected to a photo frame. I used conductive copper tape to make certain sections of the frame conductive (like keys). This transformed the frame into an instrument that added a visual element to the piece. To enhance the range of sounds, I employed a brass mortar to modify the channels. The incorporation of touch to activate sounds offered a unique experience, allowing for diverse feelings and practices during the performance.

Touch Capacitive Sensor

The sensor used in this project is a touch capacitive shield that provides 12 sensors. These sensors detect touch or proximity by measuring changes in capacitance, making them ideal for user interfaces, automotive controls, and interactive surfaces. Employing electrodes, these sensors respond to variations in capacitance caused by a user's touch. With advantages such as low power consumption, robust design, and adaptability to various materials, touch capacitive sensors continue to play a pivotal role in advancing human-machine interaction and interactive technology.



Arduino

The touch shield is linked to Arduino to capture the data, enabling the option to either process the data within Arduino or transmit it to another device for further analysis. The typical touch sensor code allows tracking both current and last touches, but, for performance reasons, only the current touch data was utilized. Incorporating both current and last touches simulates pressing and releasing a key. However, holding touch on the sensor doesn't send continuous data, requiring additional coding in either Arduino or Max for achieving sustained sound. The following is the code for this performance.

```
#include <Wire.h>
#include "Adafruit_MPR121.h"

#ifndef _BV
#define _BV(bit) (1 << (bit))
#endif

// Here, I utilized the library specific to a particular type, and it is
necessary to employ the associated library for compatibility
Adafruit_MPR121 cap = Adafruit_MPR121();

uint16_t lasttouched = 0;
uint16_t currtouched = 0;

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    delay(10);
  }

  Serial.println("Adafruit MPR121 Capacitive Touch sensor test");

  if (!cap.begin(0x5A)) {
    Serial.println("MPR121 not found, check wiring?");
    while (1);
  }
  Serial.println("MPR121 found!");
}

void loop() {
  currtouched = cap.touched();

  for (uint8_t i=0; i<12; i++) {
    if ((currtouched & _BV(i)) && !(lasttouched & _BV(i)) ) {
      Serial.print(i); Serial.println(" touched");
```

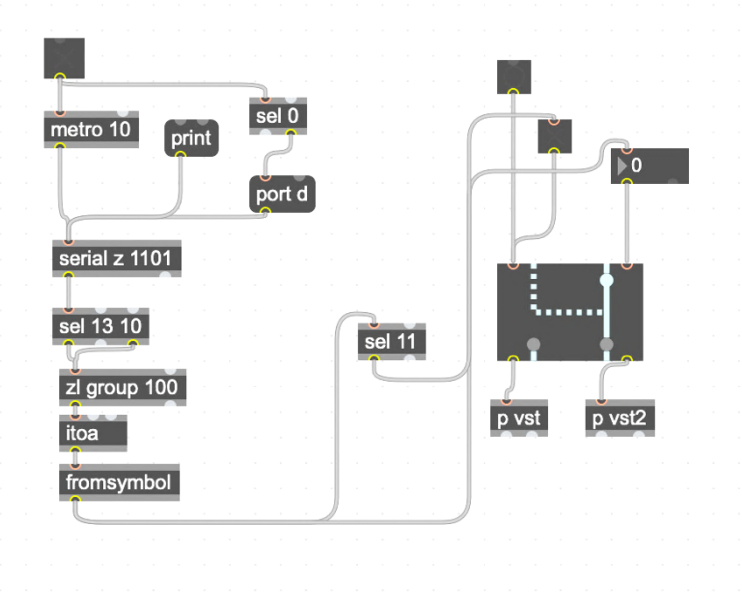
```
    }
  }

  lasttouched = currtouched;
  return;

  Serial.print("\t\t\t\t\t\t\t\t\t\t\t\t\t 0x");
Serial.println(cap.touched(), HEX);
  Serial.print("Filt: ");
  for (uint8_t i=0; i<12; i++) {
    Serial.print(cap.filteredData(i)); Serial.print("\t");
  }
  Serial.println();
  Serial.print("Base: ");
  for (uint8_t i=0; i<12; i++) {
    Serial.print(cap.baselineData(i)); Serial.print("\t");
  }
  Serial.println();
  delay(100);
}
```

Circuit Traces

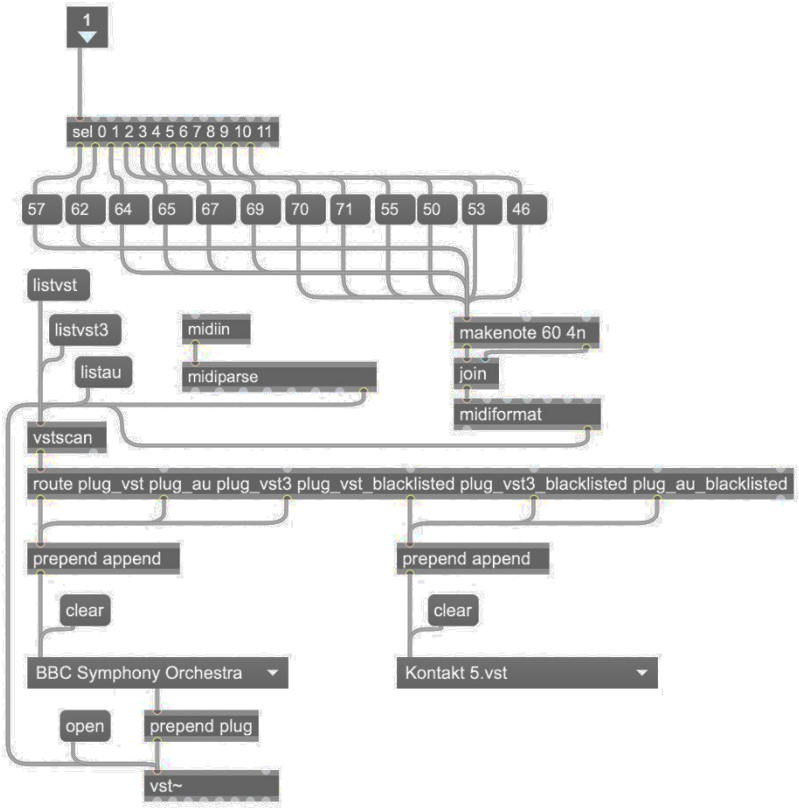
Conductive copper tape has been used to extend the sensors electrodes. There is no specific circuit design used in this project and the traces are merely for making the intended spots of surface of object conductive and turn that object to an instrument. While I have used photo frame to add some visual aspect to the performance and try to challenge my perception by playing on a non-conventional instrument and to see how it affects my perception by using my touch in different way than using usual instruments, this circuit traces can be used on almost every object or purposefully to be designed for instrument designing where touch sensitivity is the main component or desired. As it is shown in the photo, the copper tapes are attached to the bottom of photo frame which is connected on the other side to touch sensor and Arduino. When the designated spots are touched, a signal is sent to Max for triggering sounds.

Information retrieval and data processing in Max/MSP

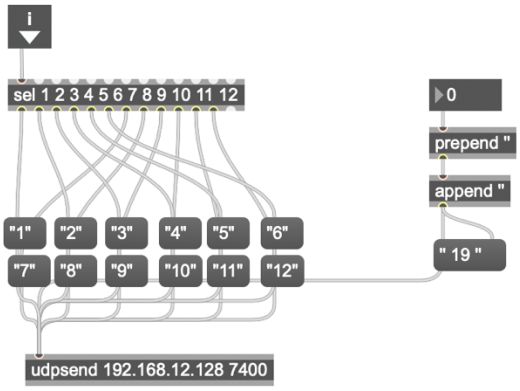
A Max patch is designed to receive data from Arduino via a serial and port connection. The left section of the patch handles data reception and conversion into numeric data. The numeric values corresponding to each sensor are then transmitted to the right section to trigger sounds—the numeric data can be utilized for various sound processing applications. Sensor number 11 is additionally assigned to switch the sound library, offering 22 distinct sounds for the performance.



Each sensor number is linked to a note in the sound library. These notes can be organized to create desired scales, or the numbers can be allocated to control various musical parameters.



Another segment of this patch is dedicated to collecting performance data and storing it for future use. My intention is to utilize this data for machine learning training, identify patterns, and subsequently replicate and transform the data into another form of performance, such as painting.



The data is sent to Python via UDP (User Datagram Protocol)—a communication protocol that allows for the transmission of data between different devices or software patches over a network.

Python; data collection

The following is the code for retrieving data from Max and collecting it in a file for machine learning purposes.

```
import socket
import struct

# UDP configuration
UDP_IP = "192.168.12.128"
UDP_PORT = 7400

# Specify the file path where you want to store the data
FILE_PATH = "SOUNDSTALLATION_ML.txt"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

with open(FILE_PATH, "w") as file:
    while True:
        try:
            data, addr = sock.recvfrom(1024)
            cleaned_data = data.replace(b'\x00', b'').decode('utf-
                  8').strip()
            print(f"Received cleaned data: {cleaned_data}")
            cleaned_data_without_commas = cleaned_data.replace(',',
                  '')

            file.write(f"{cleaned_data_without_commas}\n")
            file.flush()

            try:
                string_value = cleaned_data_without_commas
                print(f"Received string value: {string_value}")

            except UnicodeDecodeError:
                try:
                    integer_value = struct.unpack('<I', data[:4])[0]
                    print(f"Received integer value: {integer_value}")

                except struct.error as se:
                    print(f"Error decoding data: {se}")
        except Exception as e:
            print(f"Unexpected error: {e}")
```

Machine Learning

(In progress) This section utilizes collected data to apply models and algorithms for identifying patterns, with the potential to contribute insights to scientific domains for diverse purposes. The following code leverages common machine learning libraries like NumPy and Tensorflow to identify simple patterns. The training employs a supervised method where the data is labeled, and predictions are made for patterns and output. This section will be extended to encompass various patterns and algorithms, aiming to analyze performers' perceptions and movements. Subsequently, the data will be employed to replicate and generate new artwork following the training process.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

FILE_PATH = "SOUNDSTALLATION_ML.txt"

def find_pattern(sequence):
    ascending = all(sequence[i] < sequence[i + 1] for i in
range(len(sequence) - 1))
    descending = all(sequence[i] > sequence[i + 1] for i in
range(len(sequence) - 1))
    constant = all(sequence[i] == sequence[i + 1] for i in
range(len(sequence) - 1))
    alternating = all(sequence[i] * sequence[i + 2] < sequence[i + 1] ** 2
for i in range(len(sequence) - 2))

    if ascending:
        return "Ascending", len(sequence)
    elif descending:
        return "Descending", len(sequence)
    elif constant:
        return "Constant", len(sequence)
    elif alternating:
        return "Alternating", len(sequence)
    else:
        return "No discernible pattern", 0

def predict_next_number_rnn(sequence):

    X = np.array(sequence[:-1])
    y = np.array(sequence[1:])
```

```
    X = X.reshape((1, len(X), 1))


    model = Sequential()
    model.add(SimpleRNN(50, activation='relu', input_shape=(len(X[0]),
      1)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model.fit(X, y, epochs=100, verbose=0)

    # Predict the next number
    next_number = model.predict(X, verbose=0)
    return next_number[-1, 0]

def main():
    try:
        with open(FILE_PATH, "r") as file:
            lines = file.readlines()

        numbers = [int(line.strip()) for line in lines]
        print("Extracted Numbers:", numbers)

        pattern_type, pattern_length = find_pattern(numbers)

        if pattern_length > 0:
            print(f"The sequence follows a {pattern_type} pattern of
                  length {pattern_length}.")

            next_number_rnn = predict_next_number_rnn(numbers)
            print(f"Predicted next number (RNN): {next_number_rnn}")

        else:
            print("No discernible pattern in the sequence.")

    except FileNotFoundError:
        print(f"File not found: {FILE_PATH}")
    except ValueError:
        print("Error converting data to integers.")

if __name__ == "__main__":
    main()
```

The following is another in-progress machine learning section:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np

FILE_PATH = "SOUNDSTALLATION_ML.txt"
OUTPUT_FILE_PATH = "SOUNDSTALLATION2_ML.txt"
```

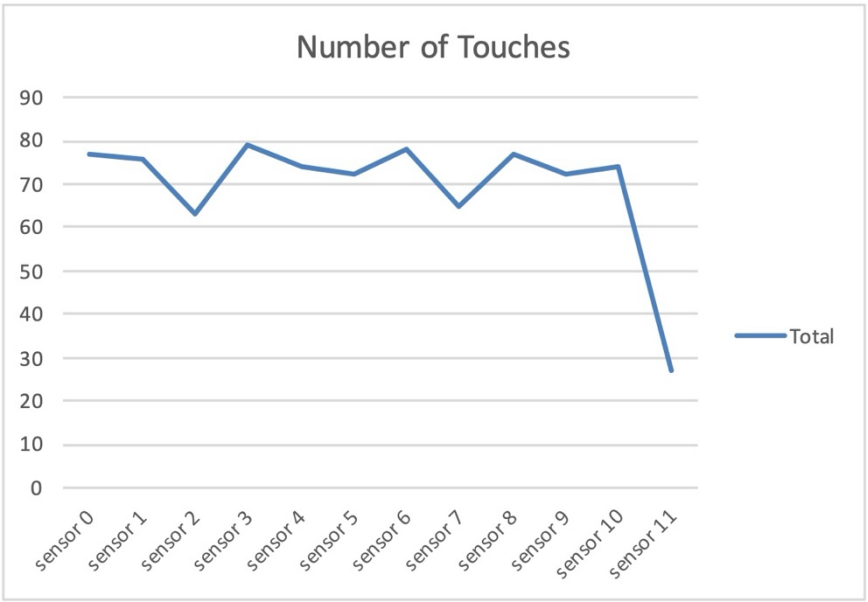
```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, h_n = self.rnn(x)
        output = self.fc(h_n[-1, :, :])
        return output

def read_sequence_from_file(file_path):
    try:
        with open(file_path, "r") as file:
            lines = file.readlines()
        sequence = [float(line.strip()) for line in lines]
        return sequence
    except FileNotFoundError:
        print(f"File not found: {file_path}")
        return None
    except ValueError:
        print("Error converting data to numbers.")
        return None

def predict_next_numbers(sequence, predict_length=3):

    sequence_tensor = torch.FloatTensor(sequence).view(1, -1, 1)
    input_size = 1
    hidden_size = 32
    output_size = 1
    model = RNN(input_size, hidden_size, output_size)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.01)

    # Training the RNN on the given sequence
    for epoch in range(1000):
        optimizer.zero_grad()
        output = model(sequence_tensor)
        loss = criterion(output, sequence_tensor)
        loss.backward()
        optimizer.step()

    # Predict the next numbers in the sequence
    with torch.no_grad():
        future_sequence = sequence_tensor[:, -1:, :].clone()
        for _ in range(predict_length):
            future_output = model(future_sequence)
            future_sequence = torch.cat((future_sequence,
future_output.unsqueeze(1)), dim=1)

    predicted_numbers = future_sequence[0, -predict_length:, 0].tolist()
    return predicted_numbers

def save_to_file(file_path, data):
    try:
        with open(file_path, "w") as file:
            for value in data:
```

```
                file.write(str(value) + "\n")
        print(f"Data saved to {file_path}")
    except IOError:
        print(f"Error saving data to {file_path}")

def main():
    sequence = read_sequence_from_file(FILE_PATH)
    if sequence:
        print("Extracted Sequence:", sequence)
        predicted_numbers = predict_next_numbers(sequence, predict_length=3)
        combined_sequence = sequence + predicted_numbers
        print("Predicted Next Numbers:", combined_sequence)

        save_to_file(OUTPUT_FILE_PATH, combined_sequence)

    else:
        print("Unable to read the sequence from the file.")

if __name__ == "__main__":
    main()
```

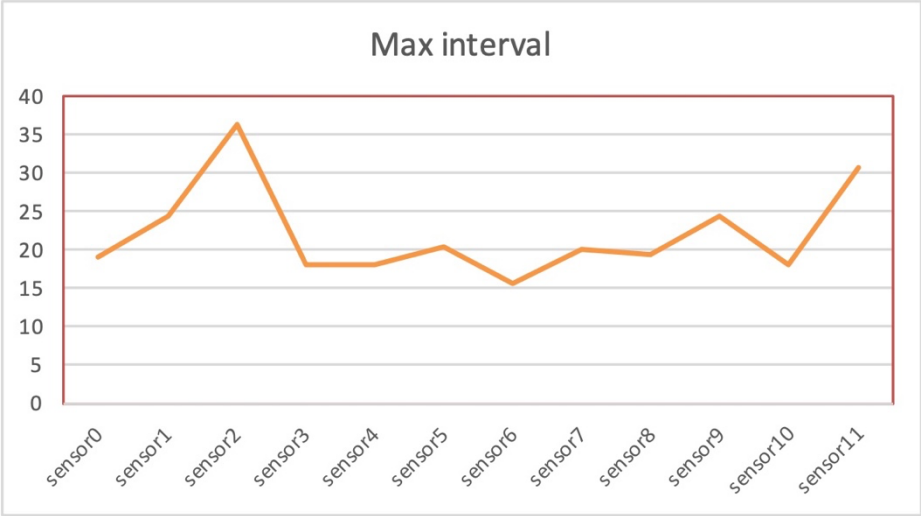
Data Analysis

The composer has conducted all analyses, illustrating the internal tempo of the music through the count of notes (equivalent to the number of sensor touches), alongside the determination of the longest and shortest intervals between timestamps (representing syntactic pauses), and calculating the average number of notes per sensor.

Sensor ID	
sensor 0	77
sensor 1	76
sensor 2	63
sensor 3	79
sensor 4	74
sensor 5	72
sensor 6	78
sensor 7	65
sensor 8	77
sensor 9	72
sensor 10	74
sensor 11	27



The MAX interval between the timestamps	
sensor0	19.2318
sensor1	24.3255
sensor2	36.3182
sensor3	18.1997
sensor4	17.9898
sensor5	20.4315
sensor6	15.6685
sensor7	20.2467
sensor8	19.475
sensor9	24.3695
sensor10	18.0092
sensor11	30.7819

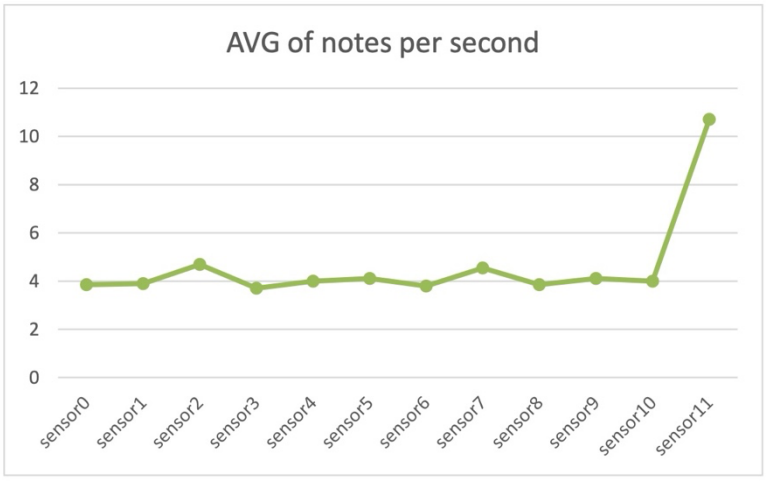


The MIN interval between the time	
sensor0	0.025432285
sensor1	0.054078668
sensor2	0.041607597
sensor3	0.012713531
sensor4	0.079677064
sensor5	0.106853324
sensor6	0.041899906
sensor7	0.015350503
sensor8	0.060808575
sensor9	0.150438155
sensor10	0.130241379
sensor11	0.007921529

First Time Touch	
Min of 0	2.57292
Min of 1	2.68594
Min of 2	8.28337
Min of 3	3.31648
Min of 4	4.36539
Min of 5	2.10041
Min of 6	6.74216
Min of 7	1.85644
Min of 8	0.57015
Min of 9	1.42533
Min of 10	2.47435
Min of 11	22.3535

Last Time Touch	
Max of 0	299.6830643
Max of 1	298.9293255
Max of 2	280.8525002
Max of 3	289.3754437
Max of 4	281.4804344
Max of 5	297.4593466
Max of 6	292.6799214
Max of 7	225.3327276
Max of 8	298.7687786
Max of 9	293.2269945
Max of 10	297.4287503
Max of 11	276.1434619

AVG of notes per second	
sensor0	3.846153846
sensor1	3.896103896
sensor2	4.6875
sensor3	3.703703704
sensor4	4
sensor5	4.109589041
sensor6	3.797468354
sensor7	4.545454545
sensor8	3.846153846
sensor9	4.109589041
sensor10	4
sensor11	10.71428571



Here is the data of the timestamps for further queries.

	0	1	2	3	4	5	6	7	8	9	10	11
0	2.572922	2.68594	8.283371	3.316477	4.365389	2.100414	6.742164	1.856439	0.570149	1.425328	2.474354	22.35351
1	2.598354	7.081363	8.914904	5.588832	8.227711	8.112401	16.48445	2.037676	2.963606	4.762559	9.398763	32.48196
2	7.449526	11.88959	8.956512	6.455067	8.807819	8.422166	20.16487	10.7874	4.971902	5.828374	11.03293	54.35544
3	14.6232	18.17138	9.618441	9.815867	9.357837	13.47496	24.48021	20.08961	8.461338	8.542242	16.8602	56.26977
4	14.88554	18.60684	17.63763	22.86886	19.83418	15.83368	28.79386	20.2607	8.980841	9.918807	17.55454	58.5637
5	16.80416	21.02484	18.24602	26.99702	27.48101	16.52345	31.20985	22.33436	11.47722	11.04155	19.75079	79.22253
6	16.96708	22.44103	18.7538	34.11993	31.1479	28.48306	31.33508	23.87902	16.4374	13.68517	21.15694	85.36867
7	23.66046	23.72354	22.65159	40.71094	38.65392	30.59631	34.52421	24.97275	19.5801	19.99529	24.59306	115.3505
8	23.86366	24.51511	30.81951	48.58536	40.69077	32.39044	34.56611	25.25151	30.46873	22.03517	32.57113	122.0758
9	28.65764	48.84066	44.79615	53.53676	41.11016	35.01596	50.23463	35.73373	41.22509	28.70092	34.83934	152.8576
10	35.22107	51.7323	45.24097	61.3235	42.02125	35.52557	50.52597	37.57682	42.02525	34.77323	37.05114	162.9467
11	41.58942	52.41801	46.37902	61.457	49.75226	35.91198	50.79297	38.80667	45.27659	34.92367	38.80463	166.5733
12	47.15529	59.79816	58.09908	62.74853	67.74206	39.44291	57.06234	42.5161	48.03877	43.0793	40.68968	167.8581
13	53.91471	64.22202	64.89387	65.13608	70.34274	45.18869	58.98304	46.77323	67.51375	52.0093	42.47773	167.8983
14	73.14655	67.19304	69.87818	65.19817	74.52816	46.27804	60.13977	47.87486	67.63377	54.41271	44.71389	179.5717
15	77.22837	71.21075	70.78779	68.16168	77.59504	48.01328	65.41093	53.94086	68.97694	56.69064	51.39836	182.8046
16	79.11733	74.11278	71.52812	75.06123	80.86716	49.3781	68.70142	55.61252	73.34842	65.90698	53.81106	189.7088
17	80.13123	75.05633	80.42198	80.26802	82.28616	51.87488	69.42629	57.83774	92.73825	69.56006	53.94622	196.6312
18	89.70637	79.92517	80.80948	81.03755	86.26039	65.6417	70.51462	64.2877	100.1481	72.77783	57.17026	201.3897
19	93.20946	79.97925	82.14105	81.6232	89.95448	83.36349	71.63598	64.36049	107.9913	73.52236	59.44652	205.4003
20	94.56846	84.94458	87.94277	81.63591	91.29796	87.4004	71.76909	66.50379	115.2596	76.62412	59.7197	206.5744
21	95.40435	91.15602	91.05751	84.12099	91.63316	88.5664	87.23595	68.04634	116.0428	83.50074	60.51978	217.7371
22	96.49098	91.37742	91.75446	94.36847	95.66174	89.39186	95.4664	74.6951	117.9851	87.12245	63.70341	229.7945
23	99.60891	92.94744	92.26474	94.95587	97.24517	100.9027	102.6641	77.75335	118.3091	88.65064	63.88573	229.8024
24	106.3197	93.4598	94.07589	96.76207	98.01422	116.0993	107.0458	78.72488	125.0263	90.28111	64.44231	240.1552
25	106.5536	95.52862	96.05136	97.37538	100.2344	116.2808	111.6952	81.19487	128.436	93.53482	65.84954	245.6327
26	108.8954	102.3528	98.80854	97.97193	106.7654	118.8265	119.8959	83.24673	128.6264	106.6975	69.50797	276.1435
27	109.839	105.0795	103.8044	108.4255	108.6081	123.952	122.9559	85.86319	129.0455	108.2757	77.57708	
28	112.0498	107.5662	106.3165	109.115	114.9715	128.3768	123.8377	87.38947	135.7791	110.4402	81.82355	
29	112.6061	109.3565	112.2353	109.7729	115.9382	128.8807	126.6776	88.84954	137.0007	117.0452	82.29182	
30	123.3759	109.5146	117.1531	116.8691	117.2435	129.7731	132.271	89.80238	137.4134	118.4394	88.29757	

31	135.0445	110.4265	120.7072	118.9181	117.4422	136.3397	133.8065	93.25721	139.4276	118.7169	89.498
32	137.5064	113.9983	121.7291	137.1178	121.5856	140.7082	134.3209	94.11175	143.5171	120.9007	91.17964
33	142.8943	116.2039	123.849	139.4316	123.858	143.3327	139.6341	100.7036	146.1075	122.564	91.30988
34	154.3516	120.3281	124.4956	141.0003	126.058	145.4983	140.117	103.6968	150.9585	139.5047	94.35902
35	155.0698	121.2613	128.2306	142.3924	130.6948	148.7773	140.6079	105.7524	160.7193	140.7038	98.76009
36	158.8462	122.4124	134.3195	144.3723	135.4121	149.3291	143.0188	106.7053	166.7747	147.8304	110.5184
37	159.4669	138.9688	139.955	146.8485	137.9555	156.1911	144.9193	112.8178	169.6463	149.2643	126.5387
38	161.0967	147.3333	141.6733	151.5199	139.1125	157.0023	146.6796	113.8259	173.9839	151.3016	136.2327
39	164.8534	150.2483	148.3703	153.8713	140.0275	166.895	148.4953	116.0996	179.6372	162.7703	143.0264
40	165.8727	154.0912	149.8408	160.0263	142.5244	169.0555	149.0692	117.4251	182.6418	165.3547	145.9074
41	174.3411	161.0363	151.359	165.7589	144.6936	169.2561	152.0317	120.6729	192.2316	166.3271	151.8551
42	176.6151	162.9345	152.4139	166.9907	147.7204	169.3629	155.6567	120.9606	192.5131	168.5896	154.7047
43	180.5444	170.2772	158.9096	168.5589	154.8624	171.3603	158.6001	123.145	195.7612	170.2483	164.6515
44	180.6384	174.3349	164.5634	175.2325	162.127	179.6072	159.9528	123.1604	197.2857	170.7043	168.3661
45	180.726	174.8576	165.426	175.4806	163.5999	181.5371	168.5496	125.4729	197.6352	172.745	170.9609
46	187.9545	179.2314	171.9774	184.416	163.7214	182.8819	168.6205	125.7465	198.6584	175.0591	178.2387
47	188.5549	180.9924	172.7591	192.5455	164.3513	183.5835	179.0164	145.9932	199.8011	178.5098	178.5984
48	192.8753	181.2893	176.1785	197.1234	176.7714	185.4947	182.0087	162.0786	204.2845	178.8646	182.6985
49	193.4203	181.3853	179.1015	198.7218	177.8024	187.4511	185.0646	169.6672	204.7143	183.8412	184.4108
50	194.1339	184.6414	181.9362	205.7809	194.8947	195.0362	187.5311	172.2844	205.4636	208.2106	193.4583
51	194.7893	185.9609	187.0601	212.3149	195.7206	202.18	189.1536	188.1065	209.0141	208.6996	199.8381
52	195.616	189.4012	191.9184	212.5796	201.8286	202.4467	201.1321	189.3588	213.225	227.2731	213.056
53	199.6907	190.2172	192.7406	213.3702	205.3769	204.3093	203.8171	200.1522	214.0161	227.8311	216.9246
54	205.8932	191.439	198.3652	216.4291	210.4373	206.8435	210.3807	204.3305	214.0769	231.11	225.4876
55	219.7461	198.5197	202.3472	218.3307	214.1556	208.5346	211.4859	205.7428	214.2169	232.6404	229.2276
56	223.5689	204.0838	208.1361	219.8783	214.2832	209.4471	215.39	207.8004	217.9137	237.7924	234.7371
57	228.4745	212.1979	212.3391	223.0749	220.1703	210.6731	228.1095	208.1766	218.1018	244.1416	236.1998
58	229.5014	213.1433	212.9877	233.9649	232.632	215.6033	228.7489	208.5166	221.5023	244.3538	254.209
59	229.7624	215.8807	214.1116	234.3823	236.2872	218.764	233.4126	212.7694	226.3966	247.1186	254.9036
60	230.6691	216.9251	225.3598	246.7266	238.4237	219.0047	247.2482	222.7482	227.778	253.0548	257.6484
61	236.6038	229.2837	244.5343	257.5412	246.5816	219.5915	248.044	223.0245	231.3627	254.0228	258.3696
62	237.7127	230.4144	280.8525	258.6964	248.2785	227.4809	252.1517	223.8819	237.7217	261.487	258.9841

63	237.7616	234.0966	258.9533	251.538	241.0145	256.0629	224.6414	241.4173	265.3457	263.4909
64	237.9568	239.7155	264.78	254.2328	243.0399	256.1406	225.3327	244.804	266.0682	264.0353
65	241.4832	249.4378	266.0578	254.3125	263.4715	256.6877		246.1229	268.825	267.7952
66	249.8182	257.9516	267.5281	260.1846	269.6533	258.2907		252.8481	277.5316	271.3584
67	251.8203	259.7312	268.5185	260.2652	275.5725	259.6326		253.857	282.7784	279.7112
68	252.8725	269.2651	269.9439	262.5197	277.6307	260.0848		256.3978	283.0055	280.4599
69	263.6747	269.757	270.7311	264.3002	280.8033	261.7092		257.8789	283.8596	281.0275
70	274.4546	276.9302	275.1431	267.3921	284.8337	266.295		262.5414	292.7346	290.5816
71	275.0475	284.4967	276.297	272.1109	297.4593	268.7609		263.3769	293.227	293.3902
72	282.8086	290.3387	279.0141	275.0409		269.0525		271.1474		296.0278
73	287.4817	291.4806	280.9248	281.4804		269.6587		274.3624		297.4288
74	287.9145	295.1866	284.8459			276.4052		277.6468		
75	298.3923	298.9293	285.1302			286.7159		286.3125		
76	299.6831		287.0624			287.4826		298.7688		
77			288.2685			292.6799				
78			289.3754							
79			299.4743							

Performance

Here is the link to one of the performances.