

Dissonanzanalyse

March 21, 2025

```
[1]: from music21 import chord, interval, converter

def Dissonanz_Finden(chordified_stream):
    """
    1.reine Quarten über der Bassnote und andere dissonante Intervalle aufgrund
    ↳des Zustand von Chordify finden
    2.eine Liste, inklusiv Offset, nameWithOctave und interval_name aller
    ↳Dissonanzen erhalten
    """
    dissonance = []
    dissonant_intervals = {'A1', 'm2', 'M2', 'A2', 'd3', 'd4', 'A4', 'd5',
    ↳'A5', 'A6', 'd6', 'm7', 'M7', 'd8', 'Dis.4'}

    for element in chordified_stream.flatten().notesAndRests:
        if isinstance(element, chord.Chord):
            notes = element.notes
            bass_note = element.bass()
            dissonant_labels = [] # ein Label erstellen, um die dissonanten
            ↳Intervalle zu speichern.

            for i, note_obj in enumerate(notes):
                for other_note in notes[i + 1:]:
                    intvl = interval.notesToInterval(note_obj, other_note)

                    # Beurteilung, ob eine reine Quarte dissonant ist.
                    if intvl.simpleName == 'P4' and (
                        note_obj.nameWithOctave == bass_note.nameWithOctave or
                        other_note.nameWithOctave == bass_note.nameWithOctave):
                        interval_name = 'Dis.4'
                    else:
                        interval_name = intvl.simpleName

                    # das Intervall protokollieren, deren Name des einfachen
                    ↳Intervalls in der Dissonanzliste steht
                    if interval_name in dissonant_intervals:
                        dissonance.append({
                            'offset_1': note_obj.offset,
```

```

        'pitch_1': note_obj.nameWithOctave,
        'offset_2': other_note.offset,
        'pitch_2': other_note.nameWithOctave,
        'interval': interval_name
    })

    dissonant_labels.append(f"{interval_name}")

    # die gefundenen Dissonanzen als Liedtext markieren
    if dissonant_labels:
        element.addLyric("\n".join(dissonant_labels))

    return dissonance, chordified_stream

```

```

[2]: file_path = r'C:
      ↪\Users\Administrator\Desktop\Dissonanzanalyse\Bach_Dissonanzanalyseprogramm\Korpus_Bachs_
      ↪15 Sinfonien\Sinfonia 2 BWV788.Musicxml'
      score = converter.parse(file_path)

```

```

[3]: #Beispiel Zeigen
      chordified = score.chordify()
      dissonances, marked_chordified = Dissonanz_Finden(chordified)
      marked_chordified.show()

```

Sinfonia 2 BWV788.Musicxml

J.S.Bach

<IPython.core.display.HTML object>

12/8

1 m7 M2 Dis.4

2 M7 M7 m2 m7 d5 d4

3 Dis.4 Dis.4 Dis.4 Dis.4 m7 M7 M2 Dis.4

5 A4 M2 M7

6 Dis.4 m2 m7

7 Dis.4 Dis.4

8 m7 m2 m7 M2 M7 M2 M7 M7 A4
A4 A4 m7

9 Dis.4 M2 d5 d4 Dis.4

10 M2 m7 A5 m7
A4

<IPython.core.display.HTML object>

11

M2 Dis.4 A5 M2 M2

13

A4 m7

14

M7 Dis.4 Dis.4

15

M2 Dis.4 Dis.4 m7

16

m2 Dis.4 m7 Dis.4 Dis.4 m7 m2 Dis.4 d5 Dis.4 d5 Dis.4 Dis.4

17

m7 d5 M2 M2 m7 m2 m7 d5 m2 m7 Dis.4 A4 m2 A4 m2 A2 A2 A4 M2 A4 A5 A4

18

Dis.4 d5 Dis.4 d5 M2 d5 Dis.4 m7 d5 m2

19

<IPython.core.display.HTML object>

20

A5 M7 d5 M2 A4 A5 A4 Dis.4 M2 m7 m7 m7 m2 d4
 Dis.4 d5 d5

21

d5 m7 Dis.4 d5 Dis.4 A4 A4 m2 m2 A4 m7 M2 Dis.4 Dis.4
 M2 Dis.4 M7 Dis.4 M2

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

29

Dis.4 M2 m7 m2 M2 Dis.4

30

Dis.4 Dis.4

31

m7 M2 m7 d5 M2 A4 A5 M7 M7 M7 M2

A4 m7 M2

32

M2 M7 m2 Dis.4 Dis.4

```
[4]: import math
#Notenanschlag
"""
Dieses Programm kann im Zustand von chordify mithilfe der Eigenschaft "tie"
↳sehr effektiv feststellen,
ob zwei Töne eines gefundenen dissonanten Intervalls einen Anschlag haben, dann
↳durch die Liedtextveränderung sie zu zeigen.

Die Logik lautet wie folgt:
Im Zustand von chordify,
wenn einer der beiden Töne eines Intervalls der mittlere ("stop") oder der
↳Endton ("continue") einer Bindungslinie ist,
dann gibt es keine Anschlag an dem Intervall. Andernfalls tritt eine Anschlag
↳auf.
"""
dissonances, marked_chordified = Dissonanz_Finden(chordified)

# Durchlaufe die Akkorde und vergleiche nicht dissonante Intervallinformationen
```

```

for element in marked_chordified.recurse().notes:
    if isinstance(element, chord.Chord): # Stelle sicher, dass es sich um
    ↳ einen Akkord handelt
        # Hole die Noten des aktuellen Akkords und ihre Offsets
        chord_notes = {note.nameWithOctave: note for note in element.notes}

        # Durchlaufe die Informationen der dissonanten Intervalle
        for dissonance in dissonances:
            # Überprüfe, ob der aktuelle Akkord das dissonante Intervall
            ↳ enthält und ob die Offsets übereinstimmen
                if (
                    dissonance['pitch_1'] in chord_notes and
                    dissonance['pitch_2'] in chord_notes and
                    math.isclose(chord_notes[dissonance['pitch_1']].offset,
            ↳ dissonance['offset_1'], rel_tol=1e-9) and
                    math.isclose(chord_notes[dissonance['pitch_2']].offset,
            ↳ dissonance['offset_2'], rel_tol=1e-9)
                ):
                    # Hole die beiden Notenobjekte des dissonanten Intervalls
                    note1 = chord_notes[dissonance['pitch_1']]
                    note2 = chord_notes[dissonance['pitch_2']]

                    # Überprüfe die Tie-Eigenschaften der beiden Noten
                    tie_states = []
                    for note in (note1, note2):
                        if note.tie:
                            tie_states.append(note.tie.type)
                        else:
                            tie_states.append(None)

                    # Klassifizierungslogik: den Text der Dissonanzen ohne
            ↳ Notenkollision zur "i." oder "I." vorläufig anpassen,
                    # mit Notenkollision zur "iv." oder "IV." vorläufig anpassen,
                    # Kleinbuchstaben der römischen Zahlen stehen für unharmonische
            ↳ Dissonanzen, Großbuchstaben für harmonische Dissonanzen.
                    if any(state in ['stop', 'continue'] for state in tie_states):
                        element.lyric = 'I.'
                    else:
                        element.lyric = 'II.'

```

```

[5]: from RhythmischeAnalyse import Bestimme_betontePosition

# Anwendung der bereits analysierten betonten-Positionen jedes Takts durch die
↳ entsprechenden Offsets auf die gefundenen Dissonanzen

part1 = score.parts[1]

```



```

betontePositionen = Bestimme_betontePosition(score)

for measure in part1.getElementsByClass('Measure'): # Iteriere über jede Takt
    measure_number = measure.measureNumber
    strong_beats = betontePositionen.get(measure_number, []) # Hole starke
    ↪ Beats für den Takt

    # und markiere deren Typ durch eine entsprechende Änderung des Liedtextes.
    for element in chordified.recurse().notes: # Iteriere über Noten und Pausen
        offset = element.offset
        if element.lyric: # Überprüfe die vorhandenen Lyrics
            if offset in strong_beats: # Überprüfe, ob die Note auf einem
            ↪ starken Beat liegt
                if element.lyric == "I.":
                    element.lyric = "III."
                elif element.lyric == "II.":
                    element.lyric = "IV."

```

Takt 1, Grundrhythmus: eighth
 Takt 2, Grundrhythmus: eighth
 Takt 3, Grundrhythmus: eighth
 Takt 4, Grundrhythmus: eighth
 Takt 5, Grundrhythmus: 16th
 Takt 6, Grundrhythmus: 16th
 Takt 7, Grundrhythmus: eighth
 Takt 8, Grundrhythmus: 16th
 Takt 9, Grundrhythmus: 16th
 Takt 10, Grundrhythmus: eighth
 Takt 11, Grundrhythmus: eighth
 Takt 12, Grundrhythmus: eighth
 Takt 13, Grundrhythmus: eighth
 Takt 14, Grundrhythmus: 16th
 Takt 15, Grundrhythmus: 16th
 Takt 16, Grundrhythmus: eighth
 Takt 17, Grundrhythmus: 16th
 Takt 18, Grundrhythmus: 16th
 Takt 19, Grundrhythmus: 16th
 Takt 20, Grundrhythmus: 16th
 Takt 21, Grundrhythmus: 16th
 Takt 22, Grundrhythmus: 16th
 Takt 23, Grundrhythmus: 16th
 Takt 24, Grundrhythmus: 16th
 Takt 25, Grundrhythmus: 16th
 Takt 26, Grundrhythmus: 16th
 Takt 27, Grundrhythmus: eighth
 Takt 28, Grundrhythmus: 16th
 Takt 29, Grundrhythmus: 16th

Takt 30, Grundrhythmus: eighth
Takt 31, Grundrhythmus: 16th
Takt 32, Grundrhythmus: 16th

```
[6]: #Test
betontPositionen = Bestimme_betontePosition(score)
for measure, positions in betontPositionen.items():
    print(f"Takt {measure}, betonte Position:{positions}")
```

Takt 1, Grundrhythmus: eighth
Takt 2, Grundrhythmus: eighth
Takt 3, Grundrhythmus: eighth
Takt 4, Grundrhythmus: eighth
Takt 5, Grundrhythmus: 16th
Takt 6, Grundrhythmus: 16th
Takt 7, Grundrhythmus: eighth
Takt 8, Grundrhythmus: 16th
Takt 9, Grundrhythmus: 16th
Takt 10, Grundrhythmus: eighth
Takt 11, Grundrhythmus: eighth
Takt 12, Grundrhythmus: eighth
Takt 13, Grundrhythmus: eighth
Takt 14, Grundrhythmus: 16th
Takt 15, Grundrhythmus: 16th
Takt 16, Grundrhythmus: eighth
Takt 17, Grundrhythmus: 16th
Takt 18, Grundrhythmus: 16th
Takt 19, Grundrhythmus: 16th
Takt 20, Grundrhythmus: 16th
Takt 21, Grundrhythmus: 16th
Takt 22, Grundrhythmus: 16th
Takt 23, Grundrhythmus: 16th
Takt 24, Grundrhythmus: 16th
Takt 25, Grundrhythmus: 16th
Takt 26, Grundrhythmus: 16th
Takt 27, Grundrhythmus: eighth
Takt 28, Grundrhythmus: 16th
Takt 29, Grundrhythmus: 16th
Takt 30, Grundrhythmus: eighth
Takt 31, Grundrhythmus: 16th
Takt 32, Grundrhythmus: 16th
Takt 1, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 2, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 3, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 4, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 5, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 6, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 7, betonte Position:[0.0, 1.5, 3.0, 4.5]

Takt 8, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 9, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 10, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 11, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 12, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 13, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 14, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 15, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 16, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 17, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 18, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 19, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 20, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 21, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 22, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 23, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 24, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 25, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 26, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 27, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 28, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 29, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 30, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 31, betonte Position:[0.0, 1.5, 3.0, 4.5]
Takt 32, betonte Position:[0.0, 1.5, 3.0, 4.5]

```
[7]: # Entferne alle vorhandenen Liedtexte aus der Partitur
for part in score.parts:
    for n in part.flatten().notes:
        n.lyric = None

# Übertragung der Markierung im Chordify zur Originalpartitur
marked_elements = []

for element in chordified.flatten().notesAndRests:
    if isinstance(element, chord.Chord) and element.lyric:
        measure_number = element.measureNumber
        offset_32nd = round(element.offset * 8)
        marked_elements.append((measure_number, offset_32nd, element.lyric))

for part in score.parts:
    for n in part.flatten().notes:
        measure_number1 = n.measureNumber
        offset_32nd1 = round(n.offset * 8)
        for measure_number, offset_32nd, lyric in marked_elements:
            if (
                measure_number1 == measure_number
```

```
        and offset_32nd1 == offset_32nd
    ):
        if not n.lyric:
            n.addLyric(lyric)
```

```
[8]: processed_positions = set()

for part in score.parts:
    for n in part.flatten().notes:
        measure_number = n.measureNumber
        offset_32nd = round(n.offset * 8)

        if (measure_number, offset_32nd) in processed_positions:
            n.lyric = None
        else:
            if n.lyric:
                processed_positions.add((measure_number, offset_32nd))
```

```
[9]: score.show()
```

Sinfonia 2. BWV788

J.S.Bach

IV. I. I. I. II. IV. I.

I. I.

3 I. I. III. I. I. I. III. I.

5 I. I. I. I. I. I.

7 III. III. III. III. III. I. I. I. I.

I. I. I. I. I.

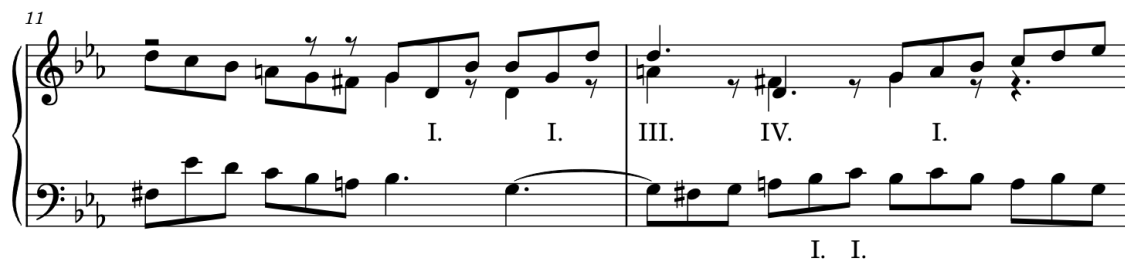
9 I. I. I. I. I. IV. I. I.

I.

<IPython.core.display.HTML object>

2

11



I. I. III. IV. I. I. I.

13



I. I. I. I.

14



I. I. I. I. IV. I.

16



III. I. I. I. III. III. III. I. I. I. I. I. I. I.

18



III. I. I. I. I. II. II. III. I. III. I. I. I. I. I.

<IPython.core.display.HTML object>

19

20

21

<IPython.core.display.HTML object>

22

III. I. I. I. II. I. IV. I. I. II. I.

I. I. I. I. I. I.

23

III. I. I. III. III.

I. I. I. I. I. I. I. I.

24

III. I. I. III. III.

I. I. I. I. I. I. I.

25

III. I. I. I. I. III. I. I. I. I. I. I. I.

26

III. II. III. II. I. III. I. III. I. III. I. IV. I. IV. III.

I. I. I. I. I. I. I. I.


```
<IPython.core.display.HTML object>
```

28

I. I. I. I. I. II. I. II. I. I. I. II.

30

III. III. I. I. I.

31

III. III. I. III. I. I. I. I. I. I. I. II. II. II. I.

```
[10]: def Ergebnis_Analyse(score):
        """
        analysiert die GesamtAnzahl und der durchschnittliche Anzahl pro Takt der
        ↪Dissonanz jeder Kategorie
        auf der Basis der Liedtextmarkierungen (i., ii., iii., usw.) in der
        ↪angegebenen Partitur.

        Rückgabewert:
        Eine Liste des Berechnungsergebnisses

        """
        # Gesamtanzahl des Takts zählen
        total_measures = len(score.parts[0].getElementsByClass('Measure'))
```

```

# Zähler für jeden Liedtexttyp Initialisieren
lyrics_counts = {
    "I.": 0, "II.": 0, "III.": 0, "IV.": 0
}

processed_positions = set()

# alle Stimmen, Takte und Elemente durchlaufen
for part in score.parts:
    for n in part.flatten().notes:
        measure_number = n.measureNumber
        offset_32nd = round(n.offset * 8) # Offset in eine feinere
        ↪ Einheit(32tel-Note) umwandeln

        # Vermeide Duplikate mit der processed_positions-Logik
        if (measure_number, offset_32nd) not in processed_positions:
            if n.lyric in lyrics_counts: # If lyric matches tracked types
                lyrics_counts[n.lyric] += 1
                processed_positions.add((measure_number, offset_32nd))

# Berechne die durchschnittliche Anzahl pro Takt und runde die Ergebnisse
↪ auf zwei Dezimalstellen
result = {}
for lyric, count in lyrics_counts.items():
    avg_per_measure = round(count / total_measures, 2) if total_measures >
    ↪ 0 else 0.00
    result[lyric] = {"Gesamtanzahl": count, "durchschnittliche Anzahl pro
    ↪ Takt": round(avg_per_measure, 2)}

return result

result = Ergebniss_Analyse(score)
for dissonanzart, Anzahl in result.items():
    # Kleinbuchstaben der römischen Zahlen stehen für unharmonische
    ↪ Dissonanzen, Großbuchstaben für harmonische Dissonanzen.
    print(f"Dissonanz der Klasse {dissonanzart}: {Anzahl}")

```

Dissonanz der Klasse I.: {'Gesamtanzahl': 180, 'durchschnittliche Anzahl pro Takt': 5.62}

Dissonanz der Klasse II.: {'Gesamtanzahl': 15, 'durchschnittliche Anzahl pro Takt': 0.47}

Dissonanz der Klasse III.: {'Gesamtanzahl': 39, 'durchschnittliche Anzahl pro Takt': 1.22}

Dissonanz der Klasse IV.: {'Gesamtanzahl': 10, 'durchschnittliche Anzahl pro Takt': 0.31}

```
[11]: def Export_To_MusicXML(score, filename="Exportierte_Partitur.musicxml"):
      """
      Exportiert die Partitur als MusicXML-Datei.
      :param score: Die Partitur, die exportiert werden soll
      :param filename: Name der MusicXML-Datei
      """
      try:
          score.write('musicxml', filename)
          print(f"Die Partitur wurde erfolgreich als '{filename}' exportiert!")
      except Exception as e:
          print(f"Fehler beim Exportieren: {e}")

      Export_To_MusicXML(score, "Analyse_Dissonanzen.musicxml")
```

Die Partitur wurde erfolgreich als 'Analyse_Dissonanzen.musicxml' exportiert!

```
[ ]:
```