universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

„Algorithmen zur Trennung von Stimmen und Lagen in Klaviermusik“

Verfasser

Harald Köhler

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag.rer.nat.)

Wien, 2008

Studienkennzahl lt. Studienblatt:	A 190 884 406
Studienrichtung lt. Studienblatt:	LA UF Informatik und Informatikmanagement, UF Mathematik
Betreuer:	a.o. Univ.-Prof. Dr. Erich Neuwirth

Algorithmen zur Trennung von Stimmen und Lagen in Klaviermusik

Kurzfassung

Zur Erstellung einer Partitur aus einer computerlesbaren Einspielung eines Musikstücks sind viele Schritte notwendig; einige davon können durch diverse gängige Software erfolgen. Im Fall eines Klavierstückes ist das Ergebnis jedoch meistens schlecht lesbar, da die getrennte Notierung der beiden Hände diese Generierung erschwert. In dieser Arbeit werden, nach einer Einführung in die Thematik der Partiturerzeugung, Algorithmen zur Umsetzung mehrerer musikalisch-technischer Ansätze zur Lösung dieses Problems präsentiert. Dabei werden die Algorithmen – je nachdem welche Daten sie verwenden – in vertikale und horizontale Algorithmen eingeteilt. Durch das Testen dieser Algorithmen an mehreren unterschiedlichen Werken der klassischen Klavierliteratur kann gezeigt werden, dass verschiedene Ansätze notwendig sind, um unterschiedliche Klaviersätze optimal zu analysieren. Insbesondere ist auch ersichtlich, dass einfache Rechenvorschriften mitunter sehr gute Ergebnisse erzielen können.

Die hier erarbeiteten Ansätze werden in Pseudocode dargestellt und überdies textuell erklärt, ein Programm zum Testen der Algorithmen wurde entwickelt und steht im WWW zur Verfügung (s. Anhang 2, Weiterführende Quellen)

Algorithms for separation of voices and positions in piano music

Abstract

Score generation from computer readable music records needs to pass through many steps, from which many can be done by existing music software. Analyzing piano pieces, they generally produce badly readable sheet music due to separate notation of the two hands. This paper describes – after an introduction to the generation of scores – algorithms on implementation of different musical-technical approaches to achieve a solution of this problem. Depending on the data they use, the algorithms will be categorized as vertically or horizontally. Testing these algorithms with various classical piano music works, it becomes obvious that different concepts are necessary for analyzing different piano works. Particularly simply designed algorithms can achieve good results too.

The concepts of this paper are described in pseudo-code and explained textual, a tool for testing the algorithms has been developed and is available on web as described in “Anhang 2, Weiterführende Quellen“.

Danksagungen

Unter den vielen netten Personen, welchen ich mich zu Dank verpflichtet fühle, möchte ich an dieser Stelle jene hervorheben, die mich im Zuge meiner Diplomarbeit unterstützt haben. Namentlich sind das meine Lebensgefährtin Maria-Christina Thomas für so viele unterstützende Hilfestellungen, dass diese hier nicht alle angeführt werden könnten, a.o. Univ-Prof. Dr. Erich Neuwirth für die kompentente Betreuung, aber auch die persönliche Begeisterung für das Thema meiner Arbeit, Veronika Pintar unter anderem für das Korrekturlesen, und nicht zuletzt meinen Eltern, die es in jeder Hinsicht erst möglich gemacht haben, bis hierher zu gelangen.

Inhalt

I. Grundlagen

1 Von der Aufnahme zur Partitur (allgemein)

Die moderne Datenverarbeitung ermöglicht prinzipiell das Erstellen einer Partitur aus einer Einspielung eines Musikwerkes. Im Folgenden werde ich den Weg zu einem fertigen Notenstück, und die Schwierigkeiten, die dabei auftreten können, kurz beschreiben, bevor ich eine nähere Spezifikation des erarbeiteten Themas geben werde.

Zunächst müssen die Audio-Daten in computerlesbare Form gebracht werden. Es gibt zahlreiche Möglichkeiten, Audio- oder Musikinformationen zu beschreiben, wie bei [Si00], S.45ff beschrieben; diese sind jedoch nicht alle für die Repräsentation durch digitale Daten geeignet.
Zur digitalen Speicherung dieser Informationen haben sich zwei konträre Ansätze durchgesetzt: Die erste Technologie wird oft als *Digitales Audio* bezeichnet, die zweite Form wird durch Ansätze wie den MIDI-Standard beschrieben.

Digitales Audio

Digitales Audio (manchmal auch *gesampeltes* Audio) beschreibt ein Musikstück (oder jegliche akustische Information) direkt durch digitalisierte Werte des im Zeitbereich abgetasteten Schalldrucks. Das bedeutet, dass in einem vorgegebenen Zeitabstand der Wert des Schalldrucks des (analogen) Audio-Signals entnommen wird. Dieser muss dann digitalisiert werden, wobei feste Wortbreiten verwendet werden und der entsprechende Wert daher nicht beliebig genau gespeichert werden kann; er wird also auf ein festgelegtes Raster angepasst. Für diesen Vorgang sind elektronische Bauteile wie Filter und Analog-Digital-Konverter notwendig.
Die akustische Qualität der Aufnahme hängt von vielen Faktoren ab. Neben den technischen Faktoren der Tonabnehmer spielen vor allem die verwendeten Werte für die Auflösung im Zeitbereich (das Zeitintervall, in dem abgetastet wird) und die Auflösung im Frequenzbereich (die Wortlänge bei der Digitalisierung) eine große Rolle. Bei geeigneter Abtastfrequenz kann das Signal nach Shannon und Nyquist originalgetreu wiederhergestellt werden.
Daher wird eine originalgetreue Wiedergabe eines Musikstücks gewährleistet, wenn man davon ausgeht, dass das menschliche Ohr nicht in der Lage ist, Frequenzen

beliebiger Höhe und Frequenzunterschiede beliebiger Genauigkeit wahrzunehmen.[1] Zahlreiche bekannte Dateiformate (wie etwa AU, WAV, MP3, etc.) erlauben die Speicherung von digitalem Audio, welches teilweise noch durch unterschiedliche Algorithmen komprimiert wird.

Digital Audio bietet im Gegensatz zum unten beschriebenen MIDI viele Vorteile für die Wiedergabe von Musikstücken, die Generierung eines Notenbildes eines derart gespeicherten Werkes ist allerdings sehr schwierig; bereits die Erlangung der Notenwerte ist nicht trivial. Weitere schwierige Schritte zur Erstellung einer Partitur sind im Abschnitt *MIDI-Dateien und Partituren* beschrieben.
Bei der Erzeugung der Notenwerte müssen beispielsweise unterschiedliche Stimmungen von Instrumenten und Abweichungen innerhalb eines Instruments berücksichtigt werden.

Es gibt dennoch einige Ansätze, eine Partitur oder zumindest eine MIDI-Datei (im folgenden Abschnitt definiert) aus den Audio-Informationen zu generieren, die Technologien hierzu sind allerdings noch nicht ganz ausgereift.

MIDI

MIDI – ein Akronym für „*Musical Instrument Digital Interface*“ ([Hu99], S.1) – setzt an einer völlig anderen Stelle an als digitales Audio, und wird daher auch in anderen Bereichen eingesetzt. Audio-Informationen werden hier durch Meta-Daten (auch Steuerbefehle, vgl. [Go02], S.118) und nicht durch ihre physikalischen Eigenschaften beschrieben. MIDI definiert – neben Hardware-Interfaces Signale – zur Übertragung bestimmter Steuerdaten zwischen MIDI-Geräten (z.B. MIDI-Keyboards, Synthesizer, PCs mit MIDI-Interfaces).
Diese Steuersignale können auch (mit einem Zeitstempel versehen) gespeichert werden, das Format hierfür wird als Standard Midi File (SMF) bezeichnet[2]. In einer Midi-Datei werden keinerlei Informationen über den konkreten Klang gespeichert, es werden Daten wie der Beginn und das Ende, die Höhe und die Lautstärke jedes Tons gespeichert. Zur Erzeugung des Klangs ist spezielle Hard- oder Software nötig, welche die gespeicherten Daten als akustische Informationen wiedergibt. Ich möchte

[1] Eine sehr gute Beschreibung der Digitalisierung von Audio-Signalen findet sich bei [Ro02], S.470ff, eine etwas kompaktere Darstellung auf Deutsch z.B. bei [Go02], S.15ff.
[2] Es gibt auch zahlreiche Erweiterungen des Standard MIDI Formats, um zusätzliche Steuerbefehle übertragen zu können. Eine mögliche Erweiterung ist bei [Si00], Kapitel 4 beschrieben.
Da MIDI-Interfaces für PCs üblicherweise jedoch nur das Standard Format unterstützen, werde ich auch nur dieses verwenden.

hier nicht näher auf die Vielzahl der mit MIDI übertragenen bzw. gespeicherten Daten eingehen, hierzu sei auf [Hu99] verwiesen. Aus Gesagtem ist dennoch bereits erkennbar, dass die Speicherung von Audio-Daten mittels der beschriebenen Meta-Informationen nicht für eine detailgetreue Wiedergabe geeignet ist und auch nicht beliebige Geräusche „aufgenommen“ werden können.

Eine Partitur kann aus den vorhandenen Daten jedoch wesentlich leichter erzeugt werden, als aus abgetasteten Audio-Signalen.
In weiterer Folge werde ich davon ausgehen, dass die Daten im MIDI-Format vorliegen.

1.1 MIDI-Dateien und Partituren

Aus technischer Sicht entspricht eine MIDI-Datei in gewisser Weise bereits einer Partitur; grundlegende Daten wie die Höhe, Dauer und Lautstärke von Noten sind direkt aus MIDI-Dateien auslesbar. Es sind allerdings noch viele Schritte notwendig, um eine tatsächliche grafische Partitur aus den vorhandenen Werten zu erzeugen.

Nach Vinci sind die wesentlichen Elemente der (grafischen) Notenschrift neben Notenliniensysteme und den Noten (bzw. Pausen) selbst, die unterschiedlichen Notenschlüssel, Taktstriche, Balkensetzungen (Richtung und Gruppierung), Halte- und Bindebögen, Tonart- bzw. Versetzungszeichen, sowie diverse Artikulationszeichen und Interpretationsangaben (vgl. [Vi91]).

Die Setzung von Taktstrichen und Notenwerten hängt stark von der Wahl des Tempos und der Quantisierung der Notenwerte (s. Abschnitt Tondauer und Quantisierung) ab, die anderen genannten Elemente werden nachfolgend behandelt.

Notenliniensysteme

In der herkömmlichen europäischen Notation ist es üblich, für die meisten Musikinstrumente Fünfliniensysteme zu verwenden. Für manche Instrumente werden auch mehrere Fünfliniensysteme untereinander gesetzt, beispielsweise um Stimmen für beide Hände getrennt zu notieren. So werden vor allem die Tasteninstrumente in zwei oder drei (bei der Orgel, wenn die Pedale getrennt notiert werden) solcher Systeme festgehalten.

Tonart und Versetzungszeichen

Die Tonart kann in einer MIDI-Datei explizit angegeben werden, das automatische Erkennen derselben ist eher schwierig und nicht verlässlich automatisiert

durchführbar, einige Notensatzprogramme versuchen dies aber durch eigene Algorithmen (vgl. [Go], S. 358). Schwierigkeiten bereiten hier sicherlich auch Änderungen der Tonart innerhalb eines Stückes.
Akzidenzien oder Versetzungszeichen dienen dem Vorschreiben der Veränderung von Tonhöhen durch Vorzeichen (s. [Mi98], S. 67). Die deutschen Namen für diese Vorzeichen lauten: Kreuz, b, Doppelkreuz, Doppel-b, sowie das Auflösungszeichen (s. z.B. [Ja01], S.161). Akzidenzien können gesetzt werden, wenn die grundlegende Tonart bekannt ist. Ein Problem hierbei bereitet die enharmonische Verwechslung, wonach (unter Einbezug von Doppelkreuz und Doppel-b) für jeden Ton zumindest zwei mögliche Schreibweisen existieren[3]. In Dur-Moll-tonaler Musik sind die „lineare Situation, [die] Auf- oder Abwärtsrichtung, sowie [der] harmonische Kontext“ ([Ja01], S. 162) ausschlaggebend für die Wahl der Versetzungszeichen – diese ist aber nicht eindeutig.

Weitere Elemente

Die weiteren Element wie die Setzung von Balken, Bindebögen, Artikulationszeichen und weiterer Interpretationsangaben – vor allem die Herstellung der Zeichen wie in der Originalpartitur – ist großteils schwer automatisiert durchführbar. Angaben wie Dynamik lassen sich zwar prinzipiell aus den mit MIDI gespeicherten Daten ablesen, Gorges merkt aber an, dass ein Computer aufgrund dieser Angaben „fast für jede Note ein Dynamikzeichen setzen würde“ ([Go89], S.353).

1.2 Tondauer und Quantisierung

Eine wesentliche Schwierigkeit der Erstellung einer grafischen Partitur aus einem als MIDI-Datei vorliegenden Musikstück besteht in der Darstellung der Notendauern, da MIDI absolute Tondauern speichert[4], in der herkömmlichen Notation jedoch relative Werte (z.B. halbe Noten, Viertelnoten) verwendet werden. Die entsprechende Umwandlung wäre sehr einfach, wenn das Grundtempo bekannt wäre und alle Noten exakt gespielt worden wären. Das Grundtempo des Stückes kann – ebenso wie die Tonart – explizit angegeben werden, bei der Aufnahme könnte ein Metronom verwendet und der entsprechende Tempowert übertragen werden. Das selbstständige Erkennen eines Tempowertes hingegen ist sehr schwierig, sogar

[3] Es wird die für das Klavier übliche gleichschwebende Stimmung vorausgesetzt. Eine ausgezeichnete Beschreibung der verschiedenen Stimmungen findet sich bei [Ne97] (die gleichschwebende Stimmung wird ab S.44 erklärt).
[4] Genau genommen wird überhaupt keine Tondauer gespeichert, sondern nur die Zeitpunkte des Tonbeginns (NoteOn) und des Tonendes (NoteOff), woraus die Dauer dann ablesbar ist.

weitaus schwieriger als das Erkennen der Tonart. Bei stark akzentuierten Stücken, wie z.B. Tänzen, könnte durch die Betonung einzelner Noten eine Taktzählzeit herausgefunden werden, bei Werken mit irregulären Betonungen ist diese Methode aber kontraproduktiv. Ein weiteres Problem bereiten größere Temposchwankungen innerhalb des Stückes. Diese sind vielfach vorgeschrieben, aber bei der Interpretation von Werken teilweise auch dem Pianisten vorbehalten (vgl. mit Abschnitt *Humanizing* in [Hu99], S.97f).
Geht man von einer Einspielung aus, deren Grundtempo bekannt ist und die keinen gröberen Temposchwankungen unterliegt, sind immer noch feine Abweichungen der eingespielten Noten zu berücksichtigen, welche durch das Spiel eines Menschen gar nicht zu vermeiden sind. Dies betrifft die zeitliche Koordination des Anschlagens und das Aushalten von Notenwerten. Um die Lesbarkeit des Notenbildes zu erleichtern, werden die eingespielten Noten daher quantisiert. Das bedeutet, dass die Werte der Tondauern auf ein vorgegebenes Raster angepasst werden. Schwierig hierbei ist vor allem die Wahl der Rastergröße, welche von der kleinsten verwendeten Note abhängen sollte. Auch ungerade Rhythmen wie Triolen können die Quantisierung erschweren. Viele gängige Sequencer-Programme bieten die Funktionalität des Quantisierens in unterschiedlicher Qualität an. Insgesamt betrachtet kann dieser Schritt zur Partiturerzeugung jedoch als gelöst betrachtet werden.

2 Niederschrift der beiden Klavierhände

Im Gegensatz zu den meisten in der klassischen europäischen Musik verwendeten Instrumenten, besteht der Notensatz eines Klavierwerkes (üblicherweise) aus zwei Fünfliniensystemen, statt aus nur einem. Die Notation in zwei verbundenen Systemen dient – neben einer besseren Lesbarkeit – vor allem der Aufteilung auf beide Hände. Es werden zwar immer wieder einzelne Noten oder Akkorde auch in der Nachbarzeile notiert, den Regelfall stellt jedoch die Notation der linken Hand in der unteren Zeile, die der rechten Hand in der oberen, dar.
Bei der Erzeugung einer Partitur[5] aus einer Einspielung eines Klavierwerkes ergibt

[5] Der Begriff „Partitur“ ist nach [Ja01] ausschließlich für „mehrstimmige Ensemble-Musik“ zu verwenden ([Ja01], S. 168f); mangels alternativer Bezeichnungsweisen werde ich diesen Begriff hier auch für einen grafischen Klaviernotensatz verwenden.

sich daher zusätzlich zu den oben beschriebenen Schritten eine weitere Aufgabenstellung: die Aufteilung des Gespielten auf zwei Notensysteme[6].

Gängige Notensatz- oder Sequenzer-Programme mit der Funktion, eine Partitur aus MIDI-Daten zu erzeugen, notieren Klavierstimmen üblicherweise nur in einer Zeile, oder trennen die gespielten Noten an einer festen Tonhöhe (etwa c^1), um sie so in zwei Zeilen darstellen zu können.
An dieser Stelle sollen die unten beschriebenen Algorithmen ansetzen, um eine sinnvollere Trennung einer Klaviereinspielung auf die beiden Hände zu ermöglichen. Der Unterschied der zuletzt genannten Methoden zur Aufteilung der Klaviereinspielung besteht darin, dass bei letzterer die beiden Notenzeilen wirklich für jeweils eine Hand stehen.

2.1 Zielsetzungen bei der Trennung der Hände

Im Folgenden möchte ich beschreiben, welche Ziele ich mir zur Bearbeitung des beschriebenen Problems und bei der Entwicklung der unten beschriebenen Algorithmen gesetzt habe.

Die grundlegende Aufgabe, die ich mir gestellt habe, ist, in einer als MIDI-Datei vorliegenden Einspielung eines Klavierwerkes die Noten so auf zwei Tracks aufzuteilen, dass jeder der beiden Tracks nach Möglichkeit von einer Hand gespielt werden kann. Es soll also keine grafische Partitur im obigen Sinn erzeugt werden – dieser aufwändige Schritt würde von der eigentlichen Thematik ablenken und kann durch diverse bestehende Software erfolgen.
Nachdem ein Notenbild aus verschiedenen Gründen niemals originalgetreu wiederhergestellt werden kann (s. nachfolgender Abschnitt), sollte auch die Aufteilung der Stimmen auf die Hände wie in der Originalpartitur nicht das höchste Ziel sein. Es sollte vielmehr versucht werden, *ein* gut lesbares und – im Kontext des jeweiligen Stückes – möglichst einfach zu spielendes Notenbild zu erzeugen.

Da die Trennung der Stimmen der beiden Hände wie oben beschrieben nicht an einer bestimmten Tonhöhe erfolgen soll, stellt sich die Frage, welche Kriterien stattdessen dafür herangezogen werden sollen. Hier eignen sich vor allem

[6] Es wird davon ausgegangen, dass beide Hände gemeinsam eingespielt wurden, und nicht getrennt.

Gesichtspunkte der Spieltechnik am Klavier, wozu etwa maximale Spannweiten der Hände oder die Anzahl der gleichzeitig zu spielenden Noten zählen.

Die Werke der Klavierliteratur der letzten Jahrhunderte unterscheiden sich sehr stark voneinander. Daher ist es schwer vorstellbar, dass ein einziger Algorithmus eine optimale Lösung für alle Werke bringen kann. Ich habe stattdessen mehrere Ansätze zur Lösung des Problems verfolgt und jeden darauf aufbauenden Algorithmus nach möglichst einfachen Prinzipien entwickelt, damit er auf eine große Zahl von Werken anwendbar, und das Ergebnis der Anwendung eines Algorithmus möglichst intuitiv nachvollziehbar ist.

3 Von der Aufnahme zur Partitur (Klavier)

Für die konkrete Aufgabenstellung lässt sich in die oben beschriebene Verarbeitungskette von der Aufnahme zur Partitur für Klaviermusik ein weiterer Schritt einfügen. Hier stellt sich noch die Frage, an welcher Stelle, dieser Arbeitsschritt einzufügen ist. Die Noten sollten jedenfalls bereits in quantisierter Form vorliegen, relative Notenwerte, Tonarten, Tempo, Takte, etc. müssen noch nicht bekannt oder errechnet worden sein. Einen möglichen Weg zeigt die folgende Abbildung.

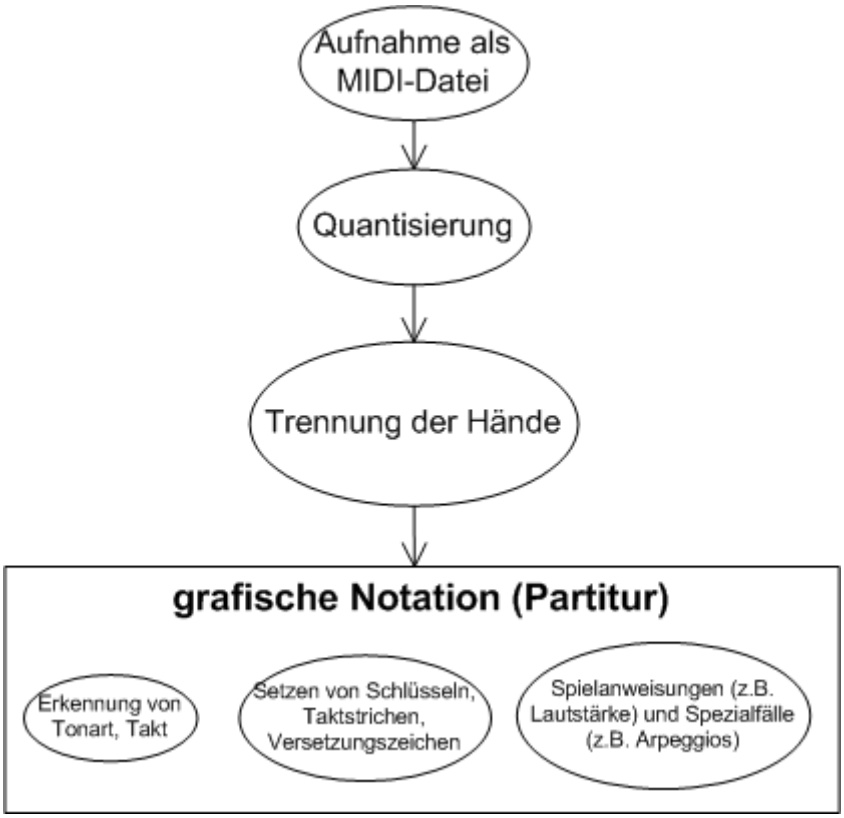


Abbildung 1: Von der Aufnahme zur Partitur

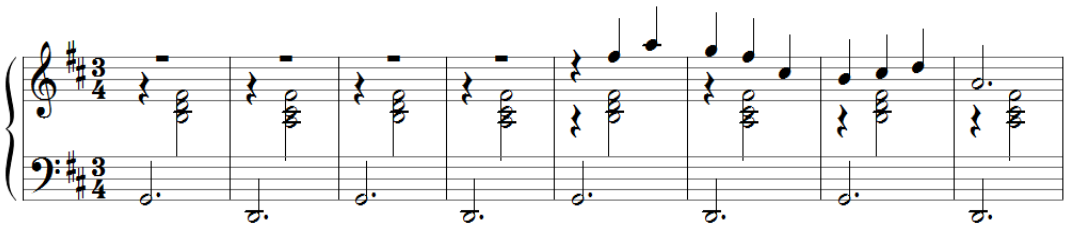
4 Wiederherstellung der Originalpartitur

Aus dem bisher Gesagten geht bereits hervor, dass die Erzeugung des Notenbildes wie in der Originalpartitur keine leichte Aufgabe ist, und manche Elemente im Prinzip nicht wiederhergestellt werden können. Die Gründe dafür sind vor allem in der „redundante[n], nicht kontextfrei[en], nicht immer konsistent[en]“ ([Si00], S.46) Schreibweise der Notenelemente auf der einen Seite, und dem inexakten Spiel jedes menschlichen Spielers auf der anderen Seite, zu suchen.

Viele Spezialfälle der Notation eines Klavierstückes können die Regenerierung einer möglichst originalgetreuen Partitur weiter stören. Hier einige Beispiele dafür:

- Werke, die nur für eine Hand geschrieben wurden,
- die Notation in mehr als zwei Notenzeilen, dies findet sich z.B. bei Rachmaninov: Prelude op.3 Nr.2: im letzten Teil wird jede Hand zur besseren Lesbarkeit in zwei Systemen notiert;
- nicht durchführbare Vorschreibungen, hier ein Beispiel aus Satie: Gymnopedie Nr.1: ab Takt 5 können nicht alle drei Stimmen gleichzeitig gespielt werden,

üblicherweise wird man eine derartige Anweisung durch die Verwendung des Haltepedals ausführen;



Notenbeispiel I-1: Satie, Gymnopedie Nr.1: Takte 1-8

- aleatorische Notation in moderner Musik;
- moderne Anweisungen, die vom „normalen“ Klavierspiel abweichen, wie z.B. das Zupfen einer Saite des Klaviers.

5 Analyse nach technischen Kriterien

Die Analyse eines Klavierwerkes zur Trennung der Hände kann prinzipiell auf zwei verschiedene Arten erfolgen; es können technische – im Sinn der Technik des Klavierspielens – oder musikalische Kriterien herangezogen werden.
Musikalische Aspekte sind schwer objektivierbar und können oft nicht einmal festgemacht werden. Technische Kriterien hingegen können eher verallgemeinert und für eine große Zahl an Stücken als gültig betrachtet werden. So kann man beispielsweise – unabhängig vom zu analysierenden Werk – davon ausgehen, dass ein Intervall von einer Duodezime nicht gleichzeitig von einer Hand am Klavier angeschlagen werden kann.

Die Merkmale, die ich in diesem Sinne für die Analyse berücksichtigt habe, sind:

1. Natürliche Handhaltung
   Im Regelfall sollte die rechte Hand auf der Klaviatur weiter rechts liegen als die linke, Überlappungen sind nach Möglichkeit gering zu halten.
2. Maximalspannweiten der Hände
   Für die maximal greifbaren Intervalle sind feste Grenzen zu setzen, welche unter keinen Umständen überschritten werden dürfen. Als absolutes Höchstintervall empfiehlt sich hier etwa eine Dezime.
3. Anzahl der Noten pro Hand
   Es kann davon ausgegangen werden, dass die Schwierigkeit der Spielbarkeit

eines Akkordes mit der Anzahl der beteiligten Noten steigt. Durch geeignete Verteilung der Noten auf die Hände können so unter Umständen technische Schwierigkeiten reduziert oder verhindert werden.

4. unterschiedliche Rhythmen gleichzeitig in einer Hand
   Akkorde mit Tönen gleichen Rhythmus‘ können von einer Hand leichter gespielt werden, als mehrere Töne unterschiedlicher Länge.
5. Lagenwechsel – Sprünge
   Jeder Lagenwechsel einer Hand bedeutet eine Erschwerung des Klavierspiels, wenn er auch durch Umverteilung der Noten auf die Hände vermieden werden könnte, da er Energie und Zeit erfordert, wobei letzeres vor allem bei schnellem Spiel relevant ist.

Die Liste könnte auch erweitert werden. – bei meinen Analysen berücksichtige ich beispielsweise das Grundtempo eines Werkes nicht. Man könnte, unter Einbezug des Tempos, die Dauer der einzelnen Töne betrachten und Entscheidungen darauf aufbauen; allerdings sind dann Grenzen für die Spielbarkeit von Läufen bestimmter Geschwindigkeit festzulegen, welche meiner Ansicht nach schwer objektivierbar sind.

Mit Analysen nach den genannten technischen Kriterien alleine können viele Stellen von Klavierwerken analysiert, und die Klavierhände so sinnvoll getrennt werden.
Es gibt aber auch Spezialfälle, die nicht auf diese Weise behandelt werden können oder sollten. Im Folgenden werde ich auf einige wichtige Spezialfälle eingehen.

6 Spezialfälle

6.1 Strenge Mehrstimmigkeit

Der Begriff der Mehrstimmigkeit im weitesten Sinn umfasst jegliche Art von gleichzeitigem Klang mehrerer Töne. Das Klavierspiel ist üblicherweise mehrstimmig, selten findet man konzertante Werke mit längeren einstimmigen Passagen. Ein Spezialfall der Mehrstimmigkeit ist in Abschnitten zu finden, welche sich in einzelne vollwertige Stimmen zerlegen lassen. Darunter fallen etwa choralartige Werke (homophon), Kanons oder Fugen (polyphon[7]).

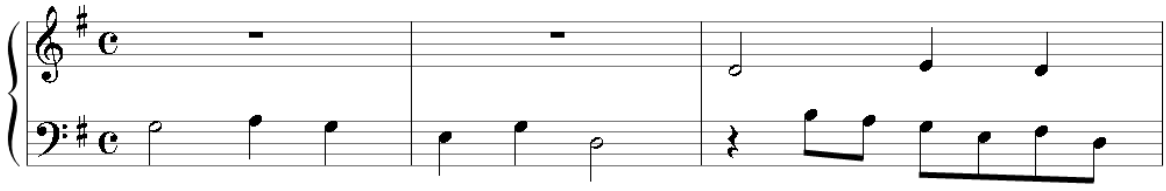
[7] Die Begriffe „homophon“ und „polyphon“ werden z.B. bei [Mi98], S.93 erklärt.

Für das Aufteilen der Hände ergibt sich hier die Aufgabe der möglichst ganzheitlichen Zuordnung jeder Stimme zu einer Hand. Durch Setzen von langen Passagen einer Stimme (oder der gesamten Stimme) in nur eine Hand kann der Zusammenhang dieser Stimme leichter wiedergegeben werden.

Algorithmisch können die Stimmen eines streng mehrstimmigen Werkes getrennt werden, indem jeder Zusammenklang von Tönen zu jedem Zeitpunkt aufgeteilt wird. Das Aneinanderreihen der tiefsten Töne ergibt dann die erste Stimme, das der nächsthöheren die zweite Stimme, und so fort.
Bei einer derartigen Zuordnung kann es aber zu Problemen oder Ungenauigkeiten kommen, wie die folgenden Punkte exemplarisch erläutern.

1. Pausen

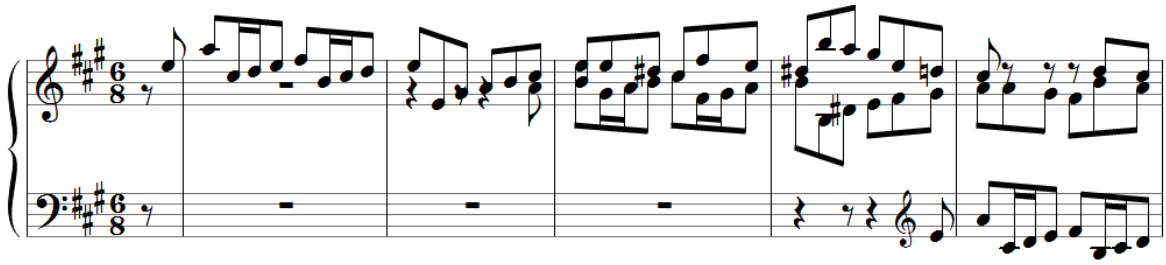
   Unterbrechungen einer Stimme werden bei der genannten Zuordnung als Verlust einer Stimme aufgefasst, da nicht verlässlich festgestellt werden kann, welche Stimme pausiert.



Notenbeispiel I-2: Armsdorff: Fughetta (Choral: Komm, heiliger Geist, Herre Gott): Takte 1-3
Die Pause in Takt 3 führt dazu, dass die beschriebene Analysemethode den ersten Ton der zweiten Stimme ebenfalls der ersten Stimme zuordnet.

2. Stimmkreuzungen

   Vor allem in polyphoner Musik kommen Kreuzungen der beteiligten Stimmen vor. In Klaviermusik können diese auch auftreten, wie das folgende Beispiel zeigt, dies kommt allerdings seltener vor, da Klavierstimmen weniger differenziert klingen können.



Notenbeispiel I-3: Schumann, Fuge: Takte 1-5
In Takt 3 kreuzen die beiden Stimmen, da es sich um eine Fuge handelt, geht dies auch aus dem Kontext hervor, die beschriebene Analyse wird aber die höhere Stimme gleichsetzen mit den höheren Stimmen an den anderen Stellen, und die Überkreuzung nicht erkennen.

3. Gleiche Töne mehrerer Stimmen

   Ebenfalls in polyphoner Vokalmusik oder Musik mit mehreren Instrumenten treffen einander verschiedene Stimmen häufig auf einem Ton. Nach strengem polyphonen Satz kann dies auch für Klavier vorgeschrieben werden – z.B. in Takt 5 der obigen Fuge von Schumann treffen auf den ersten Schlag die unteren zwei Stimmen einander auf dem a –, gespielt werden kann der Ton allerdings nur einmal. Daher wird dies in einer Einspielung als Pause einer Stimme erkannt.

Die beiden letztgenannten Fälle kommen relativ selten in Klaviermusik vor, daher wird die Analyse nach der genannten Methode dadurch nicht erheblich behindert. In einem der unten beschriebenen Algorithmen wird die hier angeführte Methode zur stimmenbezogenen Analyse verwendet.

6.2 Übergreifen

Die natürliche Haltung der Arme und Hände wird beim Klavierspiel manchmal zugunsten des Übergreifens aufgelöst. Das Übergreifen der Hände – auch als Überkreuzen oder Überschlagen bezeichnet – stellt natürlich keineswegs den Normalfall dar, wird aber oft genug vorgeschrieben, sodass es mir wichtig erscheint, diesen Spezialfall mit zu berücksichtigen, insbesondere da er unmittelbare Auswirkungen auf die Zuordnung der Hände hat. Das Übergreifen schließt auch das Untergreifen mit ein, welches nicht explizit davon unterschieden werden muss, da es im Ermessen des Interpreten liegt, ob eine Hand unter- oder übergreift.

Zunächst muss man sich vergegenwärtigen, dass das Übergreifen keine akustischen Auswirkungen hat, sondern vielmehr ein Kunstgriff ist, der in der Live-Musik als optische Komponente eines Klaviervorspiels dieses aufwerten soll. Im Zeitraum des

17.-19. Jahrhunderts, welcher keinen unbeträchtlichen Anteil der „klassischen“ Musik, auf die ich mich hier konzentriere, einnimmt, war das Konzert die einzige Möglichkeit, Musik zu präsentieren. Türk schreibt 1789, dass das „Überschlagen der Hände […] für ein sehr wesentliches Kunsstück gehalten [wurde]“, und dass „verschiedene neuere Komponisten zuweilen blos aus Tändeley die Hände überschlagen [lassen]“ ([Tü89], S.190).

Da man Übergreifen auf einer Musikaufnahme nicht hören kann, kann es bei der Analyse einer MIDI-Datei auch nicht zuverlässig erkannt werden.
Es muss daher versucht werden, anhand der erwähnten technischen Kriterien, Übergreifen sinnvoll zuzuordnen. Im Folgenden möchte ich einen Ansatz beschreiben, der in meinen Augen hierfür zweckmäßig erscheint.

Ich denke, Übergreifen bringt dann keine erhebliche technische Erschwerung, wenn die andere Hand währenddessen ihre Lage auf der Klaviatur nicht ändern muss. Um Übergreifen zuzuweisen, muss also die Lage der Hände untersucht werden, wobei hier festzulegen ist, welche Töne der Lage einer Hand – abhängig von den jeweiligen zuletzt gespielten Noten – angehören. Ich habe einen Algorithmus entwickelt, der diesem Ansatz folgt (Algorithmus X) und so teilweise erfolgreich Übergreifen wie im Original zuweisen kann.

6.3 Arpeggios und weitere spezielle Anschläge

Die bei Sevsay als „Bestandteile der regulären Klaviertechnik“ ([Se05], S. 255ff) angeführten Elemente wie Arpeggios, Triller und Tremoli sind ebenso wie die dort als Spezialeffekte titulierten Glissandi für die Niederschrift einer Klavierpartitur gesondert zu betrachten. Für die Trennung der Hände werde ich diese speziellen Anschlagsformen nicht explizit berücksichtigen. Da Arpeggios, Triller und Tremoli sich in einer Einspielung unwesentlich von entsprechenden einzeln vorgeschriebenen Noten unterscheiden, ergibt auch eine Trennung der Hände in beiden Fällen das gleiche Ergebnis. Im darauffolgenden Schritt der grafischen Notation der bereits getrennten Hände hingegen kann versucht werden, diese Elemente zu erkennen. Einige Programme zur Erzeugung einer Partitur aus MIDI-Daten sind in der Lage z.B. Triller zu notieren, wobei die Ergebnisse auch dort natürlich nicht immer der Originalschrift entsprechen.

6.4 Einander ablösende Hände

Einen weiteren Sepzialfall, welchen ich hier nicht explizit berücksichtigen werde, stellen Abschnitte dar, in denen die Hände – beispielsweise aufgrund einer hohen Geschwindigkeit – abwechselnd gleiche Akkorde oder Akkorde in gleicher Lage spielen sollen. Darunter fallen aber auch Melodielinien, welche abwechselnd gespielt werden sollen. Hierzu finden sich bei [Be65] zahlreiche Beispiele.

6.5 Handwechsel auf einer Note

Selten, aber doch kommt es vor, dass ein Wechsel der Hand auf einer liegen bleibenden Taste erfolgen muss. Ein Beispiel hierfür ist in einem der Teststücke zu finden, und ab Seite 88 beschrieben (durch Übergreifen kann dort der Handwechsel verhindert werden).

Von diesem Spezialfall habe ich bei der Entwicklung der Algorithmen abgesehen, da dies tatsächlich äußerst selten zu finden ist.

II. Algorithmen zur Analyse

Bevor ich in diesem Abschnitt die einzelnen von mir erarbeiteten Algorithmen präsentiere, möchte ich die grundlegende Vorgangsweise beim Analysieren eines Musikstückes beschreiben und Elemente zur Struktur der Analysen erklären.

1 Objekt-Struktur für die Analysen

Da ich die Algorithmen in der objektorientierten Programmiersprache JAVA implementiert habe, werde ich auch Objekte in der hier präsentierten Pseudocode-Variante verwenden. Besonders die im Folgenden beschriebenen Objekte *Note, NoteSet* und *NotePool* spielen für jeden Algorithmus bei der Analyse eine zentrale Rolle; ich werde erklären, welche Funktionalität diesen Objekten zugrunde liegt, welche Werte sie speichern und welche Funktionen sie darüber hinaus zur Verfügung stellen.

Note

Obwohl das Noten-Objekt im Wesentlichen genau das darstellt, was im Allgemeinen als Musiknote verstanden wird – abgesehen von der grafischen Darstellung, halte ich es für wichtig, hier zu erläutern, welche Eigenschaften eine Note ausmachen – und welche für die Analysen wichtig sind.

Die wichtigsten Eigenschaften einer Note sind klarerweise ihre Tonhöhe (im folgenden auch mit dem englischen Begriff *Pitch* bezeichnet) und ihre Dauer (*Duration* oder *Rhythm-Value*), wobei anzumerken ist, dass die Noten einer Partitur als Werte relativ zu einem vom jeweiligen Interpreten gewählten Tempo notiert sind (etwa eine Viertelnote, etc.), bei computergestützten Analysen jedoch bevorzugt absolute Tondauern verwendet werden.

Betrachtet man eine Note in einem größeren Zusammenhang, wird noch ihre Startzeit (und die sich aus der Summe von Startzeit und Dauer ergebende Endzeit) bedeutend.

Weiters soll das Noten-Objekt die vorzunehmende Zuordnung zur rechten oder zur linken Hand speichern können, und diese auch wieder zum Auslesen zur Verfügung stellen. Es ergibt sich folgende Struktur:

```
class Note {

 int        pitch;
 double     rhythmValue;
 double     starttime;
 double     endtime;

 boolean    left = FALSE;
 boolean    right = FALSE;
 boolean    unknown = TRUE;

}
```

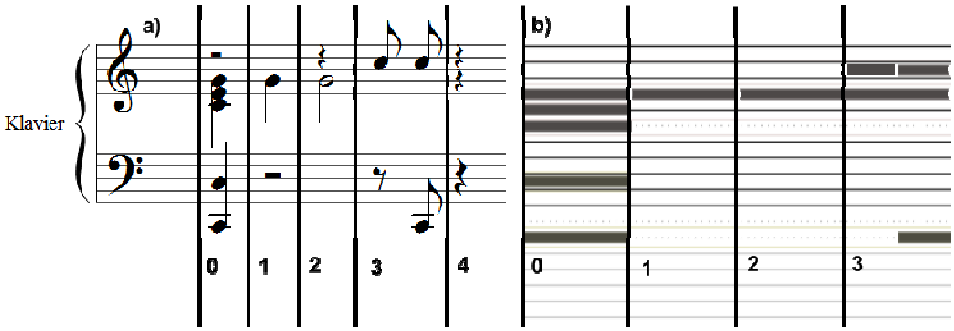
Pseudocode-Beispiel II-1: Klassenstruktur Note

Beim Einlesen einer MIDI-Datei müssen diese Noten-Objekte erst angelegt werden. Eine Note ergibt sich aus den Signalen *NoteOn* und *NoteOff* am gleichen Kanal und mit gleichem *Pitch*. Hier könnten auch weitere Werte wie die Anschlagsdynamik übernommen werden, diese werden jedoch hier nicht weiter benötigt.

NoteSet

Das zweite wichtige Objekt, das ich verwenden werde, möchte ich im weiteren Verlauf als *NoteSet* bezeichnen. Wie der Name bereits vermuten lässt, handelt es sich hierbei um einen Satz von Noten (im obigen Sinn). Darunter ist allerdings nicht eine beliebige Ansammlung von Noten zu verstehen, sondern eine Menge von gleichzeitig klingenden Noten; dabei ist es nicht wichtig, dass diese Noten zum gleichen Zeitpunkt beginnen oder gleich lange andauern. Dies unterscheidet das *NoteSet* von einem *Akkord*, welcher in der Klaviermusik häufig als gemeinsamer Anschlag mehrerer gleich langer Töne einer Hand definiert wird[8].
Eine besondere Bedeutung wird dem c*urrentNoteSet*, dem gegenwärtigen *NoteSet* zukommen, hierzu sei auf den folgenden Abschnitt verwiesen.

[8] Die Definition des Begriffs „Akkord“ variiert sehr stark mit dem jeweiligen musikalischen Zusammenhang. Nach [Fi94] beschreibt der Begriff „im musiktheoretischen Sinne […] elementar verstanden, das Zusammenklingen verschiedener Töne“ ([Fi94], S.356). In der Harmonielehre werden neben dem Zusammenklang auch harmonische Anforderungen an ebendiesen gestellt. Auch die Anzahl und Lage der beteiligten Töne kann eine Rolle spielen. (vgl. [Fi94], S.356ff)



Die Abbildung zeigt Beispiele für verschiedene *NoteSets* in zwei verschiedenen Notationsformen; links eine herkömmliche Notation, rechts das gleiche Beispiel in Blocknotation[9].

Zur Erläuterung der *NoteSet*-Beispiele gehe ich davon aus, dass eine Viertelnote hier die Dauer 1 hat.
Das *NoteSet* im Intervall [0;1) besteht aus den Noten *C*, *c*, c^1,e^1 und g^1,
im Intervall [1;2) aus g^1,
im Intervall [2;3) ebenfalls aus g^1,
im Intervall [3;3,5) aus dem noch klingenden g^1 (!) und aus dem neu angeschlagenen c^2,
im Intervall [3,5;4) aus dem noch klingenden g^1 und aus den neu angeschlagenen Noten c^2 und *C*,
im Intervall [4;5) aus einer Pause.

Da nun geklärt ist, was ich im weiteren Verlauf als *NoteSet* bezeichnen werde, möchte ich auch hier die Attribute, die ein *NoteSet* in der technischen Realisierung zur Verfügung stellen soll, erläutern.

[9] Sowohl die bei uns übliche Notenschrift als auch die Blocknotation – die in vielen Sequencer-Computerprogrammen Verwendung findet und häufig auch als Piano-Roll-Notation (oder Matrix-Editor, s. [Go89], S. 308ff) bezeichnet wird – haben Vorteile gegenüber ihrem Pendant; so lässt sich zweifellos die gewohnte Schrift leichter lesen, klare Längen gehen aus jeder Note hervor, die Blocknotation bietet dafür gerade zur Bestimmung des NoteSets zu einem beliebigen Zeitpunkt den Vorteil, alle Noten, die zu diesem gewählten Zeitpunkt klingen, unmittelbar zu sehen (als vertikalen Querschnitt) – dies ist bei der Notenschrift nicht der Fall, hier muss man sich (insbesondere bei Mehrstimmigkeit in einer Hand) oft erst überlegen, welche Note nun noch klingt.

Neben einer Liste der dem *NoteSet* angehörenden Noten-Objekte[10] werden die Startzeit (entspricht der Startzeit der Note im *NoteSet*, die zuletzt beginnt), sowie die Dauer (als die Dauer der kürzesten Note im *NoteSet*) gespeichert. Weiters sollen folgende Funktionen definiert werden:

- Funktionen zum Abfragen, ob das gesamte *NoteSet* der rechten oder der linken Hand zugeordnet wurde (bzw. ob die Zuordnung des gesamten *NoteSets* noch unbekannt ist), sowie zum Setzen ebendieser Werte,
- Funktionen, die nur die linken, rechten oder unbekannten Noten des *NoteSets* (wiederum als *NoteSet*, welches als Teilmenge des übergeordneten zu verstehen ist) zurückgeben,
- je eine Funktion zum Auslesen der höchsten und der tiefsten Note (also der Note mit dem höchsten bzw. tiefsten *Pitch*-Wert), sowie
- eine Funktion zum Auslesen der durchschnittlichen Tonhöhe.

```
class NoteSet {

 double     starttime;
 double     duration;

 boolean    left = FALSE;
 boolean    right = FALSE;
 boolean    unknown = TRUE;

 function NoteSet getRightNotes();
 function NoteSet getLeftNotes();
 function NoteSet getUnknownNotes();

 function Note getHighestNote();
 function Note getLowestNote();

 function double getAveragePitch();

}
```

Pseudocode-Beispiel II-2: Klassenstruktur NoteSet

[10] Die technische Realisierung dieser Liste ist hier nicht weiter wichtig, da auch keine Leistungs- oder Speicheroptimierung im Vordergrund steht; was ich hier – abstrahiert – als Liste bezeichne kann also beispielsweise als verkettete Liste oder als Array realisiert werden.

Die beschriebenen Funktionen sind einfach zu realisieren und werden hier nicht detailliert angeführt.

NotePool

Bevor mit einer Analyse eines Musikstückes begonnen und die Zuordnung der Noten zu den beiden Händen vorgenommen werden kann, müssen die Daten in die passende Struktur aufgenommen werden. Sobald die Noten in der oben angeführten Weise bekannt sind, können diese in *NoteSets* eingeteilt werden. Wie auch die obigen Beispiele zur Erläuterung des *NoteSet*-Begriffs zeigen, findet sich eine Note häufig in mehreren *NoteSets* wieder. Das *NotePool*-Objekt soll alle Noten eines eingelesenen Musikstücks in *NoteSets* einteilen und diese *NoteSets* verwalten können.
Ich möchte an dieser Stelle noch kein Code-Beispiel für das *NotePool*-Objekt geben, da ich im folgenden Abschnitt hierzu noch auf weitere Aspekte desselben eingehen werde.

2 Grundlegende Ansätze der Analysen

Wie bereits aus dem letzten Absatz hervorgeht, werde ich ein zu analysierendes Musikstück als eine Sammlung von aufeinanderfolgenden *NoteSets* betrachten. Mein grundlegender Ansatz, für die einzelnen Noten eine Zuordnung zu einer der beiden Hände zu treffen, ist ein sequentielles Bearbeiten aller *NoteSets* eines Stückes. Im einfachsten Fall bedeutet das, dass zuerst eine Zuordnung für die Noten des ersten *NoteSet* getroffen wird, danach eine für das zweite, und so fort. Komplexere Algorithmen durchlaufen die Noten auch rückwärts oder springen an andere Stellen im Stück, der grundlegende Ansatz bleibt allerdings auch dort derselbe.

Als weitere Anforderungen an das *NoteSet*-verwaltende Objekt *NotePool* ergeben sich das Speichern des beim sequentiellen Durchlaufen jeweils aktuellen *NoteSets*, sowie die Möglichkeit innerhalb der Liste vor- und zurückzuschreiten. Das jeweils zu bearbeitende *NoteSet* werde ich als *currentNoteSet* bezeichnen.
Es ergibt sich die nachfolgende Struktur für das Objekt *NotePool*:

```
class NotePool extends List of NoteSets {

      int currentNoteSetIndex      =  0;

      function NoteSet getCurrentNoteSet();

      function double getCurrentTime();

      function NoteSet getRecentNoteSet();

      function double getRecentTime();

      function boolean hasRecentNoteSet();

      function NoteSet getNextNoteSet();

      function double getNextTime();

      function boolean hasNextNoteSet();

      method forward();

      method backward();

}
```

Pseudocode-Beispiel II-3: Klassenstruktur NotePool

Die noch nicht erwähnten Funktionen *getRecentNoteSet*, *getRecentTime* und *hasRecentNoteSet* dienen der Abfrage des *NoteSets* vor dem *currentNoteSet*, der Startzeit und der Überprüfung auf Vorhandensein desselben (analog die Funktionen *getNextNoteSet*, etc.).

Ich möchte zwei verschiedene Ansätze zur Analyse unterscheiden: die vertikale und die horizontale.
Bei der vertikalen Analyse wird ausschließlich der vertikale Querschnitt im Notenbild zum aktuellen Zeitpunkt – das *CurrentNoteSet* – zur Entscheidung über die Zuordnung der Noten zu den beiden Händen herangezogen. Eventuelle bereits entschiedene Noten, die auch vorangehenden *NoteSets* angehören, werden nach Möglichkeit in ihrer Zuordnung nicht mehr geändert.
Die horizontale Analyse hingegen verwendet nicht nur das aktuelle *NoteSet* zur

Entscheidung, sondern auch ein oder mehrere vorangehende und nimmt somit Rücksicht auf den horizontalen Zusammenhang der Noten im Notenbild[11].

Diese Einteilung lässt sich gut mit den musikalischen Eigenschaften, die – eingangs beschrieben – der Analyse dienen können, vergleichen. Die vertikale Analyse orientiert sich vor allem an der musikalisch-technischen Einschränkung dessen, was gleichzeitig gespielt werden kann.
Horizontale Algorithmen berücksichtigen diese (und eventuell) weitere technische Einschränkungen ebenfalls, können darüber hinaus aber auch aus dem Zusammenhang, den sie definitionsgemäß in der ein oder anderen Form mitberücksichtigen, musikalische Aspekte in die Analyse aufnehmen. So ist es beispielsweise möglich, Linien (wie polyphone Stimmen) bei entsprechenden Rahmenbedingungen möglichst ganzheitlich einer Hand zuzuordnen.

Es mag nicht überraschen, dass im Allgemeinen horizontale Algorithmen leistungsfähiger sind als vertikale. Dennoch ist auf letztere nicht zu verzichten, da sie einerseits trotz wesentlich einfacher Grundideen teilweise sehr gute Resultate liefern, und andererseits eine wichtige Grundlage und Referenz für horizontale Algorithmen bilden.

In den nächsten Kapiteln werde ich meine Algorithmen präsentieren (eingeteilt in vertikale und horizontale). Neben der musikalischen Idee, technischen Details der Realisierung (wo nötig) und Ausführung in Pseudo-Code, werde ich Beispiele geben, um besondere Stärken oder Schwächen des jeweiligen Algorithmus hervorzuheben. Für detailliertere Beschreibungen der Leistungsfähigkeit der Algorithmen sei auf Teil III verwiesen. Die Präsentation in Pseudo-Code-Form werde ich kurz halten, detailliertere Ausführungen derselben finden sich im Anhang.

3 Vertikale Algorithmen

Für die Zuordnung der Noten eines *NoteSets* zu den Händen sind zwei Fälle zu unterscheiden: handelt es sich um eine einzelne Note oder um mehrere, also um einen Akkord.

[11] Korrekterweise müsste diese Form der Analyse vertikal-horizontale Analyse heißen, da die vertikalen Eigenschaften nicht ignoriert werden (dürfen). Um den Unterschied zu verdeutlichen werde ich aber bei der Bezeichnung „horizontale Analyse“ bleiben.

Einzelne Noten können nicht anhand vertikaler technischer Kriterien zugeordnet werden. Die einzige Möglichkeit würde darin bestehen, alle Noten über einem bestimmten Notenwert (z.B. c^1) der rechten Hand zuzuweisen, alle Noten darunter der linken Hand. Hier ist eine horizontale Analyse klar zu bevorzugen.

Akkorde hingegen können auch durch vertikale Aspekte allein analysiert werden.

3.1 Analyse weiter Akkorde – Algorithmus I

Bei der Analyse von Akkorden können wieder zwei Fälle unterschieden werden wobei danach vorgegangen wird, ob alle Töne eines Akkordes in einer Hand gespielt werden könnten (Fall 1) oder nicht (Fall 2).
Im zweiten Fall kann eine eindeutige Zuordnung der äußersten Töne vorgenommen werden, der höchste Ton rechts, der tiefste links[12]. Auch alle anderen Töne, die in einer der beiden Händen aufgrund der maximalen Spannweite der Hand, welche vorher festzulegen ist, nicht gespielt werden können, können der entsprechenden anderen Hand zugeordnet werden. Diese Akkorde bezeichne ich auch als „weite Akkorde“.
Als gut spielbares Maximalintervall stellt die Oktave einen guten Wert dar, unter Umständen können aber auch weitere Intervalle erforderlich sein, die Obergrenze kann man hier etwa bei einer kleinen oder großen Dezime annehmen.

Dies stellt den einzigen Fall dar, in dem die Zuordnung nicht einem willkürlich gewählten Schema folgt. Der daraus resultierende Algorithmus ist auch als einziger unbedingt notwendig, weitere Algorithmen versuchen, das betreffende Musikstück leichter lesbar bzw. spielbar zu machen.

Der Algorithmus zur Zuordnung der weiten Akkorde realisiert genau das oben beschriebene: alle Töne, welche aufgrund der Entfernung zu einem der äußersten Töne in einer Hand nicht gespielt werden können, in der anderen aber schon, werden dieser zweiten Hand zugeordnet. Töne, die in beiden Händen gespielt werden können, werden nicht zugeordnet. Die Entscheidung darüber wird einem anderen Algorithmus überlassen.

Es ergibt sich nachstehender Algorithmus.

[12] Natürlich könnten die beiden Hände auch vertauscht werden um so Übergreifen vorzuschreiben. Im Allgemeinen kann davon aber abgesehen werden.

```
method splittingAlgorithmI(NoteSet currentNoteSet) {

 Note lowestnote = currentNoteSet.getLowestNote();
 Note highestnote= currentNoteSet.getHighestNote();

 lowestnote.setLeft(TRUE);
 highestnote.setRight(TRUE);

 for( all other unknown Notes in currentNoteSet ) {

  if( (highestnote.getPitch() – note.getPitch() > MAX_INTERVAL) &&
  (note.getPitch() – lowestnote.getPitch() > MAX_INTERVAL) {

   //Illegal Interval
   break up;

  }
  else if( highestnote.getPitch() – note.getPitch() > MAX_INTERVAL ) {

   note.setLeft(TRUE);

  }
  else if( note.getPitch() – lowestnote.getPitch() > MAX_INTERVAL ) {

   note.setRight(TRUE);

  }

 }

}
```

Pseudocode-Beispiel II-4: Algorithmus I

Wie oben erwähnt, ist dieser Algorithmus getrennt von den weiteren zu betrachten, da er bewusst nicht in jedem Fall eine Lösung liefert (sondern nur bei Tönen, deren Abstand groß genug ist). Es finden sich jedoch zahlreiche Beispiele in der Klavierliteratur, bei denen dieser Algorithmus zu großen Teilen oder sogar zur Gänze eine Lösung – und damit (bis auf Übergreifen) die einzige mögliche – liefert.
Unter meinen Teststücken sei hier zum Beispiel auf große Abschnitte von Mussorgsky, *Das große Tor von Kiew*, Brahms, *Ballade* oder Schostakowitsch, *Fantastischer Tanz Nr. 2* verwiesen. Generell lassen sich virtuosere Stücke mit Akkorden in beiden Händen, wie sie ab der Romantik häufig geschrieben wurden, zu weiten Teilen bereits durch diesen Algorithmus analysieren.

Die weiteren vertikalen Algorithmen dienen der Zuordnung der Töne, die aufgrund ihrer Entfernung am Klavier zum höchsten bzw. tiefsten der gleichzeitig gespielten Töne mit jeder der beiden Hände gegriffen werden könnten.

3.2 Kleine Spannweiten der Hände – Algorithmus II

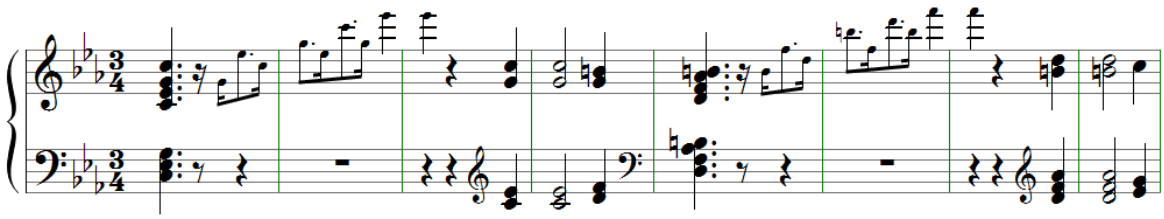
Eine naheliegende Möglichkeit, die verbliebenen Töne zuzuordnen besteht darin, die geringere Entfernung zum tiefsten bzw. höchsten Ton als Entscheidungskriterium heranzuziehen.

Detailliert bedeutet das, dass der höchste Ton rechts und der tiefste Ton links gespielt wird, für jeden übrigen Ton gilt, dass er links zugeordnet wird, falls der Abstand zum tiefsten Ton kleiner dem Abstand zum höchsten Ton ist, andernfalls rechts. Für diesen Vergleich werden die Halbtonschritte gezählt, jeder „natürliche Halbtonschritt“ im Bereich erhöht ebenfalls den Zähler, da er auf der Klaviatur ebenso viel Platz einnimmt wie zwei Halbtonschritte an anderen Stellen.
Falls der Abstand gleich groß ist, wird der Ton rechts gespielt, dies könnte allerdings auch genau umgekehrt behandelt werden und ist eine Frage der Festlegung.

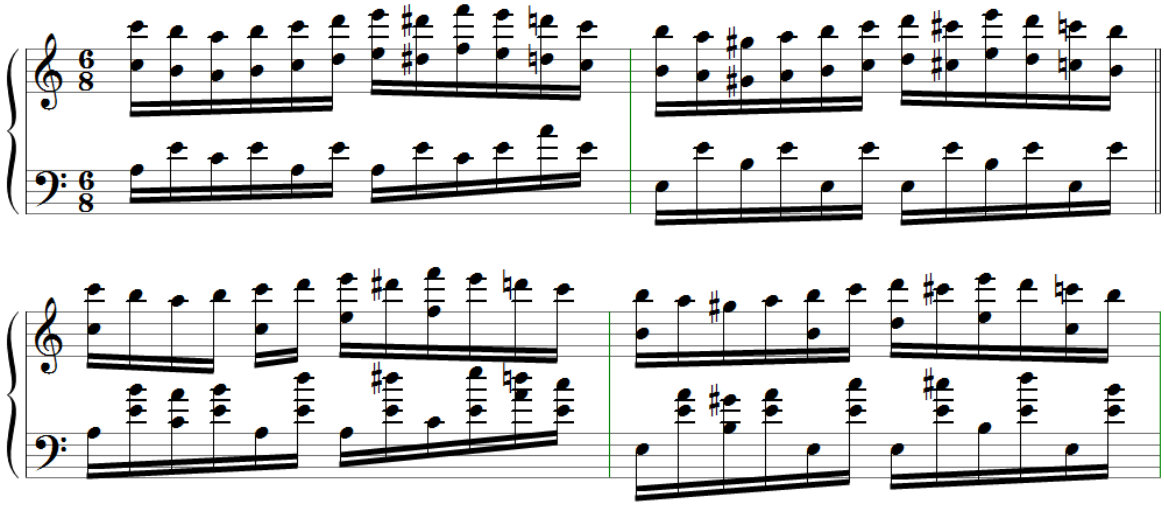
Die Zuordnung durch diesen Algorithmus bewirkt, dass beide Hände zu jedem Zeitpunkt eine minimale Spannweite benötigen. Dies bedeutet daher eine Minimierung der technischen Schwierigkeiten.
Darüber hinaus gilt anzumerken, dass dieser Algorithmus ohne jeden weiteren auskommt, er benötigt vorher nicht unbedingt eine Analyse der „weiten Akkorde“ und lässt auch keine Töne unbestimmt[13], dies gilt nicht für alle der weiteren Algorithmen.

Es finden sich aber auch einige Beispiele, bei denen dieser Algorithmus nicht hundertprozentig zufriedenstellende Ergebnisse liefert. Der Grund dafür mag darin zu suchen sein, dass mit dieser Methode jeder Akkord (jedes *NoteSet* aus mehreren Tönen) unweigerlich geteilt wird, auch wenn er in einer Hand gespielt werden könnte. Dadurch ergeben sich unnötige Sprünge oder abwechselndes akkordisches Spielen der beiden Hände, was Linien zerstören und technische Schwierigkeiten mit sich bringen kann.

[13] Dies gilt auch in diesem Fall nur unter der Voraussetzung, dass sich alle Noten auch tatsächlich mit den festgelegten Maximalintervallen zuordnen lassen.



Notenbeispiel II-1: Beethoven, Sonate Nr. 5, Takte 1-8
Die Akkorde werden so aufgeteilt, dass sie gut spielbar sind. Die klein dargestellten Noten wurden nicht durch diesen, sondern durch einen Algorithmus zur Analyse einzelner Noten (Algorithmus VIII) zugeordnet.



Notenbeispiel II-2: Mozart, Var. III aus Sonate KV300i, Takte 5-6 im Original (o.) und nach der Analyse (u.)
Die im Original vorgesehene Oktavenbewegung wird „zerstört“ und die Töne der Mittelstimme werden unregelmäßig auf die beiden Hände aufgeteilt; dadurch ergeben sich neben einem schlecht lesbaren Notenbild auch große Sprünge in der linken Hand.

Die Notenbeispiele zeigen ein sehr zufriedenstellendes Ergebnis durch die Analyse mit diesem Algorithmus, und ein weniger gutes.

Jedenfalls kann diese Entscheidungsvorschrift als Sekundäralgorithmus für andere Algorithmen dienen, wo diese keine – oder keine eindeutige – Entscheidung treffen können. Es folgt eine Beschreibung in Pseudocode.

```
method splittingAlgorithmII( NoteSet currentNoteSet ) {
 if( currentNoteSet.getHighestNote().isUnknown() ) {
  currentNoteSet.getHighestNote().setRight(TRUE);
 }
 if( currentNoteSet.getLowestNote().isUnknown() ) {
  currentNoteSet.getLowestNote().setLeft(TRUE);
 }
 for( all other unknown Notes in currentNoteSet ) {
  int differenceToLowestNote = note.getPitch() -
  currentNoteSet.getLowestNote().getPitch();
  int differenceToHighestNote = currentNoteSet.getHighestNote().getPitch()
  - note.getPitch();
  for(int j=note.getPitch(); j<noteSet.getHighestNote().getPitch(); j++) {
   if( (pitch % 12 == 4) || (pitch % 12 == 11) ) {
    differencetohighest++;
   }
  }
  for differenceToLowestNote vice-versa;
  if( differenceToLowestNote < differenceToHighestNote ) {
   note.setLeft(TRUE);
  }
  else {
   note.setRight(TRUE);
  }
 }
}
```

Pseudocode-Beispiel II-5: Algorithmus II

3.3 Gleichmäßige Aufteilung – Algorithmus III

Eine weitere, sehr naheliegende Methode der Verteilung der Töne auf die beiden Hände, ist die der gleichmäßigen Aufteilung.

Hier wird bestrebt, möglichst gleich viele Töne in beiden Händen zu spielen. Eine ungerade Anzahl an Tönen oder zu große Distanzen der Töne untereinander können einer ganz gleichmäßigen Zuordnung widersprechen.

```
method splittingAlgorithmIII( NoteSet currentNoteSet ) {

 int countLeft = 0;
 int countRight = 0;

 if( currentNoteSet.getHighestNote().isUnknown() ) {

  currentNoteSet.getHighestNote().setRight(TRUE);

 }

 if( currentNoteSet.getLowestNote().isUnknown() ) {

  currentNoteSet.getLowestNote().setLeft(TRUE);

 }

 for( all Notes in currentNoteSet ) {

  if( note.isLeft() ) {

   countLeft ++;

  } else if( note.isRight() ) {

   countRight ++;

  }

 }

 for( all unknown Notes in currentNoteSet, outer to inner ) {

  if( countRight <= countLeft ) {

   note.setRight( TRUE );
   countRight ++;

  }
```

```
  else {

   note.setLeft( TRUE );
   countLeft ++;

  }

 }

}
```

Pseudocode-Beispiel II-6: Algorithmus III

Die Noten werden nach ihrer Tonhöhe sortiert und dann von außen nach innen durchlaufen. Solange die Intervallentfernungen von den äußersten Tönen es zulassen, werden die Noten abwechselnd auf die Hände verteilt, wodurch sich eine möglichst gleiche Auslastung ergibt.

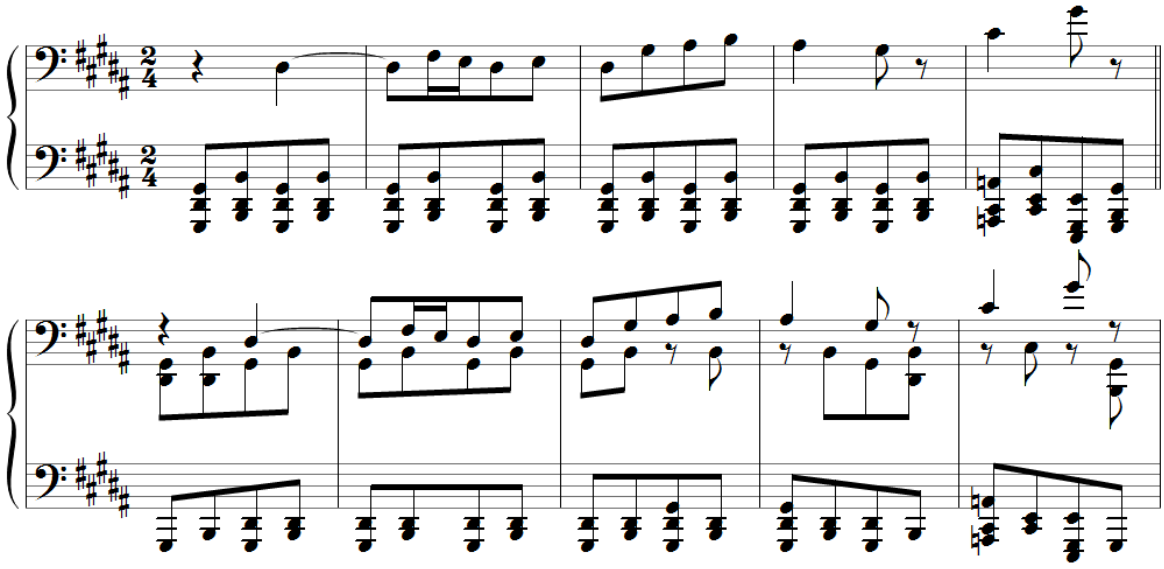
Eine gleiche Aufteilung hat sicherlich eine musikalische Berechtigung, da eine gleichmäßige Auslastung beider Hände üblicherweise technische Vereinfachung bedeutet.

Wie auch beim zuletzt beschriebenen Algorithmus II, kann die gleichmäßige Aufteilung gut als Sekundäralgorithmus eingesetzt werden, wo durch den Primäralgorithmus keine Entscheidung getroffen wird, da Algorithmus III alle (spielbaren) Töne sicher zuordnet.

Gute Ergebnisse liefert dieser Ansatz auch bei streng vielstimmigen Werken, bei denen sich die Anzahl der Stimmen nicht oft ändert. Dies gilt insbesondere bei streng zweistimmigen Werken, wie etwa *Telemann, Fantasia* (hierbei liefert auch die Analyse nach geringsten Spannweiten die gleichen Ergebnisse, da bei beiden Algorithmen die äußersten beiden Stimmen auf die Hände aufgeteilt werden).
Die Analyse solcher Werke, die generell besser durch horizontale Algorithmen zu tätigen ist, kann durch diese – eigentlich sehr einfache – Methode implizit ebenso vorgenommen werden.

Weniger optimal zeigt sich der Ansatz – wie auch der zuvor beschriebene – bei der Analyse von Akkorden, die auch vollständig von einer Hand gespielt werden könnten. Hier können sich wieder unnötige Sprünge ergeben. Auch kann sich durch die

gleichmäßige Aufteilung eine Erschwerung durch die Setzung verschiedener Rhythmen in eine Hand ergeben, wie das folgende Notenbeispiel veranschaulicht.



Notenbeispiel II-3: Musorgsky, Bydlo, Takte 1-5 im Original (o.) und nach der Analyse (u.)
Die Akkorde der – stetig gleichartig gesetzten – Begleitung werden geteilt und führen neben der Mehrbelastung der melodieführenden rechten Hand auch zu großen Sprüngen, vor allem an Stellen wo die Melodiestimme pausiert (Takte 4, 5)

3.4 Gleichartige Rhythmen – Algorithmen IV und V

Die bisher vorgestellten Algorithmen verwenden alle lediglich die Eigenschaft der Tonhöhe als Grundlage für die Zuordnung der einzelnen Noten eines Akkords.
Die wesentlichen Schwierigkeiten, die die vorangegangenen Analysemethoden nicht lösen können, sind das Auftreten von unnötig komplizierten rhythmischen Konstrukten und großen Sprüngen. Letzteres Problem ist mit vertikaler Analyse im Allgemeinen nicht zu lösen, für zuerst genanntes lassen sich aber Methoden zur Lösung des Problems finden.
Die grundlegende Idee hierfür besteht darin, nicht nur die Tonhöhe, sondern auch die Dauer des Tons zur Analyse heranzuziehen.

Die folgenden beiden Ansätze verwenden die Dauer als primäres Entscheidungskriterium, die Tonhöhe wird natürlich auch berücksichtigt, da sich sonst zu große Intervalle ergeben könnten.

Nachstehender Algorithmus IV ordnet nach Möglichkeit alle Töne des gleichen Rhythmus‘ (der oben erwähnte *RhythmValue*) auch der gleichen Hand zu. Dabei muss ein Ton dieser Dauer bereits durch einen Sekundäralgorithmus zugeordnet

worden sein. Dies erfolgt durch die Analyse der weiten Akkorde, oder – wenn nötig – durch einen explizit aufgerufenen anderen Algorithmus; hierfür kommen etwa die oben beschriebenen Varianten (II und III) in Frage.

```
method splittingAlgorithmIV( NoteSet currentNoteSet ) {

 sort Notes by pitch;

 NoteSet tempNoteSet = new NoteSet();
 add all yet decided Notes to tempNoteSet;

 boolean isLeft = FALSE;
 boolean isRight = FALSE;

 for( all unknown Notes in currentNoteSet, outer to inner ) {

  tempNoteSet.add( note );

  isLeft = FALSE;
  isRight = FALSE;

  for( all yet decided Notes of currentNoteSet as tempNote ) {

   if(starttime&rhythmValue of note exactly match the values of tempNote) {

    set isLeft = TRUE if tempNote.isLeft() or isRight = TRUE else;

   }

  }

  if( either only isLeft or only isRight ) {

   set note to the corresponding hand;

  }
  else decide tempNoteSet with secondary algorithm;

 }

}
```

Pseudocode-Beispiel II-7: Algorithmus IV

In der angeführten Pseudocode-Implementierung des Algorithmus werden alle bereits zugeordneten Noten in einem *NoteSet* (*tempNoteSet*) gespeichert. Beim Durchlaufen aller Noten des *currentNoteSet* wird die Note direkt zugeordnet falls es bereits eine zugeordnete Note derselben Startzeit und derselben Dauer gibt. Andernfalls wird das *tempNoteSet* – mit der aktuell zu entscheidenden Note als einzig noch nicht zugeordnete – durch einen Sekundäralgorithmus aufgeteilt; auf dessen Ergebnisse erfolgt dann die weitere Analyse. Die besten Resultate werden hier erzielt, wenn die Noten von außen nach innen (tiefster/höchster Ton zuerst, dann die dazwischen liegenden) bearbeitet werden.

Die Analyse mit diesem Algorithmus liefert sehr unterschiedliche Ergebnisse, stark abhängig vom jeweiligen Musikstück. Werke, die akkordisches Greifen forcieren, lassen sich sehr gut analysieren, schlechte Zuordnungen sind vor allem bei streng zwei- oder mehrstimmigen Werken zu finden, da hier bei jedem gleichzeitigen Auftreten von Tönen der gleichen Dauer (innerhalb gesetzter Grenzen) der gesamte Satz an Tönen als Akkord aufgefasst und einer Hand zugeordnet wird.

Als Beispiele für sehr gute Ergebnisse sind u. a. Chopin, *Nocturne* und Mussorgsky, *Bydlo* aus *Bilder einer Ausstellung* zu nennen.
Allgemein gilt, dass hier die Resultate nicht zuletzt von der Wahl des Sekundäralgorithmus abhängen.

Eine weitere Schwachstelle ergibt sich daraus, dass nur Rhythmen zugeordnet werden, die bereits in einer Hand vertreten sind.
Diesen Schwächen versucht der nächste Algorithmus V entgegenzuwirken:
Nach der Analyse der weiten Akkorde, werden alle Töne des *currentNoteSet* in *NoteSets* des gleichen Rhythmus (gleiche Startzeit und gleiche Dauer) eingeteilt. Innerhalb jedes dieser *NoteSets* werden alle spielbaren Noten der gleichen Hand zugeordnet, sofern bereits zumindest ein Ton zugeordnet wurde und nicht Töne beider Hände in diesem *NoteSet* sind.
Falls bis zu dieser Stelle noch keine Noten zugeordnet wurden, werden die äußersten beiden Töne auf die Hände aufgeteilt.
Alle *NoteSets*, innerhalb derer auf obig beschriebene Art noch keine Zuordnung getroffen werden konnte, werden möglichst vollständig in die Hand gesetzt, in die auch das erste rhythmisch sortierte *NoteSet* gesetzt wurde, falls das *NoteSet* in dessen Nähe ist (also innerhalb der gleichen Hälfte von rhythmisch sortierten

NoteSets), andernfalls in die andere Hand. Dieses etwas sonderbar anmutende Konstrukt, bewirkt, dass Töne, die einander in Startzeit und Dauer ähnlich (aber nicht notwendigerweise exakt gleich) sind, möglichst in der gleichen Hand gespielt werden.
Denkbar wäre auch, die letzte Zuordnung so zu verändern, dass ähnliche Rhythmen möglichst nicht von der gleichen Hand gespielt werden; die betreffende Code-Stelle ist ebenfalls abgebildet.

```
method splittingAlgorithmV( NoteSet currentNoteSet ) {
 sort Notes first by starttime, then by rhythmValue;
 NoteSet-List rhythmSets = List of NoteSets where every NoteSet contains
 only Notes of the same starttime and rhythmValue, the List is sorted by
 starttime, then by rhythmValue;
 for( all rhythmSets ) {
  sort rhythmSet by pitch;
  if( rhythmSet has left but no right Notes (or vice-versa) ) {
   set all playable Notes to left (or respectively right);
  }
 }
 if( still no Notes decided ) {
  set highestNote to right and lowestNote to left;
 }
 for( all not yet decided rhythmSets ) {
  if( rhythmSetIndex > rhythmSets.size()/2 ) {
   if( rhythmSets.get(0).isLeft() ) {
    set all playable Notes to left;
   }
```

```
    else if( rhythmSets.get(0).isRight() ) {
     set all playable Notes to right;
    }
  }
  else {
    do vice-versa;
  }
}
}
```

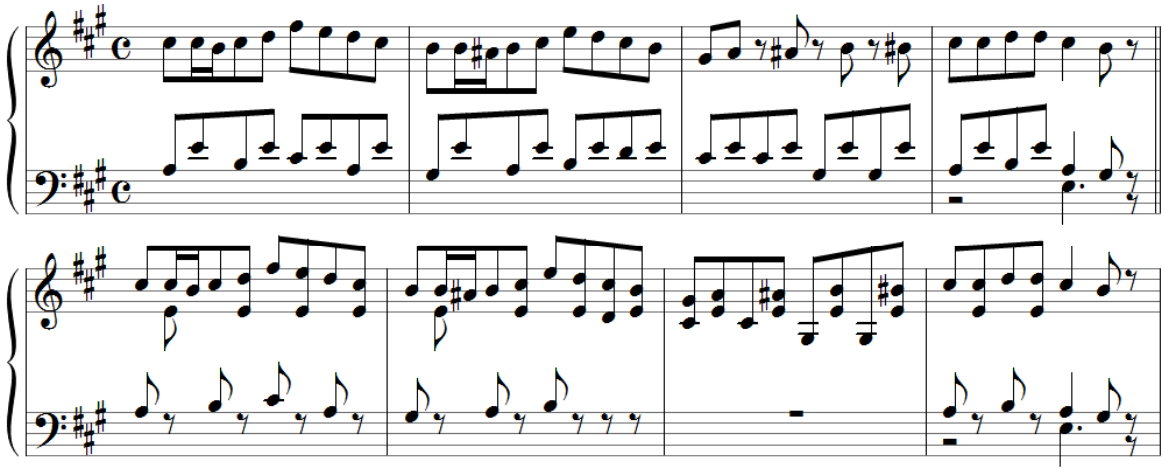
Pseudocode-Beispiel II-8: Algorithmus V

```
for( all rhythmSets ) {
 if( rhythmSetIndex <= rhythmSets.size()/2 ) {
  if( rhythmSets.get(0).isLeft() ) {
    set all playable Notes to left;
  }
  else if( rhythmSets.get(0).isRight() ) {
    set all playable Notes to right;
  }
 }
 else do vice-versa;
}
```

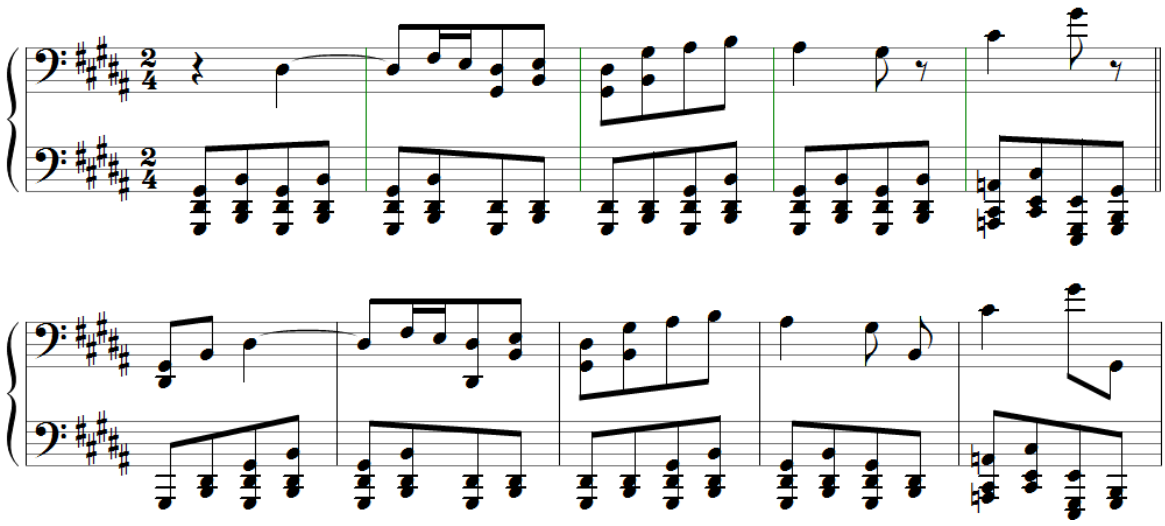
Pseudocode-Beispiel II-9: Algorithmus V - umgekehrte Zuordnung der verschiedenen Rhythmen

Dieser Algorithmus greift in vielen Fällen auf einen Sekundäralgorithmus zurück (etwa Algorithmus II oder III), und schafft es dadurch, sowohl Tonhöhen als auch Rhythmen als gleichwertige Entscheidungskriterien heranzuziehen. Es gibt daher zahlreiche Beispiele unterschiedlicher Art, bei denen dieser Algorithmus einer sehr guten Lösung entgegen strebt.

Das bedeutet aber auch, dass nicht zwangsläufig alle Töne gleichen Rhythmus‘ immer der gleichen Hand zugeordnet werden – besonders bei den in der Romantik häufig auftretenden Oktavenbewegungen liefert der erste Algorithmus zur Analyse gleicher Rhythmen (Algorithmus IV) mitunter bessere Ergebnisse; im Folgenden zeige ich zwei Beispiele, die diese beiden Methoden vergleichen.



Notenbeispiel II-4: Mozart, Sonate Variation VI, Takte 1-4: Analyse mit Algorithmus V (o., entspricht dem Original) und mit Algorithmus IV (u.)
Die Töne der beiden Stimmen, die fast an jeder Stelle die gleiche Tondauer haben, werden von Algorithmus IV überall, wo dies möglich ist, der rechten Hand zugeordnet, welche dadurch beide Stimmen spielen muss, während die linke Hand teilweise pausiert, außerdem entstehen unnötige Sprünge (Takt 3). Die Zuordnung durch Algorithmus V entspricht der des Originalsatzes. Hier sind die Stimmen gleichmäßig aufgeteilt.



Notenbeispiel II-5: Musorgsky, Bydlo, Takte 1-5: Analyse mit Algorithmus IV (o.) und mit Algorithmus V
Algorithmus IV ordnet die Töne in diesen Takten fast wie im Original (vgl. o.) zu. Die stetige Akkordbegleitung wird meistens zur Gänze in die linke Hand gesetzt, Algorithmus V weist diese Akkorde weniger konsequent einer Hand zu, das Ergebnis ist dennoch gut spielbar (vgl. mit Algorithmus II, o.)

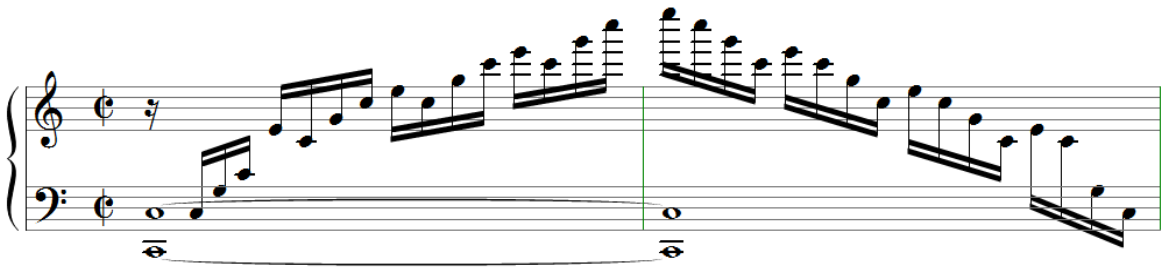
3.5 Bewertung der vertikalen Algorithmen

Die Analyse mit vertikalen Algorithmen ermöglicht – bei geeigneter Wahl der Methode – mit relativ einfachen Mitteln die Aufteilung von gleichzeitig gespielten Tönen auf die beiden Hände. Die Entscheidung erfolgt einzig durch die Werte „Tonhöhe“ und „Tondauer“ der Noten zu einem bestimmten Zeitpunkt. Bei der Analyse eines Werkes werden sequentiell alle verschiedenen vertikalen Notenquerschnitte betrachtet, und jeder einzeln entschieden.

Die wesentliche gemeinsame Schwachstelle dieser Algorithmen ist das durch das Ignorieren der Vorgeschichte eines *NoteSets* herrührende häufige Erzeugen von großen Sprüngen einer Hand, wo dies durch Umordnung der Noten leicht vermieden werden könnte. In langsam zu spielenden Werken stellt dieser Sachverhalt üblicherweise keine technische Schwierigkeit dar, kritisch wird es jedoch bei schnellen Passagen, die durch die schlechte Zuordnung nur unter großer Anstrengung oder gar nicht gespielt werden können. Da ich explizite Zeitbezüge nicht als Analysekriterium heranziehen will, gilt es generell, diese Sprünge zu vermeiden.

Bevor ich die horizontalen Algorithmen präsentiere, die versuchen, genau diese Lücke zu schließen, und teilweise auch besser auf die Vielstimmigkeit eingehen, möchte ich noch auf eine für alle Analysen (auch, oder besonders für die vertikalen

Algorithmen) relevante Gegebenheit eingehen:
Ich habe bis jetzt verschwiegen, dass es bei einer sequentiellen Analyse zu Fehlern kommen kann, die nachträglich korrigiert werden müssen. Zu jedem Zeitpunkt werden die bis dahin nicht zugeordneten Töne des *currentNoteSet* auf die beiden Hände aufgeteilt und es wird kontrolliert, ob es an dieser Stelle so gegriffen werden kann. Dass die Noten an dieser Stelle auch wirklich alle gespielt werden können, bedeutet jedoch nicht, dass sie in weiterer Folge auch noch in der gleichen Hand gespielt werden können in die sie gesetzt wurden, falls sie über mehrere *NoteSets* hinweg andauern. Zur Erläuterung möchte ich nachfolgendes Notenbeispiel II-6 hinzuziehen.



Notenbeispiel II-6: Chopin, Etude op.10 Nr.1 in C, Takte 1-2

Die einzige vernünftige Art, die angegebenen Noten zu interpretieren, ist, den Akkord *C-c* links und alle anderen Töne rechts zu spielen (wobei ich hier davon ausgehe, dass die beiden Sechzehntelnoten auf c nicht gespielt werden, da die entsprechenden Tasten bereits von der linken Hand gegriffen werden[14]) – dies ist für einen menschlichen Interpreten leicht erkennbar, da er auf einen Blick sehen kann, dass alle anderen Methoden nicht möglich sind.
Betrachten wir nun die Analyse mit einem der oben angeführten Algorithmen: Zum Zeitpunkt 0 besteht das *NoteSet* aus den beiden Tönen *C* und *c*. Weist der Algorithmus der linken Hand das *C* und der rechten das *c* zu (wie es die Algorithmen II und III tun würden), kommt es in weiterer Folge zu Problemen, da spätestens ab dem zweiten Viertelschlag die Töne keiner Hand mehr zugeordnet werden können.

Prinzipiell spricht nichts dagegen, die Analyse trotzdem so durchzuführen, allerdings müssen gegebenenfalls Noten korrigiert werden, falls dieser Fall auftritt.

[14] Selbstverständlich gibt es auch andere Möglichkeiten, mit diesen Noten umzugehen, der Einfachheit halber möchte ich jedoch bei der genannten bleiben.

Anschließend stelle ich eine Möglichkeit, die (Rückwärts-)Korrektur durchzuführen, und eine einheitliche Methode zur vertikalen Analyse vor.

```
method correct() {

 problematicNotes = List of yet decided Notes that cannot be played with
 the hand they have been assigned because the interval to the lowest or
 highest Note of this hand is too big, and that have not been marked as
 `yet corrected´;

 if this list is empty break up correction;

 double minimumStarttime = the minimum starttime of all Notes in
 problematicNotes;

 change hands of all problematicNotes;

 while ( currentTime > minimumStarttime ) { //going back to proove changes

  set all Notes in currentNoteSet with starttime >= currentTime unknown;
  notepool.backward();

 }

 if( left or right hand still not playable ) {

  set all Notes in currentNoteSet with starttime >= currentTime unknown;
  notepool.backward();

 }

 mark Notes in problematicNotes `yet corrected´;

}
```

Pseudocode-Beispiel II-10: Algorithmus zur Rückwärtskorrektur (für alle Algorithmen)

Der Algorithmus tauscht die Zuordnung der Noten, die nicht spielbar sind und geht dann so weit in der Zeit zurück, wie die geänderten Zuordnungen reichen und setzt dabei alle Noten als unbekannt, die nicht noch weiter zurückreichen. Die unbekannt gesetzten Noten müssen dann noch einmal – unter Berücksichtigung der Korrektur – mit der nachfolgend angeführten Methode für vertikale Algorithmen – analysiert werden.

```
method verticalSplittingAlgorithm( NoteSet currentNoteSet ) {
 if( currentNoteSet.size() < 2 ) {
  analyze with singlenotes algorithm; break;
 }
 analyze currentNoteSet splittingAlgorithmI;
 if( currentNoteSet.hasUnknownNotes() ) {
  analyze currentNoteSet with primary vertical splitting algorithm;
 }
 if( left or right Notes not playable ) {
  correct();
 }
}
```

Pseudocode-Beispiel II-11: Allgemeiner vertikaler Algorithmus

Diese Methode wird für jedes *NoteSet* im *NotePool* aufgerufen.

In den folgenden zwei Kapiteln werde ich die horizontalen Algorithmen vorstellen. Der erste Abschnitt zeigt Algorithmen, deren Entscheidung über die Zuordnung vom *currentNoteSet* und vom vorhergehenden *recentNoteSet* abhängt, danach werden Methoden präsentiert, durch welche ein größerer Zusammenhang analysiert wird.

4 Horizontale Algorithmen mit Berücksichtigung des direkten Vorgängers

Wie im vorangegangenen Abschnitt erklärt, liegt die wesentliche Schwäche der vertikalen Algorithmen darin, dass oftmals Sprünge vorgeschrieben werden, wo diese vermeidbar wären. Da dieses Problem mit den oben angeführten Methoden nicht zu lösen ist, muss diese Technik so erweitert werden, dass sie zu einer Lösung des Problems führt; prinzipiell kann dies durch eine minimale Erweiterung geschehen.

In diesem Kapitel werde ich Algorithmen anführen, die bei der sequentiellen Analyse eines Musikstücks ihre Entscheidung über die Zuordnung zu einer der beiden Hände neben dem *currentNoteSet* auch auf das *NoteSet* davor (das *recentNoteSet*) stützen. Da hier bereits ein (kleiner) Bezug zum horizontalen Zusammenhang des Notenbildes hergestellt wird, zähle ich diese Algorithmen bereits zu den oben definierten *horizontalen Algorithmen*.

Auch wenn der Satz an Noten, der für die Entscheidung einer Zuordnung herangezogen wird, kaum größer ist als bei den vertikalen Algorithmen, bewirkt diese Erweiterung eine wesentliche Verbesserung der Resultate.

Weiteren Analysen dieses Prinzips voran möchte ich einen Algorithmus zur Vermeidung großer Sprünge (wo diese nicht notwendig sind) präsentieren; danach werde ich weitere Verfahren aufzeigen, die ihre Entscheidung (ausschließlich) vom *currentNoteSet* und vom *recentNoteSet* abhängig machen.

4.1 Vermeidung großer Sprünge – Algorithmus VI

Auch wenn große Sprünge prinzipiell durch horizontale Algorithmen mit Berücksichtigung des direkten Vorgängers vermieden werden können, ist die Ausführung desselben nicht trivial. Die primäre Fragestellung in diesem Zusammenhang ist wohl: Was ist ein großer Sprung?
Diese Frage kann natürlich nicht eindeutig beantwortet werden, daher genügt es auch nicht, alle (horizontalen) Intervalle über einem festen Wert als großen Sprung zu werten. Das unten angeführte Resultat entspricht einem Herantasten an eine Lösung des Problems auf empirische Weise.

Der Algorithmus schreibt vor, dass ein Akkord, wenn er aufgrund seiner Spannweite auch von einer Hand gespielt werden könnte, nicht zwangsläufig geteilt wird, wenn in einer Hand der Sprung einen gewissen Grenzwert (abhängig vom zuvor gespielten Akkord) überschreitet. Tritt der Sprung nur in einer Hand auf, wird der gesamte Akkord von der anderen Hand gespielt, tritt in beiden Händen ein Sprung auf, wird die Entscheidung einem Sekundäralgorithmus überlassen.
Zur Berechnung des Grenzintervalls wird die Differenz zwischen dem höchsten Ton der linken Hand des *recentNoteSet* und dem tiefsten Ton des *currentNoteSet*, sowie die Differenz des tiefsten Tons der rechten Hand des *recentNoteSet* und dem

höchsten Ton des *currentNoteSet*, mit einem vorgegebenen Grenzwert verglichen (etwa eine Oktave, also 12 Halbtöne). Hier wird davon ausgegangen, dass kein Übergreifen stattfindet, durch Umordnung könnte allerdings auch dieser Fall berücksichtigt werden.

Eine derartige Entscheidung ist nur dann möglich, wenn im *recentNoteSet* beiden Händen zumindest jeweils ein Ton zugeordnet wurde. Falls zuvor nur eine Hand gespielt hat, wird nur der höchste oder der tiefste Ton in eine Hand gesetzt und der Rest von einem Sekundäralgorithmus entschieden. Auch in diesem Fall wird die Hand nur dann beibehalten, wenn ein gewisser Grenzwert nicht überschritten wird.

Nachfolgend nun der Code zu dem hier erklärten Algorithmus:

```
method splittingAlgorithmVI(NoteSet recentNoteSet,NoteSet currentNoteSet) {

 const MAX_INTERVAL_SAME_HAND = 12;

 if( recentNoteSet.getLeftNotes().size() == 0 ) {

  if( (currentNoteSet.getHighestNote().getPitch() >
  recentNoteSet.getHighestNote().getPitch()) ||
  (recentNoteSet.getHighestNote().getPitch() -
  currentNoteSet.getHighestNote().getPitch() < MAX_INTERVAL_SAME_HAND) ) {

   currentNoteSet.getHighestNote().setRight(TRUE);

  } else {

   currentNoteSet.getLowestNote().setLeft(TRUE);

  }

 } else if( recentNoteSet.getRightNotes().size() == 0 ) {

  do vice-versa;

 } else {

  if( (abs(currentNoteSet.getLowestNote().getPitch() -
  recentNoteSet.getLeftNotes().getHighestNote().getPitch()) <=
  MAX_INTERVAL_SAME_HAND) &&
  (abs(currentNoteSet.getHighestNote().getPitch() -
```

```
  recentNoteSet.getRightNotes().getLowestNote().getPitch()) >
  MAX_INTERVAL_SAME_HAND ) )  {

   currentNoteSet.setLeft(TRUE);

  } else if( inverse )  {

   currentNoteSet.setRight(TRUE);

  }

 }

}
```

Pseudocode-Beispiel II-12: Algorithmus VI

Dieser Algorithmus ist als Ergänzung anderer und ausschließlich zur Vermeidung großer Sprünge gedacht. Prinzipiell kann er mit jeder Analyse eines Musikstücks durch einen der präsentierten vertikalen Algorithmen zusammenarbeiten und die Analyse dadurch aufwerten.
Im Folgenden möchte ich einen weiteren Algorithmus für das Analysieren von Akkorden (*NoteSet* mit mehr als einem Ton) anführen, der diese Hilfsmethode zur Vermeidung von Sprüngen hinzuzieht.

4.2 Beibehaltung der Anzahl der Stimmen pro Hand – Algorithmus VII

Der hier vorgestellte Algorithmus stellt – im Sinne der „Horizontalisierung“ eines vertikalen Algorithmus‘ – eine Erweiterung der Methode der gleichmäßigen Aufteilung (Algorithmus III) dar. Hier wird bestrebt, nicht primär eine möglichst gleichmäßige Aufteilung der Töne auf die Hände zu jedem Zeitpunkt zu erwirken (vertikal), sondern – davon ausgehend – die Anzahl der Stimmen in einer Hand in der zeitlichen Abfolge möglichst gleich zu halten (horizontal). Dazu wird wie folgt vorgegangen:

Jedes *currentNoteSet* wird mit Algorithmus III entschieden, falls zuvor keine Noten gespielt wurden (Anfang des Stückes oder Pause). Andernfalls werden – sofern die Intervalle nicht zu groß werden – gleich viele Noten wie im *recentNoteSet* (oder weniger, falls weniger vorhanden) jeder Hand zugeordnet. Enthält das *currentNoteSet* mehr Noten als das *recentNoteSet* werden diese anschließend durch

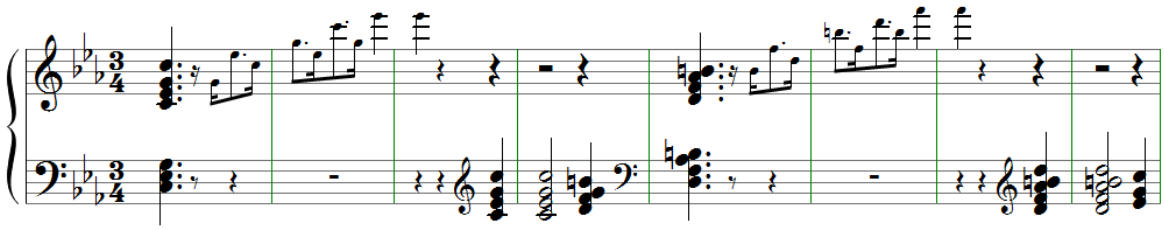
einen (vertikalen) Sekundäralgorithmus entschieden.
Die so beschriebene Zuordnung findet allerdings nur statt, wenn die Notwendigkeit besteht, das *currentNoteSet* auf beide Hände aufzuteilen, andernfalls wird der Algorithmus zur Vermeidung von Sprüngen (VI) herangezogen.

```
method splittingAlgorithmVII(NoteSet recentNoteSet,NoteSet currentNoteSet){

 Sort Notes by Pitch;

 int leftNotes = recentNoteSet.getLeftNotes().size();
 int rightNotes = recentNoteSet.getRightNotes().size();

 for( max recentNoteSet.size() unknown Notes in currentNoteSet, outer to
 innter) {

  if( Note is from lower pitch’ed Notes ) {

   if(currentNoteSet.getLeftNotes().size() < leftNotes) {

    note.setLeft(TRUE);

   }
   else {

    note.setRight(TRUE);

   }

  }

  else (Note is from upper pitch’ed) {

   do vice-versa;

  }

 }

 call secondary splitting algorithm(currentNoteSet);

}
```

Pseudocode-Beispiel II-13: Algorithmus VII

Der hier beschriebene Algorithmus ist im Zusammenhang mit dem Algorithmus zur Vermeidung weiter Sprünge bereits auf viele Stücke sehr gut anwendbar. Sowohl die Werke, bei denen der vertikale Algorithmus III zur gleichmäßigen Aufteilung der

Stimmen verwendet werden kann, als auch Werke, wo dieser wenig optimale Ergebnisse liefert, können sehr gut analysiert werden. Als Beispiele hierfür sind etwa viele Variationen der Mozart-Sonate oder die Bach-Fuge zu nennen.
Dennoch ist auch dieser Algorithmus nicht als perfekt anzusehen, und weist bei der Analyse einiger Werke kleinere Mängel auf. So ist das Ergebnis der Analyse durch den Algorithmus zur Vermeidung von Sprüngen nicht immer optimal, wie das nachfolgende Beispiel aus Beethovens 5. Klaviersonate zeigt.



Notenbeispiel II-7: Beethoven, Klaviersonate Nr.5, Takte 1-8 nach der Analyse mit Algorithmus VII
Die Akkorde in Takt 3 ff und Takt 7 f werden nicht aufgeteilt, wie es die Originalpartitur vorsehen würde. Dies ist prinzipiell kein technisches Problem, aber das Umgreifen von Takt 4 auf Takt 5 in der linken Hand könnte durch eine Teilung der Akkorde vermieden werden.

4.3 Analyse einzelner Noten – Algorithmus VIII

Wie bereits erwähnt, gibt es für die Analyse einzelner Noten – resp. *NoteSets* mit nur einem Ton – nicht viele sinnvolle Varianten. Die Entscheidung nur anhand der Tonhöhe – in Bezug auf einen festen Wert – durchzuführen, funktioniert relativ oft, kann aber auch zu schlecht lesbaren Ergebnissen führen, Linien werden mitunter unnötig geteilt.
Eine bessere Alternative stellt der hier beschriebene (und in den obigen Beispielen bereits verwendete) Algorithmus dar, der im Allgemeinen gute Ergebnisse liefert.

```
method splittingAlgorithmVIII() {

 if( recentNoteSet.isEmpty() ) {

  decide by pitch;

 } else if( recentNoteSet.size() == 1 ) {

  if( difference of the pitches in recent and currentNoteSet <
  MAX_INTERVAL) {

   assign to same hand;

  } else {
```

```
    assign to other hand;

   }

  } else {

   if( recentNoteSet has only left or only right Notes ) {

    if the pitch of the currentNote is in
    [recentNoteSet.getLowestNote().getPitch()-MAX_INTERVAL;
    recentNoteSet.getHighestNote().getPitch()+MAX_INTERVAL] assign
    currentNote to the same hand as the recentNoteSet;

   }
   else {

    if( pitch of currentNote > recentNoteSet.getHighestNote.getPitch() ) {

     assign to same hand as recentNoteSet.getHighestNote();

    }
    analog for recentNoteSet.getLowestNote();

    else if( pitch of currentNote in
    [recentNoteSet.getLeftNotes().getLowestNote().getPitch();
    recentNoteSet.getLeftNotes().getHighestNote().getPitch()] {

     assign left;

    }
    vice-versa for recentNoteSet.getRightNotes();

    else {

     decide by minimal difference of currentPitch and the lowest/highest
     Note of the recentNoteSet;

    }

   }

  }

}
```

Pseudocode-Beispiel II-14: Algorithmus VIII

Die Entscheidung über die Zuordnung der (einzigen) Note im *currentNoteSet* wird abhängig von der Aufteilung des *recentNoteSet* getroffen. Wenn es kein *recentNoteSet* gibt (etwa bei Pausen oder am Beginn eines Stückes) muss die Entscheidung anhand der Tonhöhe alleine vorgenommen werden, etwa das c^1 (Pitch 60) kann eine Grenze darstellen. Weiters wird unterschieden, ob das *recentNoteSet*

ebenfalls nur aus einer Note oder aus mehreren besteht.
Im ersten Fall wird einfach der Unterschied der Tonhöhen der beiden Töne im *recentNoteSet* und im *currentNoteSet* mit einem vorgegebenen Maximalwert (z.B. Dezime) verglichen. Ist der Unterschied „zu groß“, wird der Ton des *currentNoteSet* der Hand zugewiesen, welche nicht den Ton des *recentNoteSet* zu spielen hat, andernfalls der gleichen Hand.
Besteht das *recentNoteSet* aus mehreren Tönen, wird für die Zuordnung des Tons des *currentNoteSet* unterschieden, ob die Tonhöhe außerhalb der Lagen beider Hände, innerhalb der Lage einer Hand, oder weder noch ist. Durch die so getroffene Zuordnung (s. Pseudocode-Beispiel II-14) sollen auch größere Sprünge einer Hand vermieden werden.

5 Horizontale Algorithmen mit Berücksichtigung größerer Zusammenhänge

In diesem Abschnitt werde ich zwei Algorithmen vorstellen, die jeweils größere Zusammenhänge eines Stückes für die Analyse heranziehen.

Diese Algorithmen, die versuchen, jeweils einen der in Teil I erwähnten Spezialfälle mit zu berücksichtigen, sind bedeutend komplexer als die bisher beschriebenen.

5.1 Aufteilung nach Stimmen – Algorithmus IX

Die oben beschriebenen Algorithmen III und VII erzielen bei streng polyphonen Werken oft implizit eine gute Lösung, indem die Zahl der Stimmen in einer Hand tendenziell gleich bleibt. Ich möchte hier einen Algorithmus beschreiben, welcher einem anderen Zugang folgt und versucht, eine explizite Aufteilung der Stimmen eines vielstimmigen Werkes vorzunehmen, um horizontale Linien leichter spielbar (bzw. im Notenbild lesbar) zu machen. Das Prinzip der Aufteilung ist nicht schwierig:

Das jeweils zu analysierende Werk wird in Abschnitte geteilt, innerhalb derer an jeder Stelle gleich viele Noten zu spielen sind (die *NoteSets* innerhalb eines Abschnittes sind also alle gleich groß). Abschnittweise werden dann alle niedrigsten Töne als unterste Stimme, alle zweitniedrigsten Töne als zweite Stimme usw. betrachtet. Der Algorithmus versucht nun, jede Stimme möglichst ganzheitlich einer Hand zuzuordnen, wobei folgendermaßen vorgegangen wird:
Besteht ein Abschnitt zumindest aus zwei Stimmen, wird die unterste der linken, die

oberste der rechten Hand zugeordnet, für alle anderen Stimmen oder für einen einstimmigen Abschnitt wird der unten beschriebene Algorithmus verwendet.

Zur einfacheren Handhabung werden die Noten nicht nur in *NoteSets* sondern auch abschnittsweise in sogenannte *NoteSequences*, also in (horizontale) Sequenzen von aufeinanderfolgenden Noten eingeteilt. Die dafür benötigte Objektstruktur ist nicht aufwändig zu generieren; wichtig ist, dass eine Sequenz auch in mehrere kleinere Sequenzen unterteilt werden können muss.

```
method analyze() {

 List of NoteSequences currentSequences;
 int voices;

 while( notepool.hastNextNoteSet() ) {

  save the currentSequences to another List recentSequences (if
  currentSequences not empty);

   voices = currentNoteSet.size();

   add $voices NoteSequences to currentSequences;

   while( currentNoteSet.size() == voices ) {

    sort currentNoteSet by pitch;

    for all i add voice $i to currentSequences[i] (if not yet added);

    goForward();

   }

   splittingAlgorithmIX( currentSequences, recentSequences );

  }

 }

}
```

Pseudocode-Beispiel II-15: Algorithmus IX: Methode "analyze"

```
method splittingAlgorithmIX( currentSequences, recentSequences) {

 if( at least 2 Sequences in currentSequences ) {
     set lowest sequence to left, highest to right;
 }
```

```
 else {
      decideLeftOrRight( currentSequences[0] );
 }

 for( all other sequences in currentSequences ) {

  decideLeftOrRight( currentSequences[index] );

 }

}
```

Pseudocode-Beispiel II-16: Algorithmus IX: Methode "splittingAlgorithmIX"

Es wird versucht, eine Stimme möglichst ganzheitlich einer Hand zuzuordnen; dazu muss kontrolliert werden, ob die Intervalle zu den (bereits zugeordneten) Außenstimmen nicht zu groß werden. Gelingt dies für alle Töne einer Stimme für genau eine Hand, ist die Zuordnung klar, falls beide Hände möglich sind wird unter den Stimmen des vorherigen Abschnittes diejenige gesucht, deren durchschnittliche Tonhöhe (als arithmetisches Mittel) sich am wenigsten von der durchschnittlichen Tonhöhe der gerade zu bearbeitenden Sequenz unterscheidet[15]; die aktuelle Stimme wird dann der gleichen Hand zugeordnet wie das Ende der so gefundenen Stimme des letzten Abschnittes. Andernfalls muss die Sequenz in kleinere Teile geteilt werden, wobei versucht wird, die Stimme möglichst lange in einer Hand zu spielen. Der verbliebene Teil, ab dem die Hand gewechselt werden muss, wird dann (rekursiv) ebenso analysiert wie hier beschrieben.

```
method decideLeftOrRight( NoteSequence ) {

 NoteSequence bestSequence = sequence, where average pitch is most similar
 to average pitch of this sequence;

 check how many NoteSets in this sequence can be played left;
 check how many NoteSets in this sequence can be played right;

 if( all can be played left and right ) {
```

[15] Die durchschnittliche Tonhöhe zu nehmen ist nur dann sinnvoll, wenn die Tonhöhen einer Stimme innerhalb eines Abschnittes nicht zu stark variieren. Da die Abschnitte aber üblicherweise nicht sehr lange sind, funktioniert diese Methode im Allgemeinen gut.

```
  search the sequence in recentSequences where the average pitch is most
  similar to the average pitch of this sequence, and assign this sequence
  to the same hand as the ending of the here found recent sequence;

 }

 else if( all can be played left ) {
      set complete sequence to left;
 }

 else if( all can be played right ) {
      set complete sequence to right;
 }

 else  {

  if( more NoteSets can be played left than right (or vice-versa) ) {
      set that part to left (or vice-versa);
  }

  split the sequence at the last decided NoteSet into two parts;

  decide the second part like this full sequence recursively;

 }

}
```

Pseudocode-Beispiel II-17: Algorithmus IX: Methode "decideLeftOrRight"

Obwohl ich hier viele Details des Algorithmus‘ nicht angegeben habe, ist eindeutig ersichtlich, dass der Aufwand der Analyse wesentlich höher ist als etwa bei den oben beschriebenen vertikalen Algorithmen. Es gilt auch zu bemerken, dass hiermit wirklich vor allem polyphone Werke analysiert werden sollten. Bei Werken, in denen beispielsweise abwechselnd akkordisches und monophones Spiel gefordert wird, ist der Analyseaufwand unnötig hoch und die Ergebnisse weniger zufriedenstellend; in vielstimmigen Werken wie etwa Fugen übertreffen die Ergebnisse aber die der anderen Methoden bei Weitem.

5.2 Beibehaltung der Lage und Übergreifen – Algorithmus X

Das Ziel des im Folgenden beschriebenen Algorithmus‘ ist eine explizite technische Vereinfachung des zuzuordnenden Stückes. Die Analyse wird dabei ganzheitlich (ohne ausdrückliche Unterscheidung von Einzelnoten und Akkorden) unter Berücksichtigung einer längeren „Vorgeschichte“ durchgeführt. Die (vermeidbaren) technischen Schwierigkeiten sollen durch das Bemühen, die Lage der Hände auf der

Klaviatur beizubehalten, reduziert werden. Als weitere Besonderheit soll dadurch Übergreifen auch erlaubt werden können (s. dazu I 6.2).

Nachfolgend findet sich eine Übersicht über die Struktur des Algorithmus‘, die einzelnen Teile werden danach genauer erläutert.

Die Struktur enthält, neben den Konstanten für die erlaubten Intervalle, Variablen zur Speicherung der Lage der Hände (für jede Hand ein tiefster und ein höchster Ton), sowie die folgenden Funktionen (bzw. Methoden):

- `analyze:` die Hauptmethode, welche das Werk durchläuft und die nächste Methode für jedes neue *NoteSet* aufruft,
- `splitting_algorithm`: die Methode zur Zuordnung der Noten eines *NoteSets*.
- `updatePositions`: die Methode zur Berechnung der aktuellen Lage der Hände.
- `checkPitchInXXXPosition`: Funktionen, die überprüfen ob ein Ton in der jeweiligen Lage einer Hand liegt und das Maximalintervall zum höchsten bzw. tiefsten Ton nicht überschritten wird.
- `changeHands`: Methode, die alle bereits zugeordneten Töne des aktuellen *NoteSets* der jeweils anderen Hand zuweist.

Folgendes Code-Beispiel zeigt die wesentlichen Konstanten, Variablen und Methoden, die für diesen Algorithmus definiert werden müssen, in einer Klassenstruktur.

```
Class SplittingAlgorithmX {

 const int MAX_INTERVAL = 12;
 const int MAX_INTERVAL_POSITION = 12;
 const int ABSOLUTE_MAX_INTERVAL = 16;

 private int position_lowestPitch_left = -1;
 private int position_lowestPitch_right = -1;
 private int position_highestPitch_left = -1;
 private int position_highestPitch_right = -1;
 private boolean overlap = FALSE;

 method analyze();
```

```
 method splitting_algorithm( NoteSet currentNoteSet );

 method updatePositions( NoteSet currentNoteSet );

 function boolean checkPitchInLeftPosition( int pitch, int lowRefPitch, int
 highRefPitch );

 function boolean checkPitchInRightPosition( int pitch, int lowRefPitch,
 int highRefPitch );

 method changeHands( NoteSet currentNoteSet );

}
```

Pseudocode-Beispiel II-18: Algorithmus X: Klassenstruktur

Die Methode `analyze` durchläuft alle *NoteSets* des Werks sequentiell und ruft die Teilungsalgorithmen folgendermaßen auf:
Zu Beginn wird so lange ausschließlich ein Sekundäralgorithmus verwendet, bis beiden Händen zumindest jeweils eine Note zugeordnet wurde, damit anschließend die Lagen mit der unten beschriebenen Methode `updatePositions` bestimmt werden können.
Danach wird das nach Tonhöhe sortierte *currentNoteSet* jeweils zunächst durch den – für das Übergreifen etwas adaptierten – Algorithmus I analysiert, bevor es `splittingAlgorithmX` übergeben wird. Gibt es weitere unzugeordnete Noten, wird wieder ein Sekundäralgorithmus herangezogen.
Bei Bedarf erfolgt nun das Aufrufen der `correct`-Methode, zum Schluss werden die Lagen aktualisiert.

Die neuen Lagen der Hände werden unter Berücksichtigung der bisherigen Lage und der in der jeweiligen Hand zuletzt zugeordneten Töne berechnet. Befanden sich die zuletzt zugeordneten Noten einer Hand innerhalb der Lage, so wird die Lage gegebenenfalls verengt, sodass ein bestimmtes Intervall vom tiefsten bzw. höchsten der zugeordneten Töne nicht überschritten werden darf; musste die Lage geändert werden, gelten nunmehr diese neuen Grenzen.

```
method splittingAlgorithmX( NoteSet currentNoteSet ) {

  boolean canHoldPosition_left = TRUE;
  boolean canHoldPosition_right = TRUE;
```

```
 if( wide chords algorithms has yet decided some notes ) {
  if(any of the left Notes returns FALSE for checkPitchInLeftPosition()) {
      canHoldPosition_left = FALSE;
  }
  do for right Notes vice-versa;

  if( canHoldPosition_left && canHoldPosition_right ) {
      check each unknown Note left and right and if result is distinct set
  to left or right;
  } else if( canHoldPosition_left && !canHoldPosition_right ) {
      try to set the holdable notes to left and the other ones to right;
  } else if( !canHoldPosition_left && canHoldPosition_right ) {
      do vice-versa;
  } else if( overlapping allowed ) {
      try overlapping;
  }

 }

 else {
      try to assign notes to one hand;
 }

}
```

Pseudocode-Beispiel II-19: Algorithmus X: Methode "splittingAlgorithmX"

Die Hauptmethode `splittingAlgorithmX` unterscheidet zunächst zwei Arten von *currentNoteSets*: solche, die durch Algorithmus I bereits zugeordnete Noten haben (und daher nicht mit einer Hand gespielt werden können) und solche, deren Maximalintervall auch mit einer Hand gespielt werden könnte.

Im ersten Fall wird für alle bereits zugeordneten Noten des *NoteSets* ermittelt, ob die Lage der Hände beibehalten wird.
Können beide Lagen beibehalten werden, wird jede unbekannte Note des *currentNoteSet* überprüft, und wenn das Ergebnis eindeutig ist, der entsprechenden Hand zugewiesen. Kann hingegen die Lage nur einer Hand beibehalten werden, werden alle Töne, die dieser Lage entsprechen auch dieser Hand zugeordnet, alle weiteren der anderen Hand, sofern das Maximalintervall nicht überschritten wird.
Können beide Lagen nicht gehalten werden, wird kontrolliert, ob die Lage einer Hand beibehalten werden könnte, wenn die andere übergreifen würde (falls Übergreifen erlaubt wird).

Nachfolgend findet sich noch ein Code-Beispiel für den letztgenannten Fall.

```
boolean wasOverlapping = overlap;
canHoldPosition_left = TRUE;
canHoldPosition_right = TRUE;
overlap = TRUE;

for(all left Notes) {

  if(!checkPitchInRightPosition( currentPitch, lowPitch, highPitch)) {
   canHoldPosition_left = FALSE;
  }

}
do for right notes vice-versa;

if( canHoldPosition_left || canHoldPosition_right ) {

  changeHands(currentNoteSet);
  for( all unknown Notes in currentNoteSet) {

    if(absolute(currentPitch - lowestPitch < absoulte(currentPitch -
    highestNote) ) {
        currentNoteSet.get(i).setRight(TRUE);
    }

    else {
        currentNoteSet.get(i).setLeft(TRUE);
    }

  }

} else {

  overlap = FALSE;
  if(wasOverlapping) {
   changeHands(currentNoteSet);
  }

}
```

Pseudocode-Beispiel II-20: Algorithmus X: Auszug aus "splittingAlgorithmX" - Teil "try overlapping"

Der Erfolg der Analyse hängt auch hier nicht zuletzt von der Wahl des Sekundäralgorithmus ab, wobei dieser gegebenenfalls für das Übergreifen angepasst werden muss.
Bei den meisten der Teststücke funktioniert die Analyse hiermit sehr gut, ungewollte Sprünge können dennoch auftreten, wenn die Definition der Lage schlecht gesetzt

ist. Ein Beispiel hierfür ist *Bydlo* aus den *Bildern einer Ausstellung* (s. unten), wo der Lagenalgorithmus nicht wirkt, weil die Begleitakkorde einen ständigen Lagenwechsel erfordern. Die Entscheidung wird hier an vielen Stellen dem Sekundäralgorithmus überlassen (im Beispiel: Algorithmus III), wodurch teilweise Sprünge nicht vermieden werden können. Für Beispiele des Erfolgs der Methode sei auf das nachfolgende Kapitel verwiesen.

III. Testen der Algorithmen

In diesem Abschnitt werde ich einige Beispiele für Ergebnisse der Analyse mit den oben beschriebenen Algorithmen präsentieren. Zur zuverlässigen Untermauerung von Aussagen über die Qualität von Rechenvorschriften dieser Art müsste eine sehr große Zahl (zumindest mehrere hundert) unterschiedlichster Stücke getestet werden. An dieser Stelle sei aber noch einmal darauf verwiesen, dass die hier vorgestellten Algorithmen als Ansätze zur Lösung der Problemstellung zu verstehen sind, und nicht als fertige Programmabläufe, die eine sichere Lösung bringen. Daher ist es auch nicht zweckmäßig, die Algorithmen auf diese Art zu testen.
In diesem Sinn habe ich mich – im Rahmen dieser Arbeit – darauf beschränkt, eine kleine Zahl an Werken der klassischen Klavierliteratur mit unterschiedlicher Entstehungszeit, unterschiedlicher Charakteristik und verschiedener Schwierigkeit zu untersuchen und zu präsentieren.
Trotz der geringeren Zahl an Testobjekten können qualitative Aussagen über die vorgestellten Algorithmen getroffen werden. Anhand der folgenden Beispiele lassen sich Stärken und Schwachstellen einzelner Algorithmen aufzeigen, und man kann erkennen, dass die Wahl einer Rechenvorschrift von gewissen Faktoren der zu analysierenden Stücke abhängig gemacht werden kann.

Nachfolgend werde ich die Werke vorstellen, von denen ich danach Auszüge der Testergebnisse präsentieren werde. Dazwischen wird die Konfiguration der Algorithmen für den Testvorgang beschrieben.

1 Teststücke

Um Vergleiche mit den Originalpartituren erarbeiten zu können, habe ich „idealisierte Aufnahmen“ herangezogen, welche in den Notenwerten exakt den Originalen entsprechen. Die Quellen der Partituren sind im Literatur- und Quellenverzeichnis angeführt, die Entstehungsdaten und Lebzeiten der Komponisten sind ebenfalls diesen Partituren und ihren Begleittexten entnommen.

- Johann Krieger (1652-1735): Fughetta
  Die Fughetta über den Choral „Christum wir sollen loben schon“ beginnt wie

eine dreistimmige Fuge, der zweite Teil des 23-taktiken Werkes ist jedoch freier gestaltet, am Schluss kommt eine vierte Stimme hinzu.

- Henry Purcell (1659-1695): Almand
  Der erste Tanzsatz der Suite in d-Moll Z668 besteht aus einer reich verzierten Melodiestimme mit wechselnder Anzahl an Begleitstimmen.
- Georg Philipp Telemann (1681-1767): Fantasia in d-Moll TWV33:2
  Die Fantasia besteht aus zwei Teilen, der erste Teil ist streng zweistimmig, mit einem Unisono-Thema gesetzt, der zweite Teil ist ein eher freies Adagio, danach wird der erste Teil wiederholt.
- Johann Sebastian Bach (1685-1750): Fuge in cis-Moll BWV849
  Diese 5. Fuge aus dem ersten Band des bekannten Wohltemperierten Klaviers von J. S. Bach ist eine der wenigen 5-stimmigen Fugen des Werkzyklus und mit einer relativ großen Zahl an Themen und ihrer Länge relativ breit angelegt.
- Johann Sebastian Bach (1685-1750): Sinfonia in g-Moll BWV797
  Die zwischen den Jahren 1720 und 1723 niedergeschriebene Sinfonia in g-Moll ist – ebenso wie die anderen 14 Sinfonias – 3-stimmig gehalten, die Themen werden aber nicht durchgängig nach dem strengen Fugenschema gebracht. Die Mittelstimme wechselt an vielen Stellen die Hand, was aufgrund der Entfernung zu den äußeren Stimmen auch notwendig ist.
- Joseph Haydn (1732-1809): Klaviersonate in e-Moll 1. Satz
  Der 1. Satz der 1778 erschienenen Sonate in e-Moll zeichnet sich im Hauptthema durch abwechselndes (überlappendes), teilweise akkordisches Spiel der beiden Hände aus, im Seitenthema übernimmt eine Hand jeweils die Begleitung in Sechzehntel, während die andere Hand eine Melodiestimme in Achteln spielt.
- Wolfgang Amadeus Mozart (1756-1791): Klaviersonate in A-Dur KV300i 1. Satz
  Die „angeblich Sommer 1778“ entstandene Sonate zählt zu den bekanntesten Klaviersonaten Mozarts. Nicht nur der dritte Satz, das „Rondo alla turca“, sondern auch der hier untersuchte 1. Satz der Sonate finden häufig Eingang in Werbe- oder Kennmelodien. Dieser Satz ist als Thema mit Variationen aufgebaut, welche zahlreiche damals übliche technische Fertigkeiten des

Spielers abverlangen; in der 4. Variation beispielsweise wird Übergreifen der Hände gefordert.

- Ludwig van Beethoven (1770-1827): Klaviersonate Nr.5 in c-Moll op.10 Nr.1 1.Satz

  Dieser Sonatensatz der 1796-98 entstandenen 5. Sonate von Beethoven enthält viele unterschiedliche Stellen, wie Solo-Linien, enge Akkorde, Melodie mit akkordischer Begleitung und Melodie mit einstimmiger Begleitung aus zerlegten Akkorden.
- Franz Schubert (1797-1828): Impromptu in c-Moll

  Schuberts Impromptu Nr.1 in c-Moll beginnt, nach einem einleitenden Akkord, mit einer Melodiestimme ohne Begleitung, aus deren Variation dann die weiteren Teile des Werks bestehen. Diese fallen unterschiedlichst aus – die Melodie wird mit Akkorden begleitet wiederholt, über verschiedenen Akkordzerlegungen variiert und dazwischen in mehrstimmige Gewebe eingeflochten.
- Felix Mendelssohn Bartholdy (1809-1847): Lied ohne Worte op.19b Nr.1

  Dieses Werk in E-Dur hat eine sangbare Melodiestimme und eine beständige zweigeteilte Begleitung, welche aus einer Mittelstimme in gebrochenen Akkorden und einer Unterstimme in längeren Noten besteht.
- Frédéric Chopin (1810-1849): Nocturne in g-Moll op.37 Nr.1

  Wie auch viele andere der 21 Nocturnes für Klavier von Chopin, besteht dieses Werk im 1. Teil aus einer reich verzierten Melodiestimme, die von überlappenden Akkorden unterschiedlicher Länge begleitet wird. Der darauf folgende Mittelteil ist gänzlich akkordisch gesetzt, danach wird der 1. Teil wiederholt.
- Robert Schumann (1810-1856): Fuge in A-Dur

  Die Fuge ist dem Teil „Für Erwachsenere“ des „Album für die Jugend“ entnommen und ist dreistimmig gesetzt.
- Johannes Brahms (1833-1897): Ballade op.10. Nr.1 in d-Moll

  Die Ballade von Brahms ist dreiteilig aufgebaut, die äußeren Teile enthalten vor allem Akkorde in weiter Lage, der Mittelteil stellt abwechselnd in beiden Händen Triolen gegen gerade Notenwerte und sieht Übergreifen der rechten Hand vor.

- Modest Musorgsky (1839-1881): Bilder einer Ausstellung (Ausw.)
  Aus dem – vor allem in der Orchesterfassung von M. Ravel – bekannten Werkzyklus *Bilder einer Ausstellung* des russischen Komponisten Musorgsky von 1874 werde ich die Stücke *Bydlo* und *Das große Tor von Kiew* betrachten.
- Edvard Grieg (1843-1907): Klaviersonate in e-Moll op.7 1. Satz
  Die Klaviersonate von Grieg ist ein virtuoses viersätziges Werk, der 1. Satz enthält Passagen unterschiedlichster Art, wie Melodiespiel mit ein- bis zweistimmiger Begleitung und akkordische Stellen in enger und weiter Lage.
- Erik Satie (1866-1925): Gymnopédie Nr.1
  Die Gymnopédie, eines der bekanntesten Werke von Satie, ist ein sehr ruhiges Werk mit einer zarten Melodiestimme und akkordischer Begleitung.
- Béla Bartók (1881-1945): Zehn leichte Klavierstücke (Ausw.)
  Aus dem Werkzyklus betrachte ich die Werke *Bauernlied* und *Bärentanz*, welche sich stark voneinander unterscheiden. Das Bauernlied ist ein einfaches Werk mit einer durchgehenden „Unisono"-Stimmbewegung im Oktavabstand.
  Der Bärentanz ist eher flott zu spielen und enthält neben einer aus Repetitionen des gleichen Tons bestehenden Begleitung Akkorde, welche auch die Melodiestimme beinhalten.
- Dmitri Schostakowitsch (1906-1975): Fantastischer Tanz op.5 Nr.2
  Dieses stilisierte Tanzstück im ¾-Takt enthält neben einer akkordischen Begleitung eine Melodiestimme, welche an vielen Stellen in Oktaven gesetzt ist. Die Lage der rechten Hand reicht von der Mitte der Klaviatur bis zu ihrem oberen äußersten Ende.

2 Algorithmen

Einige der Algorithmen greifen – wie beschrieben – auf Sekundäralgorithmen zurück, oder haben andere Parameter, von deren Einstellung das Ergebnis ebenfalls abhängen kann. Hier findet sich noch einmal eine Übersicht der für das Testen verwendeten Algorithmen und dieser Einstellungen.

- Algorithmus I dient der Analyse weiter Akkorde. Alle Töne, die aufgrund ihrer Entfernung zueinander nicht in einer Hand gespielt werden können, werden auf die beiden Hände aufgeteilt. Als Intervall für die Zuordnung zur gleichen

Hand ist für das Testen eine Oktave (12 Halbtonschritte) vom höchsten bzw. tiefsten Ton festgelegt.

- Algorithmus II weist alle Töne die (in Halbtonschritten) näher zum höchsten Ton als zum tiefsten sind, rechts zu, alle anderen links.
- Algorithmus III weist der linken Hand stets gleich viele Töne zu wie der rechten, falls dies möglich ist. Für das Testen gilt: Kann eine Note in beiden Händen gespielt werden und wurde beiden Händen zu diesem Zeitpunkt bereits gleich viele Noten zugeordnet, wird dieser Ton in die rechte Hand gesetzt.
- Algorithmus IV setzt Noten gleicher Länge in die gleiche Hand (wenn spielbar). Kann in so einem Fall keine eindeutige Entscheidung getroffen werden, wird beim Testen Algorithmus II eingesetzt, falls bereits Töne zugeordnet wurden, andernfalls wird der erste Ton bei der Analyse nur anhand der Tonhöhe zugeordnet (bis *Pitch* 60 links, darüber rechts).
- Algorithmus V teilt zu jedem Zeitpunkt das *NoteSet* in Akkorde gleicher Tondauer und gleichem Anschlagszeitpunkt. Diese Akkorde werden dann jeweils möglichst einer Hand zugeordnet. Sollten einzelne Töne nicht zugeordnet werden können, wird Algorithmus II auf das *NoteSet* angewendet.
- Algorithmus VI dient als Ergänzung anderer Algorithmen zur Vermeidung größerer Sprünge. Als Intervall für die Zuordnung zur gleichen Hand wird eine Oktave verwendet.
- Algorithmus VII ordnet die Noten derart den beiden Händen zu, dass nach Möglichkeit die Anzahl der Stimmen in jeder Hand gleich groß ist wie im *NoteSet* zuvor. Wurden im *NoteSet* vor dem *currentNoteSet* keine Noten zugewiesen (Anfang oder Pause), wird Algorithmus II verwendet, andernfalls gilt: wurden in einem *NoteSet* noch keine Noten zugewiesen (z.B. durch Algorithmus I) und ist die Entfernung höchstens eine Oktave, wird Algorithmus VI verwendet. Sollten nach der Analyse weiterhin nicht zugeordnete Noten existieren, wird Algorithmus IV angewendet.
- Algorithmus VIII ordnet einzeln gespielte Noten zu. Wurden im NoteSet vor dem *currentNoteSet* keine Noten zugewiesen, wird die Note des *currentNoteSet* links zugeordnet, falls ihr *Pitch* kleiner oder gleich 60 (c^1) ist, andernfalls rechts. Als Intervall für alle Vergleiche dient die Dezime.

- Algorithmus IX teilt jedes Stück abschnittsweise in einzelne Stimmen und versucht, diese Stimmen möglichst gesamtheitlich einer Hand zuzuordnen. Es werden keine Sekundäralgorithmen verwendet.
- Algorithmus X versucht die Lage zumindest einer Hand beizubehalten und ermöglicht Übergreifen, wo die Beibehaltung so möglich ist. Als Sekundäralgorithmen fungieren Algorithmus I und Algorithmus VIII, als Intervall zur Berechnung der Lagen wird eine Oktave verwendet.

Für die Algorithmen II, III, IV, V, VII, IX gilt, dass das Maximalintervall für jede Hand zunächst als Oktave angenommen wird, können nicht alle Noten zugeordnet werden, wird das Intervall halbtonschrittweise bis zu einer Dezime erhöht und die Zuordnung jeweils erneut versucht. Diese Algorithmen verwenden außerdem jeweils Algorithmus I vor der jeweiligen Analyse.

3 Ergebnisse

Zur Präsentation der Testergebnisse habe ich die Werke (abschnittsweise) in folgende drei Kategorien eingeteilt:

1. Akkordische Homophonie
2. Melodie mit Begleitung
3. Mehrstimmige Werke

Diese Einteilung ist willkürlich und die Grenzen lassen sich nicht scharf ziehen. Es sollen aber dadurch bereits Gemeinsamkeiten der Analyse der Werke erkennbar gemacht werden. Ich werde am Ende jedes dieser Teile die Berührungspunkte anführen, für einen Gesamtvergleich sei auf Teil IV der Arbeit verwiesen.
Nach diesen drei wesentlichen Kategorien werde ich noch auf den Spezialfall des Übergreifens eingehen.

3.1 Akkordische Homophonie

Homophonie kann als „Gleichstimmigkeit mit *rhythmisch gleichen* Stimmen“ definiert werden ([Mi98], S. 93). In diesem Abschnitt werde ich Werke betrachten, die in diesem Sinn über längere Teile als homophon betrachtet werden können, wobei ich hier nur solche Stellen heranziehen möchte, wo keine strenge Mehrstimmigkeit gleichberechtigter Stimmen besteht (wie etwa in einem Choral für mehrstimmigen Chor), sondern mehrere Akkorde in Serie enthalten, welche keinen derartigen

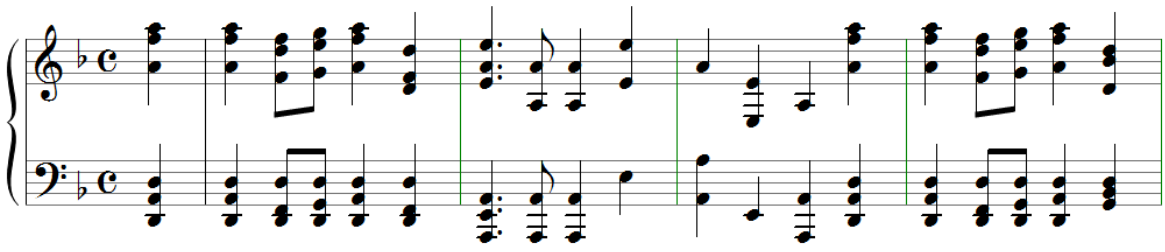
horizontalen Zusammenhang aufweisen müssen.
Hier kann keine klare Abgrenzung zu Werken der zweiten Art (Melodie mit Begleitung) erfolgen, da sehr oft eine Melodie in einer der Stimmen der Akkorde liegt (meistens in der höchsten Stimme). Unabhängig davon, ob die Akkorde eine Melodiestimme enthalten, oder „lediglich“ der Erzeugung von Klangfarben dienen, werden derart gesetzte Stellen in diesem Abschnitt behandelt.

Für die Analyse können hier zwei wesentliche Fälle unterschieden werden:

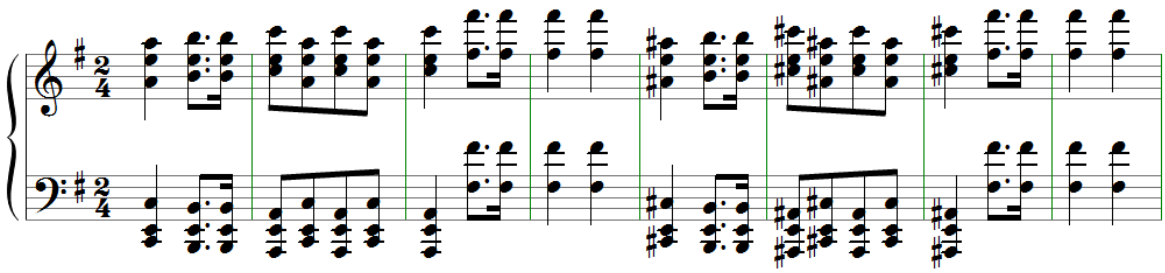
Weite Akkorde

Wie bereits weiter oben beschrieben, definiere ich den Begriff „weite Akkorde“ als Akkorde, welche aufgrund ihrer Spannweite an eindeutigen Stellen auf zwei Hände aufgeteilt werden müssen. Für diese Akkorde gibt es nur zwei mögliche Zuordnungen, wobei eine das Übergreifen einer Hand erfordert. Diesen letzteren Fall möchte ich hier allerdings nicht beschreiben, da ich hier von längeren Abschnitten ausgehe, in denen jeweils nur weite Akkorde gespielt werden, und ein Übergreifen im Allgemeinen in diesem Fall nicht vorgeschrieben wird.
Die Zuordnung kann also in jedem Abschnitt, in dem weite Akkorde auftreten eindeutig erfolgen. Für die Analyse wird Algorithmus I herangezogen, die gleiche Zuordnung kann natürlich auch etwa durch Algorithmus IX erreicht werden.

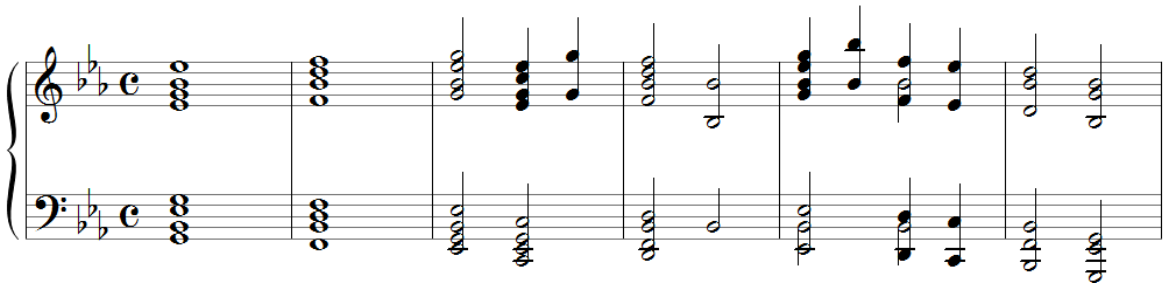
Es folgen drei Beispiele für Werke mit Abschnitten, in denen nur weite Akkorde vorkommen. Die Aufteilung entspricht der des Originals und kann fast durchgängig mit jedem Algorithmus erreicht werden.



Notenbeispiel III-1: Brahms, Ballade op.10 Nr.1: Takte 1-4



Notenbeispiel III-2: Grieg, Sonate: Takte 25-32



Notenbeispiel III-3: Musorgsky, Das große Tor von Kiew: Takte 1-6

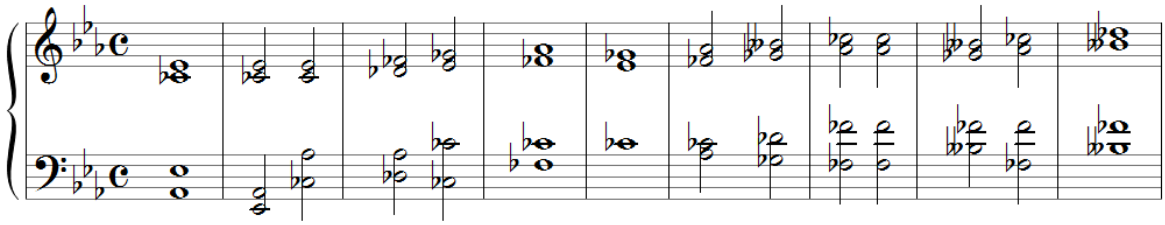
Die Notenbeispiele zeigen Abschnitte der jeweiligen Stücke, die einander in mancher Hinsicht stark ähnlich sind. Bemerkenswert ist vor allem, dass beide Hände in allen drei Fällen fast ständig Akkorde mit Oktavintervall greifen müssen. Die einzigen Stellen, an denen dies unterbrochen wird, sind bei Brahms in Takt 2 auf die letzte Viertel, sowie im folgenden Takt, und bei Mussorgsky in Takt 4. Hier muss die Zuordnung nicht zwangsläufig so erfolgen wie abgebildet, das Versetzen der Mittelstimme in die jeweils andere Hand bringt aber keine Veränderungen der technischen Ansprüche.

Die Wahl der Notenbeispiele lässt auch bereits erkennen, in welcher Musik Abschnitte dieser Art zu finden sind. Alle drei Werke wurden im 19. Jahrhundert geschrieben und sind in der Epoche der Romantik (Hochromantik bzw. Spätromantik) anzuordnen. Es finden sich zahlreiche Klavierwerke dieser Zeit und später, die abschnittsweise oder gänzlich in diesem Stil geschrieben wurden. Als Beispiele sind viele Werke von Brahms, Liszt, Rachmaninow u.v.a. zu nennen.

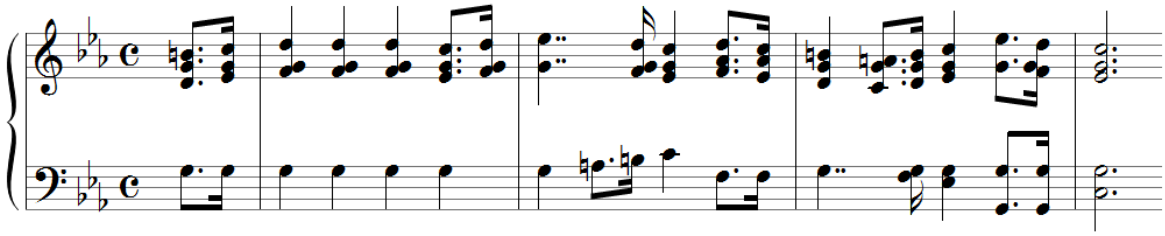
Akkorde mit geringerer Maximalspannweite

Werkteile, welche aufeinanderfolgende Akkorde enthalten, die eine geringere Maximalspannweite aufweisen, kann die Zuordnung nicht derart erfolgen. Solche

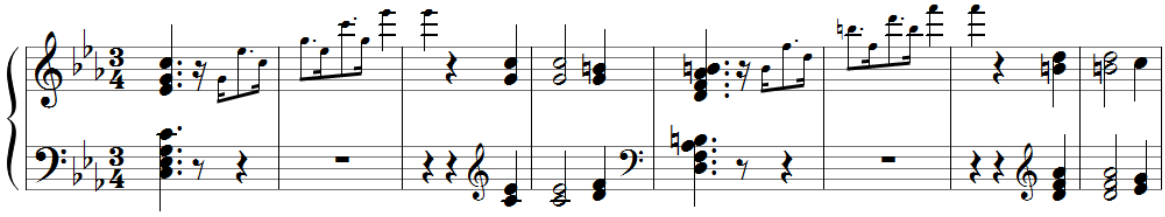
Abschnitte können sehr unterschiedlich gestaltet sein, wie die folgenden Beispiele veranschaulichen.



Notenbeispiel III-4: Mussorgsky: Das große Tor von Kiew: Takte 30-38



Notenbeispiel III-5: Schubert, Impromptu Nr.1: Takte 6-9 (mit Auftakt)



Notenbeispiel III-6: Beethoven, Sonate Nr.5: Takte 1-8

Im ersten Beispiel (Mussorgsky) bleibt die Anzahl der Stimmen pro Hand laut Originalsatz stets gleich (Ausnahme: Takt 34), die Lage der linken Hand ändert sich jedoch sehr oft. Teilweise erfolgt auch hier eine „automatische“ Zuordnung, wenn die Notenpaare sich weit genug voneinander entfernen (z.B. Takt 31).

Bei Schubert hingegen bleibt die Lage der Hände nahezu konstant, die Aufteilung der Originalpartitur sieht für die rechte Hand jeweils drei Stimmen und für die linke zunächst eine, dann zwei vor. Besonders am Anfang der Takte 7 und 8 ist diese Aufteilung zu bevorzugen, da hier jeweils zwei unterschiedliche Rhythmen zu spielen sind, bei den anderen Akkorden müsste die Aufteilung nicht so erfolgen, z.B. könnte die untere Mittelstimme von der linken Hand übernommen werden.

In der abgebildeten Sonate von Beethoven wechseln sich Akkorde in beiden Händen mit solistischen Linien ab. Die Akkorde haben unterschiedlich viele Stimmen, haben verschiedene Maximalspannweiten und behalten nicht immer die Lage.

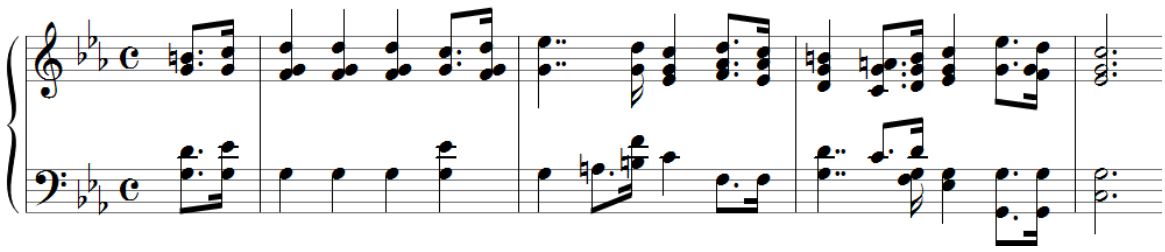
Bereits diese drei Notenbeispiele zeigen, dass die Anforderungen an die Algorithmen zur Trennung der Hände – selbst bei Abschnitten, in denen nur Akkorde vorkommen – sehr unterschiedlich sein können. Die wesentlichen Unterschiede zwischen verschiedenen akkordisch homophonen Werksteilen sind:

1. die Beibehaltung oder Änderung der Anzahl an Stimmen – sowohl insgesamt als auch pro Hand, und
2. der horizontale Verlauf der Lagen der Hände – absolut auf der Klaviatur und zueinander.

Dem entsprechend sind auch die grundlegenden Ansätze zur Analyse verschiedene:

Der gezeigte Teil des *Großen Tors von Kiew* sollte aufgrund der im Original vorgesehenen gleichbleibenden Anzahl an Stimmen pro Hand mit einem der Algorithmen III, VII oder IX zugeordnet werden. Dort, wo die Vierstimmigkeit erhalten bleibt, erfolgt eine Zuordnung wie im Original. Bei allen anderen in Frage kommenden Algorithmen werden nicht konsequent zwei Stimmen pro Hand zugeordnet, durch die Algorithmen IV und X entstehen hier sogar Pausen in einer Hand, während die andere Hand alle vier Stimmen zu übernehmen hat.

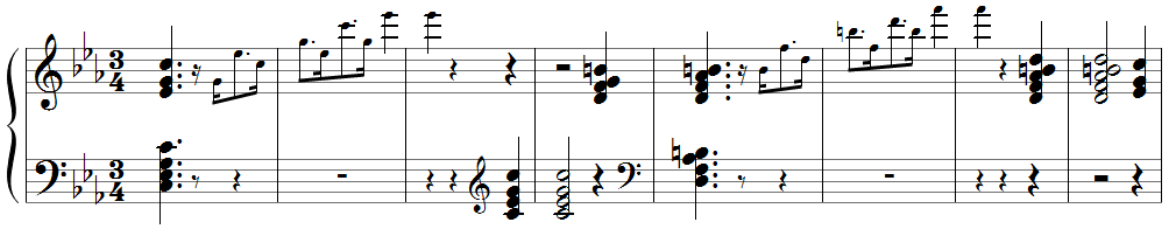
Ähnliche Zuordnungen liefern die Algorithmen bei Schubert, allerdings entspricht hier die Aufteilung der vier Stimmen auf jeweils zwei pro Hand nicht der des Originals, und ist an manchen Stellen wie erwähnt nicht optimal, da unterschiedliche Rhythmen zu spielen sind. Das Ergebnis der Analyse mit Algorithmus II etwa sieht folgendermaßen aus:



Notenbeispiel III-7: Schubert, Impromptu Nr.1: Takte 6-9 (Algorithmus II)

Das Beispiel zeigt eine Aufteilung, die durchwegs spielbar ist und in beiden Händen keine großen Intervalle fordert. Die Aufteilung der Stimmen wechselt allerdings ständig, mehrere Rhythmen in einer Hand werden dennoch nicht vermieden.
Auch der beim vorherigen Beispiel gute Algorithmus III kann im letztgenannten Fall keine besseren Ergebnisse erzielen, die Stimmen werden aber zumindest horizontal gleichmäßig aufgeteilt. Den gezeigten Ausschnitt können die Algorithmen VII und IX originalgetreu wiederherstellen. Jedoch werden an weiteren ähnlichen Stellen desselben Werkes mitunter Aufteilungen entschieden, die zwar besser als die der anderen Algorithmen sind, aber trotzdem vom Original abweichen.

Zwei verschiedene Ergebnisse der Analyse der 5. Sonate von Beethoven habe ich bereits weiter oben gezeigt (s. Seiten 31 und 49). Die Algorithmen II, III, V, IX und X liefern an diesen akkordischen Stellen jeweils gute Ergebnisse, die sich geringfügig voneinander unterscheiden. Da sich die Anzahl der Stimmen häufiger ändert, ist das Gleichbleiben der Stimmen in einer Hand weniger wichtig. Algorithmus IV ordnet – ebenso wie Algorithmus VI (von VII verwendet) – häufig Akkorde nur einer Hand zu, wenn die Maximalspannweite eine Oktave nicht überschreitet. Der Nachteil bei Algorithmus IV ist die Zuordnung solcher Akkorde zu verschiedenen Händen abhängig vom tiefsten Ton. Das folgende Beispiel zeigt, dass die Verwendung von Algorithmus IV hier daher ungünstig ist:



Notenbeispiel III-8: Beethoven, Sonate Nr.5: Takte 1-8 (Algorithmus IV)

Ein weiteres Beispiel für akkordisches Spiel findet sich im Mittelteil des o.a. Nocturnes von Chopin. Die Anzahl der Stimmen ändert sich an mehreren Stellen, die Lage beider Hände wechselt häufig und teilweise kann die Zuordnung durch Algorithmus I eindeutig erfolgen. Für die übrigen Stellen kann jeder Algorithmus herangezogen werden, die Ergebnisse sind durchwegs gut spielbar.

Der Fantastische Tanz von Schostakowitsch enthält ebenfalls viele Akkorde, die rechte Hand schreitet meistens in Oktavenbewegungen fort. Da die beiden Hände teilweise sehr weit auseinander liegen, werden viele Noten bereits durch Algorithmus I zugeordnet, lediglich in den Pausen einer Hand kommt es bei einigen Algorithmen zu großen Sprüngen dieser Hand.

Die Analyse von akkordisch gesetzten Werken kann wesentlich einfacher beurteilt werden, als die von streng mehrstimmig gesetzten oder solchen, die eine Melodie und eine dazu nicht homophone Begleitung haben. Der Grund dafür liegt darin, dass bei akkordischen Abschnitten die einzelnen *NoteSets* idealerweise keine Noten enthalten, die auch andere *NoteSets* enthalten und daher wirklich getrennt voneinander betrachtet werden können.
Welcher Algorithmus optimalerweise zur Analyse heranzuziehen ist, hängt vom Satz ab, viele solcher akkordischen Abschnitte lassen sich aber bereits durch Algorithmus I entscheiden. Für gleich bleibende Anzahl der Stimmen eignen sich III, VII und IX, bei wechselnder Zahl können auch die anderen Algorithmen herangezogen werden.

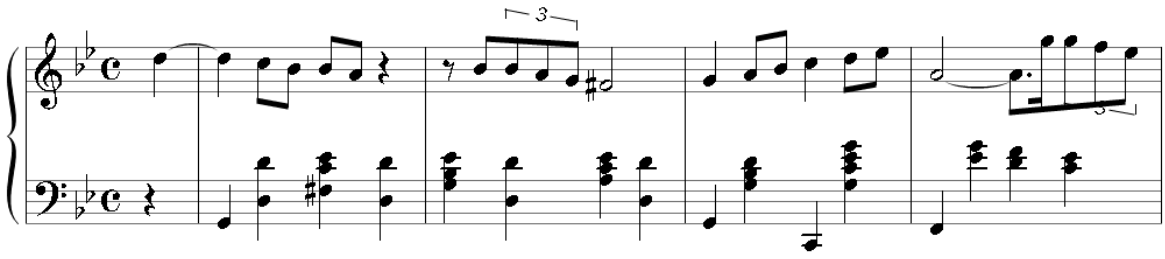
3.2 Melodie mit Begleitung

Nahezu jeder Abschnitt jedes Werkes kann als Melodiestimme mit oder ohne Begleitung aufgefasst werden. In diesem Teil möchte ich nur solche Werkteile betrachten, in denen es eine offensichtliche Melodiestimme gibt, die sich von den übrigen Tönen abhebt, welche als Begleitung aufgefasst werden.

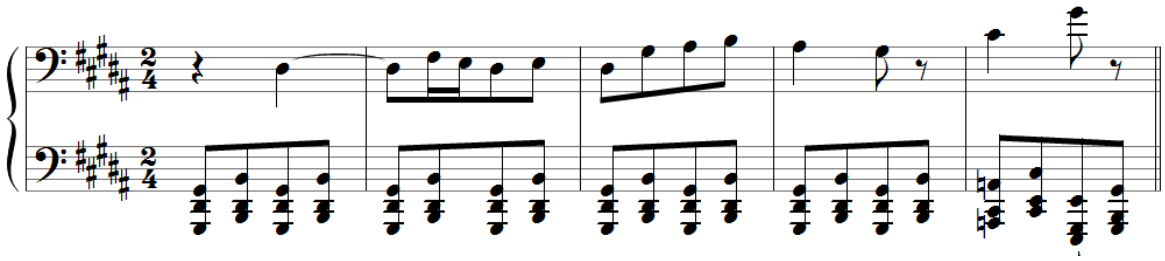
Für diesen Fall gibt es unterschiedliche Realisierungen. So kann die Begleitung etwa aus Akkorden, Arpeggien (zerlegte Akkorde) oder Stimmen aus aufeinanderfolgenden einzelnen Tönen bestehen, die Begleitung kann außerdem einem gleich bleibenden oder wechselnden Rhythmus unterliegen. In einer großen Zahl an Werken finden sich natürlich auch diverse Kombinationen dieser Schemata.

Akkordische Begleitung

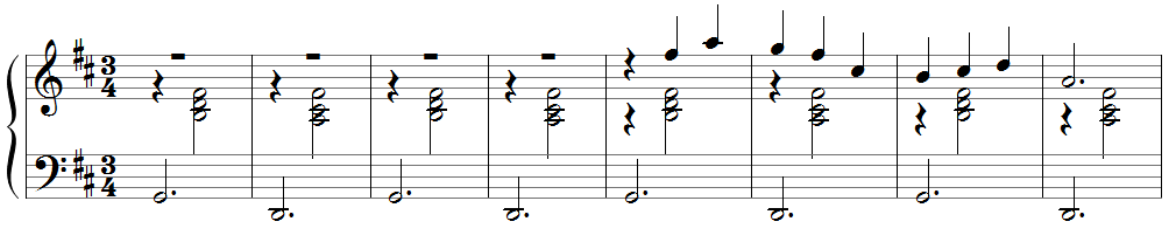
Nachfolgend sind Beispiele für Abschnitte mit Melodie und akkordischer Begleitung aus den Teststücken abgebildet.



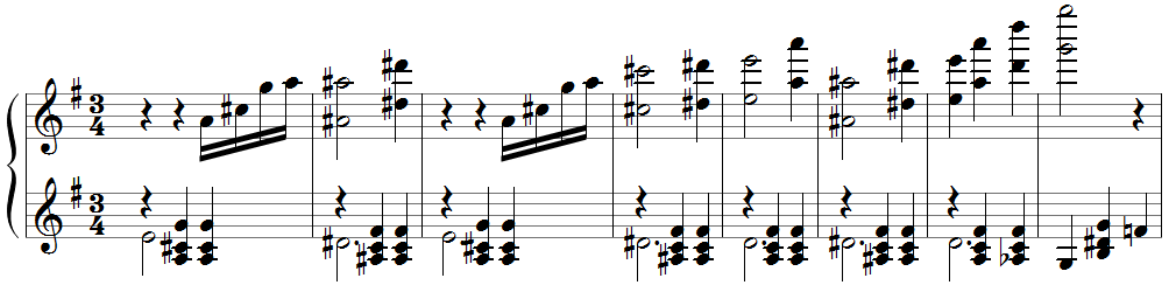
Notenbeispiel III-9: Chopin, Nocturne op.37 Nr.1: Takte 1-4 (mit Auftakt)



Notenbeispiel III-10: Mussorgsky, Bydlo aus den Bildern einer Ausstellung: Takte 1-5



Notenbeispiel III-11: Satie, Gymnopedie Nr.1: Takte 1-8



Notenbeispiel III-12: Schostakowitsch, Fantastischer Tanz Nr.1: Takte 1-8

In verschiedener Weise kann in allen abgebildeten Beispielen eine Melodiestimme in der rechten Hand und eine aus aufeinanderfolgenden Akkorden bestehende Begleitung in der linken Hand erkannt werden.

In den ersten beiden Werkausschnitten ist die Begleitung rhythmisch gleichmäßig gestaltet, bei Satie sind abwechselnd einzelne Töne und Akkorde notiert, bei Schostakowitsch findet sich ein ähnliches Muster. Die Melodiestimmen wechseln

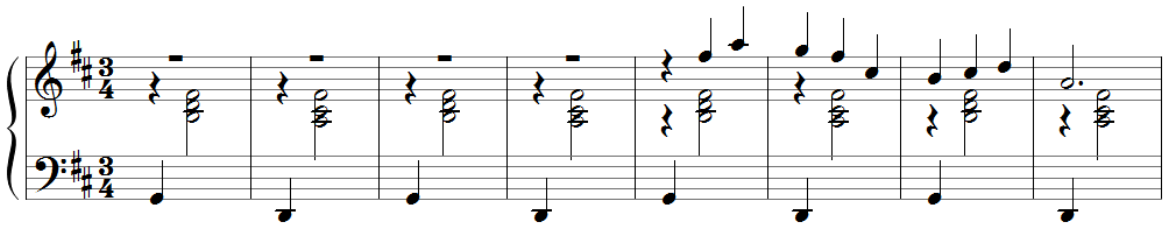
naturgemäß zumindest gelegentlich den Rhythmus, bei Satie beginnt die Melodie erst im fünften Takt, bei den anderen Werken gleich im ersten. Zu unterscheiden ist auch das Verhältnis des Begleitrhythmus zum Grundrhythmus der Melodie. Bei Mussorgsky etwa sind in der Begleitung Achtel zu spielen, was der kürzesten Zählzeit entspricht (mit Ausnahme der Sechzehntel im zweiten Takt), bei Satie und Chopin geht die Begleitung klar als langsamere Stimme hervor.

Wie auch bereits im Abschnitt 3.4 erörtert, eignen sich für Akkorde, die nicht den gleichen Rhythmuswert aufweisen wie die gleichzeitig zu spielenden Töne der Melodiestimme, vor allem die Algorithmen IV und V. Abhängig vom Kontext können auch die Algorithmen VII und X herangezogen werden. Schlechtere Resultate finden sich bei der Analyse durch II oder III. Die vertikalen Algorithmen sollten jedenfalls gemeinsam mit Algorithmus VI verwendet werden, damit große Sprünge vermieden werden können.

Der gezeigte Abschnitt des Werks von Chopin kann mit Algorithmus IV originalgetreu wiederhergestellt werden. Die anderen Algorithmen teilen die Begleitakkorde verschieden auf, wodurch die rechte Hand neben der Melodie auch Teile der Begleitung spielen muss. An Stellen, wo Verzierungen zu spielen sind, ist dies eher hinderlich, insgesamt bleibt das Werk dennoch gut spielbar.

Weiter oben in dieser Arbeit finden sich mehrere Notenbeispiele für Analyseergebnisse von Mussorgskys *Bydlo*, welche zeigen, dass eine gute Aufteilung mit den angeführten Algorithmen nicht einfach möglich ist. Die Anwendung von Algorithmus X zur Beibehaltung der Lage kann hier beispielsweise nicht erfolgreich angewendet werden, weil die Lage der linken Hand ständig wechselt; dadurch werden weder diese Akkorde konsequent links zugeordnet, noch können große Sprünge der rechten Hand vermieden werden.
Große Teile des Werkes sind aber mit Algorithmus I entscheidbar, bei den weiteren Stellen liefert Algorithmus IV die besten Ergebnisse.

Die Gymnopédie kann in der abgebildeten Form gar nicht gespielt werden (s. Seite 15), ich gehe von einer Einspielung aus, bei der die Begleitung durchgehend von der linken Hand gespielt wird:



Notenbeispiel III-13: Satie, Gymnopedie: Takte 1-8 in einer möglichen Einspielung

Diese Aufteilung kann von Algorithmus IV erreicht werden, die anderen Algorithmen teilen die Begleitung verschieden auf.

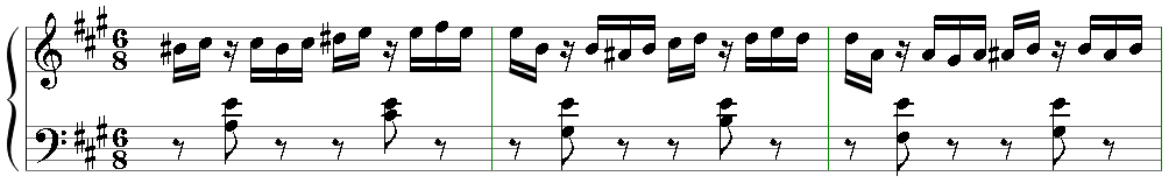
Der Tanz von Schostakowitsch weist viele Stellen auf, die mit Algorithmus I entschieden werden können, für den Rest liefern alle Algorithmen passable Ergebnisse, wobei wieder in Pausen der rechten Hand durch die vertikalen Algorithmen und Algorithmus X Sprünge zugewiesen werden. Die besten Ergebnisse werden durch IV und VII erzielt, auch Algorithmus V ist sehr gut.

Es ist klar erkennbar, dass akkordisch begleitete Melodien gut durch die Algorithmen IV und V aufgeteilt werden können (falls die Aufteilung nicht ohnehin mit Algorithmus I alleine funktioniert). Dabei weist Algorithmus IV konsequenter die gleich langen Töne einer Hand zu, was aber auch zu unterwünschten Nebeneffekten führen kann – insbesondere dann, wenn der entsprechende Ton der Melodie ebenfalls die gleiche Dauer aufweist. Zur Vermeidung großer Sprünge einer Hand ist als Ergänzung jedenfalls Algorithmus VI zu empfehlen. Durch die oft wechselnde Anzahl an gleichzeitig zu spielenden Noten und die zahlenmäßig ungleiche Aufteilung der Noten auf die Hände sind die Algorithmen II, III und IX weniger gut geeignet, bei den verbliebenen Algorithmen VII und X ist das jeweilige Ergebnis stark von der Linienführung der beiden Hände abhängig.

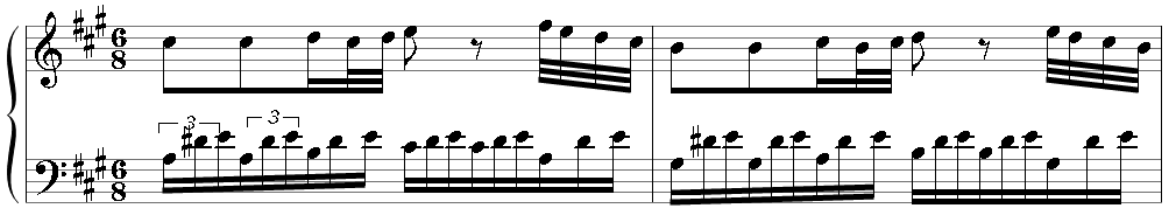
Begleitungen in Form von eigenen Stimmen und Arpeggien

Eine Begleitung kann als eigene Stimme aufgefasst werden, wenn sie aus einer wenig unterbrochenen Abfolge von einzeln gespielten Tönen besteht. Die Ausprägungen können von wenig singbaren Arpeggien bis hin zu Stimmen reichen, die alleine als Melodie fungieren könnten. Diese beiden Extremfälle sind schwer logisch zu trennen, da eine Zuweisung zu einem der Bereiche oft nicht eindeutig

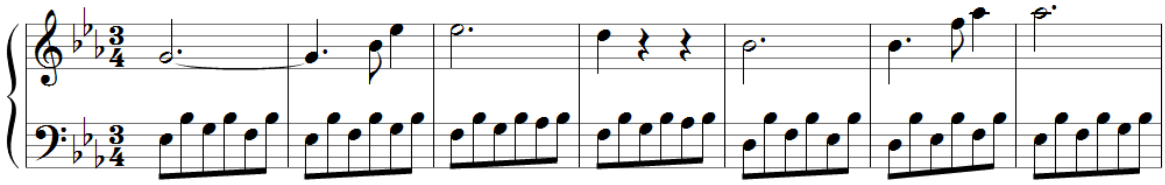
getroffen werden kann, und viele Mischformen existieren. In der Klavierliteratur finden sich sehr verschiedene Ausführungen:



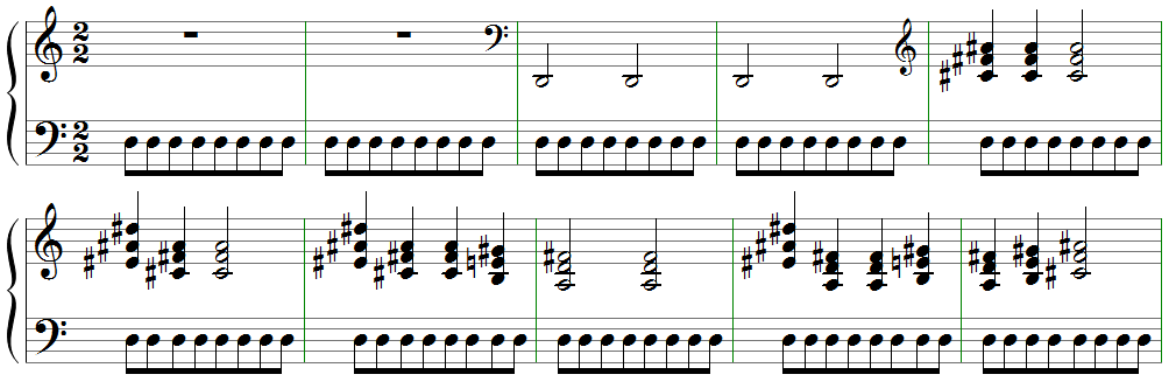
Notenbeispiel III-14: Mozart, Sonate KV300i Variation 1, Takte 1-3



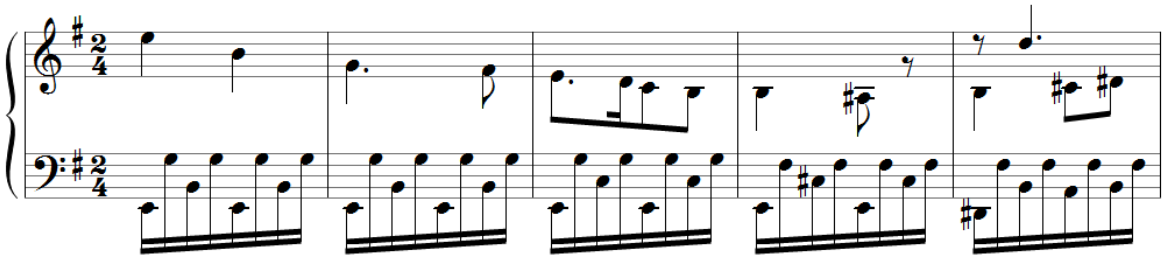
Notenbeispiel III-15: Mozart, Sonate KV300i Variation 2, Takte 1-2



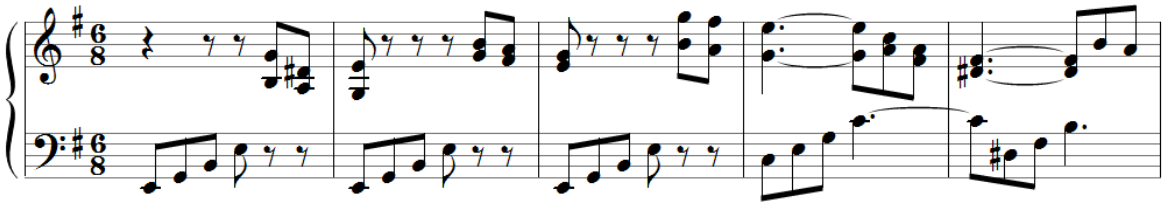
Notenbeispiel III-16: Beethoven, Sonate Nr.5: Takte 56-62



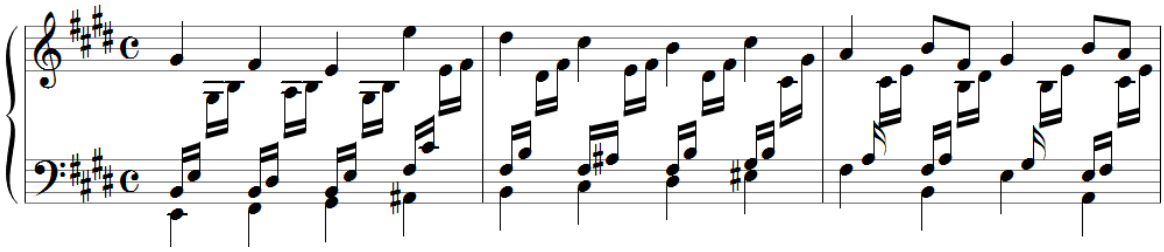
Notenbeispiel III-17: Bártok, Bärentanz: Takte 1-10



Notenbeispiel III-18: Grieg, Sonate: Takte 1-5



Notenbeispiel III-19: Haydn, Sonate: Takte 1-5



Notenbeispiel III-20: Mendelssohn, Lied ohne Worte Nr.1: Takte 3-5

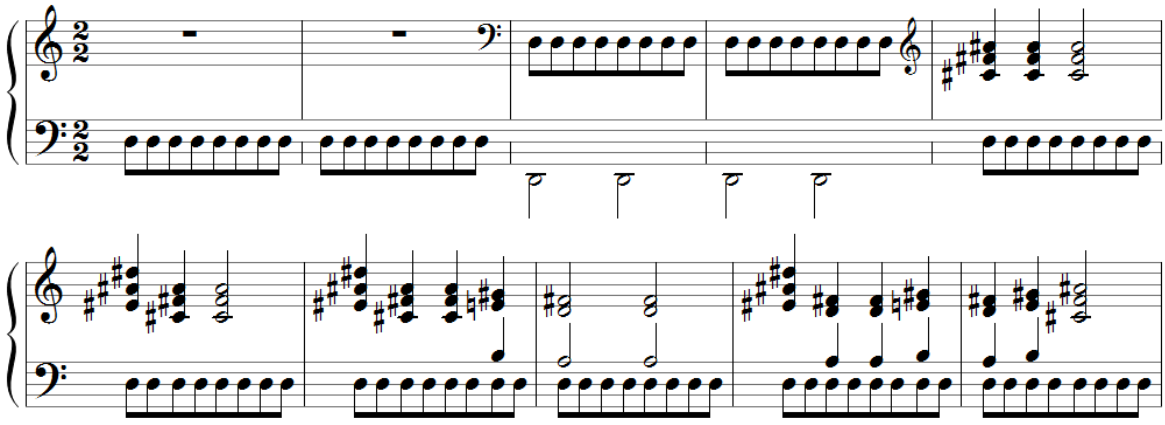
Die Abbildungen veranschaulichen unterschiedliche Ausführungen von Melodiesätzen mit Begleitungen, wobei die Begleitung jeweils aus (Akkord-) Zerlegungen oder eigenen untergeordneten Stimmen bestehen.
Die erste Variation der Sonate von Mozart sieht ein Spielen der Melodie in der rechten und dazwischen Akkorde in der linken Hand vor. Da an den Stellen, an welchen die Akkorde beginnen, jeweils eine Pause in der rechten Hand ist, bedeutet eine Zuordnung durch die Algorithmen II, III und IX das Aufteilen dieser Akkorde auf beide Hände. Besser geeignet sind hier die Algorithmen IV und X, die mehrheitlich die angegebene Zuordnung herstellen können.

Einen ganz anderen Stil der Besetzung weist die darauf folgende zweite Variation der gleichen Sonate auf. Die Begleitung ist hier durchgehend gesetzt und besteht aus Zerlegungen, gemeinsam mit der durchgehenden Melodie kann diese Variation auch als streng zweistimmig gesehen werden. Sie lässt sich daher mit den Algorithmen II, III und IX originalgetreu wiederherstellen, durch die verschiedenen

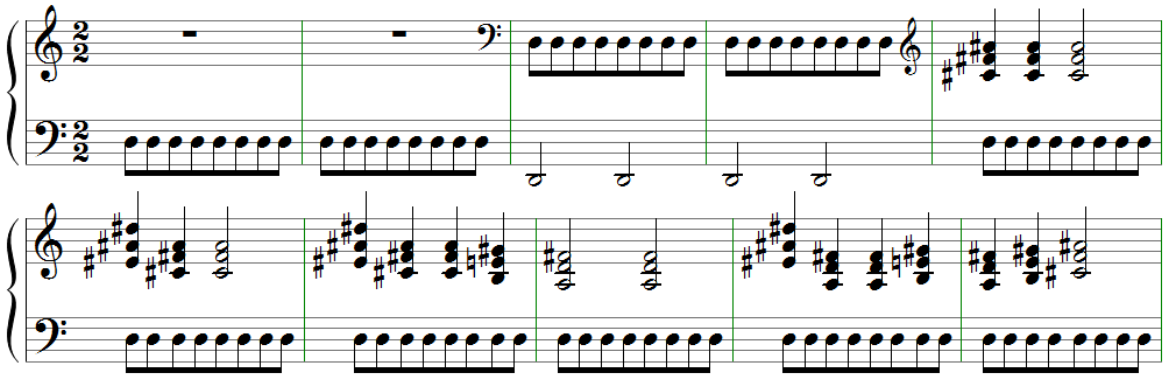
Rhythmen in beiden Händen eignen sich aber auch die Algorithmen IV und V. Die auf die verschiedenen vertikalen Algorithmen aufbauenden Algorithmen VII und X können daher auch verwendet werden.

Die nächste Abbildung zeigt ein Seitenthema aus der Sonate von Beethoven, aus der ich bereits ein anderes Beispiel präsentiert habe. In diesem Teil der Sonate spielt die rechte Hand eine sangbare Melodie, die linke Hand begleitet mit Akkordzerlegungen. Ähnlich wie bei der zweiten Variation der Mozart Sonate kann auch hier jeder Algorithmus verwendet werden.

Bei Bartók besteht die Begleitung der linken Hand aus der Repetition eines einzigen Tons, die rechte Hand spielt ab Takt 5 Akkorde, wobei jeweils der höchste Ton als Melodieton gespielt werden kann. Im Original ist in den Takten 3 und 4 ein Übergreifen der rechten Hand über die linke vorgesehen. Auf die Problematik des Übergreifens sei auf die Abschnitte I.6.2 und III.3.4 verwiesen, im Folgenden finden sich zwei Abbildungen, bei denen kein Übergreifen zugewiesen wurde.



Notenbeispiel III-21: Bartók, Bärentanz: Takte 1-10 mit Algorithmus II

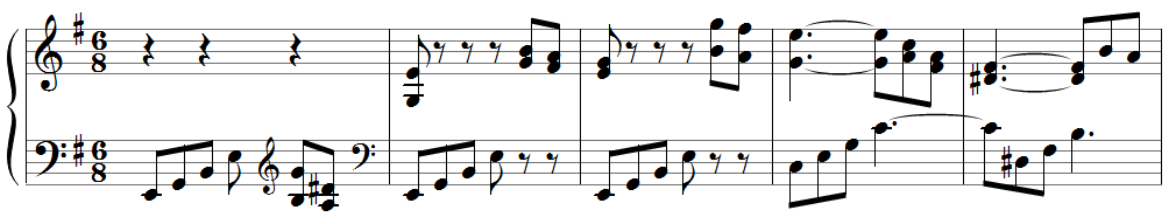


Notenbeispiel III-22: Bartók, Bärentanz: Takte 1-10 mit Algorithmus VII

Ab Takt 7 weist Algorithmus II der linken Hand immer wieder Teile der Akkorde der rechten Hand zu. Das Stück bleibt zwar spielbar, da aber ein sehr schnelles Tempo vorgesehen ist, wird es schwieriger. Ähnliche Zuweisungen werden durch die Algorithmen III und IX getätigt. Bessere Ergebnisse können mit IV und VII erreicht werden, bis auf das Übergreifen kann so der Originaltext wiederhergestellt werden (s. Abbildung).

Eine konsequente Begleitung aus Akkordzerlegungen findet sich zu Beginn der bereits bekannten Klaviersonate von Grieg. Dazu spielt die rechte Hand eine einstimmige Melodie, ab Takt 5 kommen einzelne Töne einer weiteren Stimme dazu. Wieder kann die Analyse problemlos durch die Algorithmen II, III und IX erfolgen, auch die anderen Algorithmen funktionieren aufgrund der großen Spannweiten mehrheitlich sehr gut. Wo die Maximalintervalle geringer sind, ist Algorithmus VI zu vermeiden, da dadurch teilweise die schnelle Begleitung gemeinsam mit der Melodiestimme in nur eine Hand gesetzt wird.

Das Hauptthema des 1. Satzes der beschriebenen Sonate von Haydn ist so gesetzt, dass beide Hände abwechselnd spielen sollten. Als Melodie kann die Oberstimme der Akkorde der rechten Hand gesehen werden, die linke Hand soll dazwischen Akkordzerlegungen spielen. Das wechselnde Spiel zu Beginn des Stückes bereitet beim Analysieren Schwierigkeiten und kann hier von keinem Algorithmus originalgetreu wiederhergestellt werden, ab Takt 4 spielen beide Hände gemeinsam, dieser Teil kann leichter analysiert werden. Die Algorithmen II und III beispielsweise teilen die Akkorde der rechten Hand auf, wodurch die linke Hand größere Sprünge machen muss; Algorithmus IV weist zwar die Akkorde meistens nur einer Hand zu, aber nicht immer der gleichen. Eine recht gute Lösung kann hier durch den horizontalen Algorithmus VII (unter Zuhilfenahme von Algorithmus VI) erzielt werden, wie die folgende Abbildung veranschaulicht.



Notenbeispiel III-23: Haydn, Sonate: Takte 1-5 mit Algorithmus VII

Einen Grenzfall zu den im folgenden Abschnitt beschriebenen mehrstimmigen Werken stellt das „1. Lied ohne Worte“ von Mendelssohn dar. Wie auch ein vokales Lied besteht das Werk aus einer sangbaren Melodiestimme und einer Begleitung, wobei diese hier selbst aus zwei Stimmen besteht. Die Mittelstimme soll abwechselnd von beiden Händen gespielt werden.
Das kompliziert aussehende Schema kann leicht hergestellt werden, da die Spannweiten der Hände größtenteils genau diese Aufteilung forcieren. Die wenigen Stellen, an denen dieses Kriterium nicht ausreicht werden zwar von den verschiedenen Algorithmen unterschiedlich zugewiesen, das Werk bleibt aber jedenfalls durchgehend gut spielbar.

Häufig finden sich bei Werken der hier beschriebenen Art durchgehend einstimmige Begleitungen, wodurch diese wie zweistimmige Werke analysiert werden können. Es eignen sich hierfür vor allem die Algorithmen III, VII und IX.
Wie das Beispiel von Bartók zeigt, liefert Algorithmus IV für Werke, in denen Akkorde und eine weitere Stimme vorgeschrieben sind, sehr gute Ergebnisse – unabhängig davon, ob die Akkorde den gleichen Rhythmus wie die Melodiestimme haben, oder eine Melodiestimme mit akkordischer Begleitung anzufinden ist.

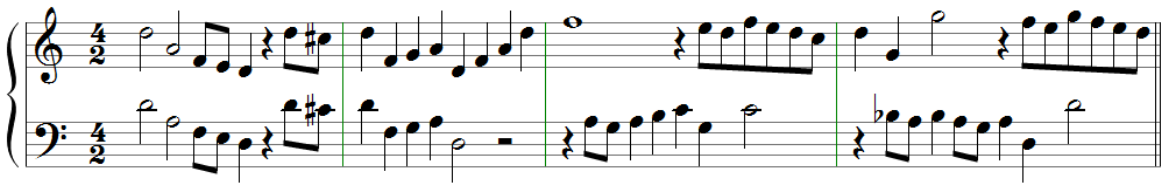
3.3 Streng mehrstimmige Werke

In diesem Abschnitt geht es nicht darum, zu erörtern, welcher Algorithmus sich am besten für die Analyse eignet. Vielmehr soll aufgezeigt werden, dass die stimmbezogene Analyse mit Algorithmus IX möglich ist und für mehrstimmige Werke Ergebnisse liefert, welche denen der anderen Algorithmen für solche Stücke überlegen sind.

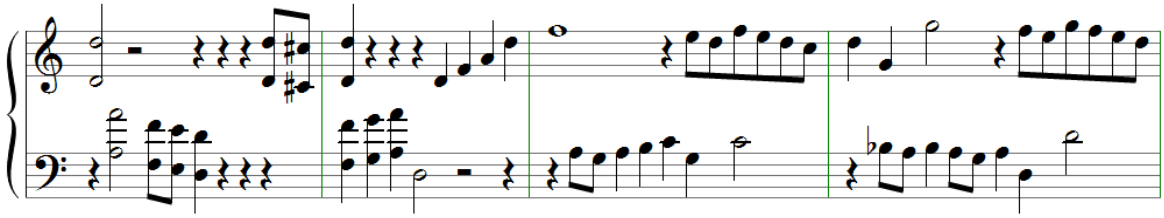
Zweistimmige Werke

Werke mit zwei gleichberechtigten unterschiedlichen Stimmen, zwei Unisono-Stimmen oder zwei Stimmen, von denen eine die Melodie, die andere die Begleitung darstellt, sind meistens so zu spielen, dass jede Hand eine dieser Stimmen übernimmt. Dort, wo in keiner Hand eine Pause gesetzt wurde, kann diese Zuordnung verlässlich mit einem der Algorithmen II, III oder IX erfolgen, an den einstimmigen Stellen wird Algorithmus VIII verwendet, welcher in den meisten Fällen

ebenfalls zufriedenstellende Ergebnisse liefert.
Denkbar sind aber auch hier andere Zuordnungen, vor allem wenn es sich um gleichartige Rhythmuswerte in beiden Stimmen handelt; hierzu folgen zwei Beispiele:

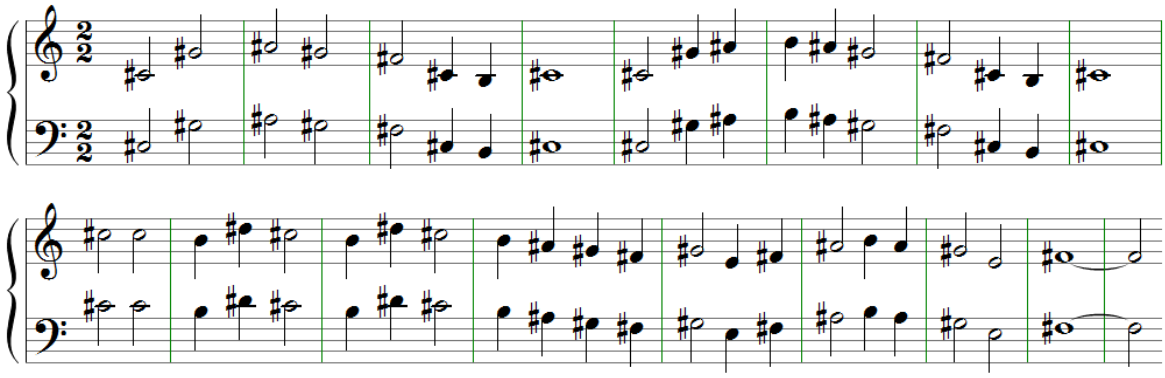


Notenbeispiel III-24: Telemann, Fantasia: Takte 1-4



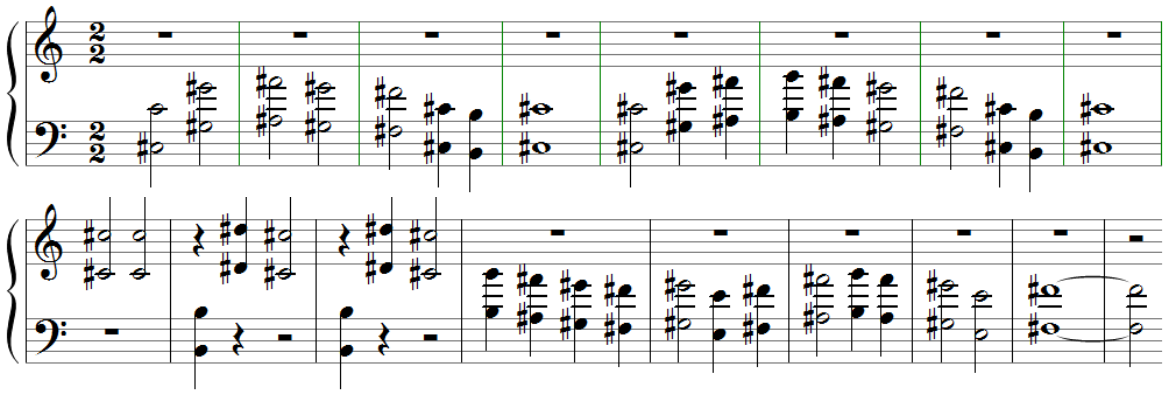
Notenbeispiel III-25: Telemann, Fantasia: Takte 1-4 mit Algorithmus IV

Die Notenbeispiele zeigen, dass die Aufteilung der Unisono-Stimmen zu Beginn der Fantasia von Telemann verschieden erfolgen kann. Durch die Algorithmen II, III, V oder IX wird die originale Aufteilung erzeugt, Algorithmus IV lässt die Stimmen jeweils als Akkorde von einer Hand spielen; da hier auf horizontale Zusammenhänge keine Rücksicht genommen wird, kommt es zu einem Wechsel der Hände bei den ersten zwei Akkorden. Abgesehen davon ist auch diese Zuordnung durchaus gut spielbar, bei langen Unisono-Stellen (bzw. Parallelbewegungen) oder schnellen derartigen Passagen ist aber eine Aufteilung auf beide Hände erstrebenswert.



Notenbeispiel III-26: Bartók, Bauernlied: Takte 1-17 mit Algorithmus II, III oder IX

Das Bauernlied von Bartók ist durchgängig durch zwei Unisono-Stimmen besetzt – auch hierfür gilt das oben gesagte, wie die beiden Beispiele veranschaulichen.

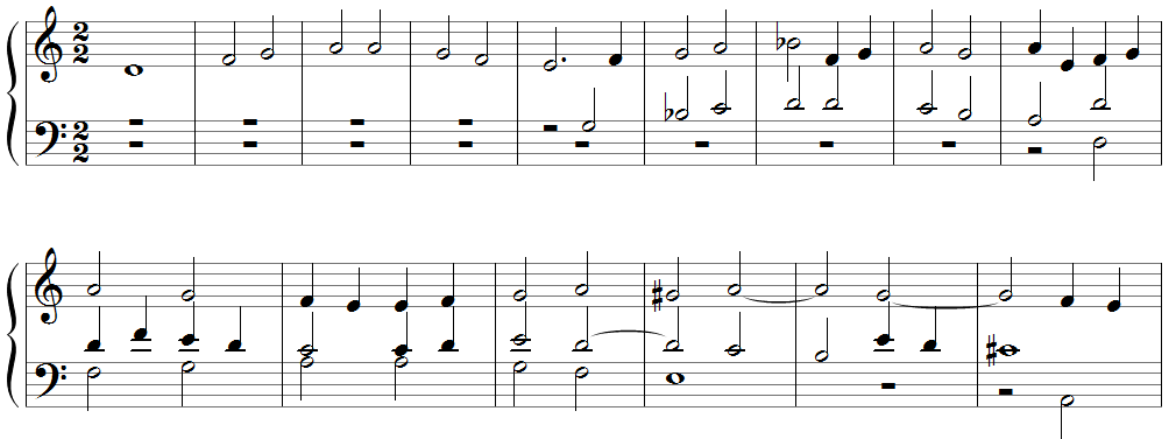


Notenbeispiel III-27: Bartók, Bauernlied: Takte 1-17 mit Algorithmus IV

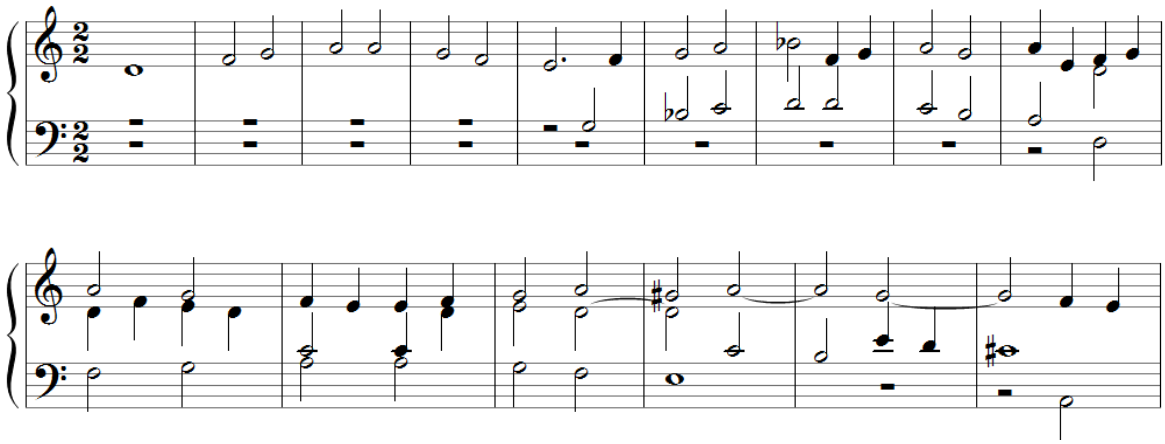
Mehrstimmige Werke

Strenge Mehrstimmigkeit in Klaviermusik findet sich gehäuft in der Epoche des Barock. Besonders die Werksform der Fuge, welche zum Beispiel von J. S. Bach häufig verwendet wurde, ist bekannt. Zu Beginn wird stets ein Thema präsentiert, welches danach in neuen Stimmen transponiert wiederholt wird, während die anderen Stimmen kontrapunktisch begleiten. Im Folgenden werde ich Ausschnitte aus drei Werken in fugenartigem Stil anhand von Notenbeispielen präsentieren. Fugen werden häufig so im Klaviernotensystem notiert, dass Stimmen abhängig von ihrer Lage auf der Klaviatur im oberen oder unteren System geschrieben werden. Dadurch ist die Hand, mit der eine Stimme gespielt werden soll, nicht immer eindeutig ersichtlich. Eine derartige Zuordnung könnte durch stimmenbezogene Analyse auch erzeugt werden, ich werde hier dennoch eine Teilung nach Händen untersuchen.

Eine durch Algorithmen generierte Zuordnung kann daher aber nicht direkt mit dem Originalsatz verglichen werden.



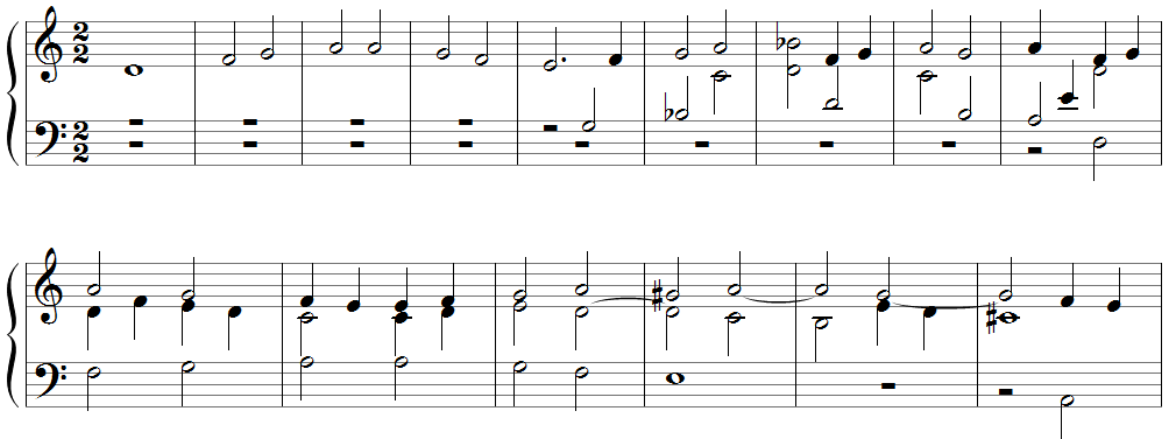
Notenbeispiel III-28: Krieger, Fughetta: Takte 1-15 mit Algorithmus IX



Notenbeispiel III-29: Krieger, Fughetta: Takte 1-15 mit Algorithmus II

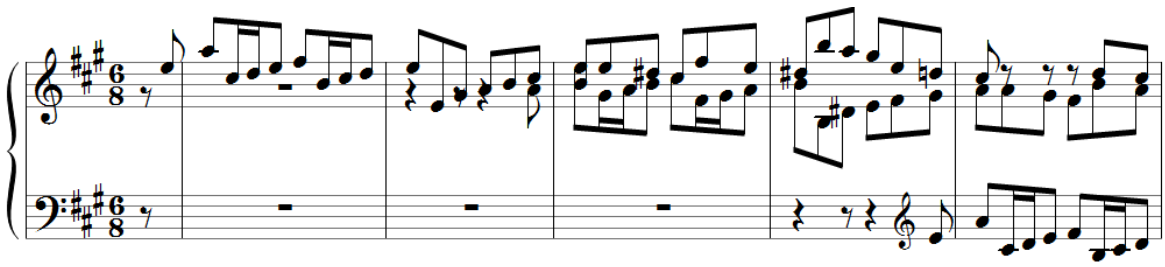
Die Abbildungen zeigen zwei verschiedene mögliche Aufteilungen der Stimmen auf die beiden Hände. Im zweistimmigen Teil (Takte 1-9) sind die beiden Ergebnisse identisch, danach unterscheiden sich beide Versionen in der Zuordnung der Mittelstimme – Algorithmus IX weist die Mittelstimme der linken Hand zu, welche dort im gesamten gezeigten Teil bleibt, Algorithmus II hingegen weist die Töne so zu, dass jeweils eine geringe Spannweite in der jeweiligen Hand vorliegt; in diesem Fall wird die Mittelstimme mehrheitlich rechts zugeordnet, Ausnahmen sind aber nicht ausgeschlossen.

Eine weitere Art der Aufteilung liefert Algorithmus X:



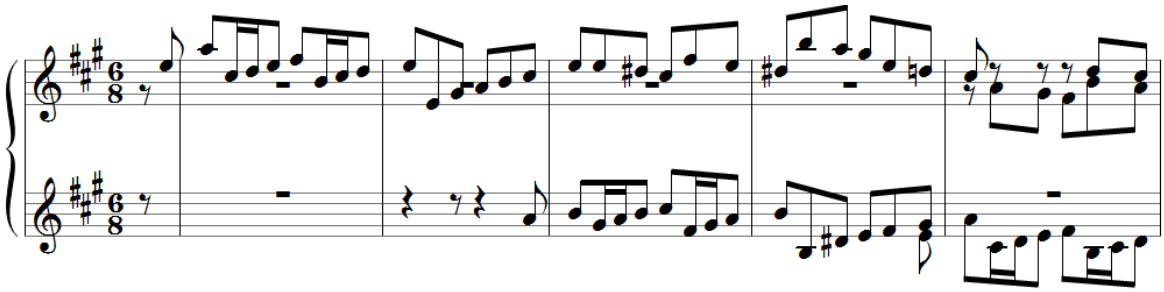
Notenbeispiel III-30: Krieger, Fughetta: Takte 1-15 mit Algorithmus X

Bei diesem Algorithmus findet keine stimmenbezogene Analyse statt und in den Takten 6-9 wechselt die Hand der Unterstimme ständig, ab Takt 10 wird eine konsequente Zuordnung der Mittelstimme in die rechte Hand vorgenommen.
Wird eine stimmbezogene Analyse gewünscht, sollte Algorithmus IX verwendet werden, aber auch andere Algorithmen liefern mitunter sehr gut spielbare Ergebnisse (z.B. II, III und X).



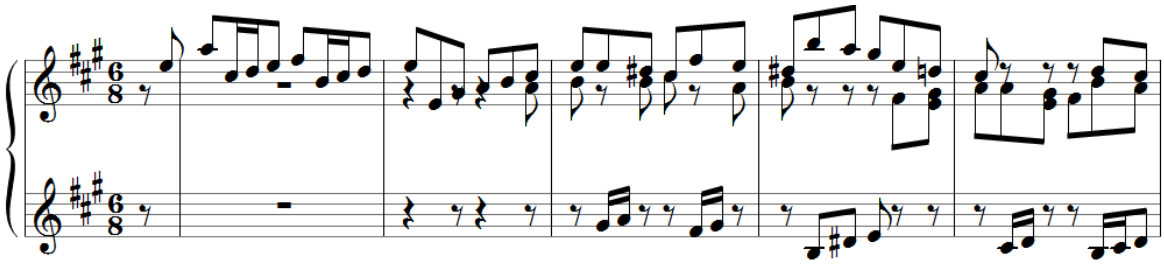
Notenbeispiel III-31: Schumann, Fuge: Takte 1-5

Als zweites Notenbeispiel behandle ich eine Fuge des Komponisten R. Schumann, welcher in der Epoche der Romantik anzusiedeln ist. Der abgebildete Ausschnitt zeigt den Anfang der Fuge, wo die drei Stimmen nacheinander das Thema präsentieren. Hier findet sich im Originalnotentext eine Aufteilung nach Stimmen auf die beiden Notensysteme (z.B. sind einige Noten der Unterstimme in Takt 4 nicht von der rechten Hand spielbar, obwohl sie oben notiert sind), sinnvollerweise ist im zweistimmigen Teil (Takte 3-4) die Unterstimme mit der linken Hand zu spielen; eine derartige Zuordnung kann durch Algorithmus IX erfolgen:



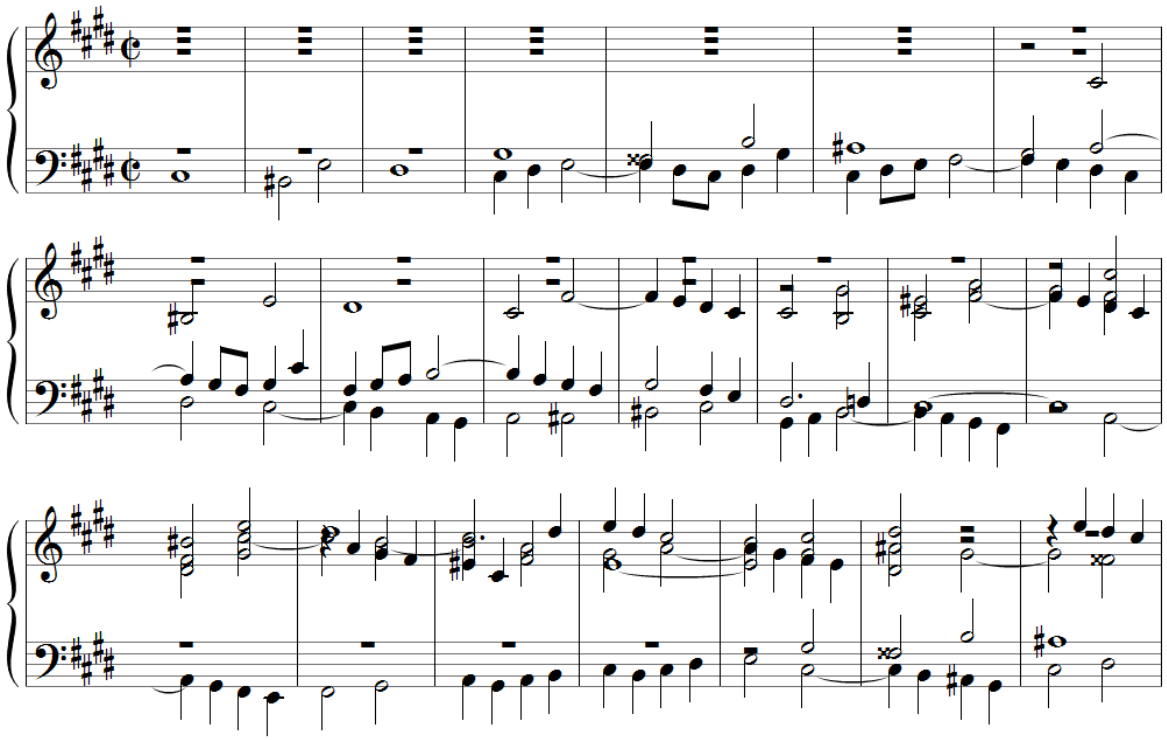
Notenbeispiel III-32: Schumann, Fuge: Takte 1-5 mit Algorithmus IX

Die Stimmkreuzung in der ersten Achtel von Takt 3 wird nicht erkannt und die Zuordnung zu den beiden Händen daher vertauscht, ansonsten findet eine sinnvolle Aufteilung der Stimmen statt, welche auch keine vermeidbaren technischen Schwierigkeiten beim Spiel bereitet.
Zum Vergleich zeige ich im Anschluss noch ein Ergebnis der Analyse mit Algorithmus IV. Im zweistimmigen Teil wird die Unterstimme unregelmäßig auf die beiden Hände verteilt, wodurch die hier melodieführende Stimme in ihrem Zusammenhang zerrissen klingen kann. Eine derartige Aufteilung ist in diesem Zusammenhang jedenfalls möglichst zu vermeiden.



Notenbeispiel III-33: Schumann, Fuge: Takte 1-5 mit Algorithmus IV

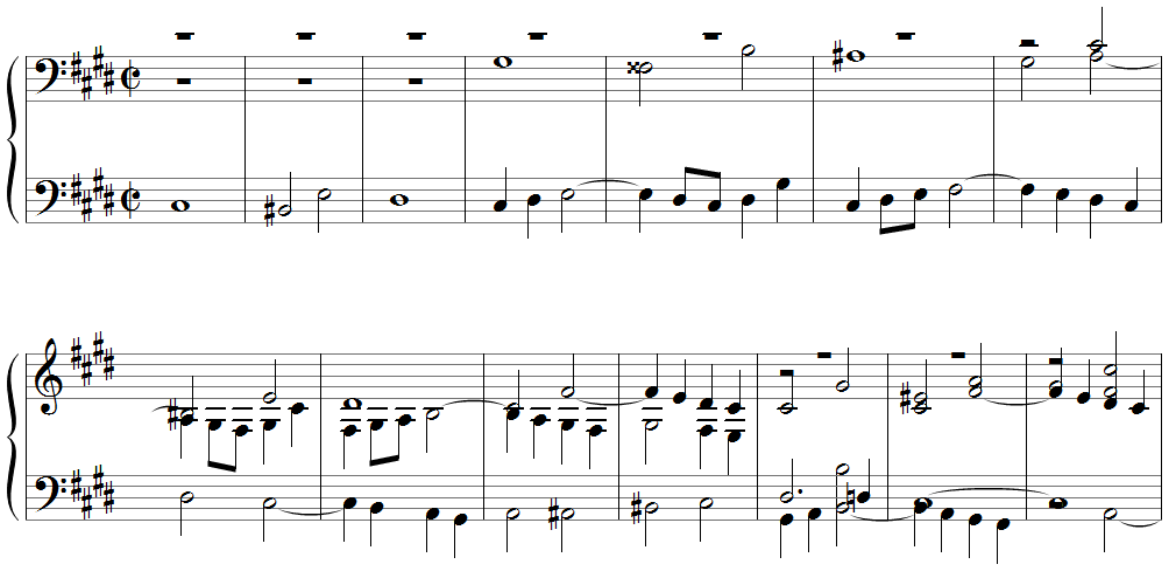
Einer der bekanntesten Komponisten von Fugen ist J. S. Bach, die folgende Abbildung zeigt den Anfang einer fünfstimmigen Fuge aus dem Werkzyklus „Das wohltemperierte Klavier Band I“.

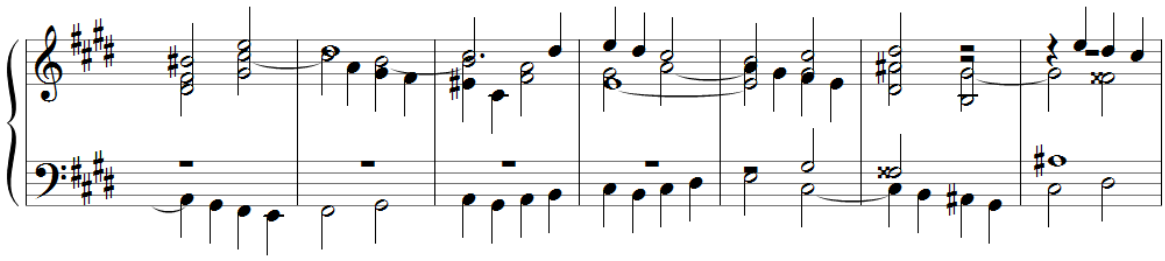


Notenbeispiel III-34: Bach, Fuge: Takte 1-21

Das dreitaktige Thema wird zunächst in der untersten Stimme präsentiert, danach setzen die anderen Stimmen bis Takt 12 sukzessive mit Transpositionen des Themas über den anderen Stimmen ein.

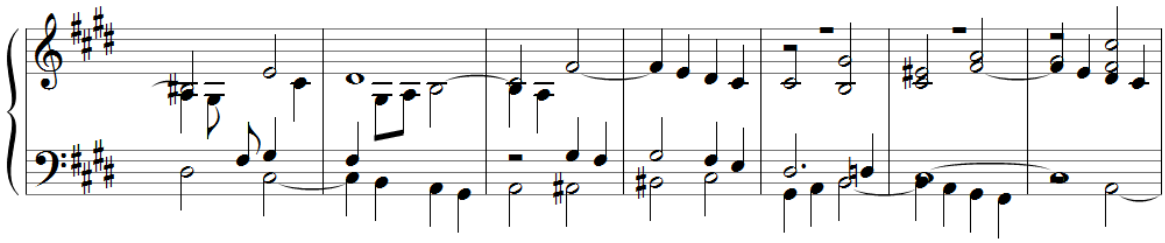
Wie bei Schumann können die beiden Notenzeilen nicht überall als Zuordnung zu den Händen interpretiert werden, da zu große Intervalle entstehen würden. So kann man abschnittsweise eine feste Zuordnung der Stimmen zu den Notenzeilen erkennen. Die Aufteilung nach Händen sieht nach der Analyse mit Algorithmus IX folgendermaßen aus:





Notenbeispiel III-35: Bach, Fuge: Takte 1-21 mit Algorithmus IX

Die erste Stimme wird der linken Hand zugeordnet, ab Takt 4 kommt die zweite Stimme in der rechten Hand hinzu, auch die dritte Stimme wird bis Takt 11 durchgehend in die rechte Hand gesetzt, danach wechselt sie in die linke. Die folgenden Takte, in denen vier bis fünf Stimmen gleichzeitig gespielt werden, lassen nur eine Aufteilung zu, da alle anderen nicht gespielt werden könnten.
Für die letzte abgebildete Zeile könnte daher auch eine nicht stimmbezogene Analyse unter Einbezug von Algorithmus I stattfinden, gute Ergebnisse finden sich bei den Algorithmen III, VII und auch X. Die verbleibenden Algorithmen erzeugen teilweise weniger gute Notenbilder; das folgende Beispiel zeigt Zeile 2 aus obiger Abbildung, die Analyse wurde hier aber mit Algorithmus II durchgeführt:



Notenbeispiel III-36: Bach, Fuge: Takte 8-14 mit Algorithmus II

In den ersten Takten des Beispiels wechselt die Mittelstimme häufig die Hand, was die Lesbarkeit des Notenbildes negativ beeinträchtigt und auch beim Spielen schwerer zu einem zusammenhängenden Klang geführt werden kann.

Auch auf die Sinfonia von Bach können die Ergebnisse übertragen werden. Die Analyse sollte mit den Algorithmen IX, III oder VII ausgeführt werden, um eine gut lesbare Aufteilung zu erhalten.
Die Allmand aus der Suite von Purcell ist zwar mehrstimmig, die Anzahl der Stimmen wechselt jedoch weit häufiger als z.B. bei der Fuge von Bach. Das Werk kann aber

zum überwiegenden Teil von allen Algorithmen gut aufgeteilt werden, da durchgehend große Intervalle zu finden sind.

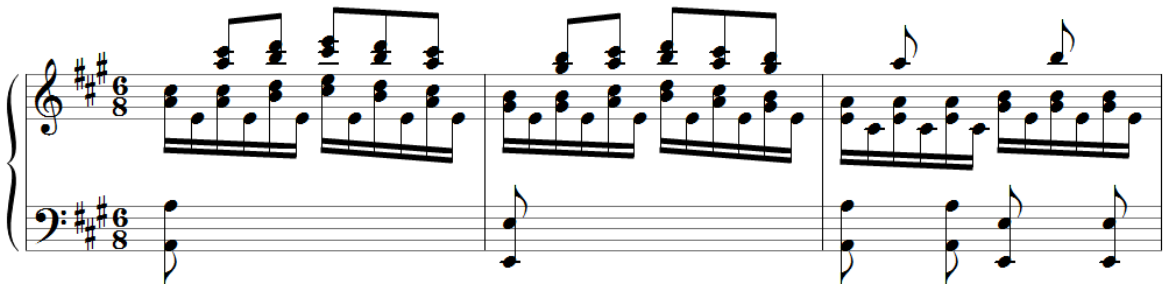
3.4 Übergreifen

In diesem Abschnitt werde ich zeigen, inwieweit – im Rahmen des in Teil I Gesagten – Übergreifen erkannt und daher zugewiesen werden kann. Für die Analysen wird jeweils Algorithmus X verwendet, welcher Übergreifen zuweist, wenn die Lage einer Hand dadurch beibehalten werden kann.

Von den oben genannten Teststücken weisen drei Stellen auf, an denen Übergreifen vorgeschrieben wird. Namentlich handelt es sich hierbei um die Variation 4 aus Mozarts Sonate, den Mittelteil aus der Ballade von Brahms und einzelne Takte aus Bartóks Bärentanz.

Eine der betreffenden Stellen des letzten Werkes ist auf Seite 75 abgebildet. In den Takten 3 und 4 ist im Originaltext Übergreifen vorgeschrieben. Dies wird durch Algorithmus X nicht erkannt, da zuvor in der rechten Hand noch nichts zu spielen und die Lage daher unbekannt war, und daher hier auf einen Sekundäralgorithmus zurückgegriffen wird. Aber auch an den weiteren – sehr ähnlichen – Stellen in diesem Werk, an denen die Lage beider Hände bekannt ist, wird kein Übergreifen zugewiesen, da die von der rechten Hand zu spielenden tiefen Töne auch jeweils innerhalb der Lage der linken Hand liegen.

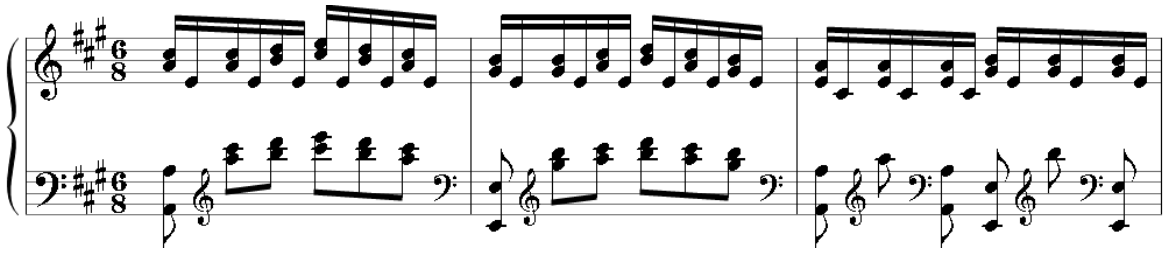
Der Anfang der erwähnten Variation der Mozart-Sonate sieht im Originaltext so aus:



Notenbeispiel III-37: Mozart, Sonate KV300i Variation 4: Takte 1-3

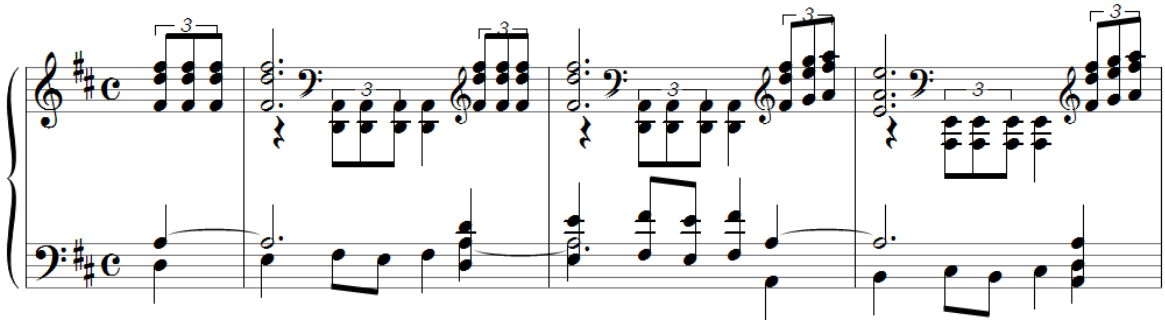
Es kann aufgrund der Schreibweise davon ausgegangen werden, dass die hohen Achtelnoten von der linken Hand gespielt werden sollen. Hier kann die Lage gleich beim ersten Schlag des ersten Taktes ermittelt werden; die rechte Hand kann ihre Lage behalten, wenn die linke für die höheren Töne übergreift. Daher weist

Algorithmus X hier auch tatsächlich Übergreifen zu. An vielen – aber nicht an allen – Stellen dieser Variation, in der fast durchgehend die linke Hand abwechselnd tiefere und höhere Töne zu spielen hat als die rechte, kann der Algorithmus dieses Übergreifen erkennen. Die ersten Takte nach der Analyse sind nachfolgend abgebildet.



Notenbeispiel III-38: Mozart, Sonate KV300i Variation 4: Takte 1-3 mit Algorithmus X

Das dritte Beispiel ist etwas schwieriger zu bewerten. Hier zunächst einige Takte des Originaltextes:

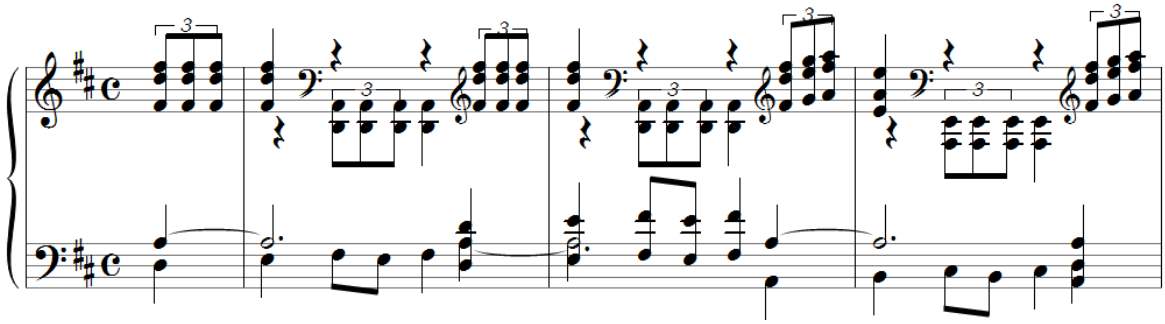


Notenbeispiel III-39: Brahms, Ballade: Takte 27-39 (mit Auftakt) – Originaltext

Das Geschriebene kann leider nicht exakt so gespielt werden wie es notiert ist. In der oberen Notenzeile steht am Beginn jeden Taktes ein Akkord in der Länge einer punktierten halben Note, auf die zweite Viertel soll aber offensichtlich die rechte Hand mehrere Oktaven tiefere Akkorde spielen, die auch nicht von der linken Hand übernommen werden können, da diese selbst in einer anderen Lage zu spielen hat. Das Aushalten der Akkorde auf den ersten Vierteln kann also nicht durch tatsächliches liegen bleiben der Hand auf der Klaviatur erfolgen, sondern etwa durch

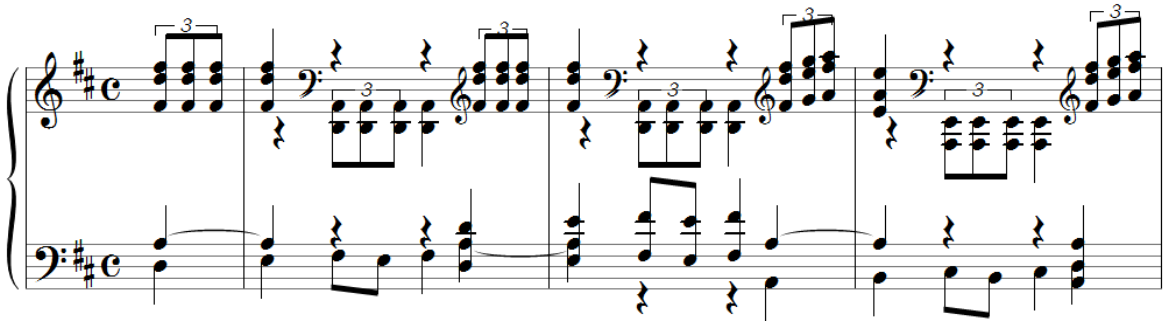
Betätigen des Haltepedals.

Eine Einspielung könnte daher etwa so aussehen:



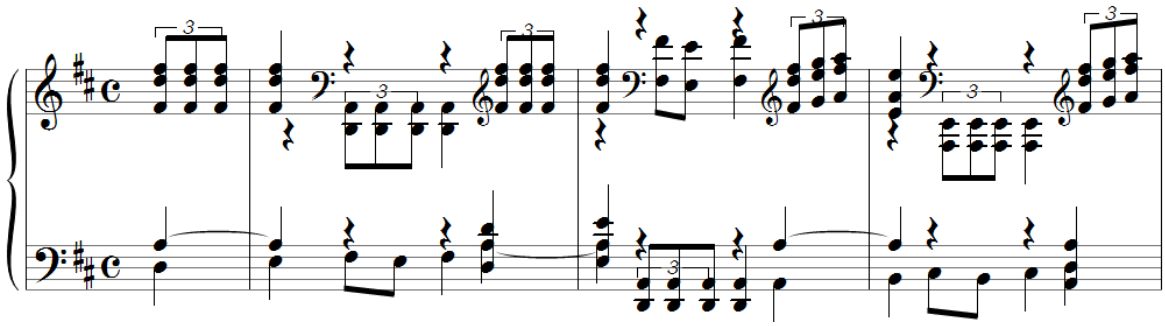
Notenbeispiel III-40: Brahms, Ballade: Takte 27-29 – Einspielung

Der Partiturausschnitt könnte nun prinzipiell gespielt werden, er erschwert die Analyse allerdings nach wie vor, da jeweils auf die zweiten Vierteln Übergreifen oder aber das Wechseln der Hand auf einem Ton zwingend erforderlich ist (jeweils das *a*) – andernfalls können nicht alle Noten gegriffen werden. Da beide Spezialfälle in keinem Algorithmus aktiv berücksichtigt werden, gehe ich von einer noch einmal geänderten Einspielung aus, in der prinzipiell weder Übergreifen noch Wechseln der Hand auf einem Ton notwendig ist:



Notenbeispiel III-41: Brahms, Ballade: Takte 27-29 – geänderte Einspielung

An den erwähnten Stellen im ersten und dritten Takt wird nun das *a* jeweils nicht über die erste Viertel hinaus ausgehalten. Jeder der Algorithmen ist nun in der Lage das Werk zu analysieren, und jeder außer X wird hier kein Übergreifen zuweisen. Algorithmus X weist in der oben beschriebenen Konfiguration teilweise Übergreifen zu, teilweise jedoch nicht, wie die folgende Abbildung veranschaulicht.



Notenbeispiel III-42: Brahms, Ballade: Takte 27-29 mit Algorithmus X

In den Takten 27 und 29 werden die tiefen Quinten tatsächlich der rechten Hand zugewiesen, weil die linke dadurch ihre Lage behalten kann, in Takt 28 hingegen müsste auch die linke Hand ihre Lage ändern (wenn auch nur um einen Ganzton) und es wird daher kein Übergreifen zugewiesen.

Die Beispiele zeigen, dass es prinzipiell möglich ist, unter geeigneten Randbedingungen, Übergreifen der Hände zu erkennen, bzw. zuzuweisen. Da Übergreifen vor allem einen optischen, nicht einen akustischen, Effekt erzeugt, ist es für die Erzeugung einer Partitur nicht als hochpriores Kriterium anzusehen.
Der Einsatz eines Algorithmus‘ zur Erkennung des Übergreifens ist mit Vorsicht zu tätigen, da es nicht auszuschließen ist, dass Übergreifen auch dort zugeordnet wird, wo es unerwünscht ist. So weist Algorithmus X etwa die ersten Noten der Melodiestimme bei Saties Gymnopédie der linken Hand zu, während die rechte den (tieferen) Begleitakkord zu spielen hat.

IV. Der Weg zu einer gesamtheitlichen Lösung – Resümee und Ausblick

Ich habe in dieser Arbeit die technischen und musikalischen Grundlagen erarbeitet, die notwendig für die Lösung des Problems der Trennung der Hände (bzw. Stimmen) einer Einspielung eines Klavierwerkes sind. Anhand verschiedener logischer Ansätze habe ich Algorithmen entwickelt und hier vorgestellt. Zuvor festgelegte Werke der „klassischen“ Klavierliteratur des 17. – 20. Jahrhunderts bildeten einen Satz an Testobjekten, aus welchen ich Ergebnisse der Analyse mit den präsentierten Algorithmen beschrieben und abgebildet habe.

In diesem letzten Teil der Arbeit möchte ich – gleichermaßen als Zusammenfassung – Thesen über die von mir entwickelten Algorithmen und die dargebotenen Testergebnisse formulieren. Danach werde ich die Arbeit mit einem Ausblick auf eine mögliche Fortführung der Bewältigung der Problematik schließen.

1 Thesen

Die Analyse von Klavierwerken unterschiedlicher Art ist prinzipiell mit allen Algorithmen möglich.

Bei geeigneter Konfiguration (beispielsweise von Maximalintervallen) und unter Berücksichtigung der jeweiligen Verwendungsziele können größte Teile der meisten Klavierwerke zu zwei Händen analysiert werden und so eine Trennung der beiden Hände erwirken.

Die Komplexität eines Algorithmus bestimmt nicht dessen Qualität.

Diese Aussage gilt prinzipiell für Algorithmen jeglicher Art, dennoch finde ich sie in diesem Zusammenhang erwähnenswert. Als Beispiel möge der oben beschriebene Algorithmus X dienen, welcher bedeutend komplexer als alle vertikalen Algorithmen ist. Die Zuordnung ist in vielen Fällen aber weniger zufriedenstellend, als die mit einem der letztgenannten Rechenvorschriften.

Horizontale Algorithmen sind leistungsfähiger als vertikale.

Die Testergebnisse haben gezeigt, dass horizontale Algorithmen in vielen

Situationen Vorteile gegenüber den vertikalen Algorithmen aufweisen. Exemplarisch seien die Vermeidung von Sprüngen, die stimmbezogene Analyse und die Zuordnung einzelner Noten genannt.
Diese Aussage steht nicht im Widerspruch zur zuletzt beschriebenen These, da die horizontale Analyse einerseits nicht per se komplexer ist als die vertikale, und andererseits nicht zwangsläufig bessere Ergebnisse liefern muss.

Unterschiedliche Werke erfordern unterschiedliche Lösungsansätze
Die im letzten Teil präsentierten Testergebnisse haben klar gezeigt, dass unterschiedliche Werke (oder Abschnitte von Werken) auch verschiedene Anforderungen an die Zuordnungsalgorithmen haben.
Insbesondere konnte dargelegt werden, dass akkordische Stellen und streng mehrstimmige Werke bestimmte Ansätze fordern. Der dritte angeführte Fall von Werken, welche aus Meldodie- und ihr untergeordneten Begleitstimmen bestehen, lässt sich weniger leicht zusammenfassen, viele Werke können aber auf einen der anderen beiden Fälle zurückgeführt werden.

2 Der Weg zu einer gesamtheitlichen Lösung

Viele Beispiele zeigen, dass die Algorithmen prinzipiell einsetzbar sind und vielfach gute Ergebnisse bei der Aufteilung der Hände liefern. Für einen Einsatz in Musiksoftware sind allerdings noch einige Schritte in der Entwicklung durchzuführen. Es gibt natürlich nicht nur einen Weg zu diesem Ziel, im Folgenden beschreibe ich mögliche Wegpunkte, die Problematik weiter zu behandeln.

Erweiterung

Der Satz an Algorithmen kann – unter Beibehaltung des Schemas der sequentiellen Analyse von einzelnen *NoteSets* – um neue erweitert werden.
Unter den nahezu unbegrenzten Möglichkeiten sei auf die Berücksichtigung der erwähnten Spezialfälle und auf den Miteinbezug absoluter Zeitbezüge besonders verwiesen.

Verfeinerung

Die gezeigten Algorithmen können weiter verfeinert werden. Insbesondere sollten die bis dahin vertikalen Algorithmen zur Vermeidung von Sprüngen auch das dem *currentNoteSet* vorangehende *NoteSet* berücksichtigen.

Test

Um zuverlässigere Aussagen über die Einsetzbarkeit und Qualität von Algorithmen machen zu können, muss eine große Zahl an Werken damit getestet werden.
Da nicht immer eine Zuordnung wie im Originalnotentext erzielt werden kann, müssen Testergebnisse aber qualitativ bewertet werden, und eine automatisierte Bewertung derselben ist daher schwer möglich.

Synthese

Den vielleicht wesentlichsten Aspekt der Fortführung der Thematik stellt die Verknüpfung der unterschiedlichen Ansätze dar. Es ist zwar auch nicht von der Hand zu weisen, dem Benutzer eines Sequencer-Programms die Wahl über den Algorithmus zur Trennung zweier Tracks einer Klavieraufnahme zu lassen, der für den Anwender komfortablere Weg beinhaltet aber einen Meta-Algorithmus, welcher entscheidet, wonach die Trennung erfolgen soll.
Hier drängt sich die Frage auf, ob so ein Meta-Algorithmus entwickelt werden kann. Meiner Ansicht nach zeigen die präsentierten Testergebnisse bereits, dass aufgrund des Notenmaterials von Abschnitten eines Werkes oftmals bereits Aussagen darüber getroffen werden können, welcher Algorithmus für optimale Ergebnisse bevorzugt zu wählen ist. Durch weiteres Testen sollten die Zuordnungen von Abschnittsarten zu Algorithmen noch klarer hervorgehen, und die Entscheidung programmatisch getroffen werden können.

Literatur- und Quellenverzeichnis

1 Literatur

[Be65] Hans von Besele: *Das Klavierspiel. Musikalische und technische Hinweise für künstlerisches Üben.* Bärenreiter-Verlag, 2. Auflage, 1965

[Fi94] Ludwig Finscher (Hg): *Die Musik in Geschichte und Gegenwart, Allgemeine Enzyklopädie der Musik, begründet von Friedrich Blume*. *Sachteil 1 A-Bog.* Bärenreiter Verlag, 1994

[Go89] Peter Gorges, Alex Merck: *Keyboards, MIDI, Homerecording*. GC Gunther Carstensen Verlag, München, 1989

[Go02] Peter Gorges:*Audio:Midi:MP3, Einführung in die digitale Musikwelt*. Voggenreiter Verlag, Bonn, 2002

[Hu99] David Miles Huber: *The MIDI Manual, A Practical Guide to the MIDI in the Project Studio.* Butterworth-Heinemann , 2nd Edition, 1999

[Ne97] Erich Neuwirth: Musikalische Stimmungen. Springer-Verlag, 1997

[Ja01] Andreas Jaschinski (Hg): *Notation*. Bärenreiter-Verlag, 2001

[Mi98] Ulrich Michels: *dtv-Atlas Musik, Band 1 Systematischer Teil, Musikgeschichte von den Anfängen bis zur Renaissance*. Deutscher Taschenbuch Verlag, 18. Auflage, 1998

[Ro02] Thomas D. Rossing, F. Richard Moore, Paul A. Wheeler: *The science of sound*. Addison Wesley. 3rd Edition, 2002

[Se05] Ertuğrul Sevsay: *Handbuch der Instrumentationspraxis*. Bärenreiter-Verlag, 2005

[Si00] Peer Sitter: *Computergestützte Arbeitsmethoden in der Musikwissenschaft.* Universitätsverlag Rasch, Osnabrück, 1. Auflage, 2000

[Tü89] Daniel Gottlob Türk: *Klavierschule*. Faksimile-Nachdruck der 1. Ausgabe 1789, herausgegeben von Erwin R. Jacobi, Bärenreiter-Verlag, 2. Auflage, 1967

[Vi91] Albert C. Vinci: *Die Notenschrift, Grundlagen der traditionellen Musiknotation*. Bärenreiter-Verlag, 2. Auflage, 1991

2 Noten

Gábor Csalog (Hg): *Frédéric Chopin: Etudes für Klavier (URTEXT)*. Könemann Music Budapest, 1995, daraus:
Etude op.10 Nr.1 in C-Dur

Jan Ekier (Hg): *Frédéric Chopin: Nocturnes*. Wiener Urtext Edition, Schott/Universal Edition, 6. Auflage, 1980, daraus:
Nocturne op.37 in g-Moll

Karl Heinz Füssl, Heinz Scholz (Hg): *Wolfgang Amadeus Mozart: Klaviersonaten, Band 2*. Wiener Urtext Edition, Schott/Universal Edition, 8. Auflage, 1973, daraus:
Sonate in A-Dur KV300i (331)

Ernst-Günter Heinemann: *Johann Sebastian Bach: Das Wohltemperierte Klavier Teil I*. G. Henle Verlag München, 1997, daraus:
Fuge Nr.4 in cis-Moll BWV849

Kurt Hermann (Hg): *Klaviermusik des 17. und 18. Jahrhunderts Band II*. Verlag Gebrüder Hug & Co., Leipzig, Zürich, 1934, daraus:
G. P. Telemann: Fantasia

Christa Jost (Hg): *Felix Mendelssohn Bartholdy: Lieder ohne Worte*. Wiener Urtext Edition, Schott/Universal Edition, 1. Auflage, 2001, daraus:
Lied ohne Worte op.19b Nr.1

Hans Kann (Hg): *Robert Schumann: Album für die Jugend op.68*. Wiener Urtext Edition, Schott/Universal Edition, 5. Auflage, 1979, daraus:
Kleine Fuge (Nr.39)

Rozsnyai Károly (Hg): *Béla Bartók: Zehn leichte Klavierstücke*. Editio Musica Budapest, 1950, daraus:
Nr.1 Bauernlied, Nr.10 Bärentanz

Eberhardt Klemm (Hg): *Erik Satie: Klavierwerke Band I (URTEXT)*. C. F. Peters Verlag, 1986, daraus:
1ère Gymnopédie

Hans Kohlhase (Hg): *Johannes Brahms: Balladen Op.10 für Klavier*. Wiener Urtext Edition, Schott/Universal Edition, 1. Auflage, 1990, daraus:
Ballade Nr.1

Ágnes Lakos (Hg): *Introduction to Polyphonic Playing*. Könemann Music Budapest, 1995, daraus:
Johann Krieger: Fughetta (Choral: Christum wir sollen loben schon)

Christa Landon, Walther Dürr (Hg): *Schubert: Impromptus für Klavier (URTEXT)*. Bärenreiter Verlag, 1984, daraus:
Impromptu D899 (op.90) Nr.1

István Máriássy (Hg): *Henry Purcell: Complete Piano Works (URTEXT)*. Könemann Music Budapest, 2000, daraus:
Suite in d-Moll Z668, Almand

Carl Adolf Matienssen (Hg): *Joseph Haydn: Sonaten für Klavier zu 2 Händen (Auswahl)*. Edition Peters, Leipzig (Nr. 4543), daraus:
Sonate in e-Moll

Sergei Merkulov (Hg): *Modest Musorgsky: Sämtliche Werke für Klavier (URTEXT)*. Könemann Music Budapest, 1996, daraus:
Bilder einer Ausstellung

Hans Sikorski (Hg): *Dmitri Schostakowitsch: Drei phantastische Tänze für Klavier op.5*. Musikverlag Hans Sikorski Hamburg, 1960, daraus:
Fantastischer Tanz op.5 Nr.2

Einar Stehen-Nökleberg, Ernst-Günter Heinemann (Hg): *Edvard Grieg: Klaviersonate in e-Moll op.7.* G. Henle Verlag München, 1995

Rudolf Steglich (Hg): *J. S. Bach: Inventionen, Sinfonien (URTEXT)*. G. Henle Verlag München, 1978, daraus:
Sinfonia Nr.11

Robert Threlfall (Hg): *Serge Rachmaninoff: Preludes for piano Authentic Edition*, Boosey & Hawkes Music Publishers Limited, 1992, daraus: *Prelude op.3 No.2*

Bertha Antonia Wallner (Hg): *Ludwig van Beethoven: Klaviersonaten Band I*. G. Henle Verlag München, 1980, daraus: *Sonate Nr.5 in c-Moll op.10 Nr.1*

Anhang

1 Algorithmen

1.1 Grundstruktur Algorithmen

```
class SplittingAlgorithm {

 method correct() {

  Note-List problematicNotes;

  for( all notes in currentNoteSet ) {

   if( note.isRight() && (currentNoteSet.getHighestNote().getPitch() -
   note.getPitch() > MAX_INTERVAL) (or vice-versa for left notes) ) {

    if( correction not yet tried for this note ) {

     problematicNotes.add(note);

    }

   }

  }

  if this list is empty {

   break up correction;

  }

  int minStarttime = problematicNotes.get(0).getStarttime();

  for( all notes in problematicNotes ) {

   starttime = minimum( starttime, notes.getStarttime() );

   set note to other hand;

  }

  //going back in time to proove changes
  while ( currentTime > minimumStarttime ) {

   set all Notes in currentNoteSet with (starttime >= currentTime) unknown;

   notepool.backward();

   if( left or right hand still not playable ) {
```

```
     set all Notes in currentNoteSet with starttime >= currentTime unknown;
     notepool.backward();
    }
  }
  mark Notes in problematicNotes `yet corrected´;
 }
}
```

1.2 Vertikale Algorithmen

Gemeinsames Schema

Diese Methode wird für alle *NoteSets* aufgerufen.

```
class VerticalSplittingAlgorithm {
 method verticalSplittingAlgorithm( NoteSet currentNoteSet ) {
  if( currentNoteSet.size() < 2 ) {
   analyze with singlenotes algorithm;
   break;
  }
  call splittingAlgorithmI(currentNoteSet);
  if( currentNoteSet.hasUnknownNotes() ) {
   analyze currentNoteSet with primary vertical splitting algorithm;
  }
  if( left or right Notes not playable ) {
   correct();
  }
 }
}
```

Algorithmus I

```
method SplittingAlgorithmI(NoteSet currentNoteSet) {
```

```
 Note lowestnote = currentNoteSet.getLowestNote();
 Note highestnote= currentNoteSet.getHighestNote();

 lowestnote.setLeft(TRUE);
 highestnote.setRight(TRUE);

 for( all other Notes in currentNoteSet ) {

  if( ! note.isUnknown() ) {

   continue with next note;

  }

  if( (highestnote.getPitch() – note.getPitch() > MAX_INTERVAL) &&
  (note.getPitch() – lowestnote.getPitch() > MAX_INTERVAL) {

   //Illegal Interval
   break up;

  }
  else if( highestnote.getPitch() – note.getPitch() > MAX_INTERVAL ) {

   note.setLeft(TRUE);

  }
  else if( note.getPitch() – lowestnote.getPitch() > MAX_INTERVAL ) {

   note.setRight(TRUE);

  }

 }

}
```

Algorithmus II

```
method splittingAlgorithmII( NoteSet currentNoteSet ) {

 int differenceToLowestNote;
 int differenceToHighestNote;

 if( currentNoteSet.getHighestNote().isUnknown() ) {

  currentNoteSet.getHighestNote().setRight(TRUE);

 }
 if( currentNoteSet.getLowestNote().isUnknown() ) {

  currentNoteSet.getLowestNote().setLeft(TRUE);
```

```
 }
 for( all other unkown Notes in currentNoteSet ) {
  differenceToLowestNote = note.getPitch() -
  currentNoteSet.getLowestNote().getPitch();
  differenceToHighestNote = currentNoteSet.getHighestNote().getPitch() -
  note.getPitch();
  for(int j=note.getPitch(); j<noteSet.getHighestNote().getPitch(); j++) {
   if( (pitch % 12 == 4) || (pitch % 12 == 11) ) {
    differencetohighest++;
   }
  }
  for(int j=note.getLowestNote().getPitch(); j<note.getPitch(); j++) {
   if( (pitch % 12 == 4) || (pitch % 12 == 11) ) {
    differencetolowest++;
   }
  }
  if( differenceToLowestNote < differenceToHighestNote ) {
   note.setLeft(TRUE);
  }
  else {
   note.setRight(TRUE);
  }
 }
}
```

Algorithmus III

```
method splittingAlgorithmIII( NoteSet currentNoteSet ) {
 int countLeft = 0;
 int countRight = 0;
```

```
 if( currentNoteSet.getHighestNote().isUnknown() ) {
  currentNoteSet.getHighestNote().setRight(TRUE);
 }
 if( currentNoteSet.getLowestNote().isUnknown() ) {
  currentNoteSet.getLowestNote().setLeft(TRUE);
 }
 for( all Notes in currentNoteSet ) {
  if( note.isLeft() ) {
   countLeft ++;
  }
  else if( note.isRight() ) {
   countRight ++;
  }
 }
 for( all unknown Notes in currentNoteSet, outer to inner ) {
  if( countRight <= countLeft ) {
   note.setRight( TRUE );
  }
  else {
   note.setLeft( TRUE );
  }
 }
}
```

Algorithmus IV

```
method splittingAlgorithmIV( NoteSet currentNoteSet ) {
 sort Notes by pitch;
 NoteSet tempNoteSet = new NoteSet();
 add all yet decided Notes to tempNoteSet;
```

```
 boolean isLeft = FALSE;
 boolean isRight = FALSE;

 for( all Notes in currentNoteSet, outer to inner ) {

  if( ! note.isUnknown() ) { continue with next Note; }
  else {

   tempNoteSet.add( note );

  }

  isLeft = FALSE;
  isRight = FALSE;

  for( all yet decided Notes as tempNote ) {

   if(starttime & rhythmValue of note exactly match the values of tempNote)
   {

    set note to same Hand as tempNote;

   }

  }

  if( either only isLeft or only isRight ) {

   set note to the corresponding hand;;

  }
  else {

   decide tempNoteSet with secondary algorithm;

  }

 }

}
```

Algorithmus V

```
method splittingAlgorithmV( NoteSet currentNoteSet ) {

 sort Notes first by starttime, then by rhythmValue;

 NoteSet-List rhythmSets = List of NoteSets where every NoteSet contains
 only Notes of the same starttime and rhythmValue, the List is sorted by
 starttime, then by rhythmValue;
```

```
 for( all rhythmSets ) {
  sort rhythmSet by pitch;
  if( rhythmSet has left but no right Notes (or vice-versa) ) {
   set all playable Notes to left (or respectively right);
  }
 }
 if( still no Notes decided ) {
  set highestNote to right and lowestNote to left;
 }
 for( all not yet decided rhythmSets ) {
  if( rhythmSetIndex > rhythmSets.size()/2 ) {
   if( rhythmSets.get(0).isLeft() ) {
    set all playable Notes to left;
   }
   else if( rhythmSets.get(0).isRight() ) {
    set all playable Notes to right;
   }
  }
  else {
   do vice-versa;
  }
 }


 if(currentNoteSet.hasUnknownNotes()) {
  call secondary splitting algorithm;
 }
}
```

1.3 Horizontale Algorithmen

Algorithmus VI

```
const MAX_INTERVAL_SAME_HAND = 12;

method splittingAlgorithmVI( NoteSet recentNoteSet, NoteSet currentNoteSet)
{

 if( recentNoteSet.getLeftNotes().size() == 0 ) {

  if( (currentNoteSet.getHighestNote().getPitch() >
  recentNoteSet.getHighestNote().getPitch()) ||
  (recentNoteSet.getHighestNote().getPitch() -
  currentNoteSet.getHighestNote().getPitch() < MAX_INTERVAL_SAME_HAND) ) {

   currentNoteSet.getHighestNote().setRight(TRUE);

  }
  else {

   currentNoteSet.getLowestNote().setLeft(TRUE);

  }

 }
 else if( recentNoteSet.getRightNotes().size() == 0 ) {

  do vice-versa;

 }

 else {

  if( (abs(currentNoteSet.getLowestNote().getPitch() -
  recentNoteSet.getLeftNotes().getHighestNote().getPitch()) <=
  MAX_INTERVAL_SAME_HAND) &&
  (abs(currentNoteSet.getHighestNote().getPitch() -
  recentNoteSet.getRightNotes().getLowestNote().getPitch()) >
  MAX_INTERVAL_SAME_HAND ) )  {

   currentNoteSet.setLeft(TRUE);

  }
  else if( (abs(currentNoteSet.getLowestNote().getPitch() -
  recentNoteSet.getLeftNotes().getHighestNote().getPitch()) >
  MAX_INTERVAL_SAME_HAND) &&
```

```
  (Math.abs(currentNoteSet.getHighestNote().getPitch() -
  recentNoteSet.getRightNotes().getLowestNote().getPitch()) <=
  MAX_INTERVAL_SAME_HAND ) )  {

   currentNoteSet.setRight(TRUE);

  }

 }

}
```

Algorithmus VII

```
method splittingAlgorithmVII(NoteSet recentNoteSet, NoteSet currentNoteSet)
{

 sort Notes by pitch;

 int leftNotes = recentNoteSet.getLeftNotes().size();
 int rightNotes = recentNoteSet.getRightNotes().size();

 for( max recentNoteSet.size() Notes in currentNoteSet, outer to innter) {

  if( !note.isUnknown() ) { continue with next Note; }

  if( Note is from lower pitch’ed Notes ) {

   if(currentNoteSet.getLeftNotes().size()<leftNotes) {

    note.setLeft(TRUE);

   }
   else {

    note.setRight(TRUE);

   }

  }

  else (Note is from upper pitch’ed) {

   if(currentNoteSet.getRightNotes().size()<rightNotes) {

    note.setRight(TRUE);

   }
   else {

    note.setLeft(TRUE);

   }
```

```
  }

 }

 call secondary splitting algorithm(currentNoteSet);

}
```

Algorithmus VIII

```
method splittingAlgorithmVIII(NoteSet recentNoteSet, NoteSet
currentNoteSet) {

 if( recentNoteSet.isEmpty() ) {

  decide by pitch;

 }
 else if( recentNoteSet.size() == 1 ) {

  if( difference of the pitches in recent and currentNoteSet <
  MAX_INTERVAL) {

   same hand;

  }
  else {

   other hand;

  }

 }
 else {

  if( recentNoteSet had only left or only right Notes ) {

   if the pitch of the currentNote is in
   [recentNoteSet.getLowestNote().getPitch()-MAX_INTERVAL;
   recentNoteSet.getHighestNote().getPitch()+MAX_INTERVAL] assign
   currentNote to the same hand as the recentNoteSet;

  }
  else {

   if( pitch of currentNote > recentNoteSet.getHighestNote.getPitch() ) {

    assign to same hand as recentNoteSet.getHighestNote();

   }
   analog for recentNoteSet.getLowestNote();
```

```
    else if( pitch of currentNote in
    [recentNoteSet.getLeftNotes().getLowestNote().getPitch();
    recentNoteSet.getLeftNotes().getHighestNote().getPitch()] {

     assign left;

    }
    vice-versa for recentNoteSet.getRightNotes();

    else {

     decide by minimal difference of currentPitch and the lowest/highest
     Note of the recentNoteSet;

    }

   }

 }

}
```

Algorithmus IX

```
method analyze() {

  List of NoteSequences currentSequences;
  int voices;

  while( notepool.hastNextNoteSet() ) {

    save the currentSequences to another List recentSequences (if
    currentSequences not empty);

     voices = currentNoteSet.size();

     for(int i=0; i < voices; i++) {

       currentSequences.add(new NoteSequence());

     }

     while( currentNoteSet.size() == voices ) {

       sort currentNoteSet by pitch;

       add voice $i to currentSequences[i] (if not yet added);

       goForward();

     }

     splitting_algorithm( currentSequences, recentSequences );

   }
```

```
 }
}


method splittingAlgorithmIX( currentSequences, recentSequences) {
 if( currentSequences.size() > 1 ) {
  currentSequences.get(0).setLeft(TRUE);
  currentSequences.get(currentSequences.size()-1).setRight(TRUE);
 }
 else {
  decideLeftOrRight( currentSequences[0] );
 }
 for( int i=1; i < currentSequences.size()-1; i++ ) {
  decideLeftOrRight( currentSequences[i] );
 }
}


method decideLeftOrRight( NoteSequence seq ) {
 NoteSequence bestSequence = sequence, where average pitch is most similar
 to average pitch of this sequence;
 check how many NoteSets in this sequence can be played left;
 check how many NoteSets in this sequence can be played right;
 if( all can be played left and right ) {
  search the sequence in recentSequences where the average pitch is most
  similar to the average pitch of this sequence and assign this sequence to
  the same hand as the ending of the here found recent sequence;
 }
 else if( all can be played left ) {
      seq.setLeft(TRUE);
 }
 else if( all can be played right ) {
      seq.setRight(TRUE);
 }
```

```
 else  {

  if( more NoteSets can be played left than right (or vice-versa) ) {
      set that part to left (or vice-versa);
  }

  split the sequence at the last decided NoteSet into two parts;

  decide the second part like this full sequence recursively;

 }

}
```

Algorithmus X

```
Class SplittingAlgorithmX {

 const int MAX_INTERVAL = 12;
 const int MAX_INTERVAL_POSITION = 12;
 const int ABSOLUTE_MAX_INTERVAL = 16;

 private int position_lowestPitch_left = -1;
 private int position_lowestPitch_right = -1;
 private int position_highestPitch_left = -1;
 private int position_highestPitch_right = -1;
 private boolean overlap = FALSE;

 method analyze() {

  while(currentNoteSet has not left and right notes && nextNoteSet exists){

   goForward();

   analyze with seconary algorithms();

  }

  updatePositions( currentNoteSet );

  while( nextNoteSet exists ) {

   goForward();

   if( currentNoteSet.isEmpty() ) {

    continue with nextNoteSet;

   }

   sort currentNoteSet by pitch;

   analyze currentNoteSet with wide chords algorithm;
```

```
   splitting_algorithmX(currentNoteSet);
   if( currentNoteSet.hasUnknownNotes() ) {
    analyze with secondary algorithm(s);
   }
   if( recentNoteSet not playable left or right ) {
    goBackward();
    correct();
    goForward();
   }
   if( currentNoteSet not playable left or right ) {
    correct();
   }
   updatePositions( currentNoteSet );
  }
}


method splittingAlgorithmX( NoteSet currentNoteSet ) {
 boolean canHoldPosition_left = TRUE;
 boolean canHoldPosition_right = TRUE;
 if( wide chords algorithms has yet decided some notes ) {
  if( any of the left Notes not checkPitchInLeftPosition() ) {
   canHoldPosition_left = FALSE;
  }
  if( any of the right Notes not checkPitchInLeftPosition() ) {
   canHoldPosition_right = FALSE;
  }
  if( canHoldPosition_left && canHoldPosition_right ) {
   check each Note Left and right and if result is distinct set this note
   to left or right;
```

```
}
else if( canHoldPosition_left && !canHoldPosition_right ) {
 if(overlap) {
  changeHands(currentNoteSet);
 }
 overlap = FALSE;
 for(int i=0; i < currentNoteSet.size(); i++) {
  if( checkPitchInLeftPosition(current pitch, lowPitch, highPitch) ) {
   currentNoteSet.get(i).setLeft(TRUE);
  } else {
   if( ! overlap ) {
    if(highestPitch – currentPitch <= MAX_INTERVAL) {
     currentNoteSet.get(i).setRight(TRUE);
    }
  } else {
   if( currentPitch - lowestPitch <= MAX_INTERVAL ) {
    currentNoteSet.get(i).setRight(TRUE);
   }
  }
 }
}
else if( !canHoldPosition_left && canHoldPosition_right ) {
  do vice-versa;
}
else if( overlapping allowed ) {
 boolean wasOverlapping = overlap;
 canHoldPosition_left = TRUE;
 canHoldPosition_right = TRUE;
 overlap = TRUE;
 for(all left Notes) {
  if(!checkPitchInRightPosition( currentPitch, lowPitch, highPitch)) {
```

```
    canHoldPosition_left = FALSE;
   }
  }
  do for right notes vice-versa;
  if( canHoldPosition_left || canHoldPosition_right ) {
   changeHands(currentNoteSet);
   for(int i=0; i < currentNoteSet.size(); i++) {
    if( currentNoteSet.get(i).isUnknown() ) {
     if(absolute(currentPitch – lowestPitch) < absoulte(currentPitch -
     highestNote) ) {
      currentNoteSet.get(i).setRight(TRUE);
     }
     else {
      currentNoteSet.get(i).setLeft(TRUE);
     }
    }
   }
  } else {
   overlap = FALSE;
   if(wasOverlapping) {
    changeHands(currentNoteSet);
   }
  }
 }
} else {
 if( any of the left Notes not checkPitchInLeftPosition() ) {
    canHoldPosition_left = FALSE;
 }
 do for canHoldPosition_right vice-versa;
 if( canHoldPosition_left && canHoldPosition_right ) {
```

```
    analyze with secondary algorithms;
   }
   else if( canHoldPosition_right (or left) ) {
    set currentNoteSet right (or left);
   }
  }
}


method updatePositions( NoteSet currentNoteSet ) {
 if(currentNoteSet.getRightNotes().size() > 0 ) {
  if( (currentNoteSet.getRightNotes().getHighestNote().getPitch() <=
  position_highestPitch_right) &&
  (currentNoteSet.getRightNotes().getLowestNote().getPitch() >=
  position_lowestPitch_right) ) {
   if( currentNoteSet.getRightNotes().getHighestNote().getPitch() -
   MAX_INTERVAL_POSITION > position_lowestPitch_right ) {
    position_lowestPitch_right =
    currentNoteSet.getRightNotes().getHighestNote().getPitch() -
    MAX_INTERVAL_POSITION;
   }
   if( currentNoteSet.getRightNotes().getLowestNote().getPitch() +
   MAX_INTERVAL_POSITION < position_highestPitch_right ) {
    position_highestPitch_right =
    currentNoteSet.getRightNotes().getLowestNote().getPitch() +
    MAX_INTERVAL_POSITION;
   }
  } else {
   position_lowestPitch_right =
   currentNoteSet.getRightNotes().getHighestNote().getPitch() -
   MAX_INTERVAL_POSITION;
   position_highestPitch_right =
   currentNoteSet.getRightNotes().getLowestNote().getPitch() +
   MAX_INTERVAL_POSITION;
   }
```

```
    }

    set position_highestPitch_left and position_lowestPitch_left vice-versa;

  }


  function boolean checkPitchInLeftPosition( int pitch, int lowRefPitch, int
  highRefPitch );

  function boolean checkPitchInRightPosition( int pitch, int lowRefPitch,
  int highRefPitch );

  method changeHands( NoteSet currentNoteSet );

}
```

2 Weiterführende Quellen

Zur Veranschaulichung der in dieser Arbeit präsentierten Algorithmen habe ich ein kleines Programm geschrieben, welches öffentlich zugänglich ist. Der werte Leser sei dazu aufgefordert, das Programm als Ergänzung zum Studium der schriftlichen Arbeit aufzufassen und dasselbe auszuprobieren.

Das Programm, welches als Java Web Start-Anwendung realisiert wurde, ist unter nachfolgendem Link im WWW verfügbar:

http://sunsite.univie.ac.at/musicfun/pianohandseparation

3 Lebenslauf

Persönliche Daten

Name	Harald Köhler
Geburtsdatum & -ort	04.04.1983, Wien
Familienstand	ledig, 1 Sohn
Staatsangehörigkeit	Österreich

Bildungsweg

1989 – 1993	Volksschule: Köhlergasse, 1180 Wien
1993 – 2001	AHS, Realgymnasium mit nw. Schwerpunkt: GRG XIX, Billrothstraße, 1190 Wien Abschluss mit Auszeichnung
Seit 2002	Universität Wien und Technische Universität Wien: Lehramtsstudium Informatik & Mathematik zweimaliges Leistungsstipendium
Seit 2005	Technische Universität Wien: Bakkalaureatsstudium Technische Informatik

Präsenzdienst: geleistet 2001

Mehrjährige einschlägige berufliche Erfahrung

in den Bereichen Systemadministration und Software-Engineering