KU LEUVEN
FACULTEIT
INGENIEURSWETENSCHAPPEN

Melodische transformatie en evalutie van muziek

Elias Moons

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen

Promotor:
Prof. dr. D. De Schreye

Assessor:
Prof. Onbekend

Begeleider:
Ir. V. Nys

Academiejaar 2015 – 2016

Voorwoord

TODO: iedereen bedanken...

Elias Moons

Inhoudsopgave

Samenvatting

Deze masterproef beschrijft methodes om melodielijnen van een muziekstuk melodisch te transformeren to nieuwe melodielijnen. Verder wordt er ook aandacht besteed aan een referentiekader waarin deze transformaties geëvalueerd kunnen worden. Tot slot wordt er ook nog gekeken naar wanneer bepaalde transformaties nuttig kunnen zijn om de consonantie van een muziekstuk te verhogen. Om dit te verwezenlijken is een algoritme beschreven dat gebaseerd is op de principes van *dynamic programming*. Dit algoritme zal, gegeven een aantal mogelijke transformaties en een melodielijn, de best mogelijke getransformeerde melodielijn teruggeven, dit volgens het gedefiniëerde referentiemodel.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen en symbolen

Afkortingen

LoG	Laplacian-of-Gaussian
MSE	Mean Square error
PSNR	Peak Signal-to-Noise ratio

Symbolen

42	“The Answer to the Ultimate Question of Life, the Universe, and Everything” volgens de [1]
c	Lichtsnelheid
E	Energie
m	Massa
π	Het getal pi

Hoofdstuk 1

Inleiding

1.1 Probleemstelling

Een van de meest voorkomende problemen voor muzikanten is de zogenaamde *writer's block*. Dit fenomeen waarbij je als muzikant merkt dat je de hele tijd op hetzelfde melodietje uitkomt en maar niet met iets nieuw kan komen kan zeer frustrerend zijn. Daarom zou het handig zijn als er een tool zou bestaan die je als het ware de inspiratie kan geven die je nodig hebt om deze *writer's block* te doorbreken. Een mogelijke oplossing voor dit probleem wordt in deze thesis onderzocht. Hier gaat het dan om het transformeren van reeds gekende melodieën tot nieuwe melodieën.

1.2 Vergelijking met voorgaand onderzoek

Onderzoek naar het bekomen van nieuwe melodieën waarbij gebruik gemaakt wordt van een computer is niet nieuw. Al gedurende tientallen jaren is er veel werk geleverd op het vlak van muziekgeneratie, waarbij het de bedoeling is om een muziekstuk te genereren waarbij vertrokken wordt van een lege partituur. Dit wordt gedaan rekening houdend met de regels uit de muziektheorie en vaak probeert men er ook een vorm van repetitiviteit en herkenning in te brengen zoals vaak ook het geval is in hedendaagse muziek. Een groot probleem dat echter altijd terugkwam was dat het heel moeilijk was om een goede verhouding te vinden tussen twee heel belangrijke eigenschappen van de muziek, namelijk verwachting en verrassing van de luisteraar. Een muziekluisteraar heeft namelijk een bepaald verwachtingspatroon voor het onmiddellijke vervolg van een melodie. Dit verwachtingspatroon strookt vaak met de regels van de muziektheorie. Wanneer deze regels dan weer te nauwgezet gevolgd worden, wordt het muziekstuk als saai ervaren, er is dus nood aan een zekere verrassing in de melodie. Te veel onverwachte wendingen worden dan weer als frustrerend gezien, kortom, het is van cruciaal belang om een goed evenwicht te vinden tussen deze twee waarden. En dit blijkt zeer moeilijk te verwezenlijken vanuit een muziek generatie-standpunt.

1.3 Doelstelling

In deze thesis wordt een onderzoek gevoerd naar de generatie van nieuwe melodieën vertrekkende van originele melodieën. Deze originele melodieën worden bij aanname verondersteld te voldoen aan de eisen van verrassing en verwachting. In deze thesis worden specifiek melodische transformaties onderzocht. Deze transformaties zullen gebaseerd zijn op wiskundige reeksen. Hierbij gaat men voor elke noot in de oorspronkelijke melodie de toonhoogte aanpassen volgens een bepaald patroon. Als voorbeeld om het concept te verduidelijken kan figuur 1.1 dienen. In deze figuur wordt met behulp van de rij van fibonacci een originele melodie getransformeerd tot een nieuwe. Er wordt getracht zo een transformatie te zoeken zodat de nieuw bekomen melodie nog steeds consonant is (nog steeds goed klinkt). Er zal ook onderzocht worden onder welke omstandigheden een transformatie al dan niet een consonante melodie oplevert. Om te bepalen of een nieuwe melodie goed klinkt is er ook nood aan een objectieve beoordeling van deze melodieën. Er wordt dus ook aandacht besteed aan het opstellen van een model dat hiervoor kan dienen. Een model dat zo goed mogelijk kan beoordelen of een gegeven melodie al dan niet, en in welke mate, consonant is. Het voordeel van het werken met transformaties in plaats van met generatie is dat men op deze manier nog een deel van de eigenheid van het originele werk kan behouden en het bekomen werk hierdoor ook minder artificieel zal overkomen bij de luisteraar.

FIGUUR 1.1: Melodische transformatie m.b.v. rij van Fibonacci: Noten op de bovenste notenbalk liggen respectievelijk 1,1,2,3,5,8 halve tonen hoger dan op de onderste notenbalk.

1.4 Assumpties en beperkingen

Op een onderdeel van hoofdstuk 3 na, behandelt deze masterproef enkel muziek met precies één melodielijn. Meerstemmige muziek waarbij meerdere noten tegelijkertijd gespeeld kunnen worden, wordt in deze thesis dus niet behandelt. Verder worden alle testen uitgevoerd op een corpus (Essen corpus [2]) bestaande uit muziekstukken van het 'folk'-genre. Tot slot zal ook telkens wanneer een transformatie uitgevoerd is, er vanuit gegaan worden dat het originele muziekstuk telkens voldoet aan de algemene voorwaarden waaraan een muziekstuk moet voldoen, zodat zijn score in het voorgestelde model telkens ook als referentie kan dienen voor zijn getransformeerde versie.

1.5 Overzicht van de tekst

In het eerstvolgende hoofdstuk, hoofdstuk 2 wordt een korte inleiding gegeven tot de muziektheorie. Hierin worden enkel deze elementen behandeld die relevant zijn voor de rest van deze thesis. Vervolgens zal er een besproken worden hoe een bepaald muziekstuk objectief kan beoordeeld worden, dit zal gebeuren in hoofdstuk 3. In hoofdstuk 4 worden verschillende melodische transformaties toegelicht en met elkaar vergeleken. Vervolgens zal hoofdstuk 5 twee algoritmes beschrijven. Deze algoritmes kunnen gebruikt worden om gegeven een aantal toegestane transformaties en een oorspronkelijke melodielijn, de meest waarschijnlijke getransformeerde melodielijn terug te geven. Hierna zullen een aantal experimenten, alsook hun resultaten besproken worden in hoofdstuk 6. Tot slot wordt er in hoofdstuk 7 nog teruggekeken op het geleverde onderzoek in een samenvattend besluit. Er wordt ook nog beschreven welk verder onderzoek zeker nog interessant zou kunnen zijn binnen dit onderwerp.

Hoofdstuk 2

Muzikale Achtergrond

Aangezien deze thesis zal handelen over melodische transformaties wordt hieronder in het kort info gegeven over elementaire begrippen rond ritme en melodie van een muziekstuk en hoe deze voorgesteld kunnen worden. Voor lezers met een voorkennis in de muziekwereld zal dit onderdeel waarschijnlijk redundant zijn en kan er bijgevolg ook meteen overgegaan worden naar het volgende hoofdstuk.

2.1 Ritme

In deze masterproef worden enkel melodische transformaties behandeld, waarbij het ritme ongewijzigd blijft. Toch is het zeker nuttig om ook een zekere voorkennis te hebben van de betekenis van ritme in een muziekstuk. Melodie en ritme van een muziekstuk gaan namelijk hand in hand. Een basiskennis van ritmische begrippen zal dus zeker ook nuttig zijn voor het begrijpen van bepaalde melodische fenomenen.

2.1.1 Tijdssignaturen

De tijdssignatuur geeft de ritmische structuur weer van het muziekstuk. Deze tijddssignatuur wordt weergegeven aan het begin van de notenbalk en ziet er uit als een breuk zonder streepje. Een voorbeeld hiervan is de vaak gebruikte $\frac{3}{4}$ (of 'drie vierden'). In deze voorstelling staat het onderste getal voor welke nootlengte overeen komt met een tel. Het bovenste getal geeft weer hoeveel tellen in een maat voorkomen. In het voorbeeld van de signatuur $\frac{3}{4}$ komt dit over met 4 tellen van lengte $\frac{1}{4}$. Voor de specifieke signatuur $\frac{4}{4}$ (of 'vier vierden') heeft men nog een andere notatie, namelijk de letter C. Deze letter komt het woord *common time*, aangezien deze signatuur zo typisch en veelvoorkomend is in moderne Westerse muziek. Figuur 2.1 geeft het gebruik van deze notatie weer. De figuur illustreert ook de 4 verschillende tellen van deze maatsoort.

2.1.2 Tempo

Een volgend belangrijk onderdeel dat het ritme van een muziekstuk bepaalt is het tempo. Het tempo van een stuk bepaalt namelijk de duur van de verschillende noten

Figuur 2.1: Maat met voortekening van *common time* tijdssignatuur.

in het muziekstuk. Het temp wordt meestal uitgedrukt in aantal tellen per minuut. De lengte van een tel zelf wordt dan weer bepaald door de tijdssignatuur. Zo betekent bijvoorbeeld een tempo van 60 tellen voor een muziekstuk met signatuur $\frac{3}{4}$ dat elke tel, ofwel elke kwartnoot, precies één seconde duurt.

2.1.3 Nootlengtes

De nootlengte is het laatste element dat de absolute duur van een noot zal bepalen. De nootlengte geeft de relatieve lengte van een noot weer ten opzichte van het tempo en de tijdssignatuur. De nootlengte wordt uitgedrukt door een breuk. Deze breuk heeft als relatieve lengte zijn verhouding tot de lengte van een tel. Zo zal een $\frac{1}{4}$-noot in $\frac{3}{4}$ één tel duren, en zal een $\frac{1}{8}$-noot in $\frac{3}{4}$ twee tellen duren.
Een note van hele lengte wordt aangeduid met een hol bolletje. Een noot van halve lengte wordt aangeduid met een half bolletje met een streep aan de rechterkant. Een $\frac{1}{4}$-noot wordt aangeduid zoals een halve noot maar dan met een vol bolletje. Een $\frac{1}{8}$-noot wordt voorgesteld door een vierde noot met een streepje vanboven. Vanaf een $\frac{1}{16}$-noot wordt er dan een streepje vanboven bijgezet telkens de nootlengte gehalveerd wordt. Deze notatie wordt geïllustreerd in figuur ??. Door het gebruik van verbindingstekens(waarbij de nieuwe nootlengte de som is van de lengtes van de twee noten die verbonden zijn) kan men dan eender welke nootlengte bekomen die men maar wenst.

Figuur 2.2: Illustratie van een aantal veelvoorkomende nootlengtes.

2.2 Melodie

Naast ritme is het andere fundamentele bestanddeel van een muziekstuk de melodie. Waar het ritme de structuur van een muziekstuk weergeeft, is de melodie het voornaamste bestanddeel van de muziek dat een gevoel meegeeft aan het stuk. De melodie geeft een toonhoogte of freqentie mee aan elke noot. Aangezien deze thesis handelt over melodische transformaties is het van belang te weten wat het begrip

A	B	C	D	E	F	G
la	si	do	re	mi	fa	sol

TABEL 2.1: Opsomming van de toonhoogtes in 2 verschillende benamingen.

'melodie' precies inhoudt. Het is namelijk dat gedeetle van een muziekstuk waarop een transformatie zal toegepast worden. Het ritme van een muziekstuk wordt in deze thesis onveranderd gelaten.

2.2.1 Toonhoogte

Toonhoogte kan in het algemeen op twee manieren voorgesteld worden. Een eerste manier is fysische waarbij elke toon met een bepaalde frequentie overeenkomt, een andere manier is meer muziek-theoretisch, waarbij de toonhoogte als discreet beschouwd wordt. In deze masterproef zal met de laatste voorstellingswijze gewerkt worden. Dit omdat we noten willen transformeren naar nieuwe noten en niet frequenties naar nieuwe frequenties (die dan niet overeen komen met een noot en bijgevolg zeer waarschijnlijk niet goed gaan klinken in het geheel). Deze toonhoogte wordt dan voorgesteld door een naam. Er zijn twee naamgevingen die vaak gebruikt worden. Eerst is er de naamgeving waarbij de letters A t.e.m. G gebruikt worden. De andere naamgeving maakt gebruik van de woorden do – re – mi – fa – sol – la – si, de relatie tussen deze 2 naamgevingen wordt weergegeven in tabel 2.1. Noten kunnen uiteraard ook op een notenbalk weergegeven worden, zie figuur 2.3 voor een visuele weergave. Hoe hoger de noot op de notenbalk, hoe hoger haar frequentie.

FIGUUR 2.3: Noten do t.e.m. si op een notenbalk.

2.2.2 Octaaf en onderverdeling in toonhoogtes

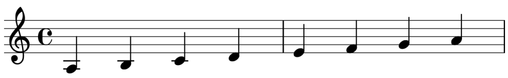
Een eenvoudige definitie van een octaaf is het interval tussen een gegeven toonhoogte en de toonhoogte met het dubbele van de frequentie van de eerste. Deze noten worden als zeer gelijkaardig ervaren en krijgen daarom dezefde benaming (bijvoorbeeld beide een A). Hierdoor gaat men vaak in muzieknotatie wanneer men een bepaalde noot benoemt, ook aangeven tot welk octaaf de noot behoort (want een bepaalde noot komt met meerdere toonhoogtes overeen). Dit doet men door in subscript de index van het octaaf weer te geven, een A (of la) in het vierde octaaf wordt dan weergegeven door A_4.
Een octaaf wordt opgedeeld in 12 toonhoogtes. De afstand tussen elk paar van 2 opeenvolgende toonhoogtes wordt een halve toon genoemd. Hiervan zijn er slechts 7

tonen die ook deel uitmaken van de toonladder. De andere 5 noten worden voorgesteld op de notenbalk relatief ten opzichte van een van de nabijgelegen tonen die slechts een halve toon hier vanaf ligt door het gebruik van een kruis ($\sharp$) of een mol($\flat$). Een kruis dient om aan te duiden dat de noot die bedoeld wordt een halve toon hoger is dan de noot die ervoor staat. Een mol gebruikt men om aan te duiden dat de noot die bedoeld wordt een halve toon lager is dan de noot die net voor het molteken staat.

2.2.3 Toonladders en toonaarden

Zoals reeds vermeld werd in het vorige deel, bestaat een octaaf uit 12 tonen waarvan er slechts 7 tot de eigenlijke toonladder behoren. De noten binnen een toonladder worden bepaald door de intervallen tussen de noten. In het algemeen zijn er twee grote onderverdelingen om een toonladder te construeren. Een eerste resultaat wordt de “grote toonladder” genoemd. deze wordt bepaald door de opeenvolging van stappen 1-1-$\frac{1}{2}$-1-1-1-$\frac{1}{2}$. Het meeste elementaire voorbeeld van een toonladder die hieraan voldoet is de toonladder van “Do-Groot” waarbij volgende noten voldoen aan de opgegeven intervalafstanden: “do - re - mi - fa - sol - la - si - do”, zoals weergegeven in figuur 2.4. Een tweede mogelijkheid is de zogenaamde “kleine toonladder”. In deze toonladder komen de intervalafstanden overeen met 1-$\frac{1}{2}$-1-1-$\frac{1}{2}$-1-1. Een concreet voorbeeld hiervan is de toonladder van “La-Klein” waarbij deze noten voldoen aan de voorwaarden: “la - si - do - re - mi - fa - sol - la”, deze wordt ook weergegeven in figuur 2.5. Deze toonladderis de kleine versie van de toonladder van Do groot, aangezien ze dezelfde noten gebruikt, de toonladder is als het ware een verschuiving van die van Do groot. Het verschil tussen beide toonladders is dus de functie van elke noot in de bijhorende toonaarden. En iets wiskundiger verwoord is er ook een verschil van frequentie waarin bepaalde noten voorkomen in muziekstukken horende bij een kleine of grote toonladder [4]. Van de kleine toonladders zijn ook nog een harmonische en melodische versie, die nog andere afstanden gebruik, maar aangezien deze niet zo vaak gebruikt worden, zou het te ver leiden deze ook te bespreken.
Het feit of een toonladder groot of klein is gaat vaak ook de sfeer bepalen van het muziekstukje. Zo zal een muziekstuk dat geschreven is in een grote toonladder eerder een vrolijk karakter hebben. Dit terwijl een stukje dat geschreven is in een kleine toonladder eerder een droeviger karakter zal hebben. Maar niet alleen de toonladder zelf, ook de noten in de toonladder hebben hun functie. Zo is bijvoorbeeld de eerste noot van een toonladder een rustpunt waarop het muziekstuk vaak beëindigd wordt. De vijfde noot wordt de dominant genoemd en is ook sterk vertegenwoordigd in het muziekstuk. Deze nooit creëert namelijk spanning. Vaak wordt deze spanning dan ook opgelost door een overgang naar de eerste noot, als rustpunt.

Figuur 2.4: Toonladder van “Do Groot”



Figuur 2.5: Toonladder van “La Klein”

Hoofdstuk 3

Objectieve Beoordeling Muziekstuk

Een hoofdstuk behandelt een samenhangend geheel dat min of meer op zichzelf staat. Het is dan ook logisch dat het begint met een inleiding, namelijk het gedeelte van de tekst dat je nu aan het lezen bent.

3.1 Eerste onderwerp in dit hoofdstuk

De inleidende informatie van dit onderwerp.

3.1.1 Een item

Een tekst staat nooit alleen. Dit wil zeggen dat er zeker ook referenties nodig zijn. Dit kan zowel naar on-line documenten[5] als naar boeken[3].

3.2 Figuren

Figuren worden gebruikt om illustraties toe te voegen. Dit is dan ook de manier om beeldmateriaal toe te voegen zoals getoond wordt in figuur 7.1.

Figuur 3.1: Het KU Leuven logo.

gnats	gram	$13.65
	each	.01
gnu	stuffed	92.50
emu		33.33
armadillo	frozen	8.99

Tabel 3.1: Een tabel zoals het niet moet.

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

Tabel 3.2: Een tabel zoals het beter is.

3.3 Tabellen

Tabellen kunnen gebruikt worden om informatie op een overzichtelijke te groeperen. Een tabel is echter geen rekenblad! Vergelijk maar eens tabel 7.1 en tabel 7.2. Welke tabel vind jij het duidelijkst?

3.4 Besluit van dit hoofdstuk

Als je in dit hoofdstuk tot belangrijke resultaten of besluiten gekomen bent, dan is het ook logisch om het hoofdstuk af te ronden met een overzicht ervan. Voor hoofdstukken zoals de inleiding en het literatuuroverzicht is dit niet strikt nodig.

Hoofdstuk 4

Melodische Transformatie

4.1 Verschillende transformaties

4.1.1 Permutatie van noten

4.1.2 Afbeelding afhankelijk van positie

4.1.3 Afbeelding afhankelijk van afstand ten opzichte van vorige noot

4.2 Vergelijking van transformaties

Hoofdstuk 5

Efficiënt Toepassen Transformatie

In dit hoofdstuk worden twee algoritmes behandeld die varianten zijn van elkaar. In beide algoritmes is het de bedoeling om een gegeven muziekstuk te transformeren tot een nieuw muziekstuk op zo een manier dat de totale consonantiescore zo hoog mogelijk is. Dit kan gebeuren door een aantal toegelaten operaties op het oorspronkelijke muziekstuk. Buiten de originele melodielijn zullen namelijk ook een aantal verschillende transformaties (zoals die beschreven zijn in onderdeel 4.1.3) meegegeven worden aan het algoritme. Deze transformaties zullen door het algoritme gebruikt worden om het originele stuk te transformeren naar een nieuw stuk met een hogere score. Voor elke noot van de melodielijn heeft het algoritme namelijk de keuze om ofwel de noot te behouden, ofwel deze te transformeren conform een van de transformaties die meegegeven werden aan het algoritme. Het algoritme dat hier voor zorgt wordt beschreven in onderdeel 5.1. In gedeelte 5.2 wordt een algoritme beschreven dat eenzelfde doel heeft maar een extra voorwaarde opgelegd krijgt. Deze voorwaarde stelt dat als met een transformatie wilt toepassen, men telkens een minimum aantal opeenvolgende noten moet transformeren met deze transformatie.

5.1 Beste sequentie

5.1.1 Doelstelling

Het algoritme dat hier beschreven wordt is afhankelijk van twee parameters. Een eerste parameter is een originele melodielijn. De tweede parameter beschrijft een aantal transformaties (gedifiniëerd zoals in onderdeel 4.1.3), die gebruikt mogen worden door het algoritme. Het doel is nu om gebruik makend van enkel deze verschillende mogelijke transformaties de originele melodielijn te transformeren tot de nieuwe melodielijn die de hoogste consonantiescore oplevert. Hierbij heeft het algoritme voor elke noot in het muzikstuk twee mogelijke keuzes. Een eerste mogelijkheid is dat de noot niet getransformeerd wordt en identiek overgenomen wordt in de nieuwe melodielijn. De tweede mogelijkheid bestaat erin dat de noot ook getransformeerd mag worden, maar denk enkel volgens een van de beschikbare transformaties.

5.1.2 Idee van het algoritme

Een eerste belangrijke observatie is dat eender welke noot in het muziekstukje zelf op slechts maximaal 14 verschillende noten kan afgebeeld worden. Elke noot kan namelijk door transformatie enkel afgebeeld worden op een noot die maximaal 5 halve tonen lager ligt dan de oorspronkelijke noot en ook maximaal 6 halve tonen hoger ligt dan de originele noot (dit is een deel van de definitie van de soort transformaties die gebruikt worden in het algoritme). Er is nu echter nog een speciaal geval waarbij een transformatie tot een van de twee extreme gevallen zou leiden en deze noot dan ook nog eens geen deel zou uitmaken van de tonaard. In dat geval is het mogelijk dat de noot nog een halve toon verder afgerond wordt om terug tot op een noot te komen die in de toonaard ligt. Elke noot kan dus theoretisch gezien (na afronding) afgebeeld worden op eender welke noot die maximaal 6 halve tonen lager en maximaal 7 halve tonen hoger ligt dat zichzelf. Dit zijn in totaal 14 verschillende mogelijkheden.
Het belangrijkste idee van het algoritme is gebaseerd op de principes van *dynamic programming*. Bij dit algoritme komt het er op neer dat achtereenvolgens voor elke noot in het muziekstuk het best pad (en bijhorende beste score) bijgehouden zal worden voor elk van de 14 mogelijke toonhoogtes die deze noot kan aannemen in de nieuwe melodielijn. En enkel op deze optimale paden zal verder gerekend worden om te bepalen wat de beste paden zijn tot de 14 mogelijke toonhoogtes die de volgende noot in het muziekstuk kan aannemen. Dit wordt dan verder herhaald tot alle noten van de originele melodielijn overlopen zijn.

5.1.3 Werking van het algoritme

In dit onderdeel zal de werking van het algoritme beschreven worden. Als extra referentie voor de lezer is de broncode bijgevoegd in appendix A.1.
Het algoritme zal tijdens de uitvoering gebruik maken van 3 hulpvariabelen. De eerste twee van deze hulpvariabelen zijn eendimensionale arrays van lengte 14. De eerste array, die de naam *past* meekrijgt, stelt de probabiliteiten voor van de beste paden eindigend bij alle mogelijke noten waarnaar de vorige beschouwde noot kan getransformeerd worden. De tweede array, die *current* genoemd wordt, bevat de probabiliteiten van de beste paden die eindigen op de huidig beschouwde noot (of een van zijn mogelijke transformaties). De waarden in deze twee arrays, zullen in het algoritme niet de probabiliteiten zelf, maar hun logaritmen zijn, dit om het rekenwerk te vereenvoudigen. De derde hulpvariabele, *matrix*, is een tweedimensionale array en heeft een dimensie van de lengte van de melodielijn op 14. Deze variabele gaat voor elke mogelijke waarde die elke noot in het muziekstukje kan aannemen, weergeven vanwaar het beste pad komt dat eindigt bij deze noot.
Om de betekenis van deze variabelen nog iets concreter te maken zal gebruik gemaakt worden van de volgende notatie. De letter 'C' voor de toonhoogte van de huidig beschouwde noot in de originele melodielijn. De letter 'P' voor de toonhoogte van

de vorige noot in het originele muziekstukje. Zo zal de waarde C+3 staan voor de noot met een toonhoogte die drie halve tonen hoger ligt dan de huidige noot in het originele stukje. P-4 zal dan bijvoorbeeld staan voor de noot met een toonhoogte die 4 halve tonen lager ligt dan de vorige noot in de originele melodielijn. Verder zal, er als er in gesproken wordt over de i-de noot in het muziekstuk, de noot op positie i bedoeld worden. Zo zal de 0-de noot in het muziekstukje de eerste noot zijn, de 1-de noot zal de tweede noot van het muziekstukje zijn enzovoort.
Stel we beschouwen momenteel de n-de noot van het muziekstukje. Dan zullen volgende stellingen gelden. Op positie *past[i]* zal de waarde voor de probabiliteit staan van het beste pad van n noten dat eindigt op de noot 'P+i-6'. Ook zal tijdens uitvoering van het algoritme de *current* array aangepast worden. Met als doelstelling: op positie *current[i]* zal de waarde voor de probabiliteit berekend worden voor het beste pad van (n+1) noten dat eindigt op de noot 'C+i-6'. De waarde van het element matrix[i][j] verwijst naar vanwaar het beste pad komt van (n+1) noten dat eindigt op de noot 'C+j-6'. Stel nu dat matrix[i][j]=k, dan betekent dit dat het beste pad van (n+1) noten dat eindigt op noot 'C+j-6', een verlenging is van het beste pad van n noten dat eindigt op noot 'P+k-6'.

Initialisatie

De eeste stap in het algoritme zal de noot op positie 1 als huidige noot bekijken en de noot op positie 0 als vorige noot. De hulpvariabelen moeten dus geïnitialiseerd worden zodat ze daaraan voldoen. Hierdoor zal de array *past* voor al zijn posities een hele lage waarde meekrijgen (-100 als logaritme van de probabiliteit in de broncode weergegeven in appendix A.1) behalve voor de noot op positie 6. Deze noot staat namelijk voor dezelfde noot als oorspronkelijke noot en aangezien de oorspronkelijke noot op positie 0 nooit getransformeerd wordt, zal *past[6] = 0.* Dit betekent dat dat kans op het behouden van de eerste noot gelijk is aan 1. De *current* array krijgt voor al zijn posities zeer lage waarden mee. De bedoeling hiervan is dat eender welk pad dat als eerste gevonden wordt dat voortgaat op de vorige noot een hogere probabiliteit zal hebben dan deze waarde. Tot slot zal ook elke waarde van *matrix* op een arbitraire startwaarde 0 gezet worden. De waarde die hier in het begin gezet wordt is van geen belang, aangezien deze waarden toch overschreven zullen worden wanneer de beste paden berekend worden.

Een stap in het algoritme

Voor elke noot in het muziekstukje, beginnend vanaf de noot op positie 1, zal nu het volgende uitegevoerd worden. Stel ook dat we momenteel als huidige noot, de noot op positie n beschowen.
Allereerst gaan we op *current[6]* de waarde zetten van het beste pad dat de huidige noot niet transformeert. Om dit te doen gaan we kijken naar de probabiliteiten van de beste paden die eindigen op alle 14 mogelijke noten die als vorige noot zouden kunnen doorgaan. Deze probabiliteiten staan uiteraard gewoon in *past.* Voor elk element *past[i]* wordt nu zijn waarde bij de afstandsprobabiliteit ten opzichte van de

huidig beschouwde noot opgeteld. Als deze waarde hoger is dan de huidige beste waarde van *current[6]*, dan wordt de waarde van *current[6]* overschreven en wordt de waarde *matrix[n][6]* op i gezet. Als tweede stap gaan we voor elke mogelijk transformatie kijken naar welke noot de noot op 'P+i-6' de noot huidige noot gaat afbeelden. En dit voor alle 14 mogelijke vorige noten. Stel nu dat de noot 'P-i+6' de huidig beschouwde noot afbeeldt op noot 'C+j-6'. Als nu de de som van *past[i]* en de afstandsprobabiliteit tussen deze twee noten groter is dan de huidige waarde *current[j]*, dan zal deze waarde overschreven worden. Alsook zal *matrix[n][j]* gelijk gesteld worden aan i. Dit aangezien het nieuwe meest waarschijnlijke pad dat gevonden is dat eindigt op 'C+j-6', komt van de noot 'P+i-6'.
Wanneer dit uitgevoerd is, bevat *current* de probabiliteiten van de beste paden van lengte (n+1) die eindigen op de 14 mogelijke toonhoogtes waarop de huidig beschouwde noot kan afgebeeld worden. Het enige wat nu nog rest is om de waarden van *current* over te zetten naar *past*. En om *current* daarna terug te initialiseren op zeer lager waarden. Dit zodat de arrays klaar zijn voor het beschouwen van de volgende noot in het muziekstukje.

Extractie van het beste pad

Na uitvoeren van het vorige gedeelte voor elke noot in het muziekstukje zal *past* de probabiliteiten bevatten van de beste paden voor elk van de 14 mogelijke eindnoten van het geheel van het nieuwe muziekstukje. Als eerste zal er dus gekeken worden welke van deze 14 toonhoogtes het meest waarschijnlijke pad heeft. Dit pad is het pad dat we zoeken in dit algoritme. Het enige wat nu nog rest is om vanaf dit punt dat pad te reconstueren. Dit gebeurt met behulp van de *matrix* array. Beginnen bij de laatste noot en zo opschuivend terug naar voor wordt hiervoor het volgende gedaan. Stel we zitten bij noot n. Noem 'C+i-6' de toonhoogte van de huidige noot die in dit beste pad zit. Noem nu *matrix[n][i]* = j. Dan is de vorige noot van het optimale pad de noot met als toonhoogte 'P+j-6', waarbij P de toonhoogte is van de overeenkomstige noot in het originele muziekstuk. *matrix[n-1][j]* geeft dan weer aanleiding tot de noot die daarvoor staat in het optimale pad, enzovoort. Op deze manier kan het volledige optimale pad weer gereconstrueerd worden.

5.1.4 Performantie en geheugencomplexiteit

De parameters waarvan het algoritme afhankelijk is, zijn de lengte van de melodielijn in de invoer en ook het aantal transformaties (in het voorbeeld beschreven in appendix A.1 wordt gebruik gemaakt van slechts 2 transformaties, maar het algoritme werkt voor eender welk aantal transformaties dat gedefiniëerd wordt).
De snelheid van het algoritme is lineair afhankelijk van beide van deze parameters. Indien de lengte van het originele melodietje met een factor f omhoog gaat en de rest constant blijft dan gaat ook het aantal stappen in het algoritme met een factor f omhoog. Het rekenwerk per stap blijft echter gelijk. Wanneer het aantal transformaties met een factor f omhoog gaat en de rest constant blijft, dan zal het aantal stappen onveranderd blijven. Het rekenwerk per stap gaat wel met een factor

f omhoog (op het constante rekenwerk per stap van het 'niet transformeren' na).
Het geheugen dat nodig is voor de uitvoer van het algoritme is lineair afhankelijk van de lengte van de originele melodie. Dit komt omdat de *matrix* array als een van zijn dimensies deze lengte heeft. Wanneer de lengte van het originele stukje met een factor f omhoog gaat, zal de grootte van deze array dus ook met een factor f omhoog gaan. De grootte van de andere twee gebruikte arrays tijdens de uitvoer van het algoritme is onveranderlijk ten opzichte van die lengte. Maar in het total geheugengebruik is de *matrix* array dominant aangezien deze zo veel groter is. Het aantal gebruikte transformaties heeft geen effect op het geheugengebruik van het algoritme.

5.2 Beste sequentie met minimum transformatie lengte

5.2.1 Doelstelling

5.2.2 Werking van het algoritme

5.2.3 Performantie en geheugencomplexiteit

Hoofdstuk 6

Experimenten en Resultaten

Een hoofdstuk behandelt een samenhangend geheel dat min of meer op zichzelf staat. Het is dan ook logisch dat het begint met een inleiding, namelijk het gedeelte van de tekst dat je nu aan het lezen bent.

6.1 Eerste onderwerp in dit hoofdstuk

De inleidende informatie van dit onderwerp.

6.1.1 Een item

Een tekst staat nooit alleen. Dit wil zeggen dat er zeker ook referenties nodig zijn. Dit kan zowel naar on-line documenten[5] als naar boeken[3].

6.2 Figuren

Figuren worden gebruikt om illustraties toe te voegen. Dit is dan ook de manier om beeldmateriaal toe te voegen zoals getoond wordt in figuur 7.1.

6.3 Tabellen

Tabellen kunnen gebruikt worden om informatie op een overzichtelijke te groeperen. Een tabel is echter geen rekenblad! Vergelijk maar eens tabel 7.1 en tabel 7.2. Welke tabel vind jij het duidelijkst?

FIGUUR 6.1: Het KU Leuven logo.

gnats	gram	$13.65
	each	.01
gnu	stuffed	92.50
emu		33.33
armadillo	frozen	8.99

Tabel 6.1: Een tabel zoals het niet moet.

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

Tabel 6.2: Een tabel zoals het beter is.

6.4 Besluit van dit hoofdstuk

Als je in dit hoofdstuk tot belangrijke resultaten of besluiten gekomen bent, dan is het ook logisch om het hoofdstuk af te ronden met een overzicht ervan. Voor hoofdstukken zoals de inleiding en het literatuuroverzicht is dit niet strikt nodig.

Hoofdstuk 7

Besluit

Een hoofdstuk behandelt een samenhangend geheel dat min of meer op zichzelf staat. Het is dan ook logisch dat het begint met een inleiding, namelijk het gedeelte van de tekst dat je nu aan het lezen bent.

7.1 Eerste onderwerp in dit hoofdstuk

De inleidende informatie van dit onderwerp.

7.1.1 Een item

Een tekst staat nooit alleen. Dit wil zeggen dat er zeker ook referenties nodig zijn. Dit kan zowel naar on-line documenten[5] als naar boeken[3].

7.2 Figuren

Figuren worden gebruikt om illustraties toe te voegen. Dit is dan ook de manier om beeldmateriaal toe te voegen zoals getoond wordt in figuur 7.1.

7.3 Tabellen

Tabellen kunnen gebruikt worden om informatie op een overzichtelijke te groeperen. Een tabel is echter geen rekenblad! Vergelijk maar eens tabel 7.1 en tabel 7.2. Welke tabel vind jij het duidelijkst?

FIGUUR 7.1: Het KU Leuven logo.

gnats	gram	$13.65
	each	.01
gnu	stuffed	92.50
emu		33.33
armadillo	frozen	8.99

Tabel 7.1: Een tabel zoals het niet moet.

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

Tabel 7.2: Een tabel zoals het beter is.

7.4 Besluit van dit hoofdstuk

Als je in dit hoofdstuk tot belangrijke resultaten of besluiten gekomen bent, dan is het ook logisch om het hoofdstuk af te ronden met een overzicht ervan. Voor hoofdstukken zoals de inleiding en het literatuuroverzicht is dit niet strikt nodig.

Bijlagen

Bijlage A

Broncode

A.1 Beste sequentie

```
__author__ = 'Elias'
import RPK
import math

zero = [0, 0, 0, 0, 0, 0, 0, 0]
transformations = [[1, 1, 2, 3, 5, -4, 1, -3],[5, -4, 1, -3,
    1, 1, 2, 3]]

def bestTransformed(int_notes, keyIndex):
    #transformation from -6 to +7
    past = [-100, -100, -100, -100, -100, -100, 0, -100,
        -100, -100, -100, -100, -100, -100]
    current = [-100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100]
    matrix = [[0 for x in range(len(past))] for y in range(
        len(int_notes))]
    for x in range(len(int_notes)-1):
        present_note = int_notes[x+1]

        #case of keeping original
        for y in range(len(past)):
            last_note = int_notes[x]+y-6
            diff = abs(present_note-last_note)
            new_note = present_note+(zero[(diff+8)%8])
            if current[new_note-present_note+6] < past[y]+
                RPK.proximity_prob_log(new_note, last_note):
                current[new_note-present_note+6] = past[y]+
                    RPK.proximity_prob_log(new_note,
                    last_note)
```

```
            matrix[x][new_note-present_note+6] = y

    #case of transform
    for t in range(len(transformations)):
        for y in range(len(past)):
            last_note = int_notes[x]+y-6
            diff = abs(present_note-last_note)
            new_note = -1;
            if (present_note-last_note <= 0):
                new_note = present_note+(transformations
                    [t][(diff+8)%8])
            else:
                new_note = present_note-(transformations
                    [t][(diff+8)%8])
            if RPK.note_prob_in_key(new_note, keyIndex)
                < 0.02:
                if (RPK.note_prob_in_key(new_note+1,
                    keyIndex) > RPK.note_prob_in_key(
                    new_note-1, keyIndex)):
                    new_note = new_note+1
                else:
                    new_note = new_note-1
            if current[new_note-present_note+6] < past[y
                ]+RPK.proximity_prob_log(new_note,
                last_note):
                current[new_note-present_note+6] = past[
                    y]+RPK.proximity_prob_log(new_note,
                    last_note)
                matrix[x][new_note-present_note+6] = y

    #reinitialize arrays
    for y in range(len(past)):
        past[y] = current[y]+RPK.range_prob_log(
            present_note+y-6)+math.log(RPK.
            note_prob_in_key(present_note+y-6, keyIndex))
        current[y] = -10000000;

#look at which endpoint had the most probable path and
    reconstruct that most probable path
maxIndex = 0
for x in range(len(past)):
    if (past[x] > past[maxIndex]):
        maxIndex = x
reversed_list = [0 for x in range(len(int_notes))]
reversed_list[0] = maxIndex
```

```
    for x in range(len(int_notes)-1):
        reversed_list[x+1] = matrix[len(int_notes)-x-2][
            maxIndex]
        maxIndex = reversed_list[x+1]
    ordered_list = [0 for x in range(len(int_notes))]
    for x in range(len(int_notes)):
        ordered_list[x] = reversed_list[len(int_notes)-x-1]
    new_int_notes = [0 for x in range(len(int_notes))]
    for x in range(len(int_notes)):
        new_int_notes[x] = int_notes[x]+ordered_list[x]-6
    return new_int_notes
```

A.2 Beste sequentie met minimum transformatie lengte

```
___author___ = 'Elias'
import RPK
import math

zero = [0, 0, 0, 0, 0, 0, 0, 0]
transformations = [[1, 1, 2, 3, 5, -4, 1, -3], [5, -4, 1,
    -3, 1, 1, 2, 3]]

#min amount of concatenated notes that need to be
    transformed.
min_length = 4
height = 14

def bestTransformed(int_notes, keyIndex):
    #transformation from -6 to +7
    #add new history to back of the array
    prob_keep_past = [[-100000, -100000, -100000, -100000,
        -100000, -100000, 0, -100000, -100000, -100000,
        -100000, -100000, -100000, -100000] for x in range(
        min_length)]
    prob_keep_current = [-100000 for y in range(height)]
    prob_transform_past = [[[-100000 for y in range(height)]
         for x in range(min_length)] for z in range(len(
        transformations))]
    prob_transform_current = [[-100000 for y in range(height
        )] for z in range(len(transformations))]
```

```
path_end_on_keep_past = [[[] for x in range(height)] for
    y in range(min_length)]
path_end_on_keep_current = [[] for x in range(height)]
path_end_on_transform_past = [[[[] for x in range(height
   )] for y in range(min_length)] for z in range(len(
   transformations))]
path_end_on_transform_current = [[[] for x in range(
   height)] for z in range(len(transformations))]

for x in range(len(int_notes)-1):
    present_note = int_notes[x+1]

    #case of keeping original
    for y in range(height):
        last_note = int_notes[x]+y-6
        diff = abs(present_note-last_note)
        new_note = present_note+(zero[(diff+8)%8])
        #extending one that already ended on a keep
        if prob_keep_current[new_note-present_note+6] <
            prob_keep_past[min_length-1][y]+RPK.
            proximity_prob_log(new_note, last_note):
             prob_keep_current[new_note-present_note+6] =
                 prob_keep_past[min_length-1][y]+RPK.
                proximity_prob_log(new_note, last_note)
             copy = list(path_end_on_keep_past[min_length
                -1][y])
             copy.append(y)
             path_end_on_keep_current[new_note-
                present_note+6] = copy
        #extending one that ended on a transform
        for z in range(len(transformations)):
            if prob_keep_current[new_note-present_note
                +6] < prob_transform_past[z][min_length
                -1][y]+RPK.proximity_prob_log(new_note,
                last_note):
                 prob_keep_current[new_note-present_note
                    +6] = prob_transform_past[z][
                    min_length-1][y]+RPK.
                    proximity_prob_log(new_note,
                    last_note)
                 copy = list(path_end_on_transform_past[z
                    ][min_length-1][y])
                 copy.append(y)
                 path_end_on_keep_current[new_note-
                    present_note+6] = copy
```

```
#case of transform
if x >= (min_length-1):
    for z in range(len(transformations)):
        #extending one that already ended on the
            same transform
        for y in range(height):
            last_note = int_notes[x]+y-6
            diff = abs(present_note-last_note)
            new_note = -1
            if (present_note-last_note <= 0):
                new_note = present_note+(
                    transformations[z][(diff+8)%8])
            else:
                new_note = present_note-(
                    transformations[z][(diff+8)%8])
            if RPK.note_prob_in_key(new_note,
                keyIndex) < 0.02:
                if (RPK.note_prob_in_key(new_note+1,
                     keyIndex) > RPK.note_prob_in_key
                    (new_note-1, keyIndex)):
                     new_note = new_note+1
                else:
                     new_note = new_note-1
            if prob_transform_current[z][new_note-
                present_note+6] < prob_transform_past
                [z][min_length-1][y]+RPK.
                proximity_prob_log(new_note,
                last_note):
                 prob_transform_current[z][new_note-
                    present_note+6] =
                    prob_transform_past[z][min_length
                    -1][y]+RPK.proximity_prob_log(
                    new_note, last_note)
                 copy = list(
                    path_end_on_transform_past[z][
                    min_length-1][y])
                 copy.append(y)
                 path_end_on_transform_current[z][
                    new_note-present_note+6] = copy

        #extending one that ended on another
            transform
        for q in range(len(transformations)):
            if q==z:
```

```
                continue
            start_probs = list(prob_transform_past[q
                ][0])
            past_paths = [list(
                path_end_on_transform_past[q][0][x])
                for x in range(height)]
            result = simulate_best_paths(start_probs
                , transformations[z], min_length,
                past_paths, x-(min_length-1),
                keyIndex, int_notes)
            end_probs = result[0]
            end_paths = result[1]
            for y in range(height):
                if prob_transform_current[z][y] <
                    end_probs[y]:
                    prob_transform_current[z][y] =
                        end_probs[y]
                    path_end_on_transform_current[z
                        ][y] = list(end_paths[y])

        #extending one that ended on keep
        start_probs = list(prob_keep_past[0])
        past_paths = [list(path_end_on_keep_past[0][
            x]) for x in range(height)]
        result = simulate_best_paths(start_probs,
            transformations[z], min_length,
            past_paths, x-(min_length-1), keyIndex,
            int_notes)
        end_probs = result[0]
        end_paths = result[1]
        for y in range(height):
            if prob_transform_current[z][y] <
                end_probs[y]:
                prob_transform_current[z][y] =
                    end_probs[y]
                path_end_on_transform_current[z][y]
                    = list(end_paths[y])

#reinitialize arrays
for y in range(height):
    for q in range(min_length-1):
        prob_keep_past[q][y] = prob_keep_past[q+1][y
            ]
        path_end_on_keep_past[q][y] =
            path_end_on_keep_past[q+1][y]
```

```
                for z in range(len(transformations)):
                    prob_transform_past[z][q][y] =
                        prob_transform_past[z][q+1][y]
                    path_end_on_transform_past[z][q][y] =
                        path_end_on_transform_past[z][q+1][y]
            prob_keep_past[min_length-1][y] =
                prob_keep_current[y]+RPK.range_prob_log(
                present_note+y-6)+math.log(RPK.
                note_prob_in_key(present_note+y-6, keyIndex))
            prob_keep_current[y] = -100000
            for z in range(len(transformations)):
                prob_transform_past[z][min_length-1][y] =
                    prob_transform_current[z][y]+RPK.
                    range_prob_log(present_note+y-6)+math.log
                    (RPK.note_prob_in_key(present_note+y-6,
                    keyIndex))
                prob_transform_current[z][y] = -100000
            path_end_on_keep_past[min_length-1][y] =
                path_end_on_keep_current[y]
            path_end_on_keep_current[y] = []
            for z in range(len(transformations)):
                path_end_on_transform_past[z][min_length-1][
                    y] = path_end_on_transform_current[z][y]
                path_end_on_transform_current[z][y] = []

    #Return path with max probability
    maxIndexKeep = 0
    for x in range(height):
        if (prob_keep_past[min_length-1][x] > prob_keep_past
            [min_length-1][maxIndexKeep]):
            maxIndexKeep = x

    maxIndexTransform = 0
    maxIndexTypeTransform = 0
    for z in range(len(transformations)):
        for x in range(height):
            if (prob_transform_past[z][min_length-1][x] >
                prob_transform_past[maxIndexTypeTransform][
                min_length-1][maxIndexTransform]):
                maxIndexTransform = x
                maxIndexTypeTransform = z

    best_path = []
    if (prob_keep_past[min_length-1][maxIndexKeep] >
        prob_transform_past[maxIndexTypeTransform][min_length
```

```
        -1][maxIndexTransform]):
            best_path = list(path_end_on_keep_past[min_length
                -1][maxIndexKeep])
            best_path.append(maxIndexKeep)
    else:
            best_path = list(path_end_on_transform_past[
                maxIndexTypeTransform][min_length-1][
                maxIndexTransform])
            best_path.append(maxIndexTransform)

    new_int_notes = [0 for x in range(len(int_notes))]
    for x in range(len(int_notes)):
        new_int_notes[x] = int_notes[x]+best_path[x]-6
    return new_int_notes

    reversed_list = [0 for x in range(len(int_notes))]
    reversed_list[0] = maxIndex
    for x in range(len(int_notes)-1):
        reversed_list[x+1] = matrix[len(int_notes)-x-2][
            maxIndex]
        maxIndex = reversed_list[x+1]
    ordered_list = [0 for x in range(len(int_notes))]
    for x in range(len(int_notes)):
        ordered_list[x] = reversed_list[len(int_notes)-x-1]
    new_int_notes = [0 for x in range(len(int_notes))]
    for x in range(len(int_notes)):
        new_int_notes[x] = int_notes[x]+ordered_list[x]-6
    return new_int_notes

#return the best paths of given length, only using the
    tranform given by the array parameter
def simulate_best_paths(start_probs, transform_array, length
    , start_paths, start_note, keyIndex, int_notes):
    past_probs = list(start_probs)
    current_probs = [-100000 for x in range(height)]
    past_paths = [list(start_paths[x]) for x in range(height
        )]
    current_paths = [[] for x in range(height)]

    for x in range(length):
        present_note = int_notes[start_note+x+1]

        for y in range(height):
            last_note = int_notes[start_note+x]+y-6
            diff = abs(present_note-last_note)
```

```
            new_note = -1
            if (present_note-last_note <= 0):
                new_note = present_note+(transform_array[(
                    diff+8)%8])
            else:
                new_note = present_note-(transform_array[(
                    diff+8)%8])
            if RPK.note_prob_in_key(new_note, keyIndex) <
                0.02:
                if (RPK.note_prob_in_key(new_note+1,
                    keyIndex) > RPK.note_prob_in_key(new_note
                    -1, keyIndex)):
                    new_note = new_note+1
                else:
                    new_note = new_note-1
            #replace if better
            if current_probs[new_note-present_note+6] <
                past_probs[y]+RPK.proximity_prob_log(new_note
                , last_note):
                current_probs[new_note-present_note+6] =
                    past_probs[y]+RPK.proximity_prob_log(
                    new_note, last_note)
                copy = list(past_paths[y])
                copy.append(y)
                current_paths[new_note-present_note+6] =
                    copy

        for y in range(height):
            past_probs[y] = current_probs[y]+RPK.
                range_prob_log(present_note+y-6)+math.log(RPK
                .note_prob_in_key(present_note+y-6, keyIndex)
                )
            current_probs[y] = -100000
            past_paths[y] = current_paths[y]
            current_paths[y] = []

    last_note = int_notes[start_note+length]
    for y in range(height):
        past_probs[y] = past_probs[y]-RPK.range_prob_log(
            last_note+y-6)-math.log(RPK.note_prob_in_key(
            last_note+y-6, keyIndex))

    return (past_probs, past_paths)
```

Bijlage B

IEEE_Paper

In de bijlagen vindt men de data terug die nuttig kunnen zijn voor de lezer, maar die niet essentieel zijn om het betoog in de normale tekst te kunnen volgen. Voorbeelden hiervan zijn bronbestanden, configuratie-informatie, langdradige wiskundige afleidingen, enz.

In een bijlage kunnen natuurlijk ook verdere onderverdelingen voorkomen, evenals figuren en referenties[1].

Bijlage C

Poster

Muziek Compositie via Melodische Transformaties

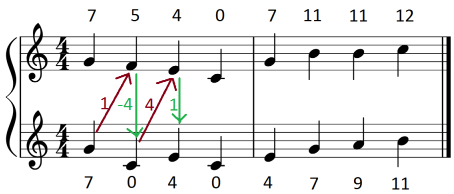
Probleemstelling

- Nood aan **Inspiratie** voor singer-songwriters met writers block
- Muziek generatie leidt vaak tot te 'Artificieel'-klinkende melodieën

→ (Melodische) muziek transformatie kan een uitweg bieden
→ Onder **welke omstandigheden** kan zo een transformatie werken?

Melodische Transformatie

- Sprong op basis van (absolute)**afstand t.o.v. vorige noot**
- Sprong in tegengestelde richting van positie t.o.v. vorige noot
- Altijd eindigen op **noot in juiste toonaard** (door afronding)
- Eenheid in halve tonen



Afstand	Sprong
0	5
1	-4
2	1
3	-3
4	1
5	1
6	2
7	3

Objectieve Beoordeling Melodie

RPK – Model [D. Temperley – 2007]

Probabiliteit van een noot in een notensequentie is het product van 3 factoren:

- R(ange): Probabiliteit van afstand toonhoogte t.o.v. centrale toonhoogte (normaal verdeeld)
- P(roximity): Probabiliteit van afstand in toonhoogte t.o.v. vorige noot (normaal verdeeld)
- K(ey) Profile: Probabiliteit van noot in gebruikte toonaard

Dit leidt tot een **consonantie-score**, een maat voor het goed klinken van een melodielijn

Gebruik transformatie voor verhoging consonantie melodie

Doel:

- Voor elke noot in de melodielijn: behoud deze noot of transformeer ze gebruik makend van één van de meegegeven transformaties
- Doe dit voor elke noot zodat de **probabiliteit** van het totale muziekstuk **zo hoog mogelijk** is
- Zorg dat het computationeel efficiënt is
- Laat toe een **minimum lengte (ML)** mee te geven die aangeeft dat transformaties enkel mogen toegepast worden op minstens ML opeenvolgende noten

Oplossing:

- Algoritme voornamelijk gebaseerd op technieken van dynamic programming
- **Tijd:** **O(AN x AT²)** **Geheugen:** **O(AN x ML x AT)**
    - AN: aantal noten in sequentie
    - ML: minimum lengte transformatie
    - AT: aantal beschikbare transformaties

Algoritme:

Voor elke beschikbare transformatie:

- Houd tabel bij met beste pad dat eindigt met deze specifieke transformatie, en dit voor alle paden eindigend op een van de laatste ML beschouwde noten

Voor elke noot in muziekstuk:

- Voor elke mogelijke transformatie 2 opties:
    - Breid pad dat eindigt met deze transformatie uit gebruik makend van deze transformatie
    - Breid pad dat eindigt met andere transformatie en exact ML in lengte korter is uit met ML keer deze transformatie
    - Pad met hoogste probabiliteit wordt bijgehouden
- Ofwel geen transformatie toepassen:
    - Uitbreiding van eender welk pad dat eindigt op huidige noot zonder transformatie

KU LEUVEN

DTAI
Declarative Languages & Artificial Intelligence

Student: Elias Moons
Begeleider: Vincent Nys
Promotor: Prof. De Schreye

Bibliografie

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy.* Del Rey (reprint), 1995. ISBN-13: 978-0345391803.

[2] U. of Oxford Text Archive. Essen corpus of german folksong melodies.

[3] T. Pratchett and N. Gaiman. *Good Omens:* The Nice and Accurate Prophecies of Agnes Nutter, Witch. HarperTorch (reprint), 2006. ISBN-13: 978-0060853983.

[4] D. Temperley. *Music and Probability.* The MIT Press, Cambridge, Massachusetts, 2007.

[5] Wikipedia. Scriptie. URL: http://nl.wikipedia.org/wiki/Masterproef, laatst nagekeken op 2010-01-07.

Fiche masterproef

Student: Elias Moons

Titel: Melodische transformatie en evalutie van muziek

Engelse titel: Melodische transformatie en evalutie van muziek

UDC: 004.9

Korte inhoud:
Hier komt een heel bondig abstract van hooguit 500 woorden. LaTeX commando's mogen hier gebruikt worden. Voorbeelden van letters met accenten zijn "éïçàô". (Gebruik ï in plaats van ï, om geen 3 puntjes te hebben.)

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen

Promotor: Prof. dr. D. De Schreye

Assessor: Prof. Onbekend

Begeleider: Ir. V. Nys