



UNIVERSITÀ DEGLI STUDI DI TRIESTE

DIPARTIMENTO DI MATEMATICA E GEOSCIENZE  
INTELLIGENZA ARTIFICIALE E DATA SCIENCE

## Musica Genetica

*Autore:*  
Leonardo Angellotti

*Relatore:*  
Luca Manzoni

Anno Accademico 2023/2024

# Contents

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Nozioni di Base</b>	<b>4</b>
2.1	nozioni algoritmiche . . . . .	4
2.1.1	fitness-function . . . . .	4
2.1.2	modello markoviano . . . . .	4
2.1.3	artificial neural network . . . . .	5
2.1.4	file MIDI . . . . .	5
2.2	nozioni musicali . . . . .	6
2.2.1	caratteristiche di una nota musicale . . . . .	6
2.2.2	scala armonica . . . . .	7
2.2.3	melodia . . . . .	8
2.2.4	accordi . . . . .	8
<b>3</b>	<b>Metodi di generazione</b>	<b>10</b>
3.1	generazione melodia . . . . .	10
3.1.1	armonia . . . . .	12
3.1.2	ritmo . . . . .	14
3.2	algoritmo genetico . . . . .	15
3.2.1	musica evolutiva . . . . .	15
3.2.2	prime applicazioni degli alg. gen. alla musica . . . . .	15
3.3	L-System . . . . .	17
<b>4</b>	<b>Progetto di musica genetica</b>	<b>20</b>
4.0.1	Run-Evolution . . . . .	21
4.0.2	fitness-function . . . . .	22
4.0.3	risultati . . . . .	25
<b>5</b>	<b>Conclusioni</b>	<b>30</b>

# Chapter 1

## Introduzione

C’è una grande differenza tra *impossibile* e *difficile da immaginare*.

La prima riguarda il *problema*; la seconda riguarda *te*.

La Musica Algoritmica ha una lunga storia nell’era pre-informatica, dai lavori ’processuali’ di George Brecht (Drip Music, 1962), Stockhausen (Setz die Segel zur Sonn, 1970), Xenakis (Metastaseis, 1971) fino a George Lewis (Voyager, 1993).

Gli algoritmi di Intelligenza Artificiale (AI) sono una delle tecnologie più ampiamente utilizzate e potenti del ventunesimo secolo. Sebbene la definizione esatta di AI si sia costantemente evoluta, dagli anni ’50 di Alan Turing, alle attuali architetture di Deep Learning come Alpha Go, rimane fisso il fatto che i correnti modelli dedicati ad attività creative, sono attualmente agli albori, e si sviluppano quasi esclusivamente il dataset su cui sono addestrati.

Un altro limite è di carattere *filosofico*, infatti l’approccio adottato da tali modelli tratta il processo creativo come un problema di ottimizzazione (non come l’artista, nella sua necessità a creare).

Esistono comunque progetti dove le composizioni che rispettano una certa simmetria e regolarità, come i corali di Bach (Deep Bach) o la Modellazione della Musica Popolare (Folk RNN).

Modellare la musica è un problema difficile per due ragioni principali:

Le gerarchie semantiche sono complesse, soggettive e operano su più scale temporali (dell’ordine di secondi a diversi minuti);

Le rappresentazioni audio a frequenze di campionamento dell'udito umano, rendono le sequenze da modellare diurne, una tipica canzone di 4 minuti in qualità CD (44 kHz, 16-bit) conterrebbe oltre 10 milioni di *timesteps*.

I progetti Magenta di Google (2016 - 2020), MuseNet di OpenAI (2019), Flow Machines di Sony CSL (2012), il Universal Music Translator di Facebook AI Research (2019) e Deep Composer di Amazon (2019) sono alcuni degli sforzi operati dalle grandi aziende tecnologiche, potendo queste far ricorso ad ampi dataset per il training dei modelli.

Tuttavia i metodi attuali risultano *rigidi*, la generazione dipende fortemente l'architettura fissata al momento dell'addestramento.  
(Una rete neurale allenata da un dataset di canzoni pop, non è in grado di generare una melodia blues).

## **Struttura della tesi**

Nella testi vengono riportate inizialmente le nozioni di base utili alla comprensione degli argomenti trattati.

Vengono poi discussi i principali metodi di generazione musicale correnti, e gli studi associati.

Infine viene presentato un algoritmo genetico, per la generazione di una sequenza musicale, discutendone i risultati conseguiti.

# Chapter 2

## Nozioni di Base

### 2.1 nozioni algoritmiche

#### 2.1.1 fitness-function

La fitness function fornisce un punteggio numerico che rappresenta l'efficacia di una soluzione candidata.

Questo punteggio guida l'algoritmo nel processo di selezione e ottimizzazione. L'obiettivo è massimizzare o minimizzare la fitness function, a seconda del problema.

Negli algoritmi genetici, le soluzioni con punteggi di fitness più alti (*genitori*) hanno una probabilità maggiore di essere selezionate per la riproduzione e la generazione di nuove soluzioni (*prole*).

#### 2.1.2 modello markoviano

In un modello markoviano, un nuovo stato dipende esclusivamente dallo stato corrente, senza tener conto degli stati passati.

Formalmente, questa proprietà si esprime come:

$$P(X_{n+1} = x \mid X_n = x_n, X_{n-1} = x_{n-1}, \dots, X_0 = x_0) = P(X_{n+1} = x \mid X_n = x_n)$$

Dove  $X_i$  rappresenta lo stato del sistema al tempo  $i$ .

Il modello è costituito da un insieme finito e numerabile di stati. Le probabilità di transizioni tra stati possono essere rappresentate in una matrice.

Per una catena di Markov con  $N$  stati, la matrice di transizione  $P$  è una matrice  $NN$  dove l'elemento  $P_{ij}$  rappresenta la probabilità di transizione dallo stato  $i$  allo stato  $j$ .

Le righe della matrice devono sommare a 1:

$$\sum_{j=1}^N P_{ij} = 1$$

Un modello markoviano raggiunge una distribuzione stazionaria, quando le probabilità di transizione non mutano nel tempo, convergendo ad un punto di equilibrio.

Nel caso specifico della tesi questo approccio è adottato per generare una consecuzione di note e accordi.

### 2.1.3 artificial neural network

Una Artificial Neural Network (ANN), o rete neurale artificiale, è un modello computazionale ispirato alla struttura e al funzionamento del cervello umano, progettato per riconoscere schemi complessi e prendere decisioni basate su input di dati.

Le ANNs sono particolarmente utili in vari campi dell'intelligenza artificiale, come il riconoscimento di immagini, il riconoscimento vocale, l'elaborazione del linguaggio naturale ed altri.

#### Funzionamento di una ANN

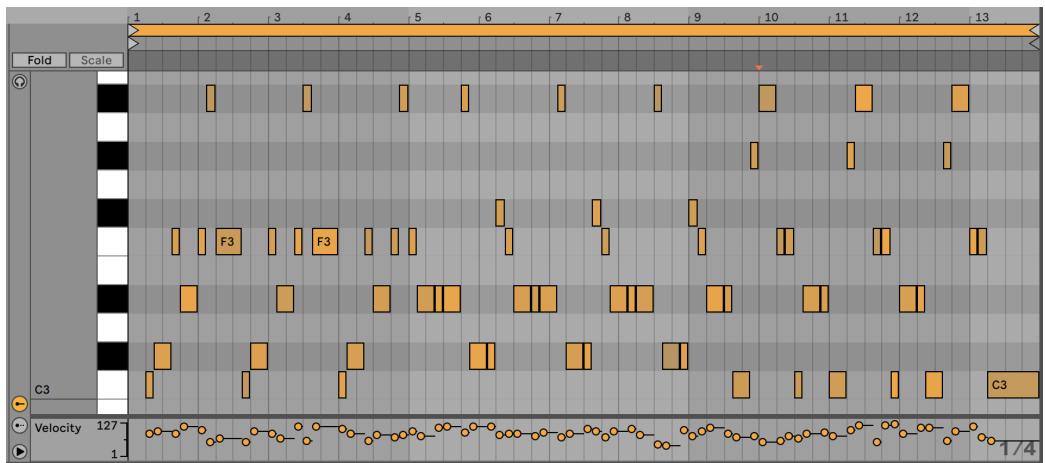
**Forward Propagation:** gli input vengono trasmessi attraverso i vari strati della rete, moltiplicando i valori in input per i pesi delle connessioni e applicando una funzione di attivazione. Questo processo persegue fino a raggiungere il livello di output, restituendo una previsione.

**Backward Propagation:** l'errore tra la previsione e il valore reale viene calcolato e usato dalla rete per aggiornarne i pesi. Viene usato il calcolo del gradiente per minimizzare l'errore, adattando i pesi in modo iterativo.

### 2.1.4 file MIDI

Un file MIDI (Musical Instrument Digital Interface) è un formato di file standardizzato nella rappresentazione di **informazioni musicali**.

Invece di memorizzare il suono, come i file audio, nel formato MIDI si memorizzano i dati relativi alle note, gli strumenti, alle dinamiche e ad altre istruzioni musicali, che possono essere riprodotte da dispositivi elettronici compatibili, come sintetizzatori, sequencer, computer e software di notazione musicale.



## 2.2 nozioni musicali



Figure 2.1: generica linea melodica di uno spartito

### 2.2.1 caratteristiche di una nota musicale

#### **Altezza (Pitch):**

L'altezza di una nota è la frequenza del suono prodotto, determinando quanto questo sia acuto o grave.

Nel pentagramma sono espresse dalla posizione verticale.

Le note musicali fondamentali nell'ottava sono Do(C), Re(D), Mi(E), Fa(F), Sol(G), La(A), Si(B).

#### **Durata:**

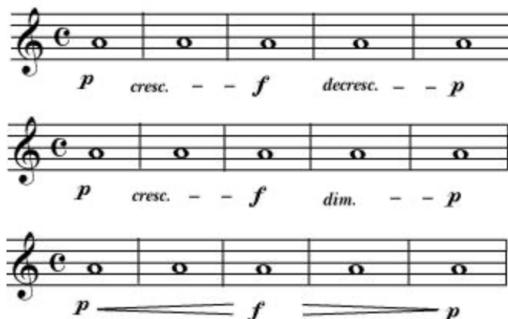
La durata di una nota indica per quanto tempo viene suonata.

Di seguito un'immagine esemplificativa:

Segno grafico	Nome	Durata
○	<i>intero</i> (o semibreve)	<b>4/4</b>
—	<i>metà</i> (o minima)	<b>2/4</b>
♩	<i>quarto</i> (o semiminima)	<b>1/4</b>
♪	<i>ottavo</i> (o croma)	<b>1/8</b>
♫	<i>sedicesimo</i> (o semicroma)	<b>1/16</b>
♬	<i>trentaduesimo</i> (o biscroma)	<b>1/32</b>
♫	<i>sessantaquattresimo</i> (o semibiscroma)	<b>1/64</b>

### Intensità (Velocity):

L'intensità di una nota si riferisce a quanto *forte* o *piano* viene suonata. Sono indicazioni come *p* (piano) per suonare piano, *f* (forte) per suonare forte, *mf* (mezzo forte), *mp* (mezzo piano), e altre variazioni come crescendo (aumentando l'intensità) e decrescendo (diminuendo l'intensità).



### 2.2.2 scala armonica

La scala maggiore è una tra le più comuni e ha un suono generalmente percepito come *felice* e *luminoso*.

La scala maggiore segue uno schema specifico di toni (T) e semitonni (S). Lo schema è:

T-T-S-T-T-T-S

Ad esempio, la scala di Do maggiore è:

Do(C)-Re(D)-Mi(E)-Fa(F)-Sol(G)-La(A)-Si(B)-Do(C)



La scala minore ha un suono generalmente percepito come *triste* o *scuro*. Segue uno schema diverso:

T-S-T-T-S-T-T

Ad esempio, la scala di La minore naturale è:

La(A)-Si(B)-Do(C)-Re(D)-Mi(E)-Fa(F)-Sol(G)-La(A)

### 2.2.3 melodia

Una melodia è una sequenza di note musicali, percepita come un'unità singola e coerente.

Essa è una componente fondamentale della musica, essendo spesso la parte più riconoscibile di un brano.

Una melodia è composta da una serie di note che vengono suonate in successione. Queste possono variare in altezza (pitch), durata (rhythm), intensità (velocity).

Il **contorno melodico** si riferisce al profilo delle altezze, ovvero come le note *ascendono* e *calano* nel corso della melodia. Questo può includere salti, scale, ripetizioni e altre variazioni.

La melodia è spesso **basata su una scala musicale**, come una scala maggiore o minore, fornendo un quadro armonico e tonale.

### 2.2.4 accordi

Gli accordi sono combinazioni di tre o più note suonate simultaneamente o in sequenza arpeggiata.

Gli accordi sono fondamentali nella musica, fornendo la struttura armonica su cui si basano melodie e progressioni.

Possono essere maggiori o minori.

Un **problema fondamentale** nella generazione di musica tramite algoritmi,

è la **scelta nella sequenza di una progressione di accordi**, che descrive lo stile musicale di una canzone e la sua *orecchiabilità*.

Esempi di Progressioni di Accordi usati comunemente in musica:

Progressione I-IV-V-I in Do Maggiore:

Accordi: Do(C)-Fa(F)-Sol(G)-Do(C)

Uso comune nelle canzoni pop e rock (*il giro di Do*)

Progressione ii-V-I in Jazz:

Accordi: Re min. settima(Dm7)-Sol settima(G7)-Do mag. settima(Cmaj7)

# Chapter 3

## Metodi di generazione

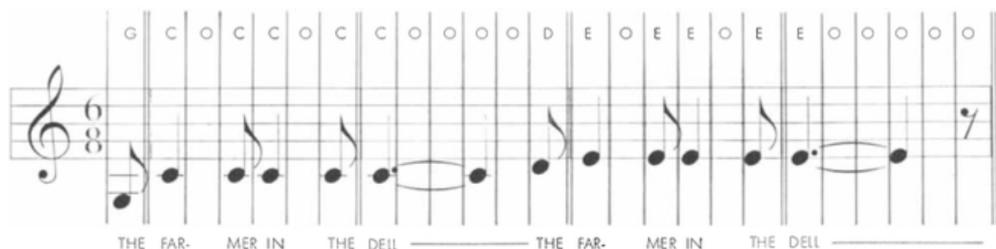
### 3.1 generazione melodia

Quando si considera il problema della generazione musicale, la forma più semplice di esercizio è la composizione di **melodie monofoniche**.

I sistemi generativi dipendono da una *fitness-function* che valuta le sequenze generate.

Tale funzione di fitness è spesso basata sulla somiglianza con un determinato corpus, stile o brano, o dipende in generale dalle regole teoriche musicali.

I primi tentativi di generare melodie con i computer risalgono al 1956, quando **Pinkerton** costruì un *modello markoviano*, il **Banal Tune-Maker**, basato su un corpus di 39 semplici filastrocche, tuttavia il modello tendeva ad essere ripetitivo.



ANALYSIS OF TUNES begins by translating their notes into the usual alphabetical symbols, using "O" to represent a rest or the holdover of a note beyond its initial beat. The colored vertical lines mark off measures (*double lines*) and beats (*single lines*).

Un altro problema nell'uso delle catene di Markov, risiede nell'eccessiva somiglianza con la melodia originale.

Il compromesso tra la composizione di brani simili a lavori esistenti, contro

la necessità di doverne creare di nuovi e creativi, è difficile da ricercare. Si ricorda a tal proposito una citazione di Stravinsky, celebre compositore Russo:

*“i buoni compositori copiano, i grandi compositori rubano”.*

Riferendosi a quest’idea tuttavia le macchine non hanno ancora la capacità di distinguere tra il furto astuto e il plagio totale.

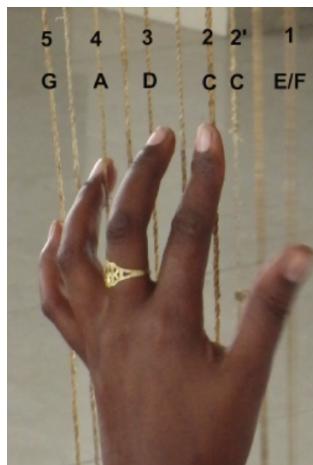
Con quest’intuizione è stata poi introdotta la variabile *MaxOrder*, ad indicare il massimo ordine di sottosequenze simili consentito in una sequenza generata, con l’intento di limitare le ripetizioni eccessive nel materiale.

Esistono diverse tipologie di tali vincoli di controllo, ad esempio la volontà che una sequenza sia globalmente ascendente, o che segua invece un intorno di note arbitrario.

**Davismoon ed Eccles** (2010) sono stati alcuni dei primi ricercatori a inquadrare la generazione musicale come un problema di ottimizzazione in un modello di Markov.

Per valutare la musica generata, il loro sistema costruisce un secondo modello, riducendo poi al minimo la distanza euclidea tra quello originale e il nuovo.

Lo studio si è basato sulle composizioni melodiche tipiche dei *bagana* (molto semplici e ripetitive), una lira a dieci corde suonata dagli Amhara, abitanti dell’Etiopia centrale e settentrionale.



Comporre una melodia monofonica può sembrare un compito semplice rispetto alla partitura di un’intera sinfonia.

Tuttavia, le melodie sono più che semplici movimenti tra le note, normalmente possiedono una **struttura a lungo termine**.

Negli ultimi anni, alcune ricerche hanno dimostrato l’efficacia dell’utilizzo di

tecniche come il **deep learning** e **ANN** per rafforzare una consecuzione coerente.

Sebbene esista molto lavoro sulla generazione di accordi data una melodia, alcuni si concentrano viceversa, sulla generazione di una melodia che si adatti a una sequenza di accordi.

**Moorer** (1972), ad esempio ha adottato quest'ultimo approccio.

Le note della melodia sono limitate solo a quelle dell'accordo corrispondente in un dato momento.

Ad ogni punto si decide, sulla base di un modello markoviano, di invertire i frammenti melodici basati sull'accordo, oppure di copiarne da quello precedente.

Le brevi melodie risultanti tuttavia hanno un suono estraneo, l'approccio melodico non è quello utilizzato dagli esseri umani, non discriminando le sequenze *gradevoli* da quelle *non familiari*.

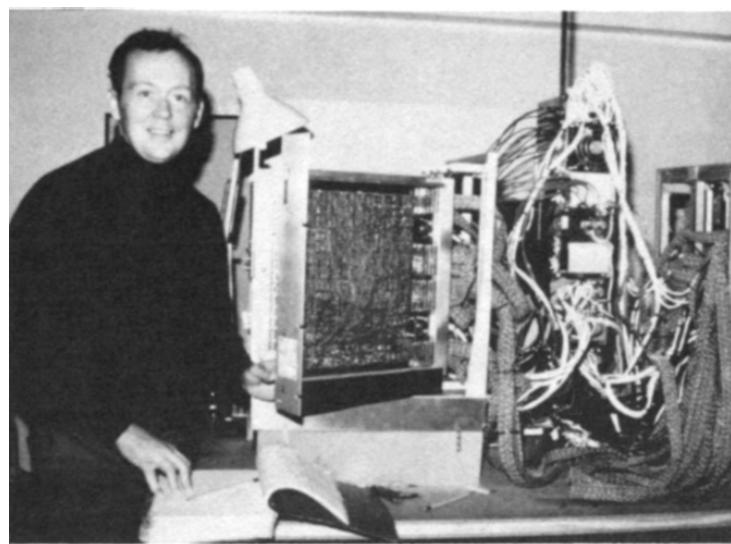


Figure 3.1: James A. Moorer con il Lucasfilm Audio Signal Processor.

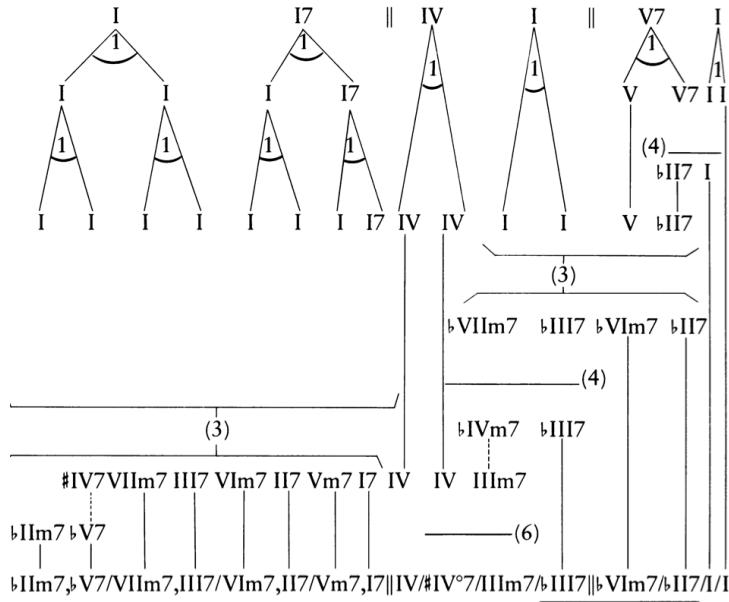
### 3.1.1 armonia

L'armonia è definita da una relazione verticale tra note, suonate simultaneamente, e orizzontale, la loro relazione nel tempo.

Il sistema deve generare sequenze riconoscibili nel genere musicale, senza mai essere *sostanzialmente uguali* a gli esempi di riferimento.

La maggior parte degli studi adotta approcci basati su un training set per determinare possibili accordi in un dato segmento melodico, garantendo una corretta transizione tra questi.

**Steedman** (1984) studiando melodie blues con lunghezza di 12 battute, e definendo un piccolo insieme di regole, è riuscito a produrre progressioni di accordi tipiche dello stile.



**Lee e Jang** (2004) hanno utilizzato il modello di Markov, integrato con programmazione dinamica, per determinare il pattern armonico di una melodia canticchiata da un utente; le probabilità nella matrice di transizione, tra le note correnti, sono state apprese da un set di 150 canzoni.

**Simon** (2008) ha adottato un approccio simile; allenando il proprio sistema su 298 brani di vari generi come jazz, rock, pop, blues ed altri.

Questi ultimi due sistemi vengono valutati tramite **feedback soggettivo**, cioè da un umano e non da una qualche fitness-function, a seguito di sessioni d'ascolto.

Uno svantaggio di questo approccio è che le sequenze di accordi generate tendono ad essere **generiche e indistinte nello stile**.

L'uso dell'uomo come funzione di fitness è certamente valido, ma costringe a

delle forti limitazioni:

se si vuole valutare immagini, queste possono essere visualizzate contemporaneamente, e molto rapidamente, mentre la musica è un fenomeno temporale, e quindi tutti i campioni devono essere ascoltati in sequenza, uno dopo l'altro.

Ciò si traduce in un costo *computazionale* elevato:

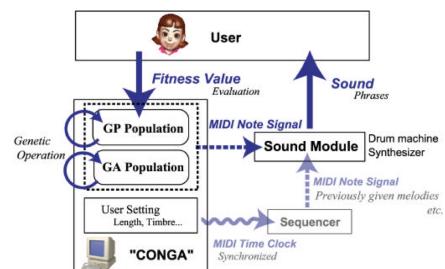
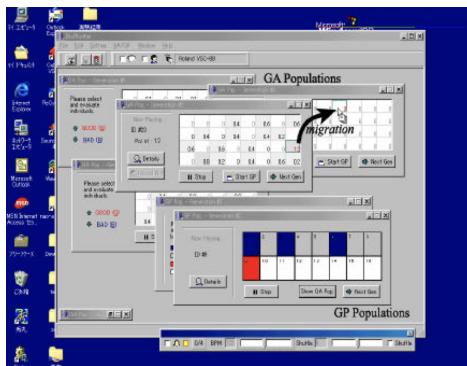
### 3.1.2 ritmo

Nei sistemi di generazione musicale il ritmo dipende dalla durata di ciascuna nota.

In generale, esistono meno sistemi di *generazione ritmica* rispetto ai sistemi melodici.

**Tokui e Iba** (2000) hanno proposto il sistema CONGA, che usa algoritmi genetici per produrre sequenze ritmiche, usando l'utente come funzione fitness.

Brevi frammenti di schemi ritmici formano gli elementi cromosomici nell'algoritmo; subendo poi trasformazioni come il crossover e mutazioni.



**Ariza** (2002), ha generato ritmiche derivanti da variazioni genetiche di un dataset, ma la funzione fitness è consistita nel calcolare la distanza tra la generazione e l'originale, attraverso differenti misure di distanza.

## 3.2 algoritmo genetico

### 3.2.1 musica evolutiva

L'applicazione dei metodi di *computazione evolutiva* nel campo musicale ha cominciato ad emergere all'inizio degli anni '90.

Ci sono tre criteri fondamentali da considerare:

*il domino del problema,*  
*la rappresentazione individuale*  
*e la misura dell'idoneità.*

Nella ricerca è fondamentale la codifica del dominio, e quindi i limiti o lo stile musicale, entro cui l'algoritmo ricerca.

Si definisce esattamente quale tipo di *musica* si desidera sviluppare; si è intenzionati a creare melodie, armonizzazioni o progressioni di accordi?

Nella rappresentazione individuale ci si chiede come rappresentare la musica: audio, spartiti stampati, messaggi MIDI?

Comunemente, per una più facile gestione dei dati, si ricorre al formato MIDI, dove il genoma contiene valori di pitch e durate.

Infine la questione della misura di idoneità: supponendo che due individui rappresentino brani musicali nel dominio desiderato, cosa rende uno *migliore* dell'altro?

### 3.2.2 prime applicazioni degli alg. gen. alla musica

**Horner e Goldberg** furono tra i primi ad applicare un algoritmo genetico al processo di composizione musicale considerando il problema della *transizione tematica*.

È stata considerata la trasformazione di una frase melodica in un'altra, dove i genomi rappresentavano la serie di transizioni necessarie.

La fitness-function misurava quanto bene il modello finale corrispondesse al modello desiderato.

Il sistema tuttavia è stato utilizzato per creare *transizioni*, piuttosto di creare effettivamente nuova musica.

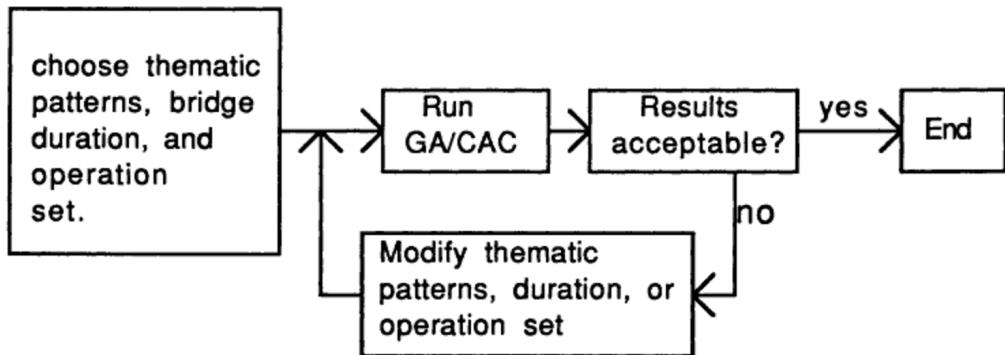


Figure 3.2: schema del processo usato da Horner e Goldberg

**John Biles** ha creato il sistema GenJam che utilizza un algoritmo genetico per evolvere assoli jazz.

GenJam ha utilizzato due popolazioni indipendenti: una per le misure di fitness ed una per la generazione di sequenze.

Questo ha permesso una valutazione dei fraseggi **in tempo reale**, valutandoli come *buoni* o *cattivi*, permettendo a Biles di suonare la sua tromba accompagnato dall'algoritmo.



Figure 3.3: John Biles suona assieme a GenJam

In studi successivi, si è **addestrata una ANN a fungere da funzione**

**di fitness**, eliminando così la necessità dell'ascoltatore umano.

Tuttavia, si è riscontrato che i **risultati non erano comunque soddisfacenti**, stabilendo che gli esseri umani ascoltano e sperimentano la musica in modi complessi e sottili, che non possono colti da modelli statistici come le ANN.

In uno studio che ha fatto uso di ANN come fitness-function, gli autori hanno limitato il loro dominio, considerando solo armonie basate su tre accordi nella tonalità di Do maggiore, e creando solo melodie di quattro battute di durata.

La composizione è stata poi suddivisa in blocchi, e per ciascuno di questi un algoritmo genetico ha generato una popolazione. Una ANN addestrata precedentemente da esempi forniti dall'utente, ha quindi giudicato i genomi candidati.

Gli autori hanno riconosciuto che questa combinazione di alg. gen. e ANN potrebbe produrre delle pseudo composizioni, ma **le restrizioni applicate al dominio di addestramento hanno limitato fortemente il modello**.

In generale si riscontra che, indipendentemente da quanto complessa o sfaccettata sia una data misura di fitness, è sicuramente *impossibile dire di trovare la misura definitiva della musica*.

### 3.3 L-System

Gli L-system (o Lindenmayer system) sono un tipo di sistema formale utilizzato per modellare la crescita di *piantericorsive*.

Sono stati introdotti da Aristid Lindenmayer nel 1968 e sono composti da un alfabeto di simboli che, usati assieme ad un set di *regole di produzione*, vengono impiegati per generare stringhe, a loro volta contenenti altri simboli, applicando appunto un approccio ricorsivo.

**Alfabeto:** I simboli dell'alfabeto possono rappresentare note, durate, dinamiche, articolazioni, e altri elementi musicali.

**Assioma:** La stringa iniziale può essere una sequenza di note.

**Le regole** descrivono come trasformare ogni simbolo in una nuova sequenza.

Alfabeto: A, B

Assioma: A

Regole di produzione:

A → AB

$B \rightarrow A$

In questo esempio:

1. Partiamo dall'assioma **A**.
2. Applichiamo le regole:

**A** diventa **AB**

**AB** diventa **ABA**

**ABA** diventa **ABAAB**

e così via.

Ora, mappiamo i simboli alle note musicali:

**A** = Do

**B** = Re

La sequenza generata può essere interpretata come una melodia: Do ,Do-Re ,Do-Re-Do , Do-Re-Do-Do-Re, e così via.

Gli L-system così permettono di generare pattern musicali ricorsivi e **auto-simili** che possono essere sia prevedibili ma anche variabili.

Nel paper *Growing music*, **Peter Worth** e **Susan Stepney**, hanno sperimentato in modo più approfondito questo sistema.

Gli L-System permettono una maggiore sensibilità per il *contesto*, cioè il rapporto tra le note lungo la sequenza, facendo crescere conseguentemente la pianta in modo diverso in base a gli elementi nell'intorno.

Questo potrebbe essere utile in musica, quando ad esempio il pezzo generato potrebbe dover raggiungere un climax o interrompersi bruscamente.

Per far sì che questo avvenga, la regola di produzione per un L-System viene applicata al simbolo solo se compare inclusa tra altri specifici.

La notazione **A < B > C** indica che agisce se la stringa **B** appare con **A** alla sinistra e **C** a destra.

Viene considerato il seguente L-System *sensibile al contesto*:

$\omega:$	F1F1F1
$p_1:$	$0 < 0 > 0 \rightarrow 0$
$p_2:$	$0 < 0 > 1 \rightarrow 1[+F1F1]$
$p_3:$	$0 < 1 > 0 \rightarrow 1$
$p_4:$	$0 < 1 > 1 \rightarrow 1$
$p_5:$	$1 < 0 > 0 \rightarrow 0$
$p_6:$	$1 < 0 > 1 \rightarrow 1F1$
$p_7:$	$1 < 1 > 0 \rightarrow 0$
$p_8:$	$1 < 1 > 1 \rightarrow 0$
$p_9:$	$+ \rightarrow -$
$p_{10}:$	$- \rightarrow +$



Questa melodia, e altre derivate in modo simile, suonano *imprevedibilmente*; ricordando un assolo jazz.

Le note non rientrano bene nella cadenza 4/4 in una partitura, perché molte hanno una durata irregolare.

Tuttavia la melodia ritorna sempre a un motivo o una frase principale, a volte trasposto o suonato in un punto diverso della battuta.

Ad esempio, nella partitura sopra, una serie di note nella prima battuta è ripetuta nella nona, trasposte in avanti per un quarto di tempo e aumentate di 2 semitonni.

Questo tipo di simmetria rispecchia la struttura ricorsiva ad albero da cui ha origine.

## Chapter 4

# Progetto di musica genetica

Viene ora introdotto un progetto di algoritmo genetico utile ad esemplificare fin quanto detto.

All'interno del progetto la funzione:

```
# Main function for running all helper functions and handling user
input.
def genetic-algorithm(key, root, tempo, rhythm)
```

scrive su disco 10 genotipi di sequenze di note, in formato MIDI, come risultato dell'algoritmo genetico.

La funzione prende in input dall'utente:

**la scala** (key) utilizzata nella scelta delle note, può essere *major* o *minor*.  
**La nota fondamentale** che indica la tonalità della canzone (root), e sulla quale verrà costruita la scala (es: C, C#, D, D#, E, F, F#, G, G#, A, A#, B)

**Il numero di battiti al minuto** (bpm), ovvero la velocità di esecuzione, della composizione

**il ritmo della composizione**, ovvero la durata delle note, a scelta tra quattro predefiniti (*rock*, *jazz*, *dance*, *bossa nova*).

All'interno della funzione troviamo

```
# Build the scale based on the root note and chosen scale type
scale = buildScale(root, key)
```

Restituisce un array, dove ogni numero rappresenta il codice MIDI per ogni nota.

L'array rappresenta la scala, maggiore o minore, costruita partendo dalla

nota fondamentale scelta.

Viene eseguita poi l'evoluzione vera e propria, attraverso la crescita del genoma.

#### 4.0.1 Run-Evolution

```
# Run the genetic algorithm to evolve a melody using the specified mutation
rate and scale
evolvedMelody = runEvolution(MUTATION_RATE, scale)
```

`runEvolution` implementa un algoritmo genetico che migliora in modo iterativo la popolazione di genomi musicali attraverso selezione, crossover e mutazione, l'esecuzione avviene finché non viene raggiunto un genoma con il punteggio `MAX-FITNESS` specificato, o il numero massimo di generazioni consentito, `MAX_GENERATIONS`.

Viene inizialmente creata un popolazione, inizializzata con metodo random. La popolazione è composta da 10 genomi.

Ogni genoma è un elenco di 8 battute, ciascuna contenente 8 note, tutte scelte casualmente dalla scala specificata.

Il codice che segue rappresenta lo sviluppo dell'algoritmo genetico utilizzato.

```
# Iterate for a maximum number of generations
for _ in range(MAX_GENERATIONS):

    # Sort the population based on fitness scores in descending order
    population = sorted(population, key=fitnessFunction, reverse=True)

    # Select the top 2 genomes (the fittest) to carry over to the next
    # generation
    nextGeneration = population[:2]

    # Generate the rest of the next generation through crossover and mutation
    for _ in range(len(population) // 2 - 1):

        # Select two parent genomes based on their fitness
        parentA, parentB = selectParents(population)
        # Perform crossover to produce two child genomes
        childA, childB = multipointCrossover(parentA, parentB)
        # Mutate the child genomes and add them to the next generation
        nextGeneration += [mutateGenome(childA, mutationRate, scale),
                           mutateGenome(childB, mutationRate, scale)]
```

```
# Update the population with the new generation
population = nextGeneration
```

Per un numero massimo di generazioni prefissato, vengono create popolazioni di genomi.

Successivamente alla prima inizializzazione randomica, viene applicata la Fitness-Function, che assegna ad ogni genoma un peso/punteggio basandosi sui propri criteri di valutazione (vedremo in seguito).

Nella successiva popolazione vengono mantenuti i primi due genomi con il punteggio più alto, mentre per i restanti 8 vengono selezionati di volta in volta due genitori, da cui, attraverso un (multipoint) crossover, danno luogo a due figli, i quali subiscono una fase di mutazione.

I processi di crossover e mutazione sono definiti di *exploration*, perché servono ad esplorare soluzioni che i due migliori genomi non sono riusciti ad ottenere.

Viceversa i due genomi mantenuti con il punteggio più alto, che si tramandano da una popolazione alla successiva, servono a garantire la migliore performance ad ogni generazione, qualora i genomi di exploration non diano risultati soddisfacenti.

Quando avviene che in una popolazione, i figli generati ottengano un punteggio superiore ai due migliori correnti, questi vengono dunque sostituiti.

Al termine del processo, viene scritta su disco l'ultima popolazione costituita da 10 genomi.

Il migliore tra questi (con punteggio più alto) viene chiamato `best_genome.mid`.

#### 4.0.2 fitness-function

La fitness function calcola il punteggio finale ponderando e sommando i punteggi dei componenti per **fluidità, ritmo e armonia**, restituendo il risultato. Vengono inizializzati i pesi e i punteggi.

```
# Weights for the different fitness components
smoothnessWeight = 15
restWeight = 5
harmonyWeight = 20

# Initialize fitness scores
smoothnessScore = 0
restScore = 0
harmonyScore = 0
```

L' `harmonyscore` riporta il punteggio di armonizzazione tra le note, ovvero come queste suonino *bene* tra loro.

Di seguito un dizionario in cui troviamo come chiave, la distanza tra la nota corrente e la precedente, e come valore, il punteggio assegnato a tale distanza armonica.

```
# Harmony intervals table for scoring harmony
harmonyIntervalsTable = {0: -20, 1: 5, 2: 5, 3: 50, 4: 50, 5: 30,
6: -10, 7: 50, 8: 10, 9: 40, 10: -2, 11: -2, 12: 10, 13: -5,
14: 5, 15: 5, 16: 50, 17: 50, 18: 30, 19: -10, 20: 50, 21: 10,
22: 40, 23: -2, 24: -2, 25: 10}
```

- Intervalli consonanti perfetti (0, 7, 12, 19, 24 semitonii): Questi intervalli, che corrispondono alle ottave e quinte perfette (e le loro ripetizioni all'ottava superiore), hanno punteggi molto alti (50 o 10), ad indicare di essere *armoniosi*.

- 0 (Unisono): 50
- 7 (Quinta giusta): 50
- 12 (Ottava giusta): 10
- 19 (Unisono all'ottava superiore): -10
- 24 (Due ottave giuste): 10

- Intervalli consonanti maggiori (4, 9, 16, 21 semitonii): Questi includono la terza maggiore e la sesta maggiore, che sono generalmente considerati molto armoniosi.

- Intervalli consonanti minori (3, 8, 17, 22 semitonii): Questi includono la terza minore e la sesta minore.

- Intervalli dissonanti (1, 2, 6, 10, 11, 13, 23 semitonii): Questi includono la seconda minore, la settima minore e la settima maggiore, che sono considerati meno armoniosi o dissonanti.

- 1 (Seconda minore): 5
- 2 (Seconda maggiore): 5
- 6 (Tritono): -10
- 10 (Settima minore): -2
- 11 (Settima maggiore): -2
- 13 (Ottava aumentata): -5
- 23 (Nona aumentata): -2

**Calcolo del punteggio di fluidità (smoothnessScore):**

Se la differenza tra le note (`noteDifference`) è 0 (unisono), il punteggio di fluidità viene diviso per 10, penalizzando fortemente la ripetizione della stessa nota.

Se la differenza è minore o uguale a 2 semitonni, aumenta il punteggio di fluidità di 1, incentivando piccoli movimenti tra le note.

Se la differenza è di 11 semitonni (quasi un'ottava), il punteggio di fluidità viene diviso per 2, penalizzando questo ampio intervallo.

Per altri intervalli, il punteggio di fluidità aumenta di  $1 / \text{noteDifference}$ , favorendo movimenti più piccoli tra le note.

```
if note is None and prevNote is None:  
    consecutiveRests += 1
```

Se entrambe `note` e `prevNote` sono `None`, significa che si è verificata una pausa consecutiva: Incrementa il contatore `consecutiveRests`.

```
if numRests * 10 <= len(flatten(genome)):  
    restScore += 10
```

Se il numero di pause (`numRests`) moltiplicato per 10 è inferiore o uguale alla lunghezza della sequenza musicale (`flatten(genome)`), aumenta il punteggio ritmico (`restScore`) di 10.

Questo premia una quantità ragionevole di pause nel contesto della lunghezza totale del genoma musicale, suggerendo che alcune pause sono considerate musicalmente desiderabili.

```
if consecutiveRests:  
    restScore -= (consecutiveRests * 10)
```

Se ci sono pause consecutive (`consecutiveRests > 0`), penalizza il punteggio ritmico diminuendo `restScore` di `consecutiveRests * 10`.

Questo penalizza pesantemente le pause consecutive, suggerendo che una lunga serie è considerata indesiderabile dal punto di vista ritmico.

```
fitness_weight_genome = (smoothnessScore * smoothnessWeight) + (restScore  
* restWeight) + (harmonyScore * harmonyWeight)
```

Il punteggio di fitness finale (`fitness_weight_genome`) è calcolato come la somma ponderata dei punteggi ottenuti.

Ogni punteggio componente (`smoothnessScore`, `restScore`, `harmonyScore`) viene moltiplicato per il rispettivo peso (`smoothnessWeight`, `restWeight`,

`harmonyWeight`).

La somma dei prodotti risultanti rappresenta il punteggio di fitness finale del genoma.

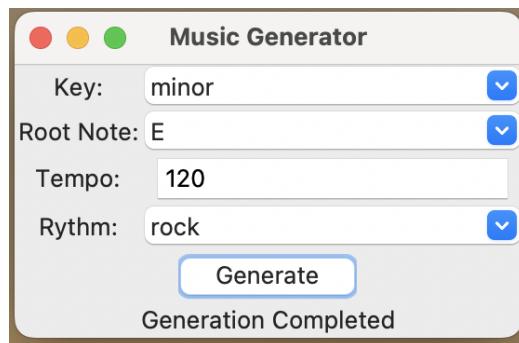
I pesi (`smoothnessWeight`, `restWeight`, `harmonyWeight`) permettono di enfatizzare o de-enfatizzare specifici aspetti della valutazione a seconda degli obiettivi desiderati per la composizione musicale.

Ad esempio, aumentando il peso della fluidità, si darà maggiore importanza alle transizioni tra le note, mentre aumentando il peso dell'armonia, si enfatizzerà la qualità armonica della melodia.

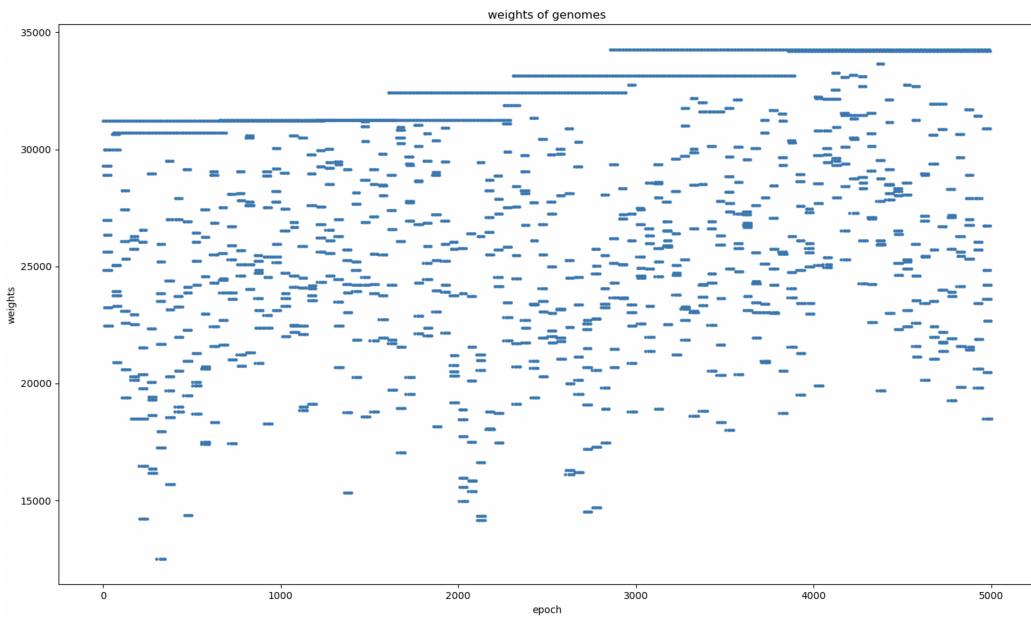
#### 4.0.3 risultati

Eseguiamo dunque il programma.

Sono riportati le opzioni scelte per una generazione musicale esemplificativa.



Al termine dell'esecuzione vengono stampati i risultati dei punteggi ottenuti dalla fitness function per i genomi di ogni popolazione.



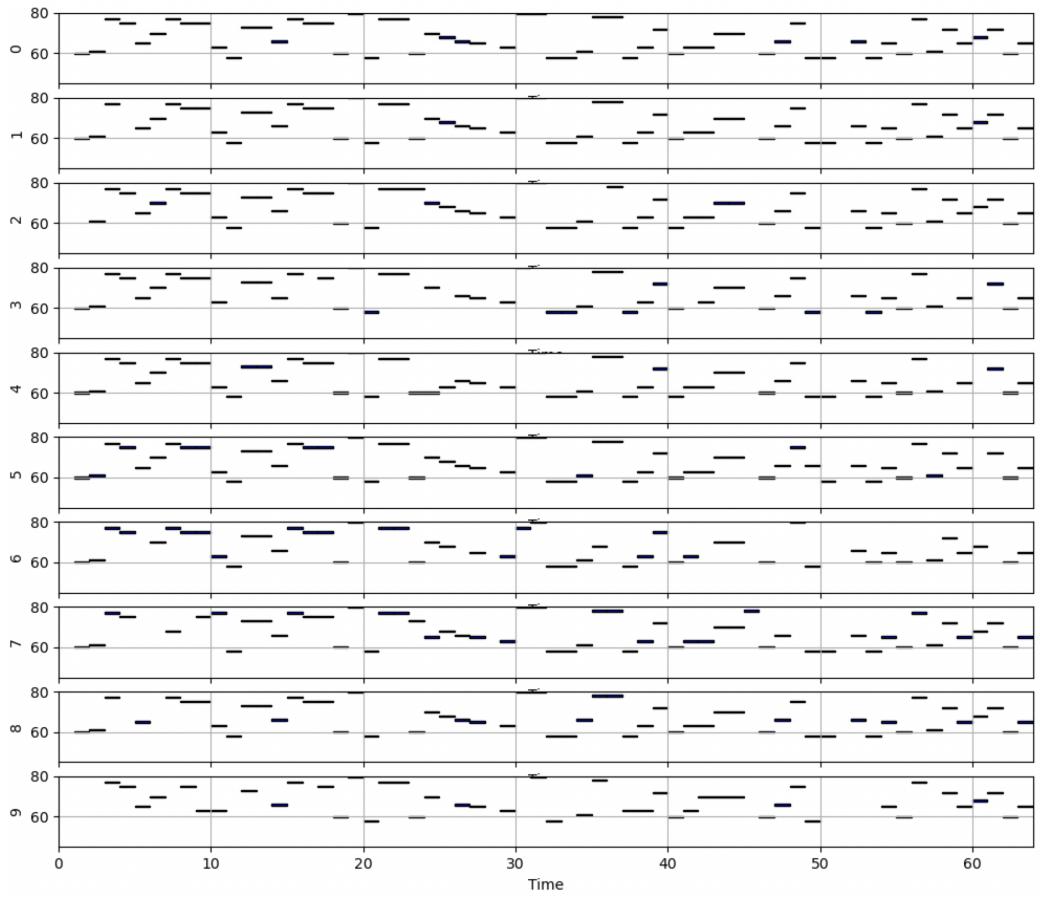
Le linee che si mantengono costanti sono i punteggi dei primi due genomi (con punteggio più alto ad ogni generazione) che nell'algoritmo scelto vengono tramandati senza alcuna alterazione (crossover, mutation).

Quando *nasce* un nuovo genoma che si verifica avere un punteggio più alto dei primi due, questo fissa un nuovo livello di punteggio, superiore ai precedenti, che viene mantenuto finché ciò non accade nuovamente.

Questa procedura garantisce un costante aumento del punteggio e un conseguente miglioramento tra le popolazioni.

La nuvola di punti che appaiono al di sotto, con punteggio inferiore, descrive il set di genomi soggetti a crossover e mutazioni.

Al termine della procedura evolutiva, vengono stampati i 10 genomi appartenenti all'ultima popolazione. (Le note nella stampa hanno durata unitaria)



La stampa rappresenta 10 file midi, dal *migliore* (numero 0) al *peggiore* (numero 9) secondo la nostra fitness-function.

Le note sono rappresentate dalle linee orizzontali che appaiono a diverse altezze di pitch, indicati nell'asse delle ordinate.

Le note sono sequenziali, appaiono dunque in momenti consecutivi.

Si ricorda che l'algoritmo genera una sequenza monofonica, infatti non appare più di una nota contemporaneamente.

Le sequenze vengono salvate su disco.

Il genoma migliore prende il nome di `best-genome.mid`, mentre ai restanti viene assegnato un numero in ordine decrescente in punteggio (`genome-0.mid`, ..., `genome-8.mid`).

Ad un primo ascolto di `best-genome.mid`, percepiamo un movimento coerente nella teoria, ordinato nel ritmo, ma *non armonioso* nel tempo.

Non viene percepita infatti alcuna vera struttura compositiva, questo perché

la fitness function si è preoccupata di fare confronti solo tra una nota e la precedente, e non tra tutte quelle nella sequenza.

Non esiste dunque una vera struttura a lungo termine, una struttura narrativa, avente inizio svolgimento e fine, propria invece di qualsiasi genere musicale.

Non è presente nemmeno il classico schema delle canzoni pop:  
*verso-ritornello-verso-ritornello-bridge-ritornello*.

Per ovviare a questo difetto si procedere con un'operazione naive ma efficace:

Si decide di generare una nuova sequenza, chiamata `repeated-melody.mid`, la quale ripete per 4 volte le prime 3 battute prese dal `best-genome.mid`.

La conseguenza risulta essere *ripetitiva* e dunque *intrinsecamente* più regolare, oltre a essere, per il nostro orecchio, più *confortevole*.

Abbiamo dunque ottenuto un risultato teorico *accettabile* e *coerente*.

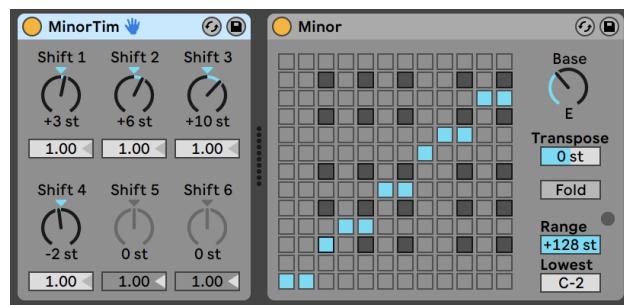
Possiamo ottenere di più nella pratica?

Si.

Importando il file midi (`repeated-melody.mid`) in una DAW (Digital Audio Workstation, in questo caso è stato utilizzato Ableton) possiamo apportare ulteriori modifiche.

Viene assegnata alla traccia lo strumento *Grand Piano*.

Su questa vengono applicate le seguenti opzioni



*MinorTim* permette di generare più note, ad una certa distanza in semitonni, partendo dalla nota corrente.

Nel caso specifico si costruisce un accordo minore ad ogni nota.

Un primo ascolto risulta essere poco piacevole, perché le note generate non considerano la scala di riferimento.

È necessario dunque usare lo strumento *Minor* per ancorare ogni nota in una posizione coerente.

Il risultato è decisamente più armonico, seppur tuttavia ancor troppo *piatto*

con nessuna variazione nell'esecuzione.

Per far fronte a questa *freddezza* ed aumentare la dinamicità, è possibile randomizzare in modo automatico, i valori di velocity nella sequenza per ciascuna nota.

Un nuovo ascolto ci rimanda ad un qualche compositore jazz, immerso nel suo fiume di sentimenti, che vuole mantenere, per piacere della clientela, un ritmo e melodia regolari.

Per rendere il risultato ancor più gradevole, e musicalmente completo, possiamo aggiungere un elemento percussivo, come una batteria.

Per il risultato ottenuto è stato utilizzato il plug-in Magenta sviluppato da Google e attualmente disponibile gratuitamente.

In particolare è stata utilizzata la funzione **Drumify**, che data una sequenza di note, genera un secondo file MIDI in cui, similmente per il piano, vengono indicate le parti della batteria da percuotere (piatto, charleston, gran cassa, rullante).

In particolare, nel risultato finale, quest'ultima sequenza MIDI è stata processata dal Plugin SuperiorDrummer3, una libreria sofisticata che raccoglie suoni di batteria accurati.

Si vuole far notare, che l'algoritmo implementato in **Drumify**, si basa su una neural network, non è di tipo genetico, come nel caso del nostro esempio, in grado dunque di cogliere e sviluppare una certa consequenzialità tra parti.

Infine, per le basse frequenze è stata aggiunta una semplice linea di basso, trasponendo la melodia due ottave inferiori, eliminando a mano, secondo il proprio gusto, le note ridondanti, lasciando invece quelle fondamentali e di maggiore durata.

Il risultato finale, nello specifico, è *quasi accettabile*, considerando che per generarlo sono state necessarie poco più di 500 linee di codice (commenti inclusi) e qualche plugin dal mondo di internet.

Il tratto fondamentale risiede nel fatto che per ottenere un file mp3, non abbiamo mai: né dovuto suonare un pianoforte, pizzicare un basso o ritmare una batteria a tempo.

Si possono sperimentare ulteriori modifiche, cambiando i pesi (`smoothnessWeight`, `restWeight`, `harmonyWeight`) o implementando un sistema nell'assegnazione della durata e velocity nelle note, ora affidate ad array con valori prefissati (rock, jazz, dance, bossa nova) e ad una scelta random rispettivamente.

# Chapter 5

## Conclusioni

Negli ultimi decenni, la ricerca sulla generazione musicale ha compiuto enormi progressi nella generazione di aspetti ben definiti della musica, come la melodia, l'armonia, il ritmo e il timbro.

Modelli statistici all'avanguardia, tecniche di ottimizzazione avanzate, database digitali più grandi su cui addestrare i modelli, e l'aumento della potenza di calcolo, hanno portato alla produzione di sistemi accurati.

*Perché allora non utilizziamo sistemi di generazione musicale nella nostra vita quotidiana?*

L'indagine di cui sopra mostra che rimane un'importante sfida generale: quella di creare musica con una struttura a lungo termine.

Per rendere i sistemi musicali generati dal computer parte della nostra vita quotidiana, c'è bisogno di sistemi più *narrativi*, in grado cioè di creare un inizio, sviluppo e conclusione nella musica.

Esiste un potenziale reale affinché il lavoro futuro si sposti verso sistemi intelligenti che non richiedano abbondanti quantità di tracce audio, o sequenze musicali, ma che siano capaci di un ragionamento innato, rispecchiando il funzionamento della mente umana.

Ciò risolverebbe anche la continua sfida di trovare un equilibrio tra la rigenerazione della musica esistente, o frammenti di questa senza ricadere nel plagio.

Come descritto da Lamport [2], e come discusso da Doe e Smith [1], i metodi utilizzati sono basati su...

# Bibliography

- [1] John Doe and Jane Smith. An example article. *Journal of Examples*, 42(1):1–10, 2023.
- [2] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 1994.