

IMprovEEsation: Intelligent Musical Evolutionary Entertainment

Davide Berardi, Matteo Martelli, Marco Melletti, Federico Montori

9 dicembre 2014

Sommario

1 Introduzione

2 Stato dell'Arte

3 Modello del Dominio

Questa sezione descriverà la composizione del nostro progetto alla luce di ciò che è già stato fatto e ciò che vogliamo introdurre. Il progetto ha come priorità l'esecuzione di un brano improvvisato da parte di più musicisti, i quali non hanno (almeno per il momento) alcuna coscienza della presenza di altri musicisti. Come mostrato in figura ??, la struttura del sistema consta di alcuni componenti differenti, che descriveremo nelle sezioni seguenti, organizzati esattamente come in un'orchestra reale (fatta eccezione per il player): il direttore d'orchestra trasferisce le proprie informazioni generali riguardo all'esecuzione a un numero arbitrario di musicisti, i quali eseguono una parte definita del brano e la passano a un player, che si occupa di tradurre in contemporanea ogni parte "scritta" da un musicista appunto in musica. È quasi come se i musicisti in questo caso, invece di produrre il suono loro stessi, producessero lo spartito misura per misura.

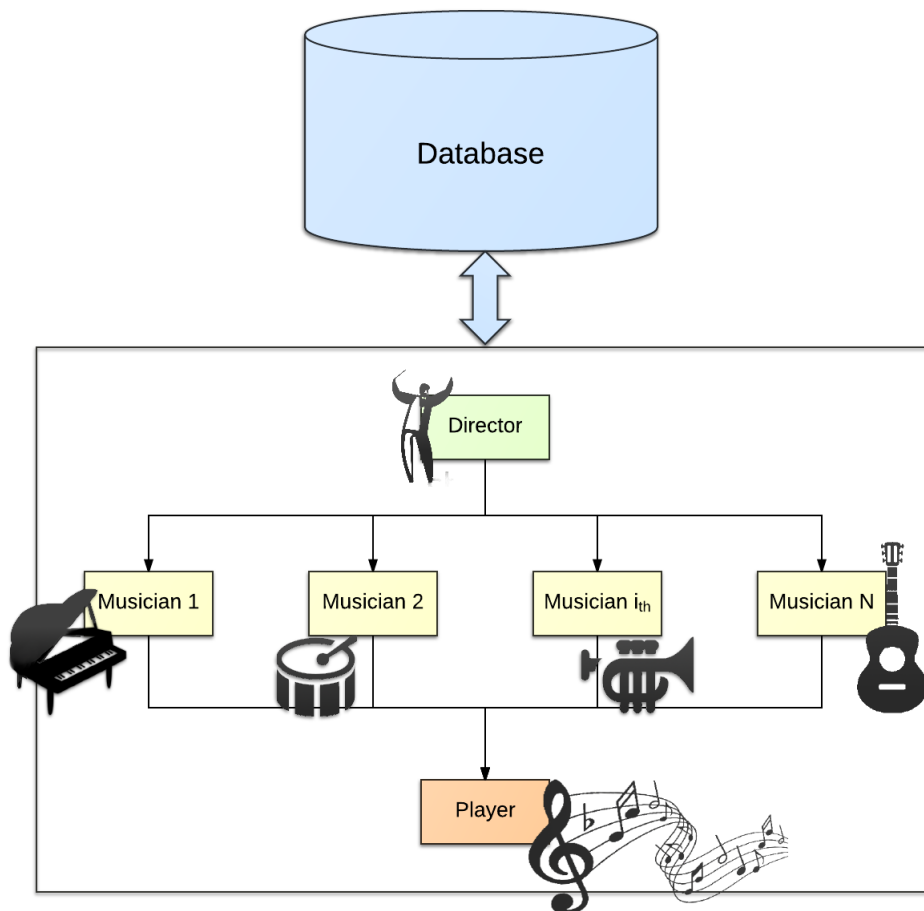


Figura 1: Schema dello scenario di progetto

Nel nostro caso, sia il direttore sia i musicisti sono da considerarsi agenti intelligenti, mentre il player è un mero esecutore (come vedremo nelle descrizioni dettagliate nei prossimi capitoli).

Tali agenti intelligenti operano in un ambiente che potremmo descrivere con la notazione PEAS:

- Parzialmente osservabile, poichè i musicisti conoscono solo lo stato corrente del direttore, ovvero soltanto ciò che esso decide di far suonare più, ovviamente, il proprio stato, ma non conosce ciò che gli altri musicisti eseguono. Inoltre, il direttore non conosce lo stato interno dei musicisti, ma si limita a decidere i parametri della misura seguente mediante un proprio algoritmo interno. Questo aspetto è presente in parte per semplificare la struttura attuale, ma potrebbe essere in seguito modificato.
- Strategico, poichè lo stato successivo dell'ambiente non è determinato dalle mosse di un agente, ma deve tener conto anche delle mosse degli altri

agenti, che, pur essendo cooperativi, potrebbero risultare imprevedibili. Il direttore decide le proprie mosse che influenzano la globalità del sistema, ma non ha controllo su ogni singolo agente.

- Episodico nel caso del musicista, poichè le azioni performate da quest ultimo non hanno ripercussioni future nè costituiscono un parametro di decisione nell'episodio seguente. La percezione del musicista è definita dalle informazioni pervenute dal direttore. Nel caso del direttore invece l'ambiente assume una connotazione sequenziale, poichè alcuni parametri dell'azione corrente determinano una probabilità di passaggio a differenti azioni successive possibili.
- Statico, poichè solo gli agenti coinvolti possono variare l'ambiente.
- Discreto poichè, pur operando in un'ottica real-time, le azioni degli agenti si basano su unità di tempo atomiche uguali per tutti, come, del resto, la teoria musicale impone.
- Multiagente, anche se le interazioni reali fra agenti sono relativamente scarse. Questo fa di un ambiente concettualmente cooperativo, in realtà un ambiente composto da unità che dagli altri agenti possono essere viste come stocastiche e imprevedibili.

4 Overview dei Componenti

Il progetto è composto principalmente di tre parti principali:

Un **direttore**, il quale svolge il compito di centralizzare l'improvvisazione, dettando regole e decidendo i parametri generali per ogni punto dell'improvvisazione; potrebbe in un certo senso essere visto come una sorta di *coscienza comune*, la quale amministrerebbe silenziosamente i vari musicisti, vedremo più avanti come la natura centralizzata del direttore inoltre aiuti, ad esempio, a sincronizzare i vari musicisti.

Un **player**, il quale assieme al direttore compone l'architettura centralizzata del progetto; il compito del player è di ricostruire e di assemblare le varie improvvisazioni provenienti dai vari musicisti, ha inoltre un'interfaccia modulare per salvare o riprodurre l'improvvisazione, per analogia con il direttore, il quale è il punto d'ingresso del progetto, il player è il punto dove viene formato l'output utile del progetto.

Vari **musicisti**, i quali prendono decisioni in base alla loro configurazione e all'output del direttore, processandole secondo vari meccanismi euristici (vedremo ad esempio implementazioni di meccanismi randomici o basati su algoritmi evolutivi).

Al fianco di questi componenti fondamentali è presente un ambiente di supporto per facilitare l'operazione, quali ad esempio il **database** dell'applicazione, responsabile dell'immagazzinamento della conoscenza dei vari componenti, ad esempio la rappresentazione dei vari generi e dei loro pattern collegati.

5 Componenti del Sistema

Come indicato in precedenza, i principali componenti in stretto contatto tra loro sono delle seguenti tre categorie, è stata scelta un implementazione non a camere stagne tra di essi, in modo da rendere il sistema il più modulare possibile, potendo separare (anche fisicamente, distribuendoli su varie macchine) i vari componenti, utilizzando dei processi singoli per ogni istanza del singolo.

Si procede quindi con la descrizione dettagliata del comportamento di ogni elemento del sistema.

5.1 Direttore

5.2 Musicista

Come il direttore, il musicista nel nostro software è essenzialmente un processo. Il suo scopo principale è quello di creare in tempo reale della musica. Della buona musica? Ci prova, infatti il processo musicista trascorre la sua esistenza suonando delle note che possano “andar bene” assieme alle note suonate dagli altri musicisti. Questi ultimi non vengono lasciati soli nelle decisioni prese durante un’improvvisazione ma il direttore li aiuta a prendere delle scelte che possano aver senso fra di loro e li aiuta a coordinarsi. Il direttore quindi, tramite un certo protocollo di comunicazione, invia determinati parametri globali a tutti i musicisti che a loro volta scandiscono il database per cercare delle note che possano avere senso nel loro attuale contesto. Ad ogni insieme di note che i musicisti ottengono ad ogni passo dell’esecuzione è correlato un set di probabilità, il quale viene utilizzato per filtrare le note scelte da utilizzare e ad introdurre il comportamento di improvvisazione.

5.3 Player

Il player è l’unico componente del progetto che non si comporta da agente intelligente, ma piuttosto da scheduler, poichè il suo compito è suonare in contemporanea le note che i musicisti hanno “scritto sul loro spartito”. Più in particolare, le informazioni che pervengono al player sono, da ogni musicista, un set di note e durate, secondo la struttura descritta nella sezione 6, della durata di una battuta. Il player resterà bloccato finchè non riceverà questo dato da ognuno dei musicisti (un numero conosciuto a priori), poi inizierà la sua esecuzione. Essa si basa sulla scansione dei dati in input secondo una base temporale atomica (che corrisponde alla durata della metà di una semicroma terzinata, ovvero un quarantottesimo di una misura intera) con la quale si possono rappresentare in base numerica intera tutte le note utilizzate. È da segnalare che non utilizziamo mai note di durata inferiore alla semicroma terzinata (quindi biscroma e semibiscroma, per i più avvezzi alla teoria musicale) per pura semplicità, poichè questo non pregiudica una buona dimostrazione del funzionamento del software.

Il player nasce come esecutore in tempo reale (o immediatamente successivo) rispetto alla creazione del brano, ma presenta una feature interessante, ovvero la scrittura su file midi del brano in creazione per esecuzioni future o studio dello spartito generato su tools come tuxguitar.

Questo componente utilizza una labeling standard degli strumenti musicali, coerente con quella utilizzata da TiMidity++¹, il tool che abbiamo sfruttato per tradurre il nostro operato in qualcosa di udibile.

6 Interazione e Comunicazione

La comunicazione è uno dei punti cruciali del progetto, sia per la quantità di informazione scambiata, che per la sincronia dei messaggi.

¹<http://timidity.sourceforge.net/>

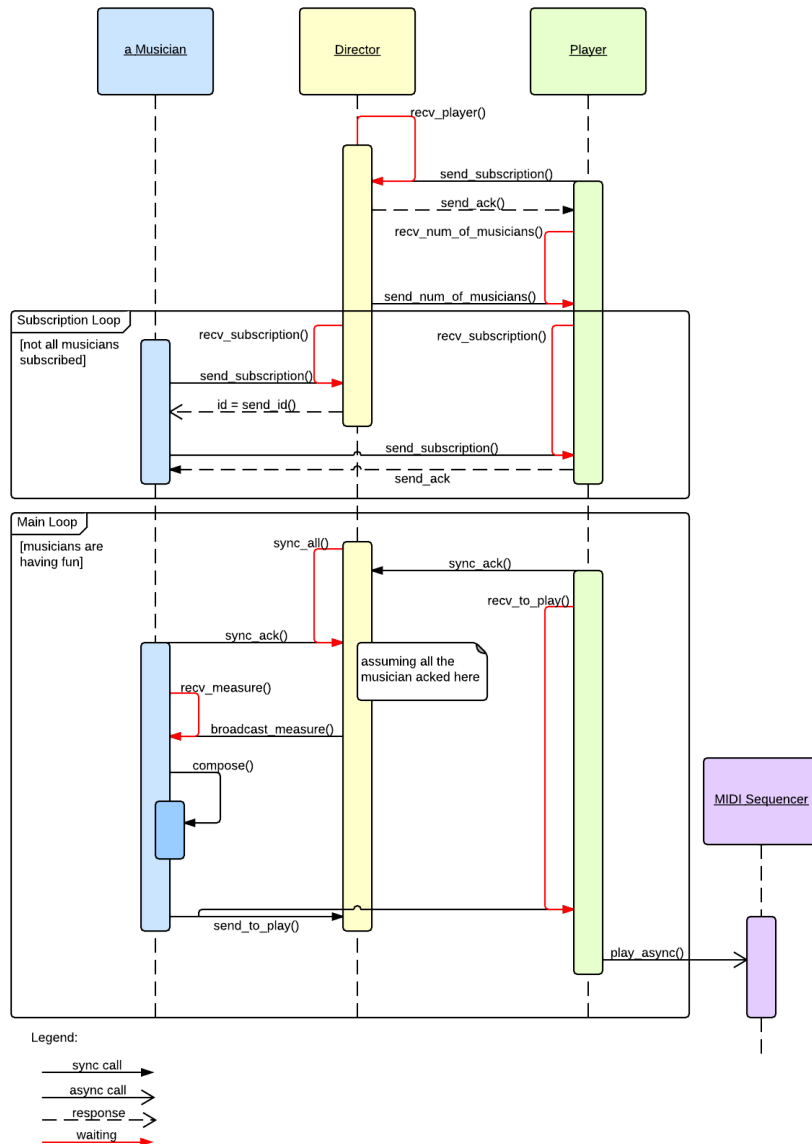


Figura 2: Diagramma del protocollo di comunicazione

Il protocollo è strutturato principalmente in tre fasi distinte.

6.1 Inizializzazione

Inizialmente il direttore riceve una sottoscrizione da parte del player. Alla sottoscrizione (opportunamente segnalata tramite *ack*), susseguirà un pac-

chetto contenente il numero di musicisti ² A questo punto l'inizializzazione tra i componenti principali (direttore e player) è completa, e la loro comunicazione, come vedremo si limiterà a messaggi di sincronizzazione.

6.2 Ciclo di Sottoscrizione

A questo punto, ogni musicista si preoccuperà di inviare la sua sottoscrizione all'improvvisazione sia al direttore che al player, in questo modo le due componenti principali avranno una chiara visione dell'improvvisazione, pur rimanendo completamente indipendenti.

6.3 Ciclo principale

Finita l'inizializzazione inizia il ciclo vero e proprio d'improvvisazione; una volta sincronizzati tutti i musicisti (e il player) a barriera, vengono inviate da parte del direttore le informazioni di improvvisazione, come vedremo più avanti questi pacchetti contengono tutte le informazioni sullo stato dell'improvvisazione, in modo da mantenere una certa coerenza tra tutti i musicisti.

Una volta composto, i musicisti inviano quindi la loro creazione ³, al player il quale si occuperà di suonare la composizione nell'ordine corretto; Mentre il direttore si occuperà della sincronizzazione, proseguendo con un nuovo passo del ciclo.

6.4 Libreria di comunicazione

L'implementazione della libreria di comunicazione è presente nel file *communication.cpp* e *struct.h*, se ne descrivono di seguito i dettagli.

Principalmente i pacchetti scambiati tra le varie istanze sono di 3 tipi: *Sottoscrizioni*, *Misure* e *Play Measures*.

6.4.1 Sottoscrizioni

Sono i pacchetti scambiati per la registrazione presso il Direttore o il Player

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
coupling																															
instrument_class																flags															

Essi contengono:

Coupling

Un indicazione (per il player) sul fatto che il musicista sia in realtà una composizione di più musicisti ⁴

²In modo da specificare solamente al direttore il numero di istanze di musicisti.

³Si rimanda alle opportune sezioni per come queste decisioni vengano prese.

⁴Immaginiamo ad esempio la mano sinistra e la mano destra dello stesso pianista, le quali devono essere assegnate allo stesso canale di output.

Instrument Class

Il tipo di strumento, utilizzato per la ricerca nel database e per l'assegnamento del corretto strumento *MIDI* in uscita.

Flags

I flag disponibili per la sottoscrizione sono:

- 0x0: Nessun flag rilevante.
- 0x1: Il musicista è un solista.
- 0x2: Il musicista utilizza pratiche di machine learning genetiche.

Da qui in poi sono riservati per utilizzi futuri.

La risposta a questi pacchetti non si limita al semplice ack, ma bensì ad un pacchetto contenente l'id (univoco per la sessione) del musicista.

É necessario indicare che la registrazione del il player presso il direttore avviene inviando questo pacchetto con un coupling pari ad un valore costante (-1).

6.4.2 Misure

Sono i pacchetti contenenti l'informazione sullo stato dell'improvvisazione e il prossimo passo d'improvvisazione, sono inviati in broadcast dal direttore a tutti i musicisti.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																		
bpm																																																	
soloist_id																																																	
tempo.upper								tempo.lower																																									
prioargs																																} 9																	
note																																scale																} tempo.upper	
chord																																mode																} tempo.upper	
tags length																																																	
tags (variable)																																																	

BPM

Un indicazione sui bpm dell'improvvisazione corrente.

Soloist ID

L'ID del musicista che improvvisa correntemente in modalità solista.

Tempo (upper e lower)

Indica la *Time Signature* dell'improvvisazione corrente.

Prioargs

Sono 9 campi costanti il quale compito é specificare una scala di priorit  con la quale effettuare le scelte di improvvisazione.⁵

Note e scale

Sono *tempo.upper* campi contenti la successione dei centri tonali della misura.

Chord e mode

Sono *tempo.upper* campi contenti la successione di accordi della misura.

Tags

  un campo testuale utilizzato per indicare attributi del pezzo da improvvisare (ad esempio il genere).

6.4.3 Play Measures

Sono i pacchetti contenenti l'informazione dettagliata sulle note suonate   la composizione che, in output dal musicista, viene passata al player pronta per essere allineata con le altre battute ed essere suonata.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
id																																
size																																
musician id																																
unchanged fst																																
note.tempo								note.id								note.triplets								note.velocity								} Notes (size)
note.chord_size																																
note.notes																																
bpm																																

Id

Un numero progressivo (per musicista) indicante la posizione della battuta da suonare.

Size

Il numero di note presenti nella misura corrente.

Musician id

L'identificativo univoco del musicista che ha generato la misura in questione.

Unchanged fst

Un booleano indicante il fatto che la prima nota sia cambiata o meno.⁶

⁵Per una spiegazione pi  dettagliata si rimanda ai capitoli sul come operano il Direttore e il Musicista.

⁶Questa feature   necessaria nel caso si debba effettuare dei legati o dei continui dalla misura precedente.

Note

Sono quindi presenti *size* strutture contenenti le seguenti informazioni

Tempo

La durata della nota (in sedicesimi).

ID

Un numero progressivo (per misura) indicante la posizione nella misura.

Triplets

Un indicazione sul fatto che la nota debba essere scandita con tempo terzinato.

Velocity

La velocity *MIDI*, é un indicazione sul volume della nota in output.

Chord_size

Il numero di note presenti (nel caso sia un accordo, altrimenti la nota sarà singola)

Notes

Il vettore di note (in notazione *MIDI*).

BPM

Un indicazione sui bpm da utilizzare nella riproduzione dell'improvvisazione.

6.5 Meccanismi di sincronizzazione

L'ultimo compito della libreria di comunicazione è provvedere ad alcuni meccanismi per la sincronizzazione dei vari componenti.

Questo obiettivo è raggiunto utilizzando le chiamate di sistema di linux basate sugli eventi quali *epoll*, vengono radunati su di una barriera (in qualsiasi ordine) i musicisti, prima che venga loro notificato tramite *ack* la possibilità di continuare.

7 Rappresentazione della Conoscenza

Da qui per le prossime 3 sezioni usiamo i titoli che piacciono tanto agli intelligentisti. Pagina 5 del libro di IA. Manca interpretazione del linguaggio naturale perchè non sapevo cosa metterci dentro.

7.1 Regole e Pattern

7.2 Database Relazionale

8 Ragionamento Automatico

8.1 Mente del Direttore

Come ragiona il direttore quando decide i pattern?

8.2 Mente del Musicista

Come ragiona il musicista quando decide le note?

9 Apprendimento

In questa sezione si vedrà l'agente musicista agire come un agente in grado di apprendere, che costituisce una base sulla quale esso tenderà sperabilmente a migliorarsi sempre più. È noto, come spiegato nelle sezioni precedenti, che il musicista non è un agente basato su un obiettivo, difatti si può dire che un vero obiettivo non esista. Vogliamo che esso agisca producendo una “bella parte musicale”, ma non abbiamo una parte musicale perfetta da assegnare, poichè, essendo un'improvvisazione, si perderebbe il concetto stesso di improvvisazione, inoltre non abbiamo un modo perfetto per assegnare a questa melodia un voto, poichè ci sarebbero da considerare un numero spropositato di variabili che in ogni caso non potrebbero valutare il prodotto in toto perchè questo dovrebbe considerare l'intera esecuzione di tutti gli strumentisti assieme. Questo ovviamente non ha senso nel dare un voto al musicista singolo perchè esso non ha coscienza degli altri musicisti e perciò potrebbe produrre un brano che suoni bene con certi musicisti e suoni male con altri. Come fare allora per produrre un brano che non sia perfetto (perchè non sarebbe improvvisato), ma che si ispiri a un brano esistente per produrre qualcosa sul genere e che possa migliorare se stesso?

La nostra risposta è stata: con un algoritmo evolutivo. Esso ha la peculiarità, una volta fornito un ideale e una buona funzione di fitness, di produrre un brano musicale “vicino” a quello fornito come ideale senza mai essere identico.

9.1 Algoritmo Evolutivo

Il nostro algoritmo evolutivo può venire applicato, nella pratica, attraverso un flag in input al programma musicista al momento del lancio, questo fa sì che alcuni musicisti possano essere “genetici” e altri no. Assieme al flag deve essere fornito il path al file che contiene il pattern ideale in formato .gme (caratteristico del progetto e spiegato nella sezione seguente). In questo modo, il musicista scriverà contemporaneamente due “spartiti” (nella pratica, due array di strutture che denotano le note suonate): lo spartito classico, improvvisato con la tecnica descritta nella sezione 8.2 e immagazzinato in una struttura apposita, invece di essere passato al player, e lo spartito ideale, estratto misura per misura dal file .gme utilizzando i parametri forniti dal direttore e immagazzinato in una struttura analoga alla precedente. Entrambe queste strutture, che chiameremo rispettivamente “iniziale” e “ideale”, vengono passate all'algoritmo evolutivo il quale, prima di processarle, si assicurerà che ambedue constino dello stesso numero di note, allungando per ripetizione la più corta.

L'algoritmo evolutivo, prima di entrare in loop, si costruisce un numero fissato (ora 512) copie della struttura iniziale in quello che è il pool genetico. Il loop seguente è costituito da un numero prefissato (ora 1500) di iterazioni che constano di quattro passaggi fondamentali[?]:

- Il primo passaggio è la **Point Mutation**, che introduce la mutazione nella generazione e agisce separatamente su tutti i membri del pool genetico. Nel nostro caso, si limita a variare il pitch e/o la durata di X note random dove X è pari a circa un ventesimo nel numero di note totali. Anche la variazione dei valori ha base randomica, chiaramente è molto più facile che il valore sia in qualche modo vicino al valore esistente piuttosto che molto lontano (è difficile che una semicroma diventi una semibreve oppure che una nota molto acuta divenga una nota molto grave).
- Il secondo passaggio è il **Sorting**, che si limita a ordinare i membri del pool genetico in base alla funzione di fitness in modo che siano disposti per similitudine decrescente con la struttura ideale. La funzione di fitness verrà spiegata in una sezione separata in seguito. Per l'ordinamento è stato scelto l'algoritmo mergesort, che ha complessità uniforme in casi ottimi e pessimi. L'algoritmo quicksort è stato scartato perchè è facile che, a ogni iterazione, il pool genetico sia vicino all'ordinamento, caso che rende il quicksort vicino al suo caso pessimo.
- Il terzo passaggio è la *Recombination*, che, come in molti algoritmi evolutivisti esistenti, opera su coppie di membri del pool genetico. Nel nostro caso, esso opera su coppie adiacenti nell'ordinamento, solo nel top 25% del pool genetico. Esso sceglie, per ogni coppia, una posizione random sulla struttura, detta crossover, che taglia entrambi i membri della coppia in due parti, poi scambia le due seconde parti all'interno della coppia. Questo set risultante di un quarto del pool genetico (nel nostro caso 128 elementi) viene sostituito al 25% inferiore nell'ordinamento, in modo che il top 24% non venga toccato, ma soltanto copiato e siagisca per ricombinazione sulla copia.
- Il quarto passaggio, proprio solo di alcuni algoritmi evolutivisti, è la **Transposon Propagation**. Questa tecnica opera singolarmente solo sui membri del pool genetico che hanno subito la Recombination. Essa sfrutta il fatto che nella musica siano frequenti le ripetizioni: determina due punti di crossover all'interno della struttura in modo che essi denotino un frammento di quest'ultima. Tale frammento viene copiato e sovrascritto su un'altra sezione della struttura in una posizione anch'essa casuale.

9.1.1 La Funzione di Fitness

L'ordinamento viene eseguito utilizzando una funzione di fitness appositamente determinata che determina la similitudine della struttura con la struttura ideale di partenza. Tale similitudine è una combinazione lineare di una serie di grandezze di seguito elencate in relazione a dei pesi predeterminati:

- La similitudine punto a punto, che sarà più alta qualora in una stessa posizione sarà presente lo stesso pitch e/o la stessa durata della nota.
- La Unigram Pitch Similarity è calcolata come segue:

$$\frac{\sum_{i=0}^U \frac{1}{\sqrt{1+(train(i)-test(i))^2}}}{U}$$

, dove U è il numero di Pitch Unigrams, ovvero di note singole distinte presenti nella struttura in esame e le funzioni $train(i)$ e $test(i)$ restituiscono il numero di occorrenze della nota di indice i all'interno dei Pitch Unigrams in esame in input rispettivamente all'interno della struttura ideale e del membro del genetic pool in esame. Questa similitudine restituisce un numero tra 0 e 1 e denota la similitudine dei pitch ovvero quanto il membro del pool genetico corrente utilizza note utilizzate anche dalla struttura ideale.

- La Unigram Tempo Similarity è calcolata esattamente come la precedente, con la differenza che fa riferimento alle durate invece che ai pitch delle note.
- La Bigram Pitch Similarity è calcolata come segue:

$$\frac{\sum_{i=0}^B \frac{1}{\sqrt{1+(train(i)-test(i))^2}}}{B}$$

, dove B è il numero di Pitch Bigrams, ovvero di coppie ordinate distinte di note presenti nella struttura in esame e le funzioni $train(i)$ e $test(i)$ restituiscono il numero di occorrenze del Bigram di indice i all'interno dei Pitch Bigrams in esame in input rispettivamente all'interno della struttura ideale e del membro del genetic pool in esame. Questa similitudine è analoga alla Unigram Pitch Similarity, ma è più forte in quanto analizza la co-occorrenza di coppie ordinate di note, e perciò avrà un peso maggiore.

- La Bigram Tempo Similarity è calcolata esattamente come la precedente, con la differenza che fa riferimento alle durate invece che ai pitch delle note.

Queste similitudini sono tratte dalla categorizzazione Bayesiana di testi in un tipico algoritmo di machine learning all'interno dell'NLP.

9.2 Il Pattern Ideale

I file di pattern ideale ospitano una serie di pattern codificati in CSV e sono semplicemente elencati come righe all'interno dei file .gme. Le caratteristiche listate in un pattern sono: dyna (come ad esempio il groove o il fill), Unchanged fst (che determina se la prima nota è un continuo di una precedente), una serie di triple <pitch, tempo, triplet>, dove triplet indica con un numero booleano, se la presente nota fa parte di una terzina o meno. Queste caratteristiche consentono al musicista di estrarre il corretto pattern conforme alle regole imposte dal direttore misura per misura. Possono essere ovviamente presenti più pattern con le stese caratteristiche, in tal caso ne verrà scelto uno a caso, in tal modo si introdurrà un fattore casuale anche nella formazione del brano ideale.

10 Risultati Sperimentali

Dopo tanto sbatto funziona tutto random!

11 Conclusioni

Ci vuole un DB supermegagigante!!!

12 Sviluppi Futuri

12.1 Sviluppo della Conoscenza

Spieghiamo qui o in sviluppi futuri? Comunque potremmo proporre nel futuro di salvare nel db il risultato del genetico facendo un match dei quarter che sono usciti dal genetico con quelli che già ci sono nel db aggiustando le probabilità che già ci sono. Quelle che non ci sono possiamo aggiungerle.

Riferimenti bibliografici

- [1] Homer J. Simpson. *Mmmmm...donuts*. Evergreen Terrace Printing Co., Springfield, SomewhereUSA, 1998
- [2] Murali S. N., *Music: Markov Chains, Genetic Algorithms and Scale Transformation*. on Youtube at <https://www.youtube.com/watch?v=FvrcvXpcfM>, 2013