IMprovEEsation: Intelligent Musical Evolutionary Entertainment

Davide Berardi, Matteo Martelli, Marco Melletti, Federico Montori

11 dicembre 2014

Sommario

1 Introduzione

2 Stato dell'Arte

Questa sezione tratta del lavoro che è già stato compiuto nell'ambito della composizione e dell'improvvisazione della macchina. È da precisare che questo progetto non si è inizialmente fondato su un'idea esistente, poichè siamo voluti partire da un'idea che è scaturita da noi e l'abbiamo voluta sviluppare. Le pubblicazoni presenti in questa sezione sono da considerarsi spunti.

Una delle applicazioni più utilizzate da ciò che è chiamato "Machine Composition" è la possibilità di creare una vera e propria band di backup con la quale un improvvisatore può suonare. Perciò quello che interessa la maggior parte di questi software è la creazione di una melodia standard, non articolata, che segua una mappa di accordi fornita dall'utente e un certo pool di stili preimpostati (che possono essere molto numerosi), con l'obiettivo di dare all'utente la facoltà di suonare il proprio strumento con una band che esegue il background. L'improvvisazione è sempre il risultato finale, ma non è esattamente quello che vogliamo ottenere. Fra questi software citiamo Band-In-A-Box[**?**], che crea basi di accompagnamento in diverse divisioni, in tutte le possibili tonalità e accordi. Recentemente è in grado di performare assoli di chitarra e sassofono, che però si riferiscono sempre a standard compresi nel database dei pattern del programma. Questo software ottiene ottimi risultati nell'accompagnamento, ma non concede piena libertà ai suoi musicisti digitali, ovvero non considera che "tutto potrebbe accadere" come in un'improvisazione reale.
Un altro aspetto molto affine con il nostro lavoro è una branca dell'IA chiamata Evolutionary Music, che è divenuta una vera e propria materia di studio a sè stante[**?**]. Sono stati performati molti tentativi di integrare l'improvvisazione e la composizione con gli algoritmi genetici, due esempi possono essere GenDash[**?**], che aiuta il musicista a comporre musica, e GenJam[**?**], il quale output è una jam session, che utilizza i risultati di Band-In-A-Box per poi mutarli a seconda del feeling che il brano sta assumendo, ad esempio, se il solista tende ad accelerare o a inserire un numero di note maggiore, anche la batteria e il basso addenseranno più colpi rendendo il ritmo più incalzante pur essendo la base la stessa.
L'Evolutionary Music si incrocia con la Computer Music in un ambito detto Machine Improvisation, che utilizza diversi sistemi per simulare l'improvvisazione di un solista su una base musicale parzialmente nota. Esistono i metodi statistici, basati su Catene di markov, HMM e processi stocastici[**?**]. Questi hanno dato origine a progetti come il Continuator[**?**], il quale fa uso di modellazioni di stili non-real time[**?**].

È da precisare che un programma in grado di produrre un brano completamente improvvisato non è mai stato realizzato, ma si è sempre cercato di partire da una base musicale definita (facendo improvvisare un solo solista) oppure musicisti virtuali che imparano da un musicista fisico in real time per poterlo accompagnare. L'intento del presente progetto è invece quello di creare ciò che pù si avvicina a una Jam session, ovvero strumentisti che si ritrovano in una sala prove e seguono una serie di direttive generiche per produrre un brano senza una base esistente.

3 Modello del Dominio

Questa sezione descriverà la composizione del nostro progetto alla luce di ciò che è gia stato fatto e ciò che vogliamo introdurre. Il progetto ha come priorità l'esecuzione di un brano imrovvisato 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'orcherstra trasferisce le proprie informazioni rgenerali 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.

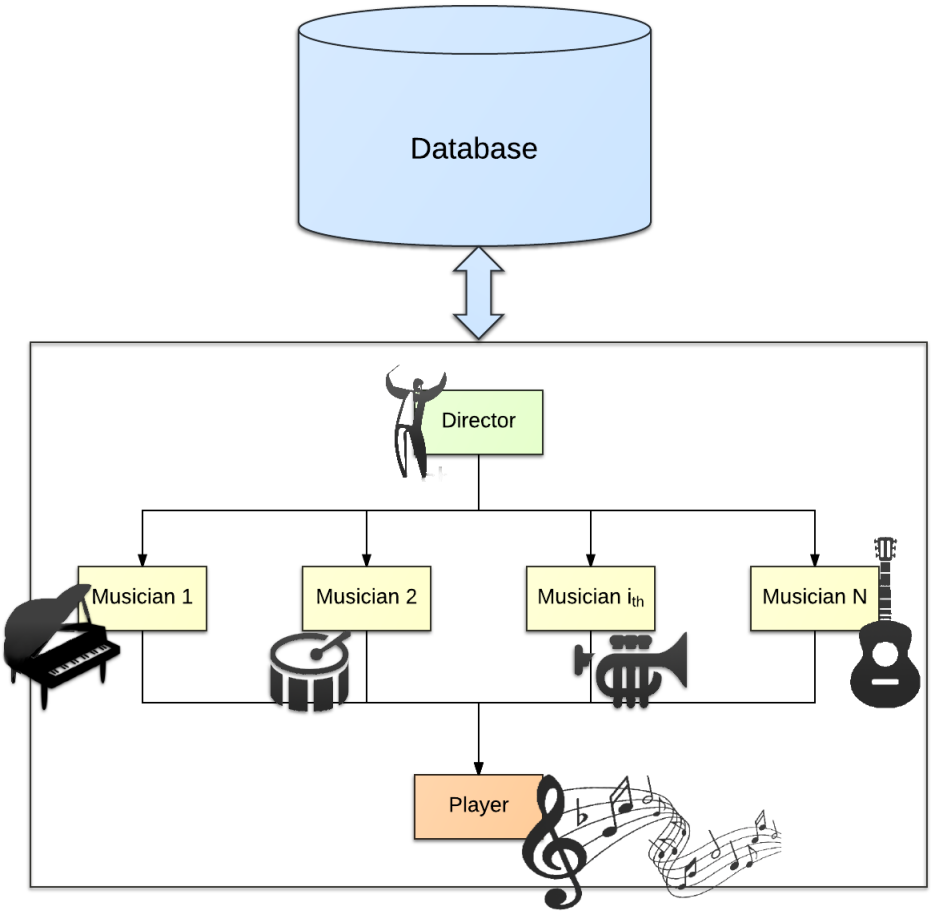

Figura 1: Schema dello scenario di progetto

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 amministrebbe silentemente i vari musicisti, vedremo piú 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 é 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 é il punto d'ingresso del progetto, il player é 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 é 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, é stata scelta un implementazione non a camere stagne tra di essi, in modo da rendere il sistema il piú 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 ùsuonando 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/`

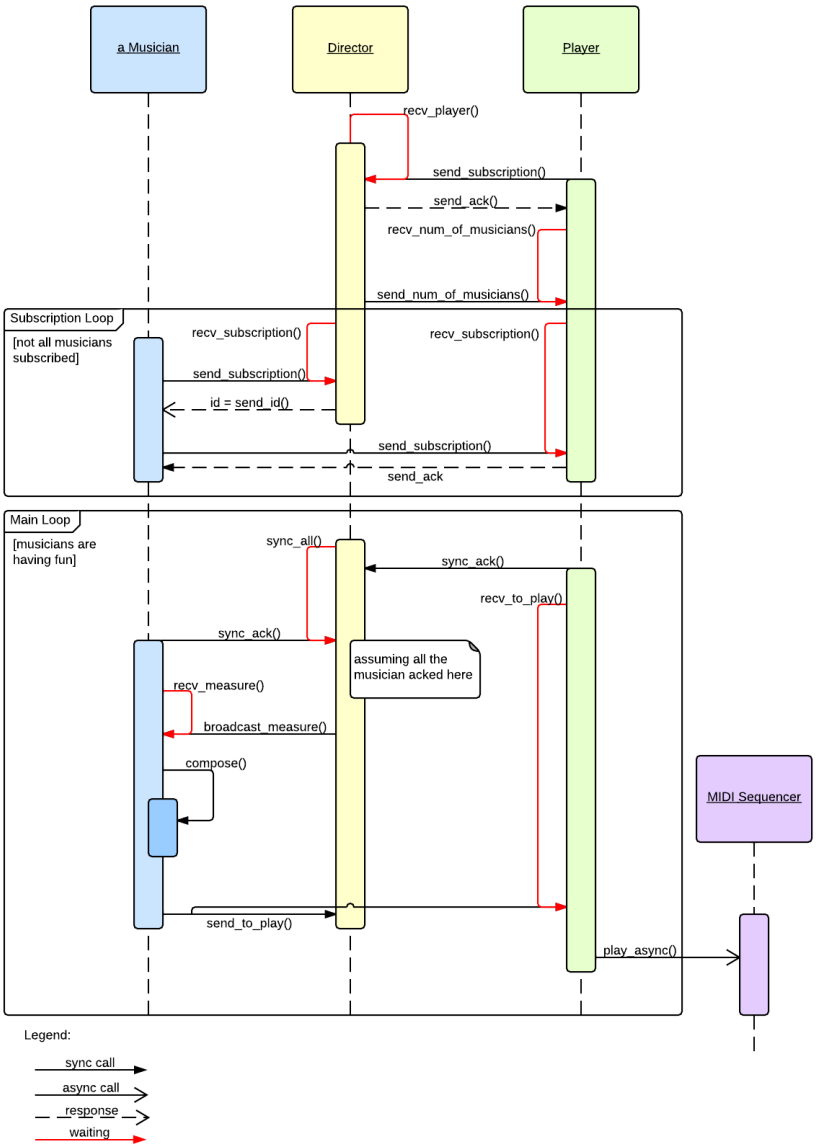

Figura 2: Diagramma del protocollo di comunicazione

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 pacchetto 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.

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

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 occuperá di suonare la composizione nell'ordine corretto; Mentre il direttore si occuperá 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

[bitwidth=1.1em]32 0-31
32coupling
8instrument_class 8flags

Essi contengono:

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

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 paccheto con un coupling pari ad un valore costante (-1).

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

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

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.

[bitwidth=1.1em]32 0-31
8bpm
32soloist_id
8tempo.upper 8tempo.lower
9 32prioargs
tempo.upper 16note 16scale
tempo.upper 16chord 16mode
32tags length
32tags (variable)

BPM
Un indicazione sui bpm dell'improvvisazione corrente.

Soloist ID
L'ID del musicista che improvvisa correntemente in modalitá 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 prioritá 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
É 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 è la composizione che, in output dal musicista, viene passata al player pronta per essere allineata con le altre battute ed essere suonata.

[bitwidth=1.1em]32 0-31
32id
32size
32musician id
8unchanged fst
Notes (size) 8note.tempo 8note.id 8note.triplets 8note.velocity
8note.chord_size
[lrtb]2note.notes
8bpm

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

Size
Il numero di note presenti nella misura corrente.

[5] Per una spiegazione piú dettagliata si rimanda ai capitoli sul come operano il Direttore e il Musicista.

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]

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 sará 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

Generalmente la conoscenza musicale di ogni musicista, cantante, compositore o direttore d'orchestra è formata da tre componenti fondamentali:

1. esecuzioni passate dell'artista stesso
2. esecuzioni altrui ascoltate precedentemente
3. regole provenienti dalla teoria musicale

In IMprovEEsation queste informazioni sono alla base del modello della conoscenza degli agenti del sistema. Le componenti 1 e 2 costituiscono i pattern musicali e le relative note associate ad essi. Inoltre nella memoria degli agenti sono presenti informazioni aggiuntive, come ad esempio scale, modo degli accordi, etc. Quest'ultime vengono messe in relazione con i pattern e le note formando così la terza componente, ovvero l'insieme di regole teoriche possedute dall'agente.
Inoltre la conoscenza degli agenti viene rappresentata come se fosse immagazzinata in un unica memoria collettiva, che viene acceduta però in regioni diverse in base al ruolo dell'agente. Ad esempio il direttore ha accesso alla regione della memoria dove sono salvate le informazioni riguardo all'andamento complessivo di un improvvisazione che comunicherà durante quest'ultima ai musicisti. Un agente musicista ha accesso alla regione di memoria dove salvate le informazioni necessarie per comporre in tempo reale delle note che siano coerenti in qualche modo con le informazioni fornite dal direttore.

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

7.1 Pattern e Regole

Definiamo i pattern come sequenze di misure di accordi. Il direttore ha la conoscenza di una collezione di diversi pattern che a loro volta possono ammettere delle varianti di dinamica e stile. Il musicista, nella memoria complessiva, ha accesso alle collezioni di informazioni riguardo note singole. Si è scelto di rappresentare una singola nota come una semicroma essendo 1/16 la suddivisione temporale più piccola che prendiamo in considerazione. Inoltre ad ogni semicroma non è associato un solo valore tonale, ma un vettore di probabilità di dimensione n pari a 13. L'indice i-esimo di ogni elemento corrisponde alla distanza tonale dalla tonalità decisa dal direttore, oppure una pausa. Ogni elemento i-esimo del vettore corrisponde al valore di probabilità p_i che la nota di distanza i dalla tonalità corrente venga selezionata ad un certo istante di tempo t. Per ogni semicroma è associato anche un valore di probabilità p_c utilizzato per decidere se quella semicroma e il suo vettore di probabilità debba essere considerato ad un certo istante di tempo t. Nella memoria del musicista sono presenti anche i quarti che raggrupprano fino a 4 semicrome ognuno. Ai quarti sono correlate ulteriori informazioni, utili a comprendere il contesto di appartenenza di un certo quarto, come:

- posizione del quarto in una misura
- l'accordo associato
- il modo dell'accordo
- strumento associatio
- dinamica del quarto
- mood (stile) del quarto
- se lo strumento associato è solista o meno

Mettendo in relazione queste informazioni con quelle fornite dal direttore durante un improvvisazione, un musicista cerca di scegliere delle note che siano il più possibile inerenti al contesto. Come ciò viene fatto verrà spiegato nella sezione **??**.

7.2 Database Relazionale

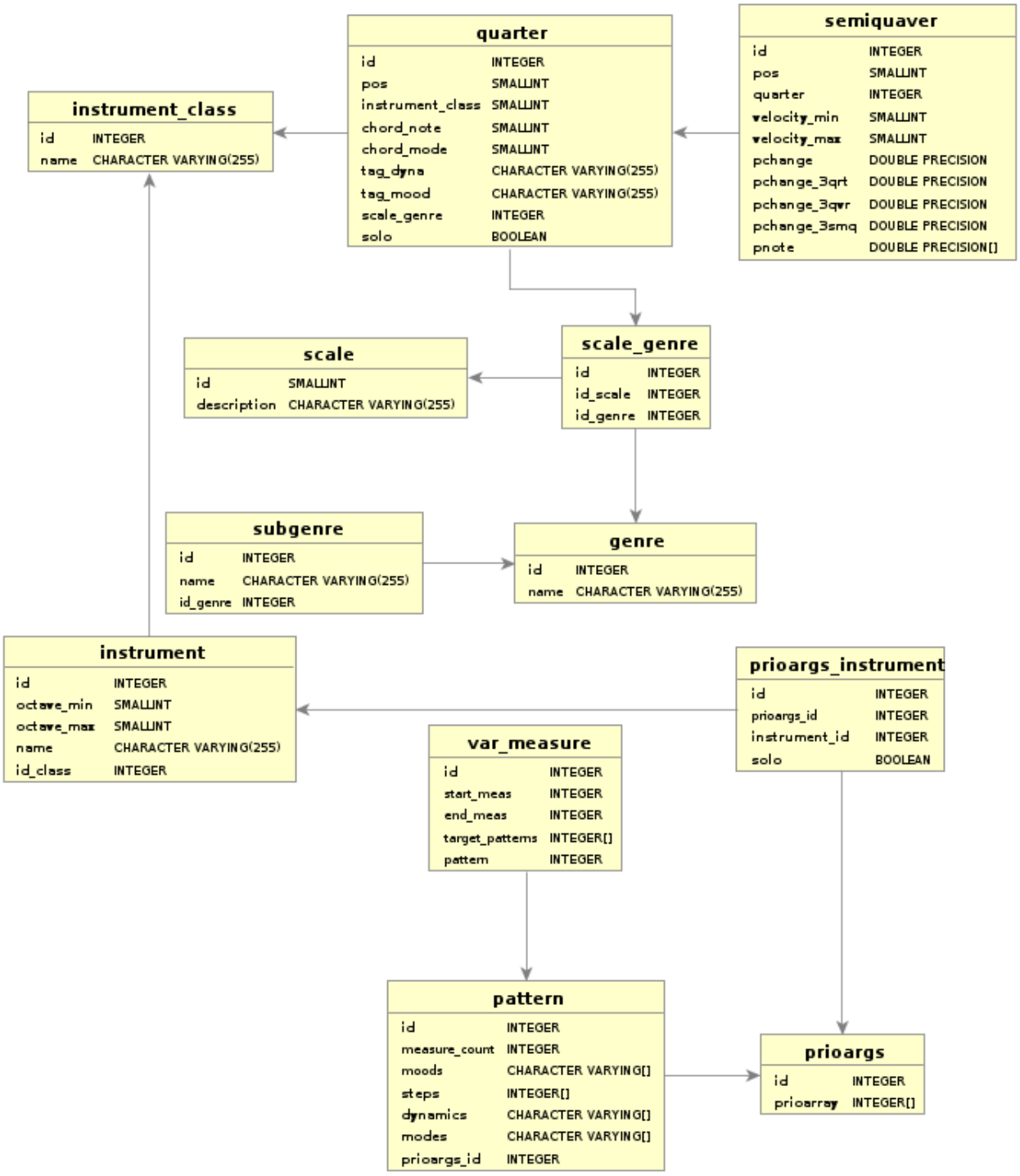

Figura 3: Schema E-R del DataBase

La struttura di conoscenza spiegata nel paragrafo precedente è stata implementata tramite un DataBase relazionale il cui schema Entitià Relazione è mostrato nella figura **??**.
La struttura si può pensare come divisa essenzialmente in due parti principali.
La prima si concentra intorno alla tabella *quarter*, attraverso la quale il musicista riesce a capire quali

semiquaver considerare in base ad un certo contesto definito dal direttore come già spiegato nella sezione precedente. È importante notare che nonostante la memoria sia collettiva a tutti gli agenti, tramite le informazioni legate ai quarti un musicista è in grado di filtrare le informazioni che meglio si adattano a se stesso. Oltre al contesto definito dal direttore infatti, il musicista può anche filtrare le informazioni che riguardano la sua natura, come ad esempio il tipo di strumento da lui suonato o se si sta comportando o meno da solista.
La seconda parte fondamentale del DataBase è concentrata intorno alla tabella *pattern*. Dalla figura **??** potrebbe sembrare che l'entità *pattern* non abbia poi così tante relazioni come l'entità *quarter* per considerare quest'utlima un parte fondamentale del DB. Questo deriva dal fatto che si è deciso di semplificare un po' il modello E-R per l'entità *pattern* utilizzando dei vettori per ogni record di quest'ultima anzichè creare delle nuove entità-relazioni. Questa scelta è giustificata dal fatto che viene inizialmente considerato un numero relativamente basso di pattern e che il modello è facilemente estendibile in quanto le due parti fondamentali del DataBase hanno due schemi abbastanza indipendenti tra loro.
Nello specifico la tabella *pattern* è caratterizzata fondamentalmente dai seguenti campi vettoriali

- steps
- modes
- dynamics
- moodes

tramite i quali un direttore riesce a scandire l'andamento dell'improvvisazione misura per misura, definendo a quali accordi riferirsi e suggerendo ai musicisti con quale dinamica e sentimento eseguire le loro perfomance. Queste direttive possono venire suggerite ai musicisti in modo stretto o lato.
A tal proposito ad ogni pattern viene associato un vettore di priorità dalla tabella prioargs che definisce con quale priorità le direttive del direttore riguardanti quel pattern debbano essere considerate dai musicisti, anche rispetto alle altre informazioni sulla loro stessa natura. È anche possibile che un musicista non consideri affatto le priorità legate al pattern ma utilizzi delle priorità ad-hoc, definite rispetto ad altri parametri.
Ad esempio un musicista solista potrebbe considerare un vetttore di priorità ad-hoc che non sia legato a nessun pattern in quanto potrebbe non essere interessato alla successione di accordi tra le misure, informazione molto forte per i pattern, ma solamente alla tonalità corrente, alle dinamiche e ai mood delle misure.
C'è poi il caso particolare del batterista che non ha alcun interesse riguardo le informazioni inerenti alla linea melodica e aromica di un improvvisazione, ma è interessato solo alla linea ritmica. Inoltre per il batterista viene utilizzato un modello diverso di vettore di probabilità di ogni semiquaver, quindi è fondamentale che la sua priorità più alta sia il tipo di strumento, altrimenti, incorrenendo in vettori di priorità associati a quarti di altri strumenti, potrebbe interpretare in maniera completamente errata tali informazioni.

Dal lato applicazione le informazioni residenti nel DB vengono recuperate tramite una libreria dinamica dedicata. Quest'ultima si occupa di effettuare le apposite query per recuperare le informazioni, con i dovuti vincoli, necessarie ai processi musisti e direttori per improvvissare.

8 Ragionamento Automatico

Quando un musicista o un gruppo di musicisti eseguono un improvvisazione, sono molteplici i fattori che incidino nelle loro azioni. Tra questi sicuramente hanno molta influenza le emozioni individuali e reciproche dei musicisti, oppure il feeling che c'è tra loro.
Indubbiamente anche se non è l'unico fattore in gioco, non può mancare un certo tipo di ragionamento nella scelte effettuate da ogni musicista, che sa in qualche modo come agire per produrre un risultato che sia apprezzabile per se stesso e per gli altri. Per un software che genera delle improvvisazioni musicali, il fattore emotivo non può però essere escluso. Anche se le macchine non sono in grado di provare delle emozioni reali, possono sempre simulare dei comportamenti che sembrino essere influenzati da delle emozioni reali, e forse possono anche scaturire emozioni a chi interagisce con tali macchine. Quest'ultimo

è un obbiettivo arduo ma sicuramente interessante per software di questo tipo. I puristi della musica e dell'arte potrebbero sollevare una critica dicendo che un musicista che non prova emozioni non è in grado di sollevare emozioni verso chi li ascolta. Lo studio emotivo nell'intelligenza artificiale è un campo aperto e osservato con interesse dal mondo scientifico, risulta quindi stimolante proporne un applicazione di studio.
In IMprovEEsation il ragionamento dei musicisti è fortemente influenzato dal ragionamento del direttore che potrebbe essere visto come una sorta di feeling o di sintonia tra i vari musicisti improvvisatori, che in un caso reale non avrebbero un coordinatore centrale.
In questa sezione vedremo meglio come ragionano il direttore e il musicista durante una sessione di improvvissazione.

8.1 Mente del Direttore

Come ragiona il direttore quando decide i pattern?

8.2 Mente del Musicista

Come ragione 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 sperabilemnte 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 evoluzionistico. 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 Evoluzionistico

Il nostro algoritmo evoluzionistico 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 contemporanemate 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 evoluzonistico il quale, prima di processarle, si assicurerà che ambedue constino dello stesso numero di note, allungando per ripetizione la più corta.

L'algoritmo evoluzionistico, prima di entrare in loop, si costrucisce 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 decrecscente 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 evoluzionistici 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 evoluzionistici, 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'utlima. 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 utlizzando una funzione di fitness apposita che determina la similitudine della struttura con la struttura ideale di partenza. Tale similirudine è una combinazione lineare di una serie di grandezze di seguito elencate in realzione 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 esattamante 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 unpeso maggiore.

- La Bigram Tempo Similarity è calcolata esattamante 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

Meritano una menzione a parte gli sviluppi futuri dedicati alla parte di machine learning. In questo momento, l'algoritmo evoluzionistico fa in modo che un musicista "impari" a improvvisare data una sequenza ideale come traccia. Come rendere questo apprendimento permanente?
La nostra proposta per un futuro sviluppo è incrociare l'algoritmo evoluzionistico con l'approccio sulle probabilità. Idealmente, un utente darebbe in pasto all'algoritmo evoluzionistico il musicista con il brano ideale finchè ciò che viene prodotto non sarà buono. Avvenuto questo, il prodotto dell'algoritmo dovrebbe venire trasformato in un set di probabilità (meglio ancora se si considerano più output di tale algoritmo, per facilitare la trasformazione in percentuali) che potrà aggiornare il database dei pattern, ovvero le probabilità già presenti sul database alle quali si fa riferimento durante l'approccio probabilistico.

Una seconda miglioria, affine con le tecniche esistenit di machine composition e machine improvisation, sarebbe quella di considerare gli altri musicisti e le loro esecuzioni. Ciò che l'algoritmo evoluzionistico potrebbe fare in questo caso è l'alterazione della funzione che computa la similitudine in modo che vengano introdotte grandezze che non confrontano più il pool genetico con il pattern ideale, ma anche con le scelte degli altri strumentisti.

Abbiamo notato che l'algoritmo evoluzionistico tende a girare attorno a un punto fisso, dopo un repentino miglioramento iniziale, dipendente dalla lunghezza del brano. Una terza miglioria consiste nel migliorarlo per aumentarne le prestazioni, soprattutto su brani di lunghezza consistente, per poter spingere quel punto fisso a percentuali di similitudine più alta.

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=FvrcvXpcfvM, 2013

[3] Gannon, P., *Band-in-a-Box*. PG Music, 1990

[4] Miranda E. R., Biles J. A., *Evolutionary Computer Music*, Springer Ed., 2007

[5] WASCHKA II, R. O. D. N. E. Y. Composing with Genetic Algorithms: GenDash. Evolutionary Computer Music. Springer London, 2007. 117-136.

[6] Biles, John. GenJam: A genetic algorithm for generating jazz solos. Proceedings of the International Computer Music Conference. INTERNATIONAL COMPUTER MUSIC ACCOCIATION, 1994.

[7] Jan Pavelka; Gerard Tel; Miroslav Bartosek, eds. (1999). Factor oracle: a new structure for pattern matching; Proceedings of SOFSEM'99; Theory and Practice of Informatics. Springer-Verlag, Berlin. pp. 291–306. ISBN 3-540-66694-X. Retrieved 4 December 2013. Lecture Notes in Computer Science 1725

[8] S. Dubnov, G. Assayag, O. Lartillot, G. Bejerano, Using Machine-Learning Methods for Musical Style Modeling, IEEE Computers, 36 (10), pp. 73-80, Oct. 2003.

[9] Pachet, Francois The Continuator: Musical Interaction with Style. In ICMA, editor, Proceedings of ICMC, pages 211-218, Göteborg, Sweden, September 2002 ICMA. best paper award