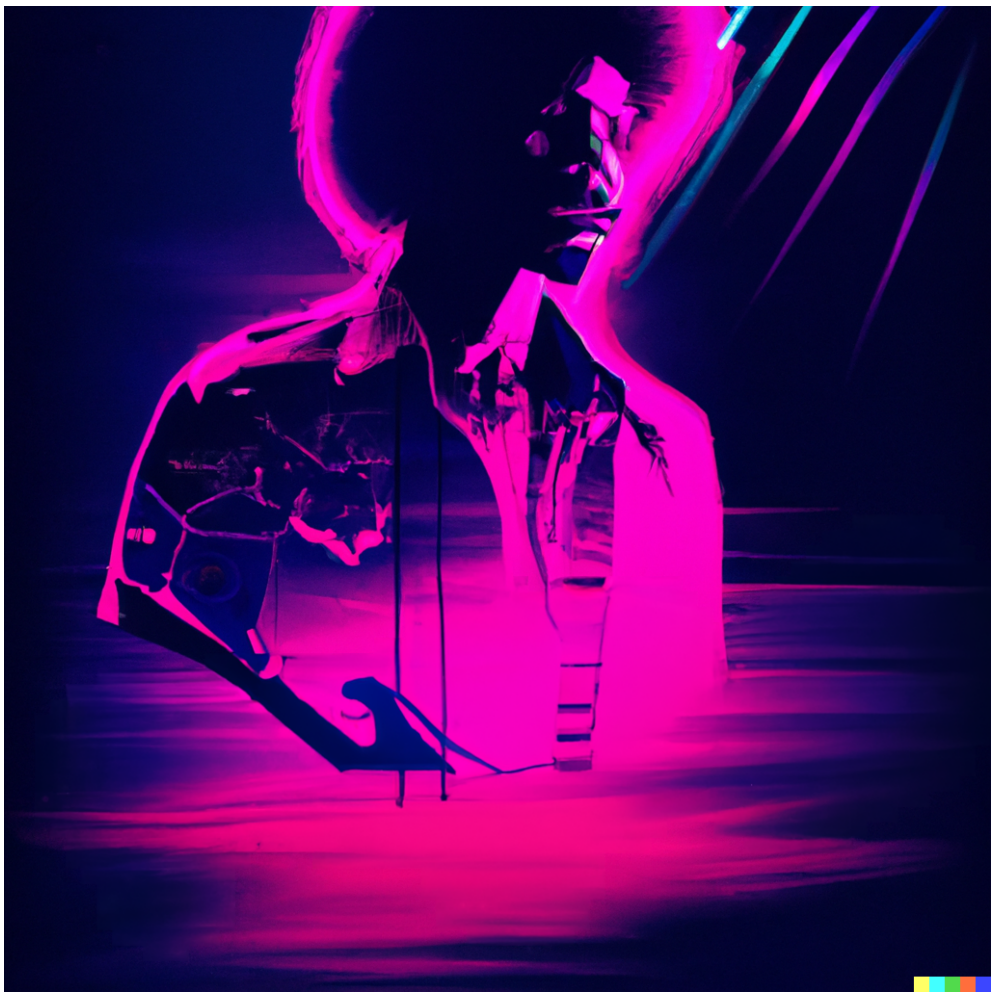


Charles Hendrix

https://github.com/SugarStoneMaster/CharlesHendrix

Carmine Iemmino (0512109893)

Indice

1 Introduzione

Il progetto nasce dall'idea di utilizzare l'intelligenza artificiale al fine di **comporre musica** (melodia e armonia) **in maniera automatica.** Quest'ultima poi, sotto giudizio umano, verrà eventualmente modificata e adattata per migliorarla secondo i propri gusti. Il sistema che ne verrà fuori è quindi da intendersi come **supporto** alle attività creative dei musicisti, i quali dovranno comunque applicare le loro conoscenze musicali per ottenere un risultato soddisfacente. L'obiettivo è che dato in input numero di battute e chiave utilizzata, venga generata in maniera automatica una composizione musicale, che sia il più "bella" possibile. Ovviamente, data la natura particolarmente **soggettiva** del problema, è impossibile definire un ottimo globale, ma ci accontenteremo di soluzioni sub-ottime che in seguito verranno manipolate manualmente dal musicista.

2 PEAS

- **Performance**: Valutato in base alla qualità delle composizioni prodotte
- **Ambiente**: input dell'utente, tutte le melodie e armonie possibili in una scala
- **Attuatori**: software di riproduzione musicale, schermo per visualizzare informazioni sulla composizione
- **Sensori**: tastiera per l'inserimento di numero di battute e chiave, files in formato .musicxml

3 Proprietà dell'ambiente

- **Completamente osservabile**: l'agente ha a disposizione il numero di battute e la chiave con cui comporre, o eventualmente una composizione preesistente in formato .musicxml
- **Semi-Deterministico**: la progressione di accordi è scelta in base alla preferenza dell'utente, per poi lasciare all'agente la possibilità di modificare la parte melodica
- **Sequenziale**: la scelta delle azioni/modifiche alla composizione dipende da modifiche precedenti
- **Statico**: una volta scelta una chiave, un numero di battute e una progressione di accordi, l'ambiente in cui delibera l'agente non sarà modificabile
- **Discreto**: le modifiche alla composizione sono in numero limitato e discreto, così come il numero di note in una scala e il numero

- **Singolo**: nell'ambiente è presente un unico grande agente composto da due agenti che hanno due scopi diversi; il primo consente di individuare la progressione di accordi (armonia) preferita tramite interazione diretta con l'utente e il secondo di prendere in input la progressione di accordi prescelta per poi comporci autonomamente una melodia sopra.

4 Risolvere il problema

Data la natura del problema, di esplorare lo spazio degli stati, ovvero le diverse composizioni possibili, ci si focalizzerà sul costruire un **algoritmo di ricerca** che riesca a trovare una soluzione/composizione migliore possibile. In questo caso non ci interessano i passi che portano alla soluzione, ma bensì la soluzione stessa. La classe di algoritmi che si occupa di risolvere questi tipi di problema è denominata **algoritmi di ricerca locale**. Dato un numero di battute e la chiave, ogni composizione possibile potrebbe essere una soluzione al problema. È quindi necessario stabilire una funzione che valuti la bontà della composizione generata.

4.1 Algoritmi genetici

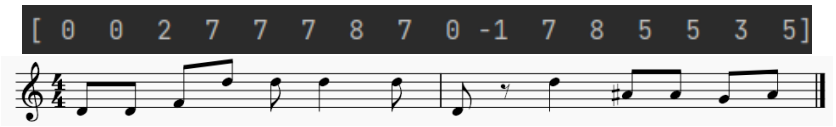
Una soluzione implementativa ricade sugli **algoritmi di ricerca locale genetici.** Essi non sono sempre capaci di trovare un ottimo globale come gli altri algoritmi di ricerca, ma, dato lo **spazio molto ampio di stati** presente nella composizione musicale, che accentuerebbe i problemi di prestazione di questi algoritmi e data **l'impossibilità di definire un ottimo globale** in quest'ambito, ci accontentiamo di soluzioni sub-ottime. Essi partono da una popolazione iniziale di individui, insieme di composizioni, per poi produrne di nuove e migliori rispetto ad una **funzione di fitness**, continuando il processo una volta raggiunto l'ottimo (assenza di miglioramenti) o dopo un specificato numero di iterazioni/generazioni. Gli algoritmi genetici si ispirano alla **teoria evoluzionistica** di Darwin e utilizzano degli operatori per creare nuove generazioni.

- **Selezione**: gli individui vengono selezionati oppure scartati dalla prossima generazione in base alla funzione di fitness
- **Crossover**: gli individui preesistenti vengono utilizzati per crearne di nuovi attraverso l'accoppiamento.
- **Mutazione**: gli individui sono soggetti a mutazioni, casuali o deterministiche, che modificano i loro geni esplorando meglio lo spazio delle soluzioni e introducendo geni che non sarebbe stato possibile introdurre solo con il crossover.

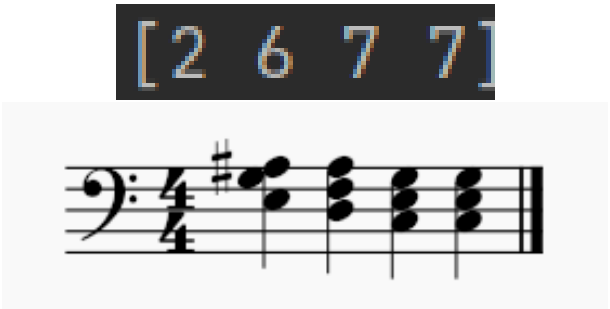
La funzione di fitness costituisce il cuore alla base degli algoritmi genetici e permette di **misurare l'adeguatezza della soluzione/composizione** rispetto al problema di trovare una composizione che sia piacevole da ascoltare.

4.1.1 Inizializzazione

Per la codifica dei geni della melodia ci si è ispirati ad un paper già esistente: *A genetic algorithm for composing music di Dragan Matic*. Si è optato per una **lista di numeri interi**, in cui ogni ogni numero può rappresentare una **pausa**, una **nota** oppure un **sustain**: aumento della durata della nota/pausa precedente della durata più piccola della composizione. Data la complessità del problema, si è deciso di impostare a priori, prima della partenza dell'algoritmo, un'unità di tempo che costituisca la **durata più piccola** nella composizione. Essa è, di solito, espressa in multipli di quattro e di conseguenza la grandezza della lista, ovvero dell'individuo/composizione, è data dal numero di battute moltiplicata la durata più piccola. Es. *se la durata più piccola scelta è 1/4 (semiminima) con 4 battute avremo una grandezza della lista/individuo di 16 geni/elementi musicali, in modo da riempire i 4/4 di ogni battuta musicale.*
Una volta scelta la durata più piccola e il numero di battute viene associata alla pausa il valore '-1', mentre invece al sustain viene associato il valore dell'indice dell'ultima nota a cui va aggiunto 1. Questo è un esempio di individuo i cui indici delle note vanno da '0' a '7', la pausa è rappresentata da '-1' e l'aumento di durata da '8'. La chiave è Re minore (Dmin), il numero di battute scelte è 2 e la durata minima è 1/8 (croma):



Per la codifica dei geni dell'armonia (progressione di accordi) si è deciso di usare una **lista a grandezza fissa di interi**, ovvero 4, dato che la progressione si ripete sempre uguale durante la composizione ogni 4 battute. Il gene rappresenta un **accordo** invece che un singolo elemento musicale e il suo valore si riferisce al suo **grado nella scala.**



Sia per la composizione melodica che quella armonica, la prima generazione di individui viene generata in maniera **casuale** nello spazio delle soluzioni ammissibili.

4.1.2 Operatori genetici

- **Selezione**: la libreria utilizzata, *pygad*, mette a disposizione diversi tipi di selezione
  - **"rws"**: Roulette Wheel
  - **"rank"**: Rank
  - **"tournament"**: k-way tournament (con valore di default di k impostato a 3)
  - **"sss"**: Steady State
  - **"random"**: Casuale
  - **"sus"**: Stocastica

  Esse sono state tutte testate per scegliere la configurazione migliore.

- **Crossover**: la libreria utilizzata, *pygad*, mette a disposizione diversi tipi di crossover
  - **"single_point"**
  - **"two_points"**
  - **"uniform"**
  - **"scattered"**

  Sono stati presi in considerazione per il testing solo il single point e il two points crossover.

- **Mutazione**: è stata creata una funzione di mutazione ad hoc, un'istanza particolare del random resetting: all'interno della classe *Composer* è possibile definire la probabilità che la mutazione dell'elemento musicale sia un aumento della frequenza (pitch up), abbassamento della frequenza (pitch down), trasformazione in pausa (rest), allungamento durata elemento musicale precedente (sustain) Da notare che:
  - se l'elemento musicale/gene è una nota ed essa non ha un'altra nota al di sopra di essa con frequenza più alta, allora la frequenza viene abbassata. Vale anche il viceversa.
  - se l'elemento musicale/gene è una pausa e viene scelto di aumentare/diminuire il pitch, allora viene scelta una nota casualmente
  - se l'elemento musicale/gene è un sustain e viene scelto di aumentare/diminuire il pitch, allora viene aumentato/diminuito il pitch della nota a cui fa riferimento il sustain

È stato creato un file python (*GeneticTester.py)* appositamente per verificare quale combinazione di algoritmo di selezione e crossover fosse migliore per il nostro problema. Sono stati effettuati 1000 test per combinazione e questi sono i risultati:

```
Average fitness for steady state with single point 8.144666666666687
Average fitness for steady state with two points 8.22376562500002
Average fitness for roulette wheel with single point 6.492458125000023
Average fitness for roulette wheel  with two points 6.594222202380974
Average fitness for stocastic with single point 7.111651041666698
Average fitness for stocastic with two points 7.115255000000033
Average fitness for rank with single point 8.008478125000023
Average fitness for rank with two points 8.35657812500002
Average fitness for random with single point 3.8451691369047563
Average fitness for random with two points 3.8018480357142828
Average fitness for tournament with single point 8.161843750000024
Average fitness for tournament with two points 8.033375000000017
The max is avg_fitness_rank_double with avg 8.35657812500002
```

La miglior combinazione si è rivelata essere la selezione con **rank** e crossover **two points**

4.1.3 Funzione di fitness

Come già menzionato prima, data la natura intrinsecamente **soggettiva** del problema, è difficile stabilire una funzione di fitness che soddisfi tutti. Ci si è quindi rifatti alla **teoria musicale** e si sono individuati diversi criteri per stabile la fitness di una composizione melodica:

- **Numero di cattivi intervalli su totale di intervalli**: in musica sono presenti alcuni intervalli oggettivamente dissonanti, che possono minare la bellezza di una composizione
- **Numero di note nella composizione su grandezza totale della composizione**: la musica non è composta solo da note ma anche da silenzi/pause
- **Numero di note che seguono l'accordo su note totali**: la melodia è più piacevole da sentire se le note suonate sono presenti nell'accordo suonato in sottofondo

Se diamo la possibilità all'agente di usare tutte le note possibili (non solo quelle in scala)

- **numero di note in scala su numero di note totali**: la maggior parte delle composizioni è scritta in una scala specifica, ma, a volte, vengono utilizzate e incoraggiate note fuori scala come, ad esempio, nella musica Jazz

La funzione di fitness è dunque combinazione lineare di questi criteri e nell'istanziazione di un *Composer* si possono associare diversi pesi a questi ultimi, a discrezione del musicista che usa il sistema.
Per la composizione armonica/progressione di accordi è stato invece optato per una **funzione di fitness interattiva**, che presenta all'utente ogni volta una nuova progressione di accordi da far valutare con un voto compreso tra 1 e 10, costituendo il valore di fitness.
Questa scelta progettuale costituisce un **collo di bottiglia** all'esecuzione dell'algoritmo, che è rallentato dall'utente che deve fornire il suo feedback alle progressioni generate. Di conseguenza non vengono spese molte iterazioni per trovare una progressione che sia piacevole all'utente, ma ciò è sufficiente per virare dalle consolidate progressioni di accordi e creare una composizione che sia **unica**.

4.1.4 Stopping condition

La libreria *pygad* utilizzata permette di definire un numero preciso di iterazioni/generazioni dopo il quale terminare l'algoritmo e restituire l'individuo migliore. Non sono stati notati miglioramenti tangibili dopo 100 generazioni, di conseguenza quest'ultimo valore è stato usato come stopping condition.

4.2 Implementazione

Come menzionato nelle sezioni precedenti, la libreria utilizzata per implementare la logica principale del sistema è stata *pygad*, libreria *Python* che permette di definire con grande facilità e personalizzazione un proprio algoritmo genetico. (pagina web di *pygad*)
Ulteriore libreria fondamentale è stata *music21*, che permette la semplice composizione di partiture tramite il paradigma ad oggetti di *Python*, restituendo in output uno spartito in formato *.midi* o *.musicxml*. L'output può essere in seguito riprodotto tramite software appositi per la manipolazione musicale. (pagina web di *music21*)
Nasce dunque il problema di mettere assieme queste due librerie, in modo da avere in output dal sistema una riproduzione istantanea della composizione generata dall'algoritmo genetico. Sono state quindi create due classi: *Key* e *Composer*

- *Key* modella la scala in cui è scritta la composizione e si occupa di effettuare il **mapping** tra valori riconoscibili da *music21*, ovvero stringhe che rappresentano il nome della nota, e valori riconoscibili da *pygad*/algoritmo genetico, ovvero interi, definendo così le note e gli accordi presenti nella scala.

  Prende in input una nota, ovvero la tonica della scala, e il modo. (i valori di default sono "*C*", ovvero *Do*, e "*Maj*", ovvero il modo maggiore)

  Sono supportate tutte le note e tutti i modi.

- *Composer* modella il compositore che si occuperà di fornire le informazioni e i metodi necessari all'algoritmo genetico per essere eseguito, definendo così la **struttura generale della composizione.**

  Prende in input numero di battute, durata dell'elemento musicale più piccolo, scala (istanza di *Key*), una progressione di accordi, una lista di intervalli considerati spiacevoli, i pesi dei criteri della funzione di fitness, le probabilità di mutazione dei diversi elementi musicali e un valore booleano che indica se comporre utilizzando tutte le note o esclusivamente quelle della scala.

  Il metodo principale della classe è "*toMusic21*" che trasforma una rappresentazione genetica di una melodia, ovvero la soluzione individuata dall'algoritmo, in uno stream riproducibile di *music21*. Altri metodi come "*in_scale*" e "*in_chord*" sono invece utilizzati dalla funzione di fitness per il calcolo del suo valore.

Allo stato attuale ci si è concentrati sulle funzionalità principali/back-end del sistema, ignorando completamente il front-end, rendendo la modifica dei parametri dell'algoritmo e/o compositore solo **direttamente tramite codice.**

5 Conclusioni e futuri sviluppi

Il sistema ottenuto può essere usato in maniera soddisfacente da un musicista come supporto alle sue attività compositive, andando a creare uno **scheletro di base** a quella che potrebbe essere una canzone. Sono inoltre stati definiti dei metodi nella classe *Composer* ("*toGeneticFromXML*" e "*toGeneticFromStream*") che permettono di ricevere in input la melodia principale di una canzone esistente e trasformarla in rappresentazione genetica: l'idea sarebbe di non inizializzare la popolazione in maniera randomica, ma bensì di sfruttare composizioni già esistenti per generare *mash-ups*.
Il sistema, allo stato attuale, soffre però **la mancanza di tocco soggettivo**, componendo semplicemente rifacendosi alla teoria: la musica, essendo una forma d'arte, è invece estremamente soggettiva. Uno sviluppo futuro potrebbe essere quindi di aggiungere all'algoritmo genetico una parte di **apprendimento**: la funzione di fitness non sarebbe più calcolata tramite combinazione lineare di criteri della teoria musicale, ma bensì tramite un **regressore**. Questo regressore sarebbe poi allenato su una classifica dei brani preferiti dell'utente, in modo che data in input una composizione generata tramite l'algoritmo genetico, il regressore restituisca in output il suo valore in classifica, ovvero il suo valore di fitness.

6 Glossario

Melodia: sequenza di note, suonate una dopo l'altra
Armonia: due o più note suonate nello stesso momento. Gli elementi primi dell'armonia sono gli accordi
Intervallo: distanza in semitoni tra due note. Fondamentale in musica per costruire le scale
Battuta: spazio nel pentagramma compreso tra due stanghette verticali. Serve a dividere le note in modo da facilitare la comprensione del ritmo
Chiave: sinonimo di scala, definisce le note principali della composizione
Modo: insieme ordinato di intervalli derivato da una corrispondente scala variando la nota iniziale
Tonica: si riferisce alla prima nota della scala e da anche il nome a quest'ultima
Grado: ruolo di una nota/accordo nella scala, es. tonica (1°), dominante (5°)
Musicxml: standard basato su xml che permette di codificare partiture musicali
Midi: protocollo standard per l'interazione degli strumenti musicali elettronici, sia tra di loro che con un computer
Mash-up: in ambito musicale, fusione tra due o più composizioni per generarne una nuova