MusiComposer

Implementazione in Prolog di algoritmi genetici per la composizione di assoli musicali per chitarra blues

***Abstract*—Il progetto è stato sviluppato dallo studente *Stefano Perniola* per il corso di *Intelligenza Artificiale* presso *UNIVPM*. La proposta progettuale è un'implementazione in Prolog di algoritmi genetici impiegati per la generazione automatica di melodie musicali. Si introduce brevemente il workflow adottato: inizialmente sono stati forniti dei frammenti musicali monofonici, che costituiscono la popolazione iniziale. Tali frammenti, chiamati nel gergo musicale riff o lick, sono particolarmente indicati per la generazione di melodie in quanto provenienti dall'ambiente blues e intensamente utilizzati dai musicisti durante delle sessioni di improvvisazione e assolo. Sui lick forniti verranno effettuate le tipiche operazioni degli algoritmi genetici (ovvero funzione di fitness, selezione, crossover e mutazione). In conseguenza di queste operazioni, si auspica di generare nuove forme ritmiche e melodiche, che sono in sostanza 'figlie' dei frammenti iniziali. Infine, è stato sviluppato un convertitore *"Prolog to MuiscXML"* per la visualizzazione grafica dello spartito musicale prodotto e per l'ascolto delle relative melodie ottenute. In virtù delle caratteristiche musicali dei lick forniti, è preferibile che il risultato finale venga interpretato da una chitarra elettrica, ma questo non esclude di certo la sperimentazione su strumenti musicali differenti. Si ringrazia infinitamente il professore *Aldo Franco Dragoni* per aver dato la possibilità di svolgere questo progetto.**

***Index Terms*—Prolog,intelligenza artificiale, algoritmi genetici, MusicXML, musica, blues, assolo, improvvisazione, chitarra**

I. Introduzione

A. Musica e processi creativi

Prima di proporre il tema centrale del progetto, bisogna fornire un quadro generale a proposito della definizione di creatività, necessaria per capire come i mezzi computazionali possono essere utilizzati per ottenere comportamenti affini. Non è facile tentare di dare una definizione del concetto di creatività in questo ambito. Intanto una possibile formulazione a livello generale potrebbe riferirsi all'abilità che alcuni individui umani possiedono nel creare qualcosa che prima non esisteva. A seguito di ulteriori riflessioni, si può dire che il più delle volte queste "creazioni" partono da concetti che quanto meno già esistevano, o almeno che potevano già esistere, ma che nessuno aveva già esplicitamente collegato in un prodotto fisso. La composizione musicale è un processo creativo tipicamente umano, che nasce dal cervello a seguito di spunti di ispirazione e sentimenti. Un tale processo impiega entrambi gli emisferi cerebrali: il sinistro, dedicato alla razionalità, alla logica, alla visione dettagliata, all'analisi, alla deduzione, al pensiero convergente e il destro, dedicato alla creatività, all'intuizione, alla visione globale, alla sintesi e al pensiero divergente. Mentre del primo emisfero, quello della logica, possiamo dire che i calcolatori elettronici siano ottimi interpreti, non sarebbe scontanto invece affermare lo stesso per l'emisfero destro, quello della creatività e dell'intuizione. L'idea di ottenere comportamenti creativi dai computer ha ispirato la scrittura di una notevole quantità di pubblicazioni scientifiche, che possono essere raccolte e catalogate nel campo multidisciplinare della creatività computazionale. Uno dei suoi sottocampi più prolifici è quello appunto della generazione musicale, che utilizza tecniche computazionali per comporre musica. A causa degli aspetti non deterministici e aleatori che caratterizzano la natura particolare della musica in generale e di questo campo di ricerca, a volte è difficile definire obiettivi precisi e tenere traccia di quali problemi possono essere considerati effettivamente risolti dai sistemi tecnologici. Quando si parla di creatività infatti, è difficile trovare un insieme di metriche oggettivamente quantificabili che la misurino. Per questi motivi, la sola e unica 'metrica' su cui si è basata la scrittura degli spartiti, è quella del cervello umano e delle sue sensazioni. Le tecniche utilizzate per lo sviluppo del compositore sono state pensate per cercare di raggiungere melodie coerenti per quanto riguarda il ritmo e la melodia dei brani, ma si demanda all'ascoltatore il giudizio della qualità e il livello gradimento della musica prodotta.

B. Motivi del progetto

Il motivo principale dello sviluppo di questo progetto ha probabilmente a che fare con la semplice curiosità di testare le potenzialità del Prolog e del suo paradigma di programmazione dichiarativo, nonchè dal desiderio di applicare i concetti appresi durante il corso di Intelligenza Artificiale in contesti pratici più o meno familiari. La natura cosi suggestiva e misteriosa degli algoritmi genetici ha subito portato a una particolare curiosità. Questa si è poi tradotta nell'intento di testare le tecniche algoritmiche nel mondo musicale, nella speranza di fare cenno a nuove forme musicali e ritmiche. Il desiderio di approfondire le tematiche del corso attraverso l'elaborazione di un progetto, è stato senz'altro ampliato dalla capacità del professore di coinvolgerci e renderci partecipi.

II. Algoritmi genetici

A. Descrizione generale

L'idea generale alla base degli algoritmi genetici è la seguente: partendo da una popolazione di soluzioni casuali a un problema, è possibile combinare tra loro alcune soluzioni per ottenerne nuove, e selezionando quelle che meglio rispondono al problema è possibile avvicinarsi sempre di più alla

soluzione ottimale del problema originale. Quindi, per risolvere un problema tramite algoritmi genetici, è necessario avere:

1. La capacità di generare soluzioni casuali ma adeguate al problema come popolazione iniziale;

2. Un modo per valutare la "idoneità" di una soluzione;

3. La capacità di mutare e ricombinare queste soluzioni.

Nel campo della generazione musicale, i punti 1 e 3 sono sicuramente disponibili (una volta scelta una rappresentazione del materiale musicale), ma è difficile valutare quanto sia buona una soluzione. Potrebbe essere difficile anche solo dare una definizione precisa di quale sia il problema.

B. Lo stato dell'arte

Forse, il più famoso generatore musicale è GenJam, progettato da Biles (1994) e pensato per l'improvvisazione Jazz. Si tratta di un software basato su algoritmi genetici a supporto di musicisti jazz alle prime armi con le improvvisazioni musicali. In sostanza il software emette una base musicale prefabbricata, mentre il musicista umano tenta di suonare un assolo. GenJam in seguito evolve l'improvvisazione umana che ha appena ascoltato. In origine, la funzione fitness veniva implementata facendo decidere a un essere umano se l'output era buono o cattivo, un approccio che di solito viene definito "Algoritmo genetico interattivo". Ciò genera un collo di bottiglia per il sistema, poiché è necessario un notevole intervento umano. Una versione successiva (Biles et al., 1996) utilizzava una rete neurale artificiale come funzione di fitness, ma portava a risultati insoddisfacenti. Alla fine, l'autore ha deciso di eliminare completamente la funzione fitness (Biles, 2001). Fondamentalmente, l'algoritmo mantiene la capacità di mutare e comporre lick, un'abilità che viene utilizzata per rispondere a input musicali in un modo che incorpora l'improvvisazione umana senza essere una semplice copia, ma poiché non c'è più valutazione della forma fisica, GenJam è non più un algoritmo genetico. Ci sono tante altre proposte in merito alla composizione automatica di musica, tra queste vengono riportate alcune che fanno uso di tecniche genetiche:

1. *A genetic algorithm for composing music - Dragan Matic Faculty of Natural Sciences University of Banjaluka, Bosnia and Herzegovina*

2. *Genetic Algorithms and Computer-Assisted Music Composition - Horner, Andrew; Goldberg, David E.*

3. *Opportunities for evolutionary music composition - Andrew R Brown 2002, Australasian Computer Music Conference*

4. *A Hybrid Neuro-Genetic Pattern Evolution System Applied to Musical Composition - Burton. PhD Thesis, University of Surrey, School of Electronic Engineering, 1998.*

5. *A Genetic Algorithm for Generating Improvised Music - Ender Özcan and Türker Erçal, Yeditepe University, Department of Computer Engineering - İstanbul, Turkey*

III. Nota e tempo musicale

A. Concetti base

La comprensione di alcuni aspetti del progetto è chiaramente influenzata dalla conoscenza del dominio musicale del lettore. Per questo motivo prima di addentrarsi nelle specifiche del programma implementato e per agevolare la comprensione delle soluzioni adottate, verranno brevemente illustrati alcuni concetti base della teoria musicale e della notazione musicale.

Di seguito viene riportata una definizione del concetto di nota musicale:

Una nota, nella notazione musicale, è un segno grafico usato per rappresentare un suono. Nella tradizione moderna occidentale, le note sono scritte sul pentagramma in modo da indicare contemporaneamente l'altezza e la durata del suono. Entrambe possono essere espresse da un unico segno o possono richiedere segni aggiuntivi: le alterazioni, che modificano l'altezza, i punti e le legature di valore, che incidono invece sulla durata.

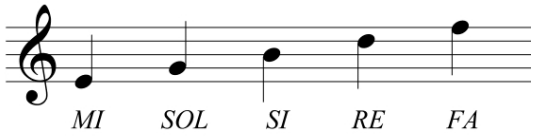


Fig. 1. Esempio di note musicali scritte sul pentagramma

Lo spazio in cui vengono scritte le note si chiama pentagramma e come suggerisce l'etimologia del termine è costituito da cinque righe. In sintensi quindi, una nota musicale è caratterizzata da due parametri fondamentali: altezza, che si riferisce a quanto una certa nota la sentiamo bassa o acuta, e durata.

Nella notazione musicale, il valore temporale di una nota, detto anche semplicemente durata, è un parametro che indica quanto deve essere prolungato nel tempo il suono rappresentato. Tale durata è relativa, ovvero dipende dal tempo di uno specifico passaggio o di una composizione. Nel sistema di rappresentazione occidentale le durate prendono i seguenti nomi: whole, half, quarter, eighth, 16th, 32th e cosi via. La durata di una cellula ritmica è esattamente la metà di quella precedente. Vi sono altresì simboli grafici che indicano la durata di pause. Altezza e durata delle note vengono infine rappresentate sul pentagramma attraverso gli opportuni simboli grafici.

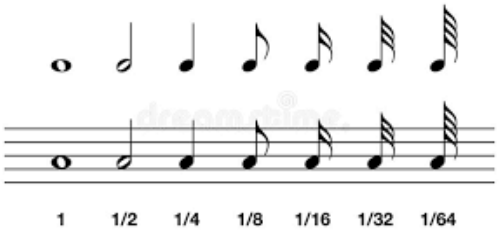


Fig. 2. Simboli delle principali cellule ritmiche

B. Nota e tempo musicale in Prolog

In Prolog, avendo la possibilità di sfruttare il paradigma dichiarativo del linguaggio, non è un'opera difficoltosa cercare di definire il concetto di nota musicale. Tuttavia, bisogna tenere conto dell'aspetto dicotomico delle note, che sono caratterizzate appunto dai due attributi fondamentali specificati prima (altezza e durata). Per quanto riguarda l'altezza della nota, è stato sufficiente esprimere un attributo *nota*, di cui è mostrato di seguito un esempio:

```
nota(c4,['C',4]).   nota(d4,['D',4]).   nota(e4,['E',4]).
nota(f4,['F',4]).   nota(g4,['G',4]).   nota(a4,['A',4]).
nota(b4,['B',4]).
```

Per indicare il nome delle note è stata utilizzata la notazione anglosassone, che fa utilizzo delle prime 7 lettere dell'alfabeto: [a,b,c,d,e,f,g]. Il numero posto accanto al nome della nota si riferisce all'ottava (altezza) di riferimento, invece la specifica del secondo parametro del predicato serve per favorire la codifica delle singole note nella struttura dati utilizzata.

Per quanto riguarda la definizione del concetto di "tempo", è stato utilizzato un predicato chiamato *cellula-ritmica*. La caratterizzazione del valore di ogni cellula ritmica è esplicitata nel secondo argomento del predicato, cosi che il programma conosca la reale durata di ogni elemento (espressa in *4/4*):

```
cellula-ritmica(whole,4).
cellula-ritmica(half,2).
cellula-ritmica(quarter,1).
cellula-ritmica(eighth,0.5).
... ... ...
```

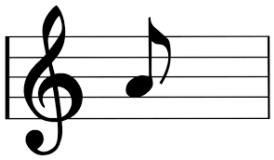
C. Estensione armonica nel progetto

L'estensione armonica è l'intervallo di tutte le altezze eseguibili da un determinato strumento musicale. Nel caso del progetto, per convenzione, è stato deciso di utilizzare l'estensione armonica della chitarra elettrica, che comprende 4 ottave. La nota più bassa della chitarra è precisamente il *Mi* di altezza 2 (*e2*), l'ultima nota invece, quella più acuta di tutte, sarebbe il *Si* di altezza 5 (*b5*).

IV. Struttura dati

Durante la fase di sviluppo del progetto, è stato fondamentale definire una struttura dati per la rappresentazione delle note musicali e del loro tempo. Tale struttura è stata definita per poter effettuare le operazioni dell'algoritmo genetico ma anche per poter trascrivere, mediante opportune codifiche, gli spartiti nel formato *MusicXML*. Si è pensato che la struttura dati ideale fosse quella che mediante una lista di elementi, riuscisse a rappresentare la nota da suonare con le sue caratteristiche in termini di altezza e tempo. Un insieme di note musicali suonate in sequenza vanno poi chiaramente ad identificare una lista di liste.

Con l'esempio seguente si tenta di chiarire ciò che è stato descritto:



La nota rappresentata in figura, un *La* di altezza 3 (*a3*), viene rappresentata in questo modo dalla struttura dati:

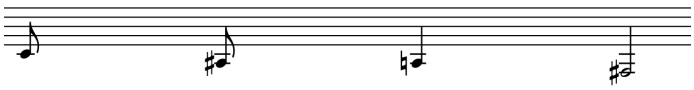
```
[['A',3],eighth].
```

La lista rappresentata ha due elementi. Il primo elemento è ancora una lista, che specifica la nota e la sua altezza, il secondo elemento invece rappresenta la sua durata. In questo caso, trattandosi di una croma, il valore utilizzato è *eighth*. Si ricorda che tutte le cellule ritmiche sono conosciute dal programma, in quanto implementate dalle dichiarazioni esposte in *cellula-ritmica(...)*.

Il programma è anche in grado di rappresentare note alterate, ovvero quelle in cui vi è l'indicazione di diesis e/o bemolle. L'alterazione serve per modificare l'altezza di una nota di un semitono in maniera ascendente o discendente. Per indicare l'alterazione di una nota nella struttura dati si è pensato di inserire un valore aggiuntivo nel primo elemento della lista, che indicasse il verso e intensità dell'alterazione:

```
[['A',3,1],eighth].
```

Sarà poi compito degli algoritmi definiti il saper gestire al meglio la struttura dati definita. Arrivati a questo punto si è in grado di mostrare un esempio di come viene codificata una successione di note musicali. La struttura dati istanziata non sarà altro che una lista di liste contenente le note e le loro caratteristiche. Esempio:



Le note rappresentate in figura possono essere descritte dalla struttura dati in questo modo:

```
[['C',3],eighth],[['A',2,1],eighth],[['A',2],quarter],[['F',2,1],half]]
```

V. MusicXML

A. Descrizione

MusicXML è un sistema di codifica basato sul noto linguaggio di markup *xml*. Più in particolare si tratta di uno standard open source utilizzato per la rappresentazione e lo scambio di spartiti musicali in formato digitale su Internet. Per questo motivo, non è difficile immaginare che i file

MusicXML siano supportati e utilizzabili da un'ampia gamma di applicazioni software di notazione musicale presenti nel mercato. La particolarità di questo formato, è proprio quella di offrire la possibilità non solo di rappresentare le note sul pentagramma, ma anche di suonarle in maniera interattiva. Proprio come i file *mp3* è diventato sinonimo di condivisione di musica registrata, diventando quindi famosissimo per la condivisione di spartiti interattivi anche fra dispositivi che utilizzano diversi programmi. Di seguito viene mostrato un esempio di come una nota musicale viene codificata nel formato *MusicXML*, in questo caso una croma di *Do diesis* di altezza 3 (in Prolog *[['C',3,1],eighth]*):

$$< note >$$
$$< pitch >$$
$$< step > C < /step >$$
$$< octave > 3 < /octave >$$
$$< alter > 1 < /alter >$$
$$< /pitch >$$
$$< type > eighth < /type >$$
$$< /note >$$

B. Convertitore Prolog-to-MusicXML

Come preannunciato prima, *MusiComposer* tra le varie funzionalità è anche in grado di convertire gli spartiti musicali prodotti nel formato *MusicXML*. Questo consente di poter subito visualizzare la composizione musicale una volta elaborata ma soprattutto sentirla suonare. La responsabilità della conversione dello spartito è assegnata al file chiamato *writer.pl* che esamina tutti i possibili casi di scrittura. Sostanzialmente una volta prodotta la struttura dati finale, vengono presi singolarmente gli elementi e a seconda dei vari casi viene scritto il relativo equivalente su *MusicXML* in termini di ottava, tempo ed eventuali alterazioni. In totale vengono esaminati 16 casi differenti, ognuno di essi con la loro caratterizzazione per quanto riguarda la conversione nel formato musicale desiderato:
1) nota non alterata, tempo naturale
2) nota non alterata, tempo croma con punto
3) nota non alterata, tempo croma iniziale della terzina
4) nota non alterata, tempo croma terzina
5) nota non alterata, tempo croma finale della terzina.
6) nota alterata, tempo naturale
7) nota alterata, tempo croma iniziale della terzina
8) nota alterata , tempo croma con punto
9) nota alterata, tempo croma terzina
10) nota alterata, tempo croma finale della terzina
11-16) 6 casi per la conversione delle pause

VI. Composer

Per quanto riguarda lo stile musicale interpretato, si è deciso di optare per melodie di stampo blues e jazz, in quanto la natura di questi generi musicali predispone in maniera più agevole alla generazione di assoli di carattere improvvisativo. Per raggiungere questo obiettivo, naturalmente il software deve avere a disposizione la 'conoscenza' di quali note possono essere suonate e quali no nella generazione di melodie di questo genere musicale. Così è stato implementato un predicato in Prolog che costruisse la scala blues musicale a partire da una tonica iniziale.

A. Scala blues

Il predicato che costruisce la scala blues ha la particolarità di conoscere solo i rapporti tra le note della scala in termini di distanze e non le note precise. Questo offre un vantaggio considerevole in quanto da una sola regola si possono ricavare tutte le scale blues possibili di ogni altezza e tonalità. Senza la regola infatti, sarebbe stato necessario enumerare esplicitamente tutte le scale blues, specificando per ognuna di esse le note precise. Nell'estensione musicale della chitarra elettrica esistono circa 90 modi diversi per suonare la scala blues in tutte le tonalità, ed enumerare esplicitamente un numero così alto di casi sarebbe stato decisamente prolisso.

B. Algoritmi per costruire la scala blues

La parte software responsabile della costruzione della scala blues è implementata nel file denominato *composer.pl* .Le distanze fra le note della scala blues sono state espresse in semitoni in questo modo: *[0,3,5,6,7,10,12]*. Per dare un'idea, la distanza *0* indica la tonica della scala, mentre la distanza *12* rappresenta sempre la tonica ma di un'ottava più alta.
I predicati che intervengono nel processo di costruzione della scala blues sono i seguenti:
I) costruisci-scala-blues(+Tonica,-Scala)
II) costruisci-indici-scalablues(+IndiceTonica,+Indici, -Res)
III) costruisci-note-scalablues(+ListaIndici,-Res1)
IV) costruisci-note2-scalablues(+ListaNote, -Scala)

Il primo di questi è il predicato principale che conosce le distanze fra le note e richiama tutti gli altri nel corpo della clausola. Il secondo predicato costruisce una lista contenente gli indici delle note della scala. Questi indici rappresentano la posizione delle note in relazione all'estensione armonica definita. Gli ultimi due predicati servono infine per codificare le note nel formato desiderato dalla struttura dati. Per chiarire meglio il concetto, forse è preferibile mostrare direttamente un esempio. Nell'esempio seguente viene illustrato passo dopo passo il workflow del predicato *costruisci-scala-blues(e3,Scala)*:
II) [12,15,17,18,19,22,24,27,29,30,31,34,36,39]
III) [e3,g3,a3,ad3,b3,d4,e4,g4,a4,ad4,b4,d5,e5,g5]
IV) [[E,3],[G,3],[A,3],[A,3,1],[B,3],[D,4],[E,4]]

VII. Descrizione dell'algoritmo genetico

Le operazioni dell'algoritmo genetico sono state implementate nel file *genesi.pl*. Attraverso l'esecuzione a riga di comando del predicato *genetico(+Tonica)*, è possibile ripercorrere i vari passi dell'algoritmo grazie alle operazioni di *write* inserite per facilitare le attività di debugging. In particolare, è possibile osservare quali siano i due genitori lick scelti, le note e i ritmi dei genitori, il punto di cut dei genitori (per il crossover), le posizioni delle note che verranno mutate nei figli, le note che andranno a mutare i figli, e infine i ritmi e le note finali dei due figli.

A. Popolazione iniziale

L'applicazione delle tecniche degli algoritmi genetici presuppone l'esistenza di una popolazione iniziale di soluzioni. Per quel che riguarda *MusiComposer*, sono stati selezionati per convenzione venti frammenti musicali di genere blues che costituissero un punto di partenza. Tali frammenti, chiamati lick, sono stati definiti attraverso dei precisi predicati che indicano le note e i tempi musicali da suonare. Per fare un esempio:

```
lick(+NumeroLick, -Lick)

lick(1, Lick) :-
costruisci-lick([0,2,-3,5],
[half,eighth,eighth,quarter], Lick), !.
```

In questo caso le note da suonare appaiono come primo argomento del predicato *costruisci-lick/2* e sono rappresentate mediante la lista contenente i seguenti indici *[0,2,-3,5]*. Invece le figure ritmiche da suonare sono quelle che appaiono al secondo argomento dello stesso predicato, ovvero *[half,eighth,eighth,quarter]*. Questo predicato, che viene utilizzato nel corpo della clausola principale, è stato utilizzato per poter codificare la struttura dati in maniera efficiente. La lista delle note e la lista delle figure ritmiche vengono mappate attraverso predicati ricorsivi definiti nel programma. In altre parole il primo elemento della prima lista si congiunge con il primo elemento della seconda lista e cosi via per gli altri elementi. Si ricorda che la lista delle note non è definita mediante una dichiarazione esplicita e diretta delle note musicali, ma bensì viene indicata la distanza in semitoni fra le note stesse e la tonica. In questo modo si ha il grosso vantaggio che ogni lick può essere suonato in una qualsiasi scala musicale conoscendo appunto i rapporti in termini di distanza che vi sono tra una nota e l'altra. Infatti, è stato sufficiente aggiungere un argomento (*Tonica*) per poter costruire frammenti musicali in qualsiasi scala desiderata: `lick(+NumeroLick, +Tonica, -Lick) :- ...`

B. Funzione di fitness e selezione

Come accennato prima, non è facile stabilire una funzione di fitness che valuti la bontà di una soluzione in questo contesto. Anche le soluzioni di altri software nel corso degli anni non sono riuscite a cogliere a pieno delle caratteristiche su cui far leva per poter valutare oggettivamente dei motivi musicali. Nella maggior parte dei casi, infatti, l'entità che stabiliva il grado di efficacia di una soluzione musicale era tipicamente umana. In *MusiComposer*, si è deciso che la qualità di un frammento musicale sia data da un solo parametro che è quello che riguarda la lunghezza in termini di battute musicali. In particolare, meno battute compongono il frammento, più a questo viene assegnato un valore di utilità più alto. La scelta di questa strategia non è del tutto casuale. Nel processo di selezione dei genitori, infatti, viene elaborata una distribuzione di probabilità sulla popolazione iniziale basata sui punteggi di fitness più alti. In altre parole, ai lick con punteggi più alti viene assegnato un valore di probabilità di essere scelto più alto. In questo modo, i frammenti musicali scelti con maggior frequenza saranno proprio quelli più 'piccoli'. Questo pensiero potrebbe offrire un vantaggio se consideriamo che il risultato finale sarà la combinazione di due genitori mediamente piccoli, e quindi con maggior speranza più dissimile e diversificato possibile da loro. Se, al contrario, la combinazione si basasse su genitori 'grandi', il risultato finale non sarebbe troppo diverso da loro, a meno che non ci fossero più punti di taglio.

C. Crossover

Nella fase di crossover viene individuato il punto di taglio dei due genitori. Questo servirà per effettuare la fase di incrocio, o appunto crossover, sia per la lista dei tempi che quella delle note. I predicati responsabili di questa fase sono *find-cut-point/5* e *cut/3*. Per mantenere un certo tipo di coerenza musicale dal punto di vista ritmico, si è pensato che un punto di taglio soddisfacente fosse quello che, una volta effettuato l'incrocio, mantenesse la lunghezza delle battute di *4/4* per entrambi i figli. Infatti, la popolazione iniziale è composta da lick che hanno un certo numero di battute da *4/4* e quindi sarebbe più agevole che anche i figli mantenessero una struttura tale.

Per fare un esempio pratico, vengono illustrati i tempi di due lick genitori: *[0.5,0.5,1,0.5,0.5,1]* e *[2,1,1]*. Un punto di taglio è ammissibile se entrambe le battute dei genitori possono essere divise in modo che le parti separate abbiano la stessa durata ritmica. Quindi il primo genitore potrebbe essere diviso in questo modo: *[0.5,0.5,1]* e *[0.5,0.5,1]*. Mentre il secondo genitore potrebbe essere diviso così: [2], [1,1]. Con questo procedimento, quando si andranno a incrociare le liste, le battute risultanti avranno ancora una lunghezza di *4/4* ciascuna. Infatti: [0.5,0.5,1,1,1] e [2,0.5,0.5,1]. Analogamente è stato effettuato tale procedimento sulla lista delle note dei lick.

D. Mutazione

Nella definizione generale dell'algoritmo genetico, la mutazione è una modifica casuale di una variabile all'interno di una soluzione per ottenere un'altra soluzione che può essere migliore o peggiore della precedente. In *MusiComposer*, questa fase si concentra sulla lista delle note dei figli. In particolare, viene selezionata una nota nella lista in maniera

casuale e successivamente la si rimpiazza con un'altra. Sono state fatte diverse prove in merito a quale possa essere una buona strategia di mutazione. Alla fine si è deciso, per mera impressione e sensazione, che una buona soluzione potrebbe essere quella di mantenere la nota mutata distante al più di un intervallo di quinta dalle note adiacenti. Questa scelta è stata dettata dal fatto che, andando a rimpiazzare delle note, si riteneva preferibile evitare eventuali 'balzi' troppo alti o bassi durante l'esecuzione di una melodia musicale e piuttosto, ci si voleva mantenere attorno ad un centro tonale delimitato.
Esempio (estratto dall'output dell'algoritmo genetico):

```
Nota adiacente 1: -7
Nota adiacente 2: -2
Nota mutante: -3
[-12,-9,-7,-5,-2,0]-->[-12,-9,-7,-3,-2,0]
```

In questo caso, la nota da mutare era *-5*, ed è stata mutata con *-3*, una nota che mantiene appunto una distanza massima di una quinta (*7* semitoni) dalle note adiacenti.

E. Figli

Una volta ottenuti i lick figli, si è pensato di salvarli mantenendo una struttura simile a quelli dei lick iniziali. Sono state utilizzate delle operazioni di scrittura su file mediante i predicati *built-in* del *Prolog*. In particolare, nel file *figli.pl* sono stati inseriti tutti i risultati delle operazioni dell'algoritmo genetico mediante le stesse regole che descrivevano i lick genitori. In questo modo, si ha lo stesso vantaggio di questi, ovvero quello che possono essere suonati in qualsiasi tonalità musicale. Il predicato responsabile di tali operazioni di *I/O* è *costruisci-predicato(+ListaNote,+ListaRitmi)*, e non fa altro che scrivere sul file in questione le informazioni salvate.
Esempio di come viene scritto un figlio nel file *figli.pl*:

```
lickfiglio(1, Lick) :-
costruisci-lick(Indice,[-12,-14,-9,-12],
[quarter,eighth,eight,half],Lick),!.
```

Sarebbe interessante far notare che il programma è autonomamente in grado di assegnare numeri sequenziali ai lick figli una volta generati. Questo è reso possibile mediante l'utilizzo del predicato *clause/2*, che, se utilizzato adeguatamente, permette di contare letteralmente il numero dei lick figli già presenti nel file:

```
findall(-,clause(lickfiglio(-,-,-),-),P).
```

VIII. Considerazioni finali

Il software sviluppato rappresenta una prima versione di quello che può essere un generatore di melodie musicali. Le possibilità di sviluppo, così come l'evoluzione delle feature già presenti, sono ancora ampie. Da un punto di vista pragmatico il programma potrebbe essere visto come plugin dedicato appunto alla creazione di riff e assoli. Ma tra le applicazioni pratiche che il programma potrebbe fornire, se ne potrebbero citare altre, soprattutto in prospettiva di avanzamenti dei lavori. Ad esempio, per i professionisti potrebbe essere impiegato come strumento di supporto alla composizione musicale, alla ricerca, alla sperimentazione e sviluppo di nuove forme ritmiche e melodiche, a partire dalle melodie iniziali fornite dai musicisti. Per i musicisti alle prime armi, potrebbe essere un buono strumento didattico per apprendere in maniera più coinvolgente l'esecuzione di assoli improvvisati, pur non conoscendo la teoria musicale. Al momento ci sono versioni forse più sviluppate nel mercato. Tra queste, si potrebbe citare *Riffer*, che si comporta in maniera molto simile a *Musi-Composer*. Il programma permette di generare una sequenza casuale ma coerente di note una volta impostata la tonalità e scala desiderate. Al di là di tutto, è opportuno ricordare che l'intuizione e la passione sono attualmente i mezzi più potenti per la composizione musicale e per la creatività artistica in generale. Mezzi che, purtroppo o per fortuna, a seconda delle prospettive del lettore, sono ancora tipicamente di natura umana. D'altronde *"ancora oggi non è chiaro se il successo degli algoritmi genetici derivi dalle loro prestazioni o dalla loro natura così suggestiva ed esteticamente piacevole. Rimane ancora molto lavoro da svolgere per identificare le condizioni in cui operano al meglio", Russell and Norvig - Intelligenza Artificiale, un approccio moderno.*