Music Composition by Interactive Evolutionary and Genetic Computation

Neil Babson

Haskore Music Library

```
> data Music = Note Pitch Dur [NoteAttribute]   -- a note \ atomic
>            | Rest Dur                        -- a rest /    objects
>            | Music :+: Music                 -- sequential composition
>            | Music :=: Music                 -- parallel composition
>            | Tempo  (Ratio Int) Music        -- scale the tempo
>            | Trans  Int Music                -- transposition
>            | Instr  IName Music              -- instrument label
>            | Player PName Music              -- player label
>            | Phrase [PhraseAttribute] Music  -- phrase attributes
>     deriving (Show, Eq)

> type Pitch      = (PitchClass, Octave)
> data PitchClass = Cf | C | Cs | Df | D | Ds | Ef | E | Es | Ff | F | Fs
>                 | Gf | G | Gs | Af | A | As | Bf | B | Bs
>       deriving (Eq,Ord,Ix,Show,Read)
> type Octave     = Int
```

Rest (3 % 16) :+: Note (F,5) (1 % 2) [],(Rest (11 % 16) :+: Note (C,5) (1 % 16) []) :=: ((Rest (13 % 16) :+: Note (C,5) (1 % 2) []) :=: ((Rest (1 % 16) :+: Note (G,5) (1 % 16) []) :=: ((Rest (9 % 16) :+: Note (B,5) (3 % 32) []) :=: ((Rest (7 % 8) :+: Note (A,5) (1 % 2) []) :=: ((Rest (0 % 1) :+: Note (B,4) (1 % 4) []) :=: ((Rest (5 % 8) :+: Note (A,4) (1 % 4) []) :=: ((Rest (3 % 16) :+: Note (G,5) (3 % 16) []) :=: (Rest (1 % 2) :+: Note (E,5) (1 % 16) [])))))))))) :+: Note (B,5) (1 % 16) []) :=: ((Rest (13 % 16) :+: Note (C,6) (3 % 32) []) :=: ((Rest (3 % 8) :+: Note (F,5) (3 % 8) []) :=: ((Rest (3 % 4) :+: Note (D,4) (3 % 16) []) :=: ((Rest (3 % 4) :+: Note (F,5) (1 % 1) []) :=: ((Rest (5 % 8) :+: Note (G,5) (1 % 4) []) :=: ((Rest (7 % 16) :+: Note (B,5) (1 % 2) []) :=: ((Rest (1 % 4) :+: Note (E,4) (1 % 4) []) :=: ((Rest (5 % 16) :+: Note (C,4) (1 % 4) []) :=: (Rest (7 % 8) :+: Note (B,4) (3 % 32) [])))))))))))),(Rest (15 % 16) :+: Note (C,6) (1 % 1) []) :=: ((Rest (1 % 4) :+: Note (F,5) (1 % 4) []) :=: ((Rest (1 % 4) :+: Note (B,4) (1 % 1) []) :=: ((Rest (1 % 2) :+: Note (A, 4) (3 % 32) []) :=: ((Rest (5 % 16) :+: Note (C,6) (1 % 4) []) :=: ((Rest (3 % 4) :+: Note (E,4) (3 % 8) []) :=: ((Rest (7 % 16) :+: Note (C,5) (1 % 4) []) :=: ((Rest (3 % 16) :+: Note (B,3) (3 % 16) []) :=: ((Rest (13 % 16) :+: Note (C,4) (1 % 4) []) :=: (Rest (0 % 1) :+: Note (E,4) (3 % 8) [])))))))))),(Rest (5 % 8) :+: Note (C,4) (1 % 1) []) :=: ((Rest (3 % 8) :+: Note (F,4) (3 % 32) []) :=:

Atomic Elements of Random Music Generation

-- 2 octaves of C Major scale used to generate random musical measures
notes = [(B,3),(C,4),(D,4),(E,4),(F,4),(G,4),(A,4),(B,4),
(C,5),(D,5),(E,5),(F,5),(G,5),(A,5),(B,5),(C,6)]

-- Starting time of a note in a measure
start = [(0%16), sn, en, den,qn, (5%16), dqn,
(7%16), hn, (9%16), (5%8), (11%16),
dhn, (13%16), (7%8), (15%16)]

-- Duration of a note. Dotted half notes and whole notes are half as likely as other durations
dur' = [sn,sn,dsn,dsn,en,en,den,den,qn,qn,dqn,dqn,hn,hn,dhn,wn]

- Each tone is a 3-tuple (note, start, dur’)

([((C,6),3 % 16,3 % 8),((E,4),0 % 1,1 % 16),((D,5),3 % 16,3 % 32),((G,5),1 % 16,1 % 4),((C,6),9 % 16,1 % 16),((F,4),13 % 16,3 % 4),((D,5),1 % 2,3 % 16),((F,5),3 % 4,3 % 16),((A,4),1 % 16,3 % 16),((E,5),5 % 16,1 % 4),((E,5),15 % 16,3 % 16),((B,3),7 % 16,1 % 16),((G,5),7 % 8,3 % 16),((E,4),3 % 16,3 % 16),((A,4),1 % 8,3 % 16)],[((E,4),3 % 8,3 % 8),((E,5),3 % 16,1 % 16),((F,4),13 % 16,3 % 4),((F,5),3 % 4,3 % 16),((A,4),1 % 16,3 % 16),((F,5),1 % 4,1 % 8),((E,5),15 % 16,3 % 16),((B,3),7 % 16,1 % 16),((E,4),3 % 16,3 % 16),((A,4),1 % 8,3 % 16)],[((A,4),7 % 16,1 % 2),((D,4),1 % 8,3 % 32),((G,5),7 % 16,1 % 4),((D,4),3 % 8,3 % 8),((A,5),15 % 16,3 % 4),((F,5),1 % 4,1 % 8),((E,5),15 % 16,3 % 16),((B,3),7 % 16,1 % 16),((E,4),3 % 16,3 % 16),((A,4),1 % 8,3 % 16)],[((A,4),7 % 16,1 % 2),((D,5),1 % 4,1 % 2),((E,4),3 % 8,3 % 8),((D,4),1 % 8,3 % 32),((G,5),7 % 16,1 % 4),((D,5),1 % 2,3 % 16),((D,4),3 % 8,3 % 8),((A,5),15 % 16,3 % 4),((E,5),0 % 1,1 % 16)],[((A,4),7 % 16,1 % 2),((D,4),1 % 8,3 % 32),((G,5),7 % 16,1 % 4),((D,4),3 % 8,3 % 8),((A,5),15 % 16,3 % 4),((F,5),1 % 4,1 % 8),((E,5),15 % 16,3 % 16),((B,3),7 % 16,1 % 16),((E,4),3 % 16,3 % 16),((A,4),1 % 8,3 % 16)],[((C,6),3 % 16,3 % 8),((E,4),0 % 1,1 % 16),((D,5),3 % 16,3 % 32),((G,5),1 % 16,1 % 4),((F,5),3 % 4,3 % 16),((A,4),1 % 16,3 % 16),((E,5),15 % 16,1 % 2),((E,4),11 % 16,1 % 2),((A,4),13 % 16,1 % 4),((C,4),7 % 8,1 % 2),((G,5),7 % 16,1 % 4),((D,5),1 % 2,3 % 16),((D,4),3 % 8,3 % 8),((A,5),15 % 16,3 % 4),((E,5),0 % 1,1 % 16)],[((G,5),13 % 16,1 % 8),((A,4),3 % 4,1 % 8),((C,6),9 % 16,1 % 16),((A,4),1 % 16,3 % 16),((E,4),11 % 16,1 % 2),((C,4),7 % 8,1 % 2),((G,5),7 % 16,1 % 4),((D,5),1 % 2,3 % 16),((A,5),15 % 16,3 % 4),((E,5),0 % 1,1 % 16),((F,5),1 % 4,1 % 8),((E,5),5 % 16,1 % 4),((E,5),15 % 16,3 % 16),((B,3),7 % 16,1 % 16),((G,5),7 % 8,3 % 16),((E,4),3 % 16,3 % 16),((A,4),1 % 8,3 % 16)],[((C,6),3 % 16,3 % 8),((E,4),0 % 1,1 % 16),((D,5),3 % 16,3 % 32),((G,5),1 % 16,1 % 4),((F,5),3 % 4,3 % 16),((A,4),1 % 16,3 % 16),((E,5),15 % 16,1 % 2),((E,4),11 % 16,1 % 2),((A,4),13 % 16,1 % 4),((C,4),7 % 8,1 % 2),((G,5),7 % 16,1 % 4),((D,5),1 % 2,3 % 16),((D,4),3 % 8,3 % 8),((A,5),15 %

Infinite Random Lists

```
-- Takes an integer seed and creates an infinite list of random numbers between 0 and 15
randNums :: Int -> [Int]
randNums gen = randomRs (0,15) (mkStdGen gen)

-- Make a list of 3-tuples, each defining a note, a start time, and a duration
makeTones 0 _ = []
makeTones n rands = ((notes !! (head rands)), (start !! (head(tail (take 2 rands)))), (dur' !! (head (
reverse ( take 3 rands))))) : makeTones (n-1) (drop 3 rands)

-- Make a new random measure with between 0 and 15 notes
initMeasure rands = makeTones (head rands) (drop 1 rands)
```

Evolutionary Algorithm

- Play population of 16 song measures to the user, who assigns each a fitness score
- Sort measures by fitness and kill bottom half
- Mutate each surviving measure to create eight new measures
- Form next generation from the survivors and their mutant offspring
- Shuffle measures and repeat until half of the population achieve perfect fitness score
- Pass eight best measures to the genetic algorithm

Mutation

```
--mutate :: Int -> [(a,b,c)] -> [Int] -> [(a,b,c)]
mutate 10 x rand = x
mutate rank x (y:ys) | y < 5     = mutate (rank+1) (dropNote x ys) (drop 2 ys)
                     | y < 10    = mutate (rank+1) (addNote x ys) (drop 7 ys)
                     | otherwise = mutate (rank+1) (changeNote x ys) (drop 7 ys)

--dropNote :: [(a,b,c)] -> [Int] -> [(a,b,c)]
dropNote x (y:ys) | (length x) == 0 =  x
                  | otherwise = take num x ++ drop (num+1) x
                    where num = (y * 7) `mod` (length x)

--addNote :: [(a,b,c)] -> [Int] -> [(a,b,c)]
addNote x y = x ++ (makeTones 1 y)

--changeNote :: [(a,b,c)] -> [Int] -> [(a,b,c)]
changeNote  x (y:ys) = addNote (dropNote x (y:ys)) ys
```

Genetic Algorithm

- Randomly construct songs from eight completed measures
- Play songs and get fitness scores
- Sort songs by fitness and kill bottom half
- Construct four new songs by mating the first and fourth ranked, and the second and third ranked
- Mutate one in three offspring by shifting measures
- Make next generation by combining survivors and offspring
- Shuffle and repeat until one of the songs gets perfect fitness ranking

Mating

```
-- Two songs combine to make two new children
mate (m0,m1,m2,m3,m4,m5,m6,m7) (n0,n1,n2,n3,n4,n5,n6,n7) (r:rs)
    | n == 0 = mate (m0,m1,m2,m3,m4,m5,m6,m7) (n0,n1,n2,n3,n4,n5,n6,n7) rs -- try again
    | n == 1 = (m0,n1,n2,n3,n4,n5,n6,n7) : (n0,m1,m2,m3,m4,m5,m6,m7) : []
    | n == 2 = (m0,m1,n2,n3,n4,n5,n6,n7) : (n0,n1,m2,m3,m4,m5,m6,m7) : []
    | n == 3 = (m0,m1,m2,n3,n4,n5,n6,n7) : (n0,n1,n2,m3,m4,m5,m6,m7) : []
    | n == 4 = (m0,m1,m2,m3,n4,n5,n6,n7) : (n0,n1,n2,n3,m4,m5,m6,m7) : []
    | n == 5 = (m0,m1,m2,m3,m4,n5,n6,n7) : (n0,n1,n2,n3,n4,m5,m6,m7) : []
    | n == 6 = (m0,m1,m2,m3,m4,m5,n6,n7) : (n0,n1,n2,n3,n4,n5,m6,m7) : []
    | n == 7 = (m0,m1,m2,m3,m4,m5,m6,n7) : (n0,n1,n2,n3,n4,n5,n6,m7) : []
    | otherwise = []
        where n = r `div` 2
```