Pembuatan Musik Tanpa Pola Dengan Menggunakan Algoritma Runut Balik

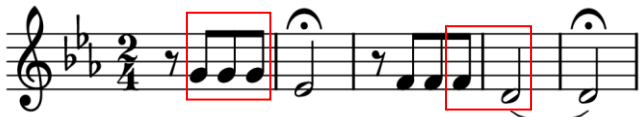
Aldrich Valentino Halim/13515081
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung (ITB)
Bandung, Indonesia
13515081@std.stei.itb.ac.id, aldrich.vh97@gmail.com

***Abstrak*—Pembuatan musik tidak pernah terlepas dari pola (*pattern*), baik pola nada, pola ritmis, maupun pola interval atau loncatan. Penggunaan pola adalah kunci dari sebuah musik yang baik dan indah. Namun, jika terdapat suatu musik yang tidak memiliki pola sama sekali, maka musik akan terdengar aneh dan tidak indah. Pembuatan musik tanpa pola ini pernah disimulasikan dengan menerapkan teori bilangan prima. Pada makalah ini, penulis akan menerapkan algoritma runut balik (*backtracking*) untuk membuat suatu musik tanpa pola dari sekumpulan not. Program dibuat dengan menggunakan bahasa pemrograman Python dan akan diuji dengan beberapa masukan untuk mengetahui performansi dari algoritma ini. Setelah itu, hasil dianalisis kemangkusannya dan akan ditarik kesimpulkan apakah algoritma ini cocok untuk menyelesaikan persoalan musik bebas pola atau tidak.**

Kata kunci—pola, musik, algoritma, runut balik, masukan, Python, kemangkusan.

I. Pendahuluan

Pada Oktober 2011, seorang matematikawan bernama Scott Rickard memformulasikan suatu musik yang dikatakan musik terburuk di dunia (*World's Ugliest Music*). Ia mengatakan bahwasannya yang membuat suatu musik indah adalah repitisi dan pola. Terdapat banyak repitisi dan pola di musik, beberapa di antaranya adalah pola nada, pola ritmis, dan pola interval. Pola nada adalah sekumpulan nada-nada yang diulang beberapa kali sebagai tema sebuah lagu. Pola ritmis adalah bagaimana sekumpulan nada-nada dimainkan sehingga memiliki waktu dan ketukan yang sama. Pola interval adalah sekumpulan nada-nada, tidak harus sama, yang memiliki jarak antara satu nada ke nada lain yang sama sehingga menghasilkan suatu pola. Sebuah musik yang baik dan indah harus mengandung pola-pola musik tersebut. Itulah tugas dari seorang *arranger*, membuat sebuah musik yang memiliki pola-pola musik yang tepat dan cukup.



Gambar 1 Pembukaan Beethoven Symphony 5 memiliki pengulangan nada Sol dan Fa sebanyak 3 kali dan Pola interval menurun 2 yang berulang sebanyak 2 kali

Namun, apabila sebuah lagu tidak memiliki pola yang direpetisi sama sekali, musik akan terdengar membosankan dan tidak memiliki alur. Hal inilah yang menjadi ketertarikan Scott Rickard, yaitu untuk membuat suatu musik yang bebas dari pola. Secara definisi, musik yang dikomposisi oleh Scott tidak dapat didengar sebagai sebuah lagu, tetapi Scott berkata dalam mendengarkannya, para pendengar disarankan untuk mencari adanya suatu pola kesamaan maupun pola yang diulang, sehingga timbul suatu kesadaran bahwa lagu tersebut tidak memiliki pola sama sekali.

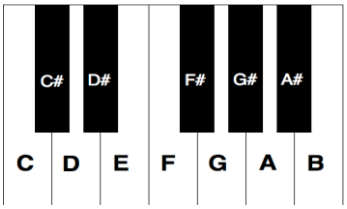
Dalam makalah ini, penulis akan membuat suatu program untuk membangkitkan sebuah musik bebas pola dengan algoritma runut balik (backtracking). Cara kerja program adalah pertama program akan menerima sebuah masukan berupa banyaknya nada atau not yang ingin disusun. Lalu, program akan menyusun nada-nada tersebut lalu memberikan hasilnya. Algoritma runut balik ini adalah metode pencarian secara *brute-force* yang sudah dioptimasi sehingga pencarian tidak perlu dilakukan terhadap semua himpunan solusi yang memungkinkan. Untuk setiap percobaan akan dilakukan pencatatan terhadap waktu eksekusi beserta banyaknya balik (*backtrack*) yang dilakukan program.

II. Dasar Teori

A. Komponen dalam Musik

1) Tangga Nada

Tangga nada adalah susunan terurut menaik dari nada-nada pokok suatu sistem nada, mulai dari nada dasar sampai dengan nada oktafnya. Contoh yang paling sederhana adalah tangga nada mayor, yaitu do, re, mi, fa, sol, la, si, do. Dalam tangga nada mayor, terdapat suatu skala baku yang diikuti oleh seluruh tangga nada mayor di kunci apapun. Skala adalah jarak antar nada dalam suatu tangga nada atau sistem nada. Contoh dalam tangga nada mayor adalah 1 - 1 - ½ - 1 - 1 - 1 - ½.



Gambar 2 Tangga Nada dalam Piano

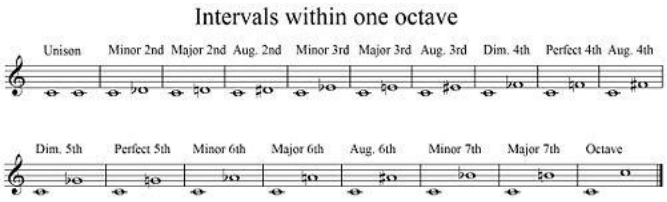
Tangga nada kromatik (*chromatic scale*) adalah suatu tangga nada dengan skala antara tiap nada adalah setengah (½) atau satu *semitone* (catatan: 1 *semitone* = ½). Sebuah tangga nada kromatik berisi dua belas not. Untuk lebih jelasnya, tangga nada kromatik berisi seluruh not yang ada dalam tangga nada mayor (dalam piano tervisualisasi dalam tuts putih dan hitam).

2) Semitone

Semitone adalah suatu istilah untuk menunjukan dua nada yang memiliki jarak setengah (½). Untuk dua nada yang memiliki jarak satu, misalnya dari do ke re atau dari fa ke sol, maka dapat dikatakan do ke re dan fa ke sol memiliki jarak dua *semitone*.

3) Interval Nada

Interval Nada adalah jarak dari suatu nada ke nada yang lain. Terdapat delapan buah interval dasar dalam sebuah tangga nada, yaitu 1-1 (unison), 1-2 (Major 2nd), 1-3 (Major 3rd), 1-4 (Perfect 4th), 1-5 (Perfect 5th), 1-6 (Major 6th), 1-7 (Major 7th), dan Oktaf.

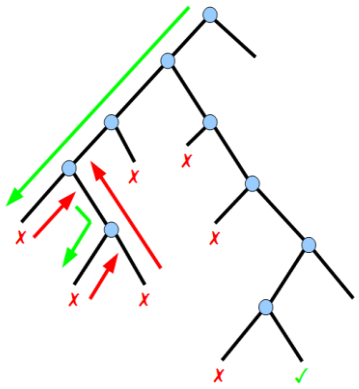


Gambar 3 Daftar Interval Nada

Untuk merepresentasikan interval dalam tangga nada kromatik, setiap interval dilengkapi dengan *minor*, *augmented*, dan *diminished*. Jika interval awalnya memiliki skala *major*, maka untuk menurunkan intervalnya sebanyak satu *semitone* cukup ditulis dengan *minor*. Contohnya dalam menuliskan 1-2^{b}, cukup ditulis dengan *minor* 2nd. Jika awalnya interval memiliki skala *perfect* (untuk 4th dan 5th), maka untuk menurunkan intervalnya sebanyak satu *semitone* ditulis dengan *diminished*. Selain itu, untuk interval yang awalnya *major* ataupun *perfect*, jika ingin menaikan intervalnya sebanyak satu *semitone*, maka skala dituliskan dengan *augmented*. Contohnya untuk menuliskan interval 1-5$^{\#}$, cukup ditulis dengan *augmented* 5th.

B. Algoritma Runut Balik

Algoritma Runut Balik atau *backtracking* adalah algoritma komputasi berbasis *Depth First Search* (DFS) yang sudah dioptimasi agar dapat mencari solusi persoalan secara lebih mangkus. Algoritma runut balik adalah perbaikan dari algoritma *brute-force* sehingga pencarian tidak dilakukan terhadap seluruh kemungkinan solusi.



Gambar 4 Visualisasi Algoritma Runut Balik

Beberapa komponen dari algoritma runut balik adalah sebagai berikut.

1) Solusi Persoalan

Solusi dinyatakan sebagai sebuah vektor yang beranggotakan *n* buah elemen, di mana *n* adalah besarnya masukan persoalan. Setiap elemen merupakan anggota dari himpunan solusi (S).

$$X = (x_1, x_2, x_3, \ldots, x_n);\ x_i \in S_i$$

2) Fungsi Pembangkit

Dilambangkan sebgai predikat T(k), yaitu fungsi yang bertugas untuk membangkitkan suatu nilai untuk x_k dan merupakan anggota dari himpunan vektor solusi.

3) Fungsi Pembatas

Suatu predikat (dilambangkan dengan B) yang merupakan vektor beranggotakan *n* buah elemen yang dapat bernilai benar atau salah. Suatu pembatas B bernilai benar berarti (x_1, x_2, …, x_n) mengarah ke solusi, dan tidak jika bernilai salah.

$$B(x_1, x_2, x_3, \ldots, x_n);\ B \in \{True, False\}$$

III. Penerapan Algoritma Runut Balik

Persoalan musik bebas pola dimodelkan dalam sebuah persoalan yang akan diselesaikan dengan algoritma runut balik. Pertama, terdapat sebuah *n* buah not dalam tangga nada, baik tangga nada mayor maupun kromatik. Dengan menggunakan algoritma runut balik, akan dicari suatu susunan dari *n* buah not tersebut sehingga tidak ada not yang berulang dan tidak ada interval yang sama antara satu not dengan not yang lainnya.

A. Representasi Tangga Nada dalam Larik

Tangga nada akan dimainkan berdasarkan waktu. Agar tidak ada pengulangan nada, setiap nada akan dimainkan tepat sekali dalam satu satuan waktu, maka untuk *n* buah nada akan ada *n* satuan waktu untuk memainkannya. Setiap nada yang dimainkan akan memiliki satu waktu eksekusi. Jadi, akan dibuat sebuah larik berukuran *n* elemen yang melambangkan nada ke berapa untuk waktu eksekusi tertentu.

Waktu eksekusi	1	2	3	4	…
Not	5	1	2	6	…

B. Representasi Pencatatan Pola Interval

Pola interval adalah perbedaan jarak antara suatu nada dengan nada yang lain. Untuk pencatatan dalam program, selisih antara satu nada dengan nada yang lain direpresentasikan sebagai bilangan bulat (1, 2, 3, …). Karena tidak boleh ada satu pun not yang berulang, maka interval unison (1 ke 1 atau

perbedaan 0 semitone) dapat dihilangkan. Untuk *n* buah not yang akan disusun, terdapat (n - 1) buah interval, yaitu banyaknya lompatan antara *n*.

$$\sum Interval\ Between\ Notes = n - 1$$

Untuk setiap intervalnya, naik dan turun dapat dianggap sebagai interval yang berbeda. Sehingga, jumlah interval dikalikan dengan dua.

$$\sum Intervals = 2(n - 1)$$

Dalam program, pencatatan interval dibuat dalam bentuk *map* yang seluruh elemennya bertipe *boolean*. Suatu elemen dengan *key* ke-i memiliki nilai benar jika interval ke-i sudah digunakan, dan salah jika belum digunakan. Berikut adalah visualisasi dari *map* yang akan digunakan.

Interval	-2	-1	1	2
Apakah interval sudah terpakai?	False	False	True	False

C. Komponen dalam Algoritma Runut Balik

Solusi persoalan direpresentasikan dalam bentuk sebuah *tuple* beranggotakan banyaknya not. Tiap elemen ke-k dalam *tuple* akan merepresentasikan satu satuan waktu dan isi dari elemen ke x_k adalah not yang akan dimainkan.

$$X = (x_1, x_2, x_3, \ldots, x_n);\ x_k \in \{1, 2, 3, \ldots, n\}$$

Fungsi pembangkit untuk persoalan ini adalah membangkitkan nada yang mungkin untuk dimainkan dalam satu satuan waktu. Fungsi pembangkit tidak memperhitungkan apakah not atau nada yang dibangkitkan sudah pernah dipilih atau tidak.

Fungsi pembatas adalah fungsi yang bertugas untuk menghentikan pembangkitan simpul ke arah yang tidak menuju kepada tujuan. Ada dua hal yang diuji dalam fungsi pembatas, yaitu apakah dalam waktu ke-k terdapat dua buah nada yang sama dan apakah interval dari nada ke-k dan ke-(k-1) sudah pernah digunakan atau belum. Bila nada belum pernah ada dan interval belum pernah digunakan, nada tersebut sahih untuk diletakan pada waktu ke-k. Jika salah satu saja dari kedua kondisi tersebut salah, maka nada tidak sahih untuk diletakan pada waktu tersebut dan harus dilakukan *backtrack* atau pencarian dilanjutkan ke nada selanjutnya.

D. Algoritma Runut Balik dalam Pseudocode

Berikut adalah deklarasi variabel dari program.

```
DEKLARASI KAMUS
n       : integer
Pattern : map [-(n-1) . . (n-1)] of boolean
Times   : array [1 . . n] of integer
```

Terdapat tiga buah variabel utama, yaitu n, Pattern dan Times. Variabel n adalah banyaknya not yang akan diatur posisinya. Variabel Pattern adalah sebuah struktur data *map* yang memasangkan nilai interval dengan nilai *boolean*. Ketiga, sebuah variabel Times yang merupakan larik berisi angka. Indeks larik merepresentasikan waktu, sedangkan elemen larik ke-i merepresentasikan not yang akan ditempatkan pada waktu ke-i.

Fungsi pembatas dinamakan dengan fungsi *Verify*. Di dalam fungsi pembatas *Verify*, terdapat sebuah *loop* besar yang bertugas untuk mengecek apakah terdapat nada yang sama sebelumnya dengan nada yang sekarang. Bila ada yang sama, *loop* langsung berhenti dan ditandai sebagai tidak sahih. Selain untuk mengecek kesamaan dengan nada sebelumnya, pola interval juga harus dicocokan. Untuk mengecek interval yang sudah pernah digunakan, pertama harus diketahui jarak antara not sekarang dengan not sebelumnya. Hasil selisihnya digunakan sebagai key untuk mengakses apakah interval sudah digunakan atau belum.

Ketika fungsi dipanggil, fungsi akan menerima sebuah masukan berupa bilangan bulat yang melambangkan waktu ke berapa dalam larik Times yang akan diverifikasi. Berikut adalah pseudocode dari fungsi tersebut.

```
function Verify(time: integer) → boolean
{Fungsi pembatas yang menerima sebuah note}
{Cek apakah not pada time sahih atau tidak}
KAMUS
   available : boolean
   stop      : boolean
   i         : integer
ALGORITMA
   available ← True
   stop ← False
   i ← 1
   while i < time and not(stop) do
      if Times[i] = Times[time] then
         available ← False
         stop ← True
      else
         i ← i + 1
   {i = time or stop}

   if time > 1 then
      available ← available and
         not(Pattern[Times[time] -
         Times[time-1]])
      {cek apakah interval sudah digunakan}

   return available
```

Fungsi terakhir adalah fungsi untuk mencari solusi dari *n* buah nada. Terdapat sebuah *loop* besar yang akan melakukan iterasi hingga solusi ditemukan. Di dalam *loop* tersebut, terdapat sebuah *loop* lagi yang akan digunakan untuk mendapatkan not yang tepat pada sebuah satu satuan waktu. Pembangkitan not ini memanggil fungsi *Verify* agar not yang dibangkitkan hanya yang menuju solusi. Jika didapat not yang sahih, maka interval dari not yang baru tersebut terhadap not sebelumnya akan dicatat, yaitu dengan mengubah *map* dengan *key* interval baru tersebut menjadi *true*.

Akhir dari fungsi ini akan mengecek kondisi berhenti. Bila seluruh larik waktu sudah terisi, maka solusi didapatkan dan program akan mengembalikan nilai banyaknya runut balik yang dilakukan. Jika belum seluruh larik waktu terisi, maka fungsi akan melanjutkan iterasi ke satuan waktu yang berikutnya. Namun, ketika seluruh not sudah dicoba untuk suatu satuan waktu dan ternyata tidak ditemukan yang cocok, maka program akan melakukan runut balik ke satuan waktu sebelumnya sambil menambahkan perhitungan banyaknya runut balik yang dilakukan. Sebelum dilakukan runut balik, interval yang tadinya sudah digunakan harus diinisiasi menjadi *false* kembali.

Saat fungsi dipanggil untuk mulai komputasi, selalu masukan parameternya dengan satu (1) yang berarti fungsi akan memulai dari waktu pertama. Berikut adalah fungsinya dalam pseudocode.

```
function Compute(time: integer) → integer
{Fungsi pencari solusi}
{Mengembalikan jumlah backtrack yang dilakukan}
KAMUS
  stop  : boolean
  back  : integer
ALGORITMA
  stop ← False
  back ← 0
  while not(stop) do
     if setelah backtrack then
        Interval dari nada yang telah
        digunakan sebelumnya
        dijadikan False
        Times[time] ← Times[time] + 1

     while (not(Verify(time)) and
            masih ada not) do
        Times[time] ← Times[time] + 1
     {Verify(time) or not habis}

     if dapat tempat then
        Pattern[Times[time] -
                Times[time-1]] ← True
        {Pattern ditandai sudah dipakai}

     if Times[time] <= n then
        if Times[time] = n then
           write('Hasil ditemukan')
           return back
        else  {maju ke waktu selanjutnya}
           time ← time + 1
           Times[time] ← 1  {inisiasi}
     else if Times[time] > n and time = 0 then
        write('Solusi tidak ditemukan')
        return -1
     else  {harus backtrack}
        Ganti pattern yang sudah diset True
        pada iterasi ini kembali jadi False
        Times[time] ← 1
        time ← time - 1
        back ← back + 1
```

IV. Hasil Percobaan

Penulis telah membuat program yang menerapkan pseudocode pada bab sebelumnya dengan bahasa pemrograman Python. Berikut adalah pembahasan mengenai hasil keluaran, performansi program dengan masukan yang bervariasi, serta analisisnya.

A. Hasil Keluaran Program

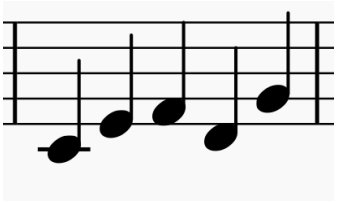
Pertama, program akan meminta masukan pengguna, lalu program akan mengeluarkan waktu eksekusi, jumlah backtrack, dan solusi yang dihasilkan.

```
D:\>python freepattern.py
Input number of notes = 5
Executed in 0.0 seconds
Number of backtracks 8

time : note
1 : 1
2 : 3
3 : 4
4 : 2
5 : 5
```

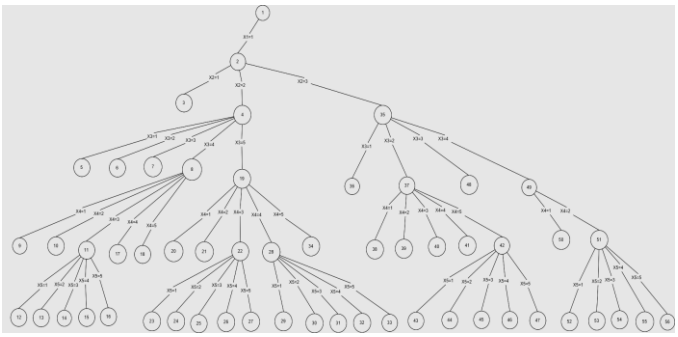
Gambar 5 Hasil keluaran program dalam bahasa Python

Hasil susunan 5 not secara berurutan adalah 1, 3, 4, 2, dan 5. Interval yang digunakan dalam susunan ini adalah 2 (3 kurang 1), 1 (4 kurang 3), -2 (2 kurang 4), dan 3 (5 kurang 2).



Gambar 6 Hasil susunan 5 not dalam not balok, dimainkan dalam tangga nada C mayor

Jika digambar pohon ruang statusnya, setiap mencapai simpul yang tidak bisa dibangkitkan lagi, maka akan dilakukan backtracking. Jika masih dapat dibangkitkan, maka pembangkitan diteruskan. Hal tersebut adalah penyebab mengapa algoritma backtracking menghasilkan simpul yang sangat banyak. Pencarian yang diutamakan adalah ke dalam, bukan ke samping (*Depth First*). Dalam kasus 5 buah not saja, simpul yang dihasilkan dalam satu kali menjalankan program adalah sebanyak 56 simpul. Namun, jika dibandingkan dengan



Gambar 7 Pohon ruang status dari 5 not

algoritma *brute-force*, maka algoritma runut balik telah memotong hampir setengah dari seluruh kemungkinan solusi (banyaknya solusi = n! = 5! = 120).

B. Tabel Data Waktu Eksekusi dan Jumlah Runut Balik

Penulis melakukan pengujian program dari masukan 4 sampai dengan 27. Di atas masukan sebesar 27, waktu eksekusi menjadi terlalu lama dan jumlah backtrack sudah lebih dari 30 juta kali. Berikut adalah hasil yang didapat.

Masukan	Waktu (skon)	Jumlah Backtrack
4	0	0
5	0	8
6	0	4
7	0	10
8	0	9
9	0	20
10	0.00099	18
11	0.00399	258
12	0.00600	291
13	0.03400	1172
14	0.00300	38
15	0.03600	1519
16	0.00900	353
17	0.0420	1461
18	0.05399	1924
19	0.737999	25142
20	1.05400	31591
21	5.66200	165372
22	2.25300	59248
23	19.8610	486984
24	29.05299	684298
25	495.5480	11749653
26	61.02399	1360371
27	1468.65799	30151276

Tabel 1 Tabel eksekusi dari beberapa masukan yang menunjukan waktu dan jumlah backtrack yang bervariasi

Eksekusi program dilakukan oleh interpreter Python 2.7.13 dengan sistem operasi Windows 10. Pengambilan waktu dilakukan untuk eksekusi fungsi **Compute** saja. Dilihat dari tabel, waktu eksekusi naik sangat signifikan ketika jumlah masukan sudah di atas 20. Hal ini dikarenakan runut balik yang dilakukan oleh program sangat banyak ketika jumlah masukan semakin besar. Namun, tidak seluruh hasil eksekusi program memiliki waktu eksekusi yang monoton naik. Fenomena ini dapat terlihat dalam jumlah masukan dari 13, 14, 15 dan 25, 26, 27. Fenomena tersebut dapat terjadi ketika program mendapatkan jumlah runut balik yang sedikit, atau dengan kata lain kasus yang dikerjakan adalah kasus terbaik (*best case*).

C. Analisis

Dalam fungsi **Compute**, banyaknya iterasi *loop* besar sebanyak not yang dimasukan. Di dalam setiap iterasi ke-i, terdapat sebuah *loop* yang mencari not yang tepat untuk waktu tersebut. Setiap pencarian not dimulai dari not pertama lagi, sehingga banyaknya pengulangan sebanyak i – 1 kali karena not ke-i tidak dicek. Sehingga, untuk banyaknya perbandingan sejauh ini dapat diformulasikan sebagai berikut.

$$T(n) = 1.0 + 2.1 + 3.2 + \cdots + (n-1).(n-2) + n.(n-1)$$

Untuk setiap iterasi yang mencari not, dilakukan pemanggilan fungsi **Verify** yang akan menguji kelayakan dari not yang dipilih pada waktu tersebut. Banyaknya perbandingan dalam fungsi pembatas tersebut adalah sebanyak i – 1, yaitu mengecek dari satuan waktu terkecil hingga ke waktu ke i – 1. Maka, formulasi T(n) diperbaiki menjadi sebagai berikut.

$$T(n) = 1.0.0 + 2.1.1 + 3.2.2 + \cdots + (n-1).(n-2).(n-2) + n.(n-1).(n-1)$$

Maka, untuk kasus di mana tidak terjadi runut balik sama sekali atau kasus terbaik (*best case*), program memiliki kompleksitas polinomial $O(n^3)$. Hal ini sangat cepat bila dibandingkan dengan algoritma *brute-force* yang mencoba segala kemungkinan, yaitu jika ada *n* buah not yang ingin disusun, maka ada n! kemungkinan yang dapat dicoba.

Selain itu, kasus terburuk dari program ini terjadi ketika jumlah *backtrack* sangat banyak sehingga pembangkitan not menjadi sangat banyak. Banyaknya kemungkinan yang dicek memiliki jumlah yang sama dengan algoritma *brute-force*, yaitu sebanyak n!. Sehingga, dapat dikatakan kompleksitas program untuk kasus terburuk adalah eksponensial, yaitu $O(n!)$.

KESIMPULAN

Pembuatan musik terburuk di dunia atau musik bebas pola dapat dilakukan dengan menerapkan algoritma runut balik. Kompleksitas algoritma untuk kasus terbaiknya adalah $O(n^3)$, sedangkan untuk kasus terburuknya memiliki kompleksitas $O(n!)$. Setelah dilakukan pengujian, program sangat baik untuk mencari susunan not bebas pola dengan jumlah not yang kurang dari 20. Namun, ketika masukan diperbesar, maka jumlah runut balik menjadi sangat banyak dan waktu eksekusinya menjadi sangat lama dan tidak wajar. Untuk itu, perlu dicari suatu algoritma lain untuk menyelesaikan persoalan ini. Beberapa algoritma yang dapat diterapkan antara lain algoritma *Branch and Bound*, *Greedy Best First Search*, dan penerapan Teorema Bilangan Prima (algoritma ini telah digunakan oleh penemu persoalan musik bebas pola, Scott Rickard).

UCAPAN TERIMA KASIH

Pertama, penulis ingin mengucap syukur kepada Tuhan Yang Maha Esa atas rahmatNya sehingga makalah ini dapat terselesaikan dengan baik. Terima kasih juga penulis sampaikan kepada Bapak Rinaldi Munir, Ibu Masayu Leylia Khodra, dan

Ibu Nur Ulfa Maulidevi selaku dosen pengajar IF2211 Strategi Algoritma dan sumber inspirasi untuk penulisan makalah yang membahas tentang algoritma runut balik ini.

REFERENSI

[1] Munir, Rinaldi. 2009. *Strategi Algoritma*. Bandung: Penerbit Institut Teknologi Bandung.

[2] Conservapedia. 2016. *Music Interval*. http://www.conservapedia.com/Interval_(music), diakses 15 Mei 2017.

[3] Phenotype, Steven. 2011. *Backtracking Visualisation*. https://www.w3.org/2011/Talks/01-14-steven-phenotype/backtracking.png, diakses 15 Mei 2017.

[4] Kumar, Shailendra. 2014. *The Basics For Guitar Lesson 1*. http://www.thebasicsofguitar.com/lesson-1-eng/, diakses 15 Mei 2017.

[5] TEDxTalks. 2011. *The world's ugliest music | Scott Rickard | TEDxMIA*. https://www.youtube.com/watch?v=RENk9PK06AQ, diakses 17 Mei 2017.

[6] Liszewski, Andrew. 2011. *Only a Mathematician Could Love the World's Ugliest Music*. http://gizmodo.com/5856721/only-a-mathematician-could-love-the-worlds-ugliest-music, diakses 17 Mei 2017.

[7] TSP. 2015. *The World's Ugliest Music by Scott Rickard (Full Transcript)*. https://singjupost.com/the-worlds-ugliest-music-by-scott-rickard-full-transcript/, diakses 17 Mei 2017.

[8] Wikipedia. 2016. Tangga Nada. https://id.wikipedia.org/wiki/Tangga_nada, diakses 18 Mei 2017.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Mei 2017

Aldrich Valentino Halim
13515081