Penerapan Algoritma Pattern Matching untuk Mengidentifikasi Musik *Monophonic*

Fahziar Riesad Wutono (13512012)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512012@std.stei.itb.ac.id

***Abstract*—Pattern matching atau pencocokan string adalah sebuah proses untuk mencari dan menemukan sebuah *pattern* atau pola di dalam string yang lebih panjang. Pattern di sini tidak harus berupa kata atau kalimat. Namun, tipe dari pattern dan string yang lebih panjang harus bersesuaian. Musik dapat diidentifikasi dengan memanfaatkan *pattern matching*. Makalah ini akan membahas bagamaina memanfaatkan algoritma *pattern matching* untuk mengidentifikasi musik.**

***Index Terms*—Identifikasi Musik Monophonic, Pattern Matching.**

I. PENDAHULUAN

Saat ini musik semakin sering dimainkan. Dengan adanya pemutar musik yang kecil dan dapat dibawa kemana saja seseorang dapat mendengarkan musik dimanapun. Selain menggunakan alat yang khusus digunakan untuk memainkan musik, alat yang tidak dikhususkan untuk memainkan musik dapat digunakan untuk memainkan musik. Alat seperti telepon genggam atau komputer tablet dapat dapat digunakan untuk memainkan musik.

Pada pemutar musik tersebut, musik disimpan dalam sebuah media penyimpanan. Media penyimpanan ini dapat berupa CD, DVD, kaset atau flash memory. Saat pengguna ingin memainkan sebuah musik, alat pemutar musik me-load musik pada media penyimpanan kemudian memainkannya. Dengan menggunakan teknik seperti ini, pengguna dapat memainkan musik sesuai dengan pilihan pengguna dan kapanpun ia inginkan.

Selain memainkan musik yang disimpan, beberapa pemutar musik dapat memainkan musik melalui radio. Musik tidak disimpan di pemutar musik. Musik disimpan oleh stasiun radio kemudian stasiun radio memainkan musik tersebut. Pemutar musik menangkap sinyal radio dan mengubah sinyal radio tersebut menjadi suara yang dapat didengar manusia.

Kedua metode ini memiliki kelebihan serta kekurangan. Metode pertama miliki kelebihan pengguna dapat memilih musik yang ingin dimainkan. Selain itu pengguna juga dapat memainkan musik yang diinginkan kapanpun ia mau. Namun pengguna harus membeli musik terlebih dahulu sebelum dimainkan. Pada cara yang kedua, pengguna tidak perlu membeli musik terlebih dahulu. Namun, pengguna tidak dapat memilih musik yang akan dimainkan. Musik yang dimainkan dipilih oleh stasiun radio.

Karena pemilihan musik dipilih oleh stasiun radio, terkadang saat mendengarkan musik melalui radio seseorang dapat menemukan sebuah musik baru yang bagus dan belum ia kenal. Agar dapat memainkan musik itu kapanpun, ia harus membeli musik itu dan menyimpan di pemutar musik. Untuk membelinya, tentu saja ia harus mengetahui identitas dari musik, misalnya judul lagu atau komposer. Namun pendengar belum tentu mengetahui judul tersebut. Disinilah program pengidentifikasi musik berguna.

Program identifikasi musik menerima input berupa beberapa bagian dari musik yang akan diidentifikasi yang menjadi *pattern* yang akan dicari. Setelah itu, program mencari musik yang mengandung *pattern* input. Selain mengandung seluruh bagian musik, basis data tersebut juga mengandung identitas musik. Dari situ didapat identitas dari musik yang ingin dicari.

Ada dua jenis musik dilihat dari jumlah nada yang dimainkan secara bersamaan. Salah satu jenis tersebut adalah musik *monophonic*. Pada musik *monophonic*, hanya satu nada yang dimainkan secara bersamaan. Makalah ini hanya membahas bagaimana cara untuk mengidentifikasi musik *monophonic* dengan menggunakan algoritma *pattern matching.*

Untuk menyederhanakan masalah, makalah ini membahas pengidentifikasian musik yang sudah berbentuk teks. Jika musik masih berbentuk suara, musik tersebut perlu dikonversi terlebih dahulu ke dalam bentuk teks.

II. PATTERN MATCHING

Pattern matching, disebut juga *string* matching atau pencocokan string, adalah proses untuk mencari sebuah *pattern* pada sebuah teks. Di sini ada dua buah istilah penting yang menjadi bagian dari *pattern matching*, yaitu *pattern* dan teks.

Istilah pertama yaitu *pattern. Pattern* adalah sesuatu yang akan dicari di dalam teks. *Pattern* yang beradai di

dalam teks disebut dengan target.

Istilah kedua yaitu teks. Teks adalah sebuah string dimana pattern ingin dicari didalamnya. Teks harus lebih panjang dari *pattern*. Jika teks lebih sedikit dari *pattern* dapat dipastikan *pattern* tidak dapat ditemukan.

Persoalan *pattern matching* adalah diberikan sebuah *pattern* yang panjangnya n karakter. Carilah *pattern* tersebut dalam teks, dimana panjang dari teks lebih besar dari n[1].

Meskipun digunakan istilah teks, teks di sini tidak harus berisi sekumpulan karakter. Teks di sini dapat berupa gambar, suara atau data yang tidak berbentuk teks. Namun, tipe dari *pattern* dan teks harus saling kompatibel. Meskipun *pattern* dan teks bukan berisi karakter, dalam melakukan *pattern matching pattern* dan teks diperlakukan seperti teks yang isinya adalah huruf-huruf seperti teks biasa

. *Pattern matching* ada dua jenis, yaitu *pattern matching* yang tipenya adalah *exact matching* dan tipenya *approximate matching.* Pada *exact matching, pattern* harus sama persis dengan target di teks. Berbeda dengan *exact matching,* pada *approximate matching pattern* tidak harus sama persi dengan target yang ada di teks. Pada *approximate matching,* ditentukan sebuah *threshold* atau ambang batas minimum untuk sebuah bagian dari string menjadi target. Dengan begitu, target dari *approximate matching* tidak perlu sama persis dengan *pattern.*

A. Algoritma Pattern Matching dengan Sifat Exact Matching

Algoritma *pattern matching* yang sifatnya adalah*exact matching* ada beberapa jenis. Beberapa diantaranya adalah algoritma *brute force*, algoritma Knuth-Morris-Pratt dan algoritna Boyer-Moore. Algoritma dengan kompleksitas terburuk adalah *brute force.* Algoritma Knuth-Morris-Pratt dan Boyer-Moore adalah perbaikan dari algorituma *brute force.*

1. Algoritma *brute force*

Algoritma *brute force* memiliki kompleksitas o(nm). Penerpaan algoritma *brute force* adalah sebagai berikut:

1. *Pattern* dicocockan dengan bagian awal teks.
2. *Pattern* dikatakan telah menemukan target jika setiap karakter *pattern* telah cocok dengan sebagian dari teks.
3. Jika ditemukan ketidakcocokan, geser *pattern* sebanyak satu karakter.

Algoritma ini dijalankan hingga sebuah target pertama ditemukan atau seluruh teks teks telah diperiksa. Jika seluruh teks telah diperiksa, berarti *pattern* tidak ditemukan di dalam teks.

2. Algoritma Knuth-Morris-Pratt

Pada algoritma Knuth-Morris-Pratt, *pattern* dicocokan seperti pada algoritma *brute force*. Perbedaannya yaitu algoritma Knuth-Morris-Pratt menghindari *wasteful comparison.* Pada algoritma *brute force*, setiap ditemukan ketidakcocokan *pattern* hanya dimajukan satu karakter. Berbeda dengan *brute force*, algoritma Knuth-Morris-Pratt memajukan *pattern* sejauh beberapa karakter.

Untuk mengetahui seberapa jauh *pattern* dapat dimajukan, digunakan fungsi pinggiran. Fungsi pinggiran bergantung pada karakter yang berada di *pattern.*

Fungsi pinggiran biasa ditulis b(k). Nilai k adalah posisi dimana *pattern* tidak cocok dengan teks. Nilai k=1 atau k=0 adalah posisi karakter pertama pada *pattern,* tergantung dari bahasa pemrograman yang digunakan. Nilai k=j-1 atau k=j adalah posisi karaktr terakhir pada *pattern.*

Misalkan ada sebuah *pattern* P. Nilai b(k) adalah panjang substring terpanjang dari P[1..k] yang menjadi *prefix* atau awalan dari string dan juga menjadi *suffix* atau akhiran dari string. Jika posisi karakter pertama pada string adalah 0, maka definisi b(k) dapat diubah menjadi nilai substring terpanjang dari P[0..k-1] yang menjadi *prefix* atau awalan dari string dan juga menjadi *suffix* atau akhiran dari string.

Nilai dari b(k) tidak perlu dihitung setiap kali dilakukan pencocokan karakter. Nilai ini cukup dihitung sekali saja di awal proses *pattern matching.* Pada awal proses pencocokan string, nilai b(k) untuk seluruh k dihitung. Hasil penghitungannya disimpan dalam sebuah tabel atau *array*. Setiap kali ditemukan ketidakcocokan, nilia dari b(k) bisa didapatkan dengan hanya melihat hasil penghitungan di awal [2].

Sebagai contoh, misal ada *pattern* abcabc. Tabel satu berisi nilai fungsi pinggiran dari *pattern* tersebut.

j	1	2	3	4	5	6
b(j)	0	0	0	1	2	3

Tabel 1. Nilai fungsi pinggiran dari *pattern* abcabc

Setiap ditemukan ketidakcocokan, *pattern* tidak harus digeser 1 karakter. Jumlah pergeserannya ditentukan oleh fungsi pinggiran. Perhitungannya yaitu:

$$jumlah\ pergeseran = k - b(j-1) \quad (1)$$

disini k adalah panjang dari *pattern.* Nilai j adalah posisi dimana ketidakcocokan ditemukan.

Kompleksitas dari algoritma Knuth-Morris-Pratt adalah o(n+m)[3]. Kompleksitas ini lebih baik dibandingkan dengan algoritma *brute-force*.

4. Algoritma Boyer-Moore

Berbeda dengan algoritma *brute-force* dan Knuth-Morris-Pratt, pada algoritma Boyer-Moore, *pattern* dibandingkan mulai dari karakter paling belakang dari *pattern*. Mirip seperti algoritma Knuth-Morris-Pratt, pada algoritma Boyer-Moore *pattern* dapat digeser lebih dari

satu karakter setiap ditemukan ketidakcocokan. Hal ini membuat algoritma Boyer-Moore lebih efisien dibandingkan dengan algoritma *brute-force*.

Langkah-langkah *pattern matching* dari algoritma Boyer-Moore adalah sebagai berikut:

1. Posisikan *pattern* di awal teks.
2. Bandingkan setiap karakter di *pattern* dengan karakter di teks dimulai dari karakter paling belakang *pattern*. Lakukan hingga seluruh karakter pada *pattern* telah dibandingkan atau ditemukan ketidakcocokan pada karakter *pattern* dengan karakter pada teks.
3. Jika seluruh karakter *pattern* sudah cocok dengan teks, berarti target telah ditemukan. Jika tidak, geser pattern dengan jumlah pergeseran sesuai dengan hasil perhitungan fungsi *last occurence*. Jika *pattern* sudah berada di ujung text, berarti *pattern* tidak terdapat pada teks.

Pergeseran algoritma Boyer-Moore bergantung pada fungsi *last occurence*. Fungsi *last occurence* menerima masukan sebuah karakter dan mengembalikan posisi karakter paling akhir dari karakter di dalam *pattern*. Jika karakter tersebut tidak ada pada *pattern*, fungsi *last occurence* mengembalikan nilai -1.

Fungsi *last occurence* tidak perlu dihitung setiap kali dilakukan pencocokan string. Fungsi *last occurence* cukup dihitung sekali di awal untuk setiap karakter pada *character set*. Hasilnya disimpan di dalam sebuah tabel atau *array*.

Misal ada sebuah pattern abcabc. Sebagai contoh, nilai fungsi *last occurence* dari *pattern* tersebut dapat dilihat di tabel 2. Untuk menyingkat tabel, tabel hanya berisi karakter yang terdapat pada *pattern*. Nilai karakter lainnya adalah -1. String pada *pattern* tersebut dimulai dari angka 1, bukan 0.

Karakter	a	b	c
Last occurence	4	5	6

Tabel 2. Nilai fungsi *last occurence* dari *pattern* abcabc

Misal x adalah karakter pada teks yang tidak cocok dengan karakter pada pattern, cara penghitungan jumlah pergeseran adalah sebagai berikut[4]:

1. Jika *last occurence* dari karakter x di sebelah kiri posisi saat ini, geser *pattern* sehinga posisi x sama posisi *last occurence* x di *pattern*.
2. Jika *last occurence* dari karakter x di sebelah kanan posisi saat ini, geser *pattern* sejauh satu karakter.
3. Jika karakter x tidak ada di dalam *pattern*, geser *pattern* sebanyak jumlah karakter *pattern*.

Kompleksitas dari algoritma Boyer-Moore adalah O(m + n)[5]. Kompleksitas terburuk dari algoritma Boyer-Moore adalah O(nm + A)[4]. Jika n sama dengan m, kompleksitasnya menjadi $O(n^2 + A)$, lebih buruk daripada *brute* force.

B. Algoritma Pattern Matching dengan Sifat Approximate Matching

Pada algoritma *pattern matching* yang sifatnya *approximate matching*, target tidak perlu tepat sama dengan *pattern*. Perbedaan satu atau dua karakter target dapat dikatakan sama dengan *pattern*. Hal ini membantu jika karakter pada target tidak diketahui secara persis urutan dan komposisinya.

Ada beberapa algoritma *pattenr matching* yang sifatnya adalah *approximate matching*. Salah satu diantaranya adalah *pattern matching* dengan memanfaatkan *edit distance*.

Definisi dari *edit distance* adalah diberikan dua buah string, misal string a dan string b, *edit distance* dari a dan b adalah jumlah *point mutation* minimum untuk mengubah dari string a ke string b [6]. Istilah lain dari *edit distance* adalah *string metric*.

Salah satu istilah kunci dari *edit distance* adalah *point mutation*. Yang termasuk ke dalam *point mutation* adalah:

1. Perubahan karakter, misalnya string *wake* menjadi string *woke*.
2. Penambahan karakter, misalnya string *buy* menjadi *buys*.
3. Penghapusan karakter, misalnya string *buys* menjadi *buy*.

Edit distance dapat digunakan untuk merepresentasikan sebarapa besar perbedaan antara dua buah string. Dua buah string dikatakan sama jika *edit distance* dua buah string tersebut adalah 0. Semakin besar nilai *edit distance*, semakin besar perbedaan antara dua buah string.

Pattern matching dapat dilakukan dengan memanfaatkan *edit distance*. *Pattern matching* dilakukan dengan menggabungkan algoritma *brute force* dan *edit distance*.

Pada dasarnaya *pattern matching* dengan *edit distance* mirip dengan algoritma *pattern matching brute force*. Namun, operasi perbandingan diganti dengan perhitungan nilai fungsi *edit distance* antara pattern dan bagian teks yang dibandingkan. Setiap kali bagian teks dikatakan tidak sesuai, *pattern* dimajukan satu karakter.

Sebelum proses *pattern matching* dimulai, *threshold* atau ambang batas agar sebuah bagian dari string dinyatakan sesuai dengan *pattern* ditentukan. Ambang batas dapat dinyatakan dengan nilai eksak yang menyatakan nilai maksimum *edit distance*. Selain itu, ambang batas dapat dinyatakan dalam persen yang

menyatakan seberapa tinggi kesamaan antara dua buah string agar dikatakan. Sebelum *pattern matching* dimulai, amban batas dalam persen dikonversi terlebih dahulu ke dalam nilai eksak dengan rumus:

$$threshold = \frac{persen\ ambang\ batas}{100} \times panjang\ pattern \quad (2)$$

Pemberian ambang batas dengan menggunakan persen mempunyai keunggulan yaitu memberikan ambang batas yang lebih seragam untuk setiap *pattern.* Jika menggunakan nilai eksak, nilai ambang batas sebesar 1 memiliki akurasi berbeda bila digunakan dengan *pattern* dengan panjang 5 karakter dan *pattern* dengan panjang 10 karakter.

III. Istilah Dalam Musik

Ada beberapa istilah dalam musik. Istilah-istilah yang berkaitan dengan makalah ini antara lain nada, monophonic dan polyphonic.

A. Nada, Monophonic dan Polyphonic

Nada atau *pitch* adalah frekuensi yang dirasa dari sebuah not[7]. Secara sederhana, nada adalah seberapa tinggi atau rendah frekuensi dari sebuah not. Musik merupakan kumpulan dari nada.

Dilihat dari jumlah nada yang dimainkan secara bersama, musik dibagi menjadi dua jenis. Jenis pertama yaitu musik *monophonic* dan musik *polyphonic*.

Musik yang pertama adalah musik *polyphonic*. Pada musik jenis ini, dalam satu waktu terdapat beberapa nada yang dimainkan. Kata *polyphonic* berasal dari bahasa latin yang artinya "banyak suara". Sebagian besar musik yang didengar saat ini adalah musik *polyphonic*[7]. Contoh musik *polyphonic* ada pada figur 1. Bagian yang dikotakkan adalah bagian musik yang dimainkan secara bersama.

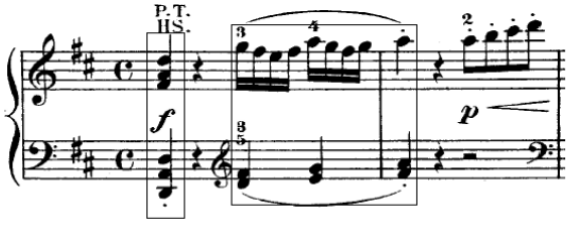


Fig. 1. Sebuah musik *polyphonic*. (Sumber: A Geometric Approach to Pattern Matching in Polyphonic Music, Luke Andrew Tanur)

Musik yang kedua yaitu musik *monophonic* adalah musik dimana dalam satu waktu hanya dimainkan satu buah nada. Contoh musik *monophonic* ada pada figur 2. Figur ini berisi potongan not balok dari lagu "Ibu Kita Kartini". Musik jenis ini yang akan dibahas lebih lanjut dalam makalah ini.

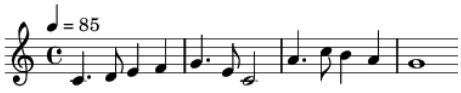


Fig. 2. Sebuah musik *polyphonic*, potongan dari lagu "Ibu Kita Kartini". (Sumber: id.wikipedia.org)

B. Notasi Musik

Ada beberapa cara untuk menuliskan musik ke dalam sebuah media. Salah satunya adalah dengan menggunakan notasi not balok. Not balok merupakan cara yang bagus untuk menulis musik ke dalam sebuah media karena not balok dapat melambangkan nada, panjang nada dan bagaimana nada tersebut dimainkan.

Namun not balok memiliki kelemahan. Melakukan *pattern matching* pada not balok sulit untuk dilakukan. Untuk memudahkan sebaiknya not balok diubah terlebih dahulu ke dalam sesuatu yang dapat direpresentasikan oleh string.

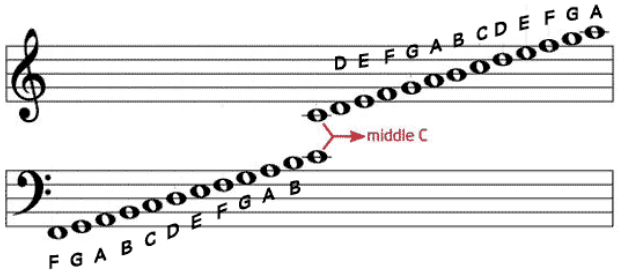


Fig. 3. Nama nada-nada pada not balok. (Sumber: www.ace-your-audition.com)

Cara yang pertama untuk mengubah not balok ke dalam yaitu menuliskan nada-nada pada not balok dalam bentuk nama nadanya. Pemetaan not balok menjadi huruf-huruf dapat dilihat di figur 3. Cara ini mempunyai kelemahan yaitu menghilangkan sejumlah informasi yang mampu disimpan oleh not balok. Informasi tersebut antara lain panjang pendeknya nada serta bagaimana nada tersebut dimainkan. Contoh hasil perubahan lagu "Ibu Kita Kartini" dari figur dua adalah sebagai berikut:

CDEFGEC

Cara kedua adalah menuliskannya dalam bentuk not angka. Cara ini mirip dengan cara sebelumnya, namun not bukan dituliskan dalam bentuk huruf melainkan dalam bentuk angka.Nada C dalam notasi ini adalah 1, nada D adalah 2, nada E adalah 3, nada F adalah 4 dan seterusnya hingga nada B adalah 6. Contoh hasil perubahan lagu "Ibu Kita Kartini" dari figur dua dalam not angka adalah sebagai berikut:

Kelemahan dari cara yang sama dengan kelemahan dari cara pertama, yaitu ada sejumlah informasi yang hilang.

IV. Penerapan Pattern Matching Pada Untuk Identifikasi Musik

Pattern matching, baik *exact matching* maupun *approximate matching* dapat diterapkan untuk mengidentifikasi musik. Sebelum proses *pattern matching* dimulai, musik harus dikonversi terlebih dahulu ke format yang dapat dilakukan *pattern matching* dengan mudah.

Input dari proses ini dapat berupa suara atau sudah berbentuk not balok. Jika input berupa not balok, proses konversi mirip seperti proses konversi pada bab III. Jika input dari proses ini sudah dalam bentuk yang mudah untuk dilakukan *pattern matching* maka input tidak perlu diubah. Input dikatakan bentuknya mudah untuk dilakukan *pattern matching* jika input tersebut sudah berbentuk string.

Setelah konversi berhasil dilakukan, proses *pattern matching* baru bisa dilakukan. Baik *exact matching* maupun *approximate matching* baru bisa dilakukan setelah inputnya sudah dalam berbentuk string.

A. Identifikasi dengan Menggunakan Algoritma Exact Matching

Identifikasi musik dengan menggunakan algoritma *exact matching* melibatkan algoritma *exact matching* dan sebuah basis data. Basis data ini berisi identitias dari lagu, misalnya judul lagu dan komposernya, serta seluruh nada pada musik. Nada ini sudah berbentuk string yang diproses dengan aturan yang sama pada bab III.

Proses identifikasinya yaitu:

1. Ambil entri pertama dari basis data.
2. Proses *pattern matching* dengan *pattern* adalah input dari proses ini dan teks adalah seluruh nada dari musik.
3. Jika target pada teks ditemukan, maka musik berhasil diidentifikasi. Ambil identititas musik dan keluarkan sebagai output proses identifikasi. Jika target tidak ditemukan, identitas dari musik belum ditemukan. Ambil entri musik selanjutnya kemudian ulangi poin dua dan tiga. Jika sudah tidak ada entri musik yang belum diperiksa di basis data, berarti musik gagal diidentifikasi.

Pada proses *pattern matching*, algoritma yang dipilih adalah algoritma *pattern matching* yang sifatnya adalah *exact matching*. Baik algoritma *brute-force*, Knuth-Moris-Pratt maupun Boyer-Moore dapat digunakan.

Untuk mempercepat pencarian, algoritma yang disarankan adalah algoritma Knuth-Moris-Pratt atau Boyer-Moore. Hal ini karena baik algoritma Boyer-Moore maupun Knuth-Moris-Pratt mempunyai kompleksitas yang lebih kecil dibandingkan dengan algoritma *brute-force*.

B. Identifikasi dengan Menggunakan Algoritma Approximate Matching

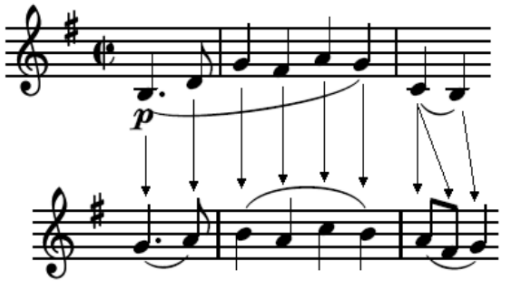
Algoritma *pattern matching* yang sifatnya adalah *exact matching* memiliki kelemahan. Algoritma-algoritma tersebut menuntu target dan *pattern* harus sama persis.

Ketika seseorang melakukan konversi sebuah musik ke dalam bentuk not balok atau not angka, orang tersebut hanya mengira-ngira saja bagaimana bentuk not balok tersebut. Karena hanya mengira-ngira saja, not balok yang dihasilkan bisa saja sedikit berbeda dengan not balok yang dibuat komposer. Jika konversi menggunakan sebuah alat, bisa saja alat mempunyai galat sehingga tidak bisa menghasilkan not yang sama persis dengan not yang dibuat oleh komposer. Hal ini membuat *exact matching* tidak bisa menemukan *pattern* sama sekali[8].

Salah satu solusi dari masalah ini adalah dengan menggunakna *edit distance*. *Edit distance* memungkikan proses *pattern matching* menemukan sebuah target walaupun target tersebut tidak sama persis dengan *pattern*.

Proses identifikasi musik dengan menggunakan *edit distance* mirip dengan proses identifikasi musik dengan algoritma *exact matching*. Perbedaannya yaitu sebelum proses *pattern matching* dimulai ditentukan terlebih dahulu berapa perbedaan *pattern* dan target maksimal agar dapat dikatakan *pattern* dan target *match*. Kemudian setiap kali dilakukan *pattern matching*, hasil dari perhitungan *edit distance* dibandingkan dengan perbedaan maksimal antara target dan *pattern* yang sudah ditentukan di awal. Jika perbedaannya lebih sedikit, *pattern* dikatakan telah menemukan target.

Selain *edit distance* biasa, ada jenis *edit distance* yang dikhususkan untuk musik yang disebut *musical edit distance*. *Musical edit distance* pada dasarnya sama dengan *edit distance* biasa, namun pada *musical edit distance* terdapat tambahan operasi yaitu penggabungan beberapa not menjadi satu dan pemisahan satu not menjadi beberapa not. Operasi ini tidak bermakna jika digunakan pada string biasa. Setiap operasi ini memiliki *cost* yang berbeda. Perbedaan yang kedua yaitu *cost* dari setiap perubahan not dipengaruhi oleh seberapa jauh perbedaan *pitch* antara kedua not[8]. Contoh *musical edit distance* ada pada figur 4.



$$w_{interval}: \; 0.35 \quad 0.1 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad (0.35 + 0.5) \; 0.35$$

$$ED(A,B) = \sum_{w_{interval}} = 2.45$$

Fig. 4. Contoh operasi *musical edit distance*. Hasil dari operasi adalah jumlah *cost* dari sebuah interval. (Sumber: A Geometric Approach to Pattern Matching in Polyphonic Music, Luke Andrew Tanur)

V. KESIMPULAN DAN SARAN

Pattern matching dapat digunakan untuk identifikasi musik. Meskipun dapat digunakan, *pattern matching* untuk string tersebut perlu dimodifikasi agar memberikan hasil yang optimal.

Algoritma ini disarankan untuk digunakan bersama *pitch detection algorithm* yang mampu memperkirakan nada-nada dari lagu. Dengan menggunakan menggabungkan algoritma *pattern matching* untuk identifikasi musik dengan *pitch detection algorithm* proses identifikasi musik dapat berjalan secara otomatis.

REFERENSI

[1] Rinaldi Munir, *Dikat Kuliah IF2211 Strategi Algoritma*. Bandung: ITB, 2009, pp. 186

[2] Rinaldi Munir, *Dikat Kuliah IF2211 Strategi Algoritma*. Bandung: ITB, 2009, pp. 189-191

[3] stackoverflow.com/questions/9182651/

[4] informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/Pencocokan String (2014).pptx

[5] www.math.tau.ac.il/~haimk/seminar04/boyer-moore-algorithm.ppt

[6] www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Edit/

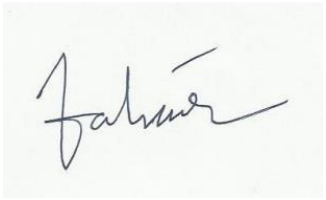
[7] cgm.cs.mcgill.ca/~athens/cs507/Projects/2004/Eric-Blais/definitions.html

[8] Luke Adrew Tanur, *A Geometric Approach to Pattern Matching in Polyphonic Music*, Waterloo:University of Waterloo,2004

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, 18 Mei 2014



Fahziar Riesad Wutono - 13512012